Structured Petri Nets for the Design and Implementation of Manufacturing Control Software with Fault Monitoring Capabilities

Thesis Submitted to the University of Wales for the Degree of

Doctor of Philosophy

By

Martin Stanton, BSc
Department of Engineering
University of Wales College, Newport

1998

Declarations

DECLARATION

This work has not previously been accepted in any substance for any degree and is not being currently submitted in candidature for any degree.

Signed ....[illegible]............ (candidate)

Date ....1/12/98............

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated.
Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ....[illegible]............ (candidate)

Date ....1/12/98............

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ....[illegible]............ (candidate)

Date ....1/12/98............

Acknowledgements

I would like to thank Arthur Buck for his advice, encouragement and constructive criticism throughout the project, Geoff Roberts and Hefin Rowlands for their friendship and support, Bill Arnold for his time and effort, and my fellow research students for their comradeship and discussion during our time in B14. Finally I would like to thank my wife Denise and my daughter Hannah for letting me work late and for wanting me to finish.

Summary

The thesis describes a method for the design and implementation of manufacturing control software using structured Petri nets. An earlier design method is presented from which a more formal approach is developed, and a definition for structured Petri nets is given. This definition is then compared to other classes of Petri net found in the literature. A comparison is also made between the proposed design method and other methods described in the literature. The structured Petri nets are then used to create a control structure, which is shown to have properties that allow the detection and diagnosis of faults originating both in the hardware and the software of the system. A detailed discussion is also presented concerning the implementation of structured Petri nets on various types of manufacturing controller and on general-purpose computers. In particular, results are presented from experiments with various implementation methods on programmable logic controllers. Conclusions are then drawn on the various aspects of the work and details of further research possibilities are described.

Table of Contents

Declarations i

Acknowledgements ii

Summary iii

Table of Contents iv

List of Figures xi

List of Tables xiv

Chapter 1 Introduction to the Research 1-1

1.1 Introduction 1-1

1.2 Objectives of the Research 1-2

1.3 Achieving the Objectives 1-2

1.4 Petri Nets 1-4

1.5 Application to Manufacturing Systems 1-5

1.6 Petri Nets for Control 1-7

1.6.1 Centralised control 1-7

1.6.2 Decentralised control 1-7

1.6.3 Different levels of control model 1-8

1.7 Overview of Thesis 1-8

1.8 References 1-10

Chapter 2 A Control Structure 2-1

2.1 Petri nets 2-1

2.1.1 Transition firing rule 2-2

2.2 The Control Structure 2-2

2.2.1 Places transitions and arcs 2-2

2.3 The Structure 2-4

2.3.1 Safety net 2-5
2.3.2 Linking the control net and the subnets 2-5
2.3.3 Subnet/Output net link 2-6
2.4 Implementation 2-7
2.5 Fault Monitoring 2-9
2.6 Limitations of the Current Method 2-10
2.6.1 The design method 2-10
2.6.2 Implementation 2-11
2.7 Important Properties 2-12
2.7.1 Size of the control code 2-12
2.7.2 Complexity 2-12
2.7.3 Fault diagnosis 2-13
2.8 Areas of Improvement 2-13
2.8.1 Modelling and control of complex concurrent systems 2-13
2.8.2 Structural analysis 2-13
2.8.3 Enhance fault diagnosis 2-14
2.8.4 Implementation on other types of controller 2-14
2.9 Chapter Summary 2-14
2.10 References 2-14

Chapter 3 Structured Petri Nets 3-1
3.1 Introduction 3-1
3.2 A Standard Petri Net Definition 3-1
3.2.1 Petri nets 3-2
3.3 Important Properties 3-2
3.3.1 Safeness 3-2
3.3.2 Liveness 3-3
3.3.3 Conflicts 3-3
3.4 Analysis 3-5
3.5 Marked Graphs 3-7
3.5.1 Definition 3-8
3.6 Decision Free Petri Nets 3-8
3.6.1 Operation and resource places 3-9
3.7 Free Choice Petri Nets 3-10
3.8 Petri Nets with External Inputs and Outputs 3-11
3.8.1 Controlled Petri nets 3-12
3.9 A Definition for Structured Petri Nets 3-13
3.9.1 Definition 3-13

3.10 Implications on Properties 3-15
3.10.1 Liveness 3-15
3.10.2 Safeness 3-16
3.10.3 Conflicts 3-17
3.11 Chapter Summary 3-17
3.12 References 3-18

Chapter 4 Petri Net Modules 4-1
4.1 Modelling a Pneumatic Cylinder 4-1
4.2 Monitoring and Control 4-3
4.2.1 Monitoring 4-4
4.2.2 Controlling the hardware 4-5
4.2.3 Controlling the software 4-6
4.3 Elements of Structured Petri Nets 4-8
4.4 Interpreting Net Elements 4-8
4.4.1 Transitions 4-9
4.4.2 Places 4-9
4.4.3 Control places 4-12
4.5 The Controller and its Environment 4-12
4.5.1 Hardware 4-13
4.5.2 Safety subsystem 4-13
4.5.3 Hardware and software I/O 4-14
4.5.4 Hardware controls 4-14
4.5.5 Software controls 4-15
4.5.6 Feedback places 4-15
4.6 Subnets 4-16
4.7 Creating a Hierarchical Structure 4-17
4.8 Joining Petri Net Modules 4-18
4.9 Chapter Summary 4-20
4.10 References 4-20

Chapter 5 Developing a Control Structure 5-1
5.1 The Aims of the Method 5-1
5.2 The Method 5-2
5.3 Specifications of Systems Tasks 5-3
5.3.1 The initialisation task 5-3
5.4 Defining the communications 5-4
5.4.1 Describing the interface 5-5

5.4.2 Control/feedback pairs 5-6

5.5 Decomposing the System into its Constituent Subsystems 5-8

5.5.1 Subsystem actions 5-10

5.6 Mapping Subsystem Actions to System Tasks 5-10

5.7 Constructing the Control Net 5-11

5.7.1 Describing the paths 5-12

5.7.2 Merging the paths 5-13

5.7.3 Ensuring the net is conflict free 5-13

5.8 Subsystem Development 5-16

5.9 Other Approaches to Controller Design 5-16

5.9.1 Venkatesh 5-17

5.9.2 Net condition/event systems 5-17

5.9.3 Zhou and DiCesare 5-18

5.9.4 Resource Control Nets 5-20

5.10 Chapter Summary 5-20

5.11 References 5-21

Chapter 6 Application of the Control Structure 6-1

6.1 Example: A Raw Materials Station 6-1

6.1.1 Function of the raw materials station 6-2

6.2 System Requirements of the Raw Materials Station 6-2

6.3 Defining the Communications 6-3

6.4 Breakdown of Subsystem Components 6-5

6.4.1 Effect of the number of subsystems on the control net 6-5

6.5 Mapping Systems and Subsystem Tasks 6-7

6.6 Creating the Petri Net 6-8

6.6.1 Describing the paths 6-8

6.6.2 Mapping tasks to places 6-9

6.6.3 Describing the transitions for the Petri net 6-10

6.7 Merging the Paths 6-13

6.7.1 Removing Decisions form the net 6-14

6.8 Generating the Subnets 6-15

6.8.1 The pallet manipulator subsystem 6-16

6.8.2 The loading bay subsystem 6-17

6.8.3 The cylinder storage subsystem 6-17

6.9 Designing the Output Nets 6-17

6.10 Software Design for a Manufacturing Cell 6-18

6.10.1 Development of the manufacturing cell control code 6-19

6.10.2 Designing the cell control net 6-20
6.10.3 Designing the robot control net 6-20
6.10.4 CNC control code 6-21
6.10.5 Implementation 6-21
6.10.6 Further developments and enhancements 6-21
6.11 Further Developments for the Control Structure 6-22
6.11.1 Shop floor controller 6-22
6.11.2 Computer Integrated Manufacture 6-23
6.12 Chapter Summary 6-24
6.13 References 6-25

Chapter 7 Implementation 7-1
7.1 Interpretation and Implementation 7-1
7.2 Methods of Implementation 7-2
7.2.1 The 'token player' 7-3
7.2.2 Structured program approach 7-4
7.3 Implementation on a PLC 7-4
7.4 Representation of a Petri Net by Ladder Logic 7-6
7.4.1 Implementation using ladder logic 7-7
7.5 The Need for a New Approach 7-8
7.5.1 Defining a 'correct' implementation 7-9
7.6 Testing the Implementation Methods 7-10
7.6.1 Results of applying this implementation method 7-12
7.6.2 A more 'correct' implementation 7-13
7.7 Testing the Implementations 7-16
7.7.1 Experiment 1 7-17
7.7.2 Experiment 2 7-19
7.7.3 Experiment 3 7-19
7.8 Comparing the results 7-20
7.9 Implementation in Higher Level Languages 7-23
7.9.1 Robot controller 7-23
7.9.2 Relational database 7-24
7.10 Chapter Summary 7-25
7.11 References 7-26

Chapter 8 Fault Monitoring 8-1
8.1 Faults and Failures 8-1
8.2 Hard and Soft Faults 8-2

8.3 Classifying Failures 8-3
8.3.1 Hardware failures 8-3
8.3.2 Software failures 8-4
8.3.3 Product faults 8-6
8.3.4 The need to reduce software faults 8-6
8.4 Fault prevention 8-7
8.4.1 Strutured programming 8-7
8.4.2 Parallelism 8-8
8.4.3 Modularity 8-8
8.5 Considerations for Monitoring 8-9
8.5.1 Initial state 8-9
8.5.2 Timing Information 8-10
8.6 A Fault Monitoring Architecture 8-11
8.7 Monitoring a Sequence 8-15
8.8 Space Cost of the Method 8-15
8.9 Fault Monitoring Mechanism 8-16
8.10 Implementation Issues 8-16
8.11 Dealing with Choices 8-17
8.11.1 Concurrency 8-17
8.12 Monitoring 8-17
8.13 Diagnosis and Containment 8-18
8.14 Using Standard Software 8-18
8.15 Chapter Summary 8-19
8-16 References 8-19

Chapter 9 **Conclusions and Further Research** 9-1
9.1 Structured Petri Nets 9-2
9.1.1 Relationship with Free Choice Petri Nets 9-2
9.2 Petri Net Modules 9-2
9.2.1 Centralised and distributed systems 9-4
9.2.2 Systems integration 9-4
9.3 The Design Method 9-5
9.4 Implementation 9-6
9.4.1 Implementation on a relational database 9-7
9.5 Fault Monitoring 9-7
9.6 Further Research 9-8
9.6.1 Structured Petri nets 9-9
9.6.2 Forbidden state problems 9-9

9.7 The Design Method 9-10
9.8 Implementation 9-10
9.8.1 Analysis of a PLC 9-11
9.8.2 Relational Database and CASE 9-11
9.8.3 Fault Monitoring 9-12
9.9 Work on Safety Systems 9-12
9.10 Chapter Summary 9-13
9.11 References 9-15

Appendix 1 Petri Nets for the Raw Materials Station A1-1

Appendix 2 Updated Net for the Pallet Manipulator A2-1

Appendix 3 Manufacturing Cell Documentation A3-1

Appendix 4 Papers A4-1

List of Figures

Figure 2.1 Symbols used in graphical representation of Petri nets 2-3
Figure 2.2 Petri net structure for a single machine 2-4
Figure 2.3 Control net/subnet link 2-6
Figure 2.4 Subnet/output net link 2-7
Figure 2.5(a) Petri net segment 2-8
Figure 2.5(b) Ladder representation for place p_3 2-8
Figure 2.6 Structure of the Ladder Logic program 2-9
Figure 3.1 A Petri net showing structural conflict when there is a choice of operations 3-4
Figure 3.2 The Petri net of Figure 3.1 with the conflict resolved 3-5
Figure 3.3 Example Petri net with external inputs and outputs 3-14
Figure 4.1 A simple Petri net description of a pneumatic cylinder 4-2
Figure 4.2 An extended Petri net description of the pneumatic cylinder 4-3
Figure 4.3 A Petri net with external inputs 4-5
Figure 4.4 A Petri net with both external inputs and outputs 4-6
Figure 4.5 A full Petri net description of a pneumatic cylinder controller 4-7
Figure 4.6 Symbols used in the graphical representation of structured Petri nets 4-9
Figure 4.7 A primitive place, p_1, is used to indicate that the system is ready 4-11
Figure 4.8 The relationship between a controller and its immediate environment 4-13
Figure 4.9 A control structure with three levels of control 4-17
Figure 4.10 A communicating pair of Petri net modules 4-18
Figure 5.1 Sequence of steps for the Petri net controller development 5-2
Figure 5.2 Control/Feedback Diagram for a system with many environmental entities 5-6
Figure 5.3 Action 2 is shared by both tasks and therefore causes a conflict between transitions t_5 and t_6. 5-14
Figure 5.4 Place p_7 now resolves the conflict 5-15
Figure 6.1 Layout of raw materials station 6-1
Figure 6.2 Communications between the raw materials station and its supervisory controller 6-4
Figure 6.3 Petri net structure for the Raw Materials Station 6-7
Figure 6.4 Initialisation path for the raw materials station 6-12
Figure 6.5 Paths describing the function of getting a block and pallet 6-12
Figure 6.6 Paths describing placing a cylinder from slope 1 and a pallet 6-12
Figure 6.7 The paths describing placing a cylinder from slope 2 and a pallet 6-13
Figure 6.8 Paths that describe getting an empty pallet and placing a loaded (or empty) pallet onto the conveyor 6-13

Figure 6.9 Control net for the raw materials station 6-14
Figure 6.10 The paths of Figure 6.5 with added state places 6-15
Figure 6.11 The path for getting an empty pallet with additional state place 6-15
Figure 6.12 Control net including additional state places 6-16
Figure 6.13 Control structure for a manufacturing cell 6-18
Figure 6.14 Layout of the Manufacturing Cell 6-20
Figure 6.15 Example of a Conveyor System 6-22
Figure 6.16 Proposed architecture for Computer Integrated Manufacturing System 6-23
Figure 7.1 Example net before the firing of transition t_2 7-8
Figure 7.2 Example net after the firing of transition t_2 7-9
Figure 7.3 Sample net used to test implementation methods 7-11
Figure 7.4 Ladder logic program for Petri net of Figure 7.3 using the original implementation method 7-12
Figure 7.5 Second ladder logic implementation of the Petri net of Figure 7.3 7-14
Figure 7.6 Third ladder logic implementation of the Petri net of Figure 7.3 7-15
Figure 7.7 Comparison between markings of different implementations 7-22
Figure 8.1 Relationship between a manufacturing station and the elements in its environment 8-11
Figure 8.2 Architecture of the proposed fault monitoring system 8-13
Figure 8.3 General Petri net structure used for the control of manufacturing systems including communications signals 8-14
Figure A1.1 Control net A1-2
Figure A1.2 Hardware initialisation A1-3
Figure A1.3 Hardware initialisation continued... .... A1-4
Figure A1.4 Hardware initialisation continued... .... A1-5
Figure A1.5 Hardware initialisation continued... .... A1-6
Figure A1.6 Get pallet from conveyor A1-7
Figure A1.7 Get pallet from storage A1-8
Figure A1.8 Get cylinder A1-8
Figure A1.9 Place block on pallet A1-9
Figure A1.10 Get cylinder 2 .... A1-9
Figure A1.11 Place cylinder on pallet A1-10
Figure A1.12 Place pallet on conveyor A1-11
Figure A1.13 Check storage areas A1-12
Figure A1.14 Safety net A1-13
Figure A2.1 Subnet showing sequence for Pallet Manipulator A2-2
Figure A2.2 Subnets for pneumatic cylinders making up the pallet manipulator A2-3
Figure A2.3 Output nets for the pallet manipulator A2-4
Figure A3.1 Layout of Flexible Manufacturing System A3-2
Figure A3.2 Cell controller hierarchy A3-2
Figure A3.3 Lathe Control net A3-9

Figure A3.4 Mill Control net A3-10
Figure A3.5 Robot Control net A3-11
Figure A3.6 Cell control net for the FMS A3-12

List of Tables

Table 6.1 System tasks of raw materials station 6-2
Table 6.2 Control and feedback signals for the raw materials station 6-4
Table 6.3 The subsystems of the raw materials station and their functions 6-6
Table 6.4 Task sequences for the raw materials station 6-7
Table 6.5 Paths for the raw materials station 6-9
Table 6.6 Full descriptions of the non-primitive places from Table 6.4 6-10
Table 6.7 Place descriptions for the raw materials station control net 6-10
Table 6.8 Transition labels for control net 6-11
Table 7.1 Distribution of tokens in a single cycle of the Petri net 7-16
Table 7.2 Distribution of tokens for the first implementation 7-18
Table 7.3 Distribution of tokens for the second implementation 7-18
Table 7.4 Distribution of tokens for the third implementation 7-18
Table A2.1 Task sequences for the pallet manipulator A2-1
Table A3.1 PLC output signals A3-13
Table A3.2 PLC input signals A3-14

Chapter 1
Introduction to the Research

1.1 Introduction

This work describes a method for the development of manufacturing control software, based on the Petri net formalism. The method incorporates the principles of modularity and the stepwise refinement of manufacturing processes, resulting in a software structure that clearly specifies both the manufacturing process and the control signals passing between system components.

In addition the software structure provides the necessary information to allow the detection and diagnosis of a variety of system failures. This information gives an indication of the origins of such a failure, whether it is caused by a fault in the system hardware or produced by an error in the software. These diagnosis capabilities are a by-product of combining the control structure with the structured Petri net formalism. They are thus inherent to the design process, and not appended as an afterthought.

The work also raises some issues associated with the implementation of Petri nets and in particular implementation on Programmable Logic Controllers using Ladder Logic Diagrams.

The method has been applied to two separate manufacturing workstations. The first workstation, which is part of a larger system, places different types of raw material onto a conveyor. The second is a self-contained manufacturing cell consisting of a lathe and a mill, which are loaded and unloaded by a shared robot.

1.2 Objectives of the Research

The purpose of automating the development of manufacturing control software is to provide a system that allows the process designer to describe the desired process in a manner that can be easily and unambiguously interpreted by the system, and yet still be understood by the operator. The transition between process description and implementation should be transparent to the operator. There should be some means of verifying the input to ensure it does not violate any basic rules, and the intended user must be able to understand the results of any such verification. Such a system should allow the user to test possible configurations, and to reuse manufacturing information as much as possible. In addition, if the system malfunctions in some way, for whatever reason, the software system should at a minimum be able to detect that malfunction and issue a warning. Ideally on malfunction, the system should be able to take some corrective action itself in order to resolve the malfunction or minimise its effect to the overall system performance.

1.3 Achieving the Objectives

The work carried out here is an attempt towards achieving the objectives of automating software development. There are a number of elements of this work that help towards that achievement.

The proposed design method is based on a modular form of Petri net, which has been called a structured Petri net. The interface between Petri net modules is well defined and acts as their only point of communication. This reduces the coupling between modules, and creates a higher degree of modular cohesion.

A method for designing manufacturing control code is presented. This is then formalised to make it amenable to automatic generation from the process specifications, for which there is currently no universally accepted standard notation. It is a simple step from an elementary process description to a Petri net description which, having a graphical element, will clearly indicate any concurrency within the process.

The implementation method for the control software depends heavily on the type of controller being used. Here the majority of the work has been carried out using Programmable Logic Controllers (PLC's) for which different methods of implementation have been compared. Other implementations such as high level languages have been considered in the literature, which are suited to more general-purpose controllers such as Personal Computers. Also during the course of this work an implementation using a Relational Database System has been investigated.

Automation will be more readily achieved if the same formalism can be used for specification, design and control of a system. This formalism must be simple enough to permit the implementation on those controllers commonly found in modern manufacturing systems (such as PLC's, robot controllers, CNC Tools). There must be a degree of simplicity to the method enabling managers, systems engineers, and others involved with the implementation of manufacturing systems, to communicate with each other effectively. However, the formalism must also be expressive enough to accurately reflect the workings of the system to which it is being applied.

It is for these reasons that the work here proposes a method of designing and implementing control code for manufacturing systems based on structured Petri nets. By carefully selecting the interpretations for the basic elements of the formalism, many of the problems associated with low-level Petri nets, such as state explosion and complexity, can be reduced. The complexity of the models can be further reduced by using the same basic elements to describe inter module communication.

It is the use of the Petri net elements for describing communication between the modules that provides information concerning the nature of the signals passing between different parts of the control structure. At the lowest level in the control structure, communication is carried out between the controller and the hardware of the plant itself. By careful specification of these communications any discrepancies between the signals that actually pass between the controller and the plant and those which, from the specification, are expected, can be captured. This information provides the framework for a novel method of distinguishing between malfunctions caused by machine hardware and those caused by control software. Such a distinction has become necessary because of the need to reduce the downtime of manufacturing facilities when attempting to correct hardware failures caused by software errors.

The proposed methodology has its origins in the development of manufacturing control software. However, the Petri net formalism, based on interpretations of its elements, combined with the general nature of the design method has application to other types of system.

1.4 Petri Nets

In his thesis (Petri, 1962) Carl Adam Petri describes 'the conceptual foundations of a theory of communication". His theory concerned the transmission of information applied to the design and programming of "information-processing machines". Net theory was adopted by the Information Systems Theory Project and Project MAC at the Massachusetts Institute of Technology (see (Peterson, 1977)) where it was developed further into what is today known as Petri net theory. Petri's original net theory developed into a general net theory (Genrich et al, 1979) which is closely related to the general systems theory of Ludwig von Bertalanffy (Bertalanffy, 1968).

By 1978, Petri net applications included distributed database systems, communication protocols, and computer hardware modelling (Agerwala, 1978), (Agerwala, 1979). The further application of Petri nets to a number of different fields has been noted in (Peterson, 1981) and (Murata, 1989) including analysis and design of manufacturing systems.

The first book on Petri nets was published in 1981 and the author described Petri nets as "a tool for the study of systems" (Peterson, 1981). Another author describes Petri nets as a tool for modelling communication between parallel processes (Reutenauer, 1990). Both descriptions are in line with the ideas of systems theory since systems consist of interacting processes, and those interactions must be described by some formalism. It makes sense for that same formalism to be used for the process description and for describing the communications between those processes (a point of Petri's thesis). This same commonality between the description of systems and the communications between them is, in part, behind the work of this thesis.

1.5 Application to Manufacturing Systems

As part of the project MAC a Masters thesis (Hack, 1972) dealt with the analysis of production systems using free-choice Petri nets (a sub-class of ordinary Petri nets). It is claimed that this is one of the oldest application areas of Petri nets (Silva and Valette, 1990). Even so there were very few papers published on the subject until the late 1970's and early 1980's, with most of that work being carried out in France. A survey paper (Silva and Valette, 1990) cites a number of papers in French from 1978 and 1979, and another survey (D'Souza and Khator, 1994) cites a paper in English from 1980 (Chocron and Cerny, 1980). Industrial process control is cited as one of the applications of Petri nets in (Johnsonbaugh and Murata, 1982) and (Andre et al, 1980), where much of the effort was in developing hardware implementations of Petri nets. The mid 1980's saw the publication of more French papers dealing with Petri net controllers for flexible manufacturing systems (e.g. (Silva and Velilla, 1982) and (Valette et al, 1985)), and an important paper on the synthesis of FMS models by merging Petri nets of individual sub-tasks (Narahari and Viswanadham, 1985). Some work on modified Petri nets was also published around this time (Beck and Krogh, 1986). Many of the approaches described attempted to incorporate a modular approach in order to reduce the size and complexity of models for large systems. They also introduced more complex modelling formalisms such as coloured Petri nets (Gentina, et al, 1988).

The late 1980's saw the introduction of Controlled Petri nets in (Krogh, 1987) and (Holloway and Krogh, 1990), which were applied to the supervisory control of discrete event dynamic systems.

In (Zurawski and Zhou, 1994) a tutorial is presented with an introduction to industrial applications of Petri nets and an up to date bibliography. In the late 1990's there is a large concentration on more high level Petri net models which incorporate other techniques such as fuzzy logic (Hanna et al, 1994) or object oriented methods.

One of the important developments to come out of the research was the development of Grafcet (David and Alla, 1992) or Sequence Function Charts, for programming Programmable Logic Controllers (PLCs). However, despite the popularity of Grafcet on the continent, most manufacturing organisations in the UK and in the USA are still using programming methods such as Ladder Logic, Boolean Logic, and assembly type languages.

1.6 Petri Nets for Control

A Petri net is an abstract model and only represents a system when some meaning is ascribed to its elements (Agerwala, 1979). The particular meaning, or interpretation, is forced upon a net in cases of implementation. The abstract Petri net model is a parallel system, but it is implemented on a sequential machine. This forced interpretation will effect the behaviour of the Petri net and its properties to varying degrees (Grafcet is a prime example).

In (Silva and Velilla, 1982) a comparison was made between Petri net implementations on different Programmable Logic Controllers. These comparisons highlight the importance of the interpretation on the behaviour of the (implemented) net.

1.6.1 Centralised control

Manufacturing control can be either centralised or decentralised. According to (Silva and Valette, 1990), centralised control requires a co-ordinator (or manager) and a set of tasks. The co-ordinator plays the 'token game' on the net model. The tasks are attached to fired transitions.

The problems associated with centralised control are that the co-ordinator is a weak point for catastrophic failure, and there is an overhead associated with the indirect communications between tasks (both in execution time and in size of code).

1.6.2 Decentralised control

Again according to (Silva and Valette, 1990) decentralised control requires a set of sequential processes, and some communication/synchronisation mechanism.

Processes communicate with each other directly rather than through a central co-ordination system. This does however make communications more complex.

1.6.3 Different levels of control model

Petri nets are applied to modelling manufacturing systems at different levels:

- Structural analysis of a system can be carried out using low level Petri nets, such as Ordinary Petri nets, Simple Petri nets, Marked graphs (Murata, 1989), (Peterson, 1981) and free-choice Petri nets (Desel and Esparza, 1995).
- Performance analysis tasks such as measuring throughput or scheduling exercises are performed using timed Petri nets (Murata, 1989), (Merlin, 1976) or stochastic Petri nets (Murata 1989), (Marsan, 1989).
- Higher level nets such as coloured Petri nets (Jensen, 1997) and more recently Object Oriented Petri nets (see (Adamou et al, 1998)) and Fuzzy Petri nets (Hanna et al, 1994) are now being used for more complex simulation of flexible manufacturing systems.

1.7 Overview of Thesis

Chapter 2 describes the Petri net structure used as a starting point for the rest of the work presented in later chapters. The advantages and limitations of the structure are presented, along with a discussion of the need for its improvement.

Chapter 3 presents the Petri net formalism in more detail and describes a number of Petri net classes which are related to structured Petri nets. Finally a description of structured Petri nets is provided.

The structured Petri nets of *Chapter 3* provide a formal descriptive device for modelling manufacturing system elements. *Chapter 4* appends this formalism with a set of interpretations that may be applied to the elements of structured Petri nets. These interpretations allow a model to be constructed from the basic net elements that has more meaning to the user of the system. With each of these interpretations comes a unique graphical descriptor that provides easy understanding of the model. Finally the chapter discusses the issues behind modularity and how the structured Petri nets allow such a modular structure to be created.

Chapter 5 presents a method for the development of control code using structured Petri nets. The method relies heavily on the concepts of modularity, and stepwise refinement (Wirth, 1971). The chapter also presents similarities with commonly used systems analysis and design techniques.

Chapter 6 provides an example of where the development method has been applied to a real system. The system presented is a workstation for supplying raw materials to a larger manufacturing system. Some of the issues arising from the application of the development method are also discussed in this chapter.

Chapter 7 discusses some issues arising from the implementation of Petri nets on sequential machines. It describes some of the problems of interpretation and some of the techniques used to overcome such problems.

The implementation issues of *Chapter 7* provide the groundwork for the ability to use the control structure as a means of fault detection and diagnosis. In *Chapter 8* some definitions are provided for possible faults and failures in manufacturing systems and areas where the control structure can be used to detect the existence of such faults are proposed.

Chapter 9 draws some conclusions from the work described earlier and critically assesses the practicalities and usefulness of both the structured Petri net formalism and the control structure development method. The chapter also provides some pointers for further development of the system, and other related work that could arise from that described here.

Finally *Appendix 4* contains three conference papers which have been produced during the course of this research, and which are referred to at various points in this text.

1.8 References

Adamou, M., Zerhouni, S. N. and Bourjault, A., 1998, "Hierarchical modelling and control of flexible assembly systems using object-oriented Petri nets." *International Journal of Computer Integrated Manufacturing*, **11**, pp. 18-33.

Agerwala, T., 1978, "Some applications of Petri nets." In *Proc. 1978 National Electronics Conference*, Chicago, USA, vol. 12, pp. 88-94.

Agerwala, T., 1979, "Putting Petri nets to work." *IEEE Computer*, **12**, pp. 88-94.

André, C., Diaz, M., Girault, C., and Sifakis, J., 1980, "Survey of French research and applications based on Petri nets." In *Proc. Advanced Course on General Net Theory of Processes and Systems,* Berlin, Germany: Springer-Verlag, pp.321-345.

Beck, C. L. and Krogh B. H., 1986, "Models for simulation and discrete control of manufacturing systems." In *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA, pp. 305-310.

Bertalanffy, L. von, 1968, *General Systems Theory.* New York, USA: Braziller.

Chocron, D. and Cerny E., 1980, "A Petri net based industrial sequencer." *In Proc. Conference on Applications of Mini and Micro Computers*, Philadelphia, PA, USA, pp. 18-22.

David R. and Alla, H., 1992, *Petri nets and Grafcet: Tools for modelling discrete event systems.* London, England: Prentice Hall.

Desel, J. and Esparza, J., 1995, *Free choice Petri nets.* Cambridge, England: Cambridge University Press.

D'Souza, K. A. and Khator S. K., 1994, "A survey of Petri net applications in modeling controls for automated manufacturing systems." *Computers in Industry*, **24**, pp. 5-16.

Genrich, H. J., Lautenbach, K. and Thiagarajan, P. S., 1979, "Elements of General Net Theory." In *Proc. Advanced Course on General Net Theory of Processes and Systems,* Berlin, Germany: Springer-Verlag, pp. 20-163.

Gentina, J. C., Bourey, J. P. and Kapusta, M., 1988, "Coloured Adaptive Structured Petri Nets." *Computer Integrated Manufacturing Systems*, **1**, pp. 39-47.

Hack, M., 1972, "Analysis of production schemata by Petri nets." Masters Thesis, Massachusetts Institute of Technology.

Hanna, M., 1994, "Determination of product quality from an FMS cell using Fuzzy Petri nets." In *Proc. IEEE International Conference on Systems, Man and Cybernetics,* San Antonio, TX, USA, pp. 2002-2007.

Holloway, L. E. and Krogh, B. H., 1990, "Synthesis of feedback control logic for a class of controlled Petri nets". *IEEE Transactions on Automatic Control,* **35**, pp. 514-523.

Jensen, K., 1997, *Coloured Petri-Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 1*, London: Springer-Verlag.

Johnsonbaugh, R. and Murata, T., 1982, "Petri nets and marked graphs - Mathematical models of concurrent computation." *American Math Monthly,* **89**, pp. 552-566.

Krogh, B. H., 1987, "Controlled Petri nets and maximally permissive feedback logic." In *Proc. 25th Annual Allerton Conference,* University of Illinois, USA, pp. 317-326.

Marsan, M. A., 1990, "Stochastic Petri nets: An elementary introduction." In *Advances in Petri Nets 1989,* G. Rozenberg, Ed., (Lecture Notes in Computer Science 424). Berlin, Germany: Springer-Verlag, pp. 1-29.

Merlin, P. M., 1976, "A methodology for the design and implementation of communication protocols." *IEEE Transactions on Communications,* **COM-24**, pp. 614-621.

Murata, T., 1989, "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE,* **77**, pp. 541-581.

Narahari, Y. and Viswanadham, N., 1985, "A Petri net approach to the modelling and analysis of flexible manufacturing systems." *Annals of Operations Research,* **3**, pp. 449-472.

Peterson, J. L., 1977, "Petri nets", *Computing Surveys.* **9**, pp. 223-252.

Peterson, J. L., 1981, *Petri net theory and the modeling of systems.* Englewood Cliffs, NJ, USA: Prentice Hall.

Petri, C. A., 1966, "Communication with automata", *English translation of 'Kommunikation mit Automaten',* Griffiss Air Force Base Technical Report RADC-TR-65-377 Vol. 1 Supplement 1.

Reutenauer, C., 1990, *The Mathematics of Petri Nets.* Prentice Hall International.

Silva, M. and Valette, R., 1990, "Petri nets and flexible manufacturing." In *Advances in Petri Nets 1989.* G. Rozenberg, Ed., (Lecture Notes in Computer Science 424). Berlin, Germany: Springer-Verlag, pp. 374-417.

Silva, M. and Velilla, S., 1982, "Programmable logic controllers and Petri nets: A comparative study." In *Proc. IFAC Conference on Software for Computer Control,* Madrid, Spain, pp. 83-88.

Valette, R., Courvoisier, M., Demmou, H., Bigou, J. M. and Desclaux, C., 1985, "Putting Petri nets to work for controlling flexible manufacturing systems." In *Proc. International Symposium on Circuits and Systems,* Kyoto, Japan, pp. 929-932.

Wirth, N., 1971, "Program development by stepwise refinement." *Communications of the ACM,* **14**, pp. 221-227.

Zurawski, R and Zhou, M., 1994, "Petri nets and industrial applications: A tutorial." *IEEE Transactions on Industrial Electronics.* **41**, pp. 567-583.

Chapter 2

A Control Structure

The Petri net structure described in this chapter represents the starting point of the research presented in the rest of this thesis. This chapter describes the method by which the basic Petri net elements were initially used to create a control structure. It then proceeds to describe the control structure itself and outlines the initial method by which the control structure was implemented on a PLC using ladder logic. Finally, the chapter goes on to describe the problems associated with both the structure and implementation and details the need for a more formal approach to the design and implementation of manufacturing control code using Petri nets.

2.1 Petri Nets

A basic definition of Petri nets is given here in order that it may be compared with the Petri net elements described in the following sections.

A Petri net is a 5-tuple, $PN = \{P, T, I, O, \mu_0\}$ where:

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

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

$P \cup T = \varnothing$ and $P \cap T = \varnothing$ ($\varnothing$ is the empty set).

$I : T \to P$ is the input function mapping from transitions to places,

$O : T \to P$ is the output function mapping from transitions to places,

$\mu_0 : P \to \mathrm{N}$ is the initial marking (N is the set of non-negative integers).

Places are represented graphically by circles and transitions by bars. Arcs are drawn as arrows between places and transitions. These represent the input and output functions. The marking is represented by the distribution of tokens amongst places. These are represented graphically by dots that appear within places.

2.1.1 Transition firing rule

When all the input places to a transition contain tokens, the transition is enabled. An enabled transition will fire by removing tokens from its input places and placing tokens in it output places. A more complete description of Petri nets can be found in (Peterson, 1981) or (Murata, 1989).

2.2 The Control Structure

For the control structure presented in this chapter, the behaviour and representation of Petri net elements are very similar to those described above.

2.2.1 Places, transitions, and arcs

Places are represented graphically by circles, and are used to represent non-primitive actions (e.g. placing an item on a conveyor), or states (e.g., machine is idle). Places representing non-primitive actions are called non-primitive places, and places representing states are called primitive places.

Hardware places

In addition, places may be used to represent hardware elements such as switches or sensors. These are called hardware places. A hardware place will contain a token when its associated switch/sensor is on and will not contain a token when the switch/sensor is off. Hardware places therefore represent binary control signals that can be used to enable or disable transitions. For primitive and non-primitive places, the firing rule is the same as that for ordinary Petri nets (as described in section 2.1.1). However, if an enabled transition has a hardware place as one of its inputs, and the transition subsequently fires, the token in the hardware place is not removed. The reason for this is that the marking of a hardware place is not dependent on the action of the Petri net, but is instead dependent on the state of its associated device (e.g. a limit switch). The Petri net controlling the system may cause the hardware switch to be shut off when the transition fires. In this case, the hardware place would lose its token, but this is due to the change in state of the hardware device rather than the behaviour of the net elements.

Graphical representation

The graphical representations of the Petri net elements used in the control structure are shown in Figure 2.1.

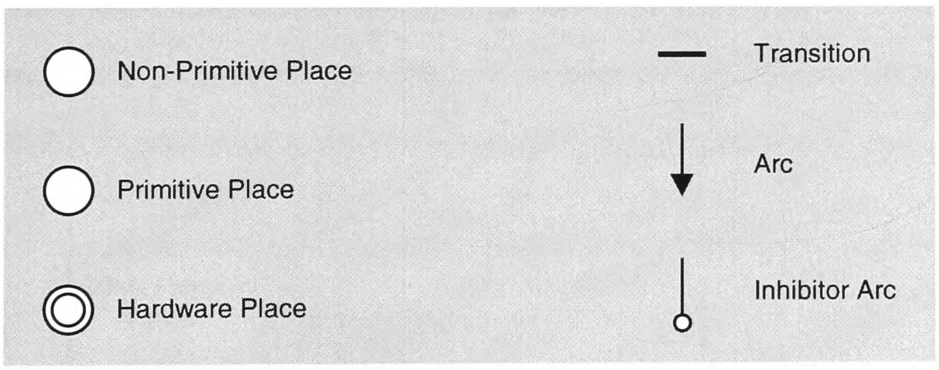

Figure 2.1 Symbols used in graphical representation of nets

Places of all types may contain at most one token. For non-primitive places, the presence of a token indicates that its associated action is currently being carried out. For primitive places the presence of a token indicates that its associated state holds, or is true. As noted previously, if a hardware place contains a token (is marked), then its associated hardware switch is on.

Transitions are drawn as horizontal bars, and represent either the transition between states or the start, or completion, of an action. In the Petri net model, it is assumed that all enabled transitions fire simultaneously. A transition's input places have their tokens removed, and output places receive their tokens at the same instant. In other words, transitions have no time associated with their firing.

Arcs may be either ordinary arcs or inhibitor arcs that allow testing for zero (see (Murata, 1989)).

2.3 The Structure

The Petri net structure is designed to reflect the hierarchy inherent in a manufacturing system. Each net in the structure is drawn from left to right with the higher levels of the hierarchy to the left (see Figure 2.2 below). The system under consideration is divided into a set of sub-systems referred to as axes. The co-ordination of these axes is described by a top level Petri net called the Control Net. Each axis is itself be described by a Petri net, which is called a subnet. The diagram in Figure 2.2 shows a system with three axes, each represented by a subnet.

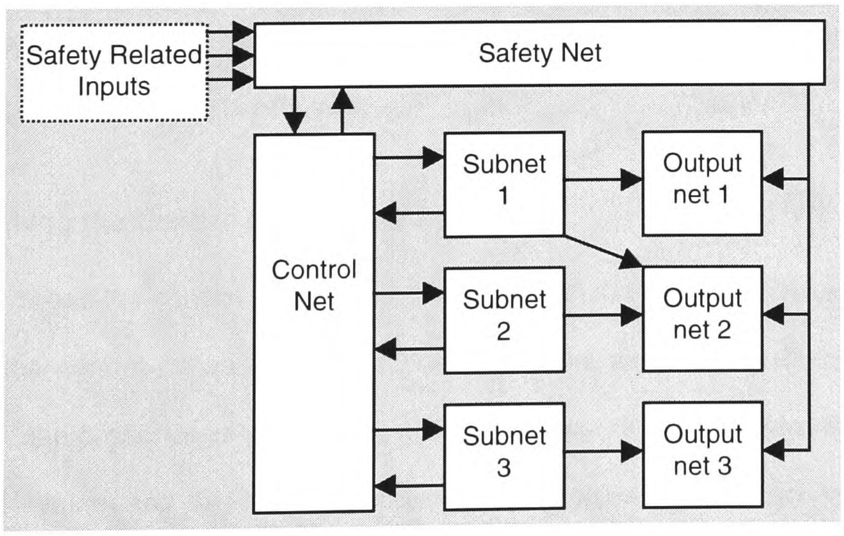

Figure 2.2 Petri net structure for a single machine

A third layer of nets, called output nets (because they are linked to the system's output devices), makes the physical link between the Petri net model and the machine hardware. Output nets are used to represent solenoids, electric motors etc. depending on the devices attached to the machine.

Figure 2.2 shows that the link between subnets and output nets is in one direction only. The subnets cause the output nets to activate or deactivate output devices. To indicate that actuation has taken place, they receive feedback from hardware places attached to sensors and limit switches. Thus, the feedback is not directly from the output nets but is instead from the machine hardware. For an example consider a subnet that causes an output net to activate a solenoid, which is attached to a pneumatic cylinder. Once this cylinder has completed actuation, it activates a limit switch. The limit switch is represented in the subnet as a hardware place.

2.3.1 Safety net

The safety net is used to monitor any safety related inputs attached to the system and if an unsafe condition is detected, to handle the orderly shutdown of the system. The safety net is linked to the control net to indicate that the system is safe to start. It is also linked to the output nets in order to take direct control of all the output devices if an emergency shutdown is required.

2.3.2 Linking the Control Net and the Subnets

The link between the control net and a subnet is shown in Figure 2.3. All subnets are linked to the control net in this way. The link works as follows. Transition t_1 is enabled by the presence of tokens in places p_1 and p_2. It will therefore fire, placing tokens in places p_3 and p_s. Place p_3 is a non-primitive place and thus according to the definition given previously, represents a non-primitive action or task. As p_3 is now marked this indicates that the action it represents is being carried out. This action is described by a subnet, which starts by consuming the token that is now in place p_s. The subnet carries out its action and on completion produces a token at place p_f. This enables the transition t_2 (p_3 is still marked from the firing of t_1), which can now fire removing tokens from both p_3 and p_f and placing a token in place p_4.

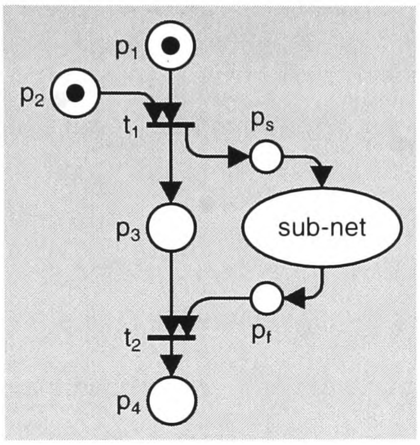

Figure 2.3 Control net/subnet link

The token remains in place p_3 all the while its associated sub-net is carrying out its operation, and is only removed on completion of that operation. The completion of the sub-net's operation is indicated directly by the production of a token in place p_f and indirectly by the firing of transition t_2.

2.3.3 Subnet/Output net link

The link between the subnets and the output nets is similar to that shown above for the control net/subnet link. However, in this case there is no feedback from the output net indicating completion of its task. Instead, feedback is obtained from sensors attached to the axis that the subnet represents. This is shown in Figure 2.4, where p_f is no longer associated with the action carried out by the subnet. It is now linked to a sensor and thus there is no direct feedback from the output net.

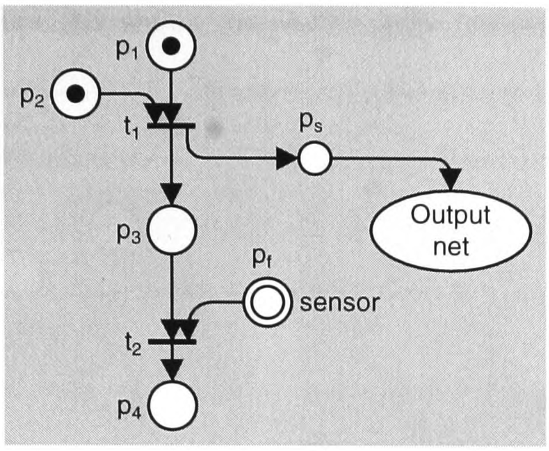

Figure 2.4 Subnet/output net link

2.4 Implementation

The nets are implemented on a PLC in ladder logic. In the ladder logic program, an output coil is used to represent each place. The reasoning behind the representation is as follows.

A place becomes marked when one of its input transitions fires. As stated in section 2.2, transitions fire instantaneously, as soon as they are enabled. Therefore, a place becomes marked when one of its input transitions is enabled.

A transition becomes enabled when all of its input places are marked. Therefore, a transition's output places become marked when all of its input places are marked.

In this way, the marking of a place can be expressed in terms of the other places in the net, removing the need to explicitly represent transitions in the ladder logic program.

In a similar manner, the marking is removed from a place if one of its output transitions fires. A transition firing can be recognised by all of its output places becoming marked. There is no need for the explicit representation of transitions.

The example in Figure 2.5 shows the ladder logic representation for place p_3 (Stanton et al, 1996).

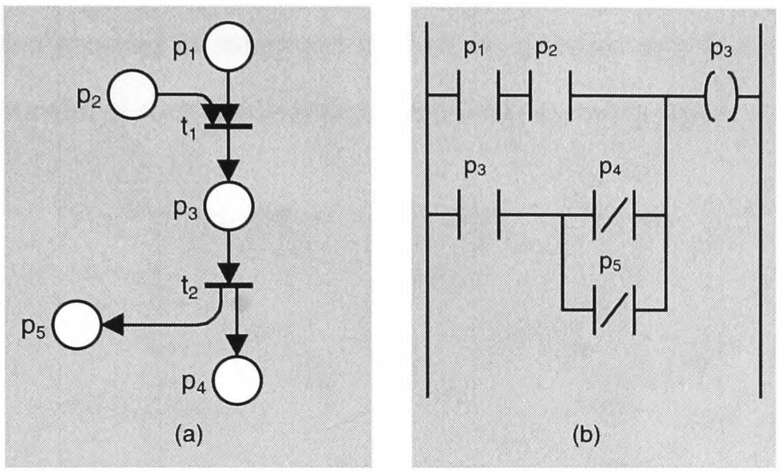

Figure 2.5 (a) Petri net segment (b) Ladder representation for place p_3

In Figure 2.5(a), transition t_1 is enabled when places p_1 and p_2 both contain a token. The transition fires instantaneously and the tokens are removed from places p_1 and p_2 and a new token is placed in place p_3. This token will remain in place p_3 until transition t_2 fires. The arrival of tokens in places p_4 and p_5 can be viewed as an indication that transition t_2 has completed firing, and so when these tokens arrive, the token can be removed from place p_3.

This behaviour is reflected in the ladder logic rung of Figure 2.5(b). It is assumed that initially all contacts and coils are de-energised. When contacts p_1 and p_2 become energised, output p_3 will then become energised (it is also assumed that when p_3 becomes energised p_1 and p_2 are then de-energised again). Coil p_3 will remain energised since it is acting as a latch. Only when both contacts p_4 and p_5 become energised will p_3 become de-energised again. The remainder of the ladder logic program is constructed in this way, with one coil for each place in the net.

Within the ladder program the nets, of which the structure comprises, are ordered from top to bottom with the control net first, followed by the safety net, then the sub-nets, and finally the output nets. This reflects the structure of the Petri net description and provides a structured method for programming PLC's with ladder logic. The structure of such a ladder logic program is shown in Figure 2.6.

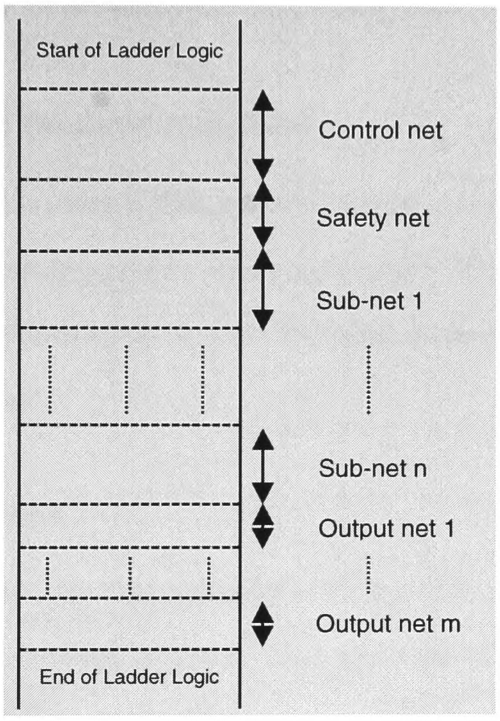

Figure 2.6 Structure of the Ladder Logic program

2.5 Fault Monitoring

Using the ladder logic representation described in Figure 2.6, in conjunction with the Petri net graph from which it is constructed, a certain level of fault diagnosis is possible. If the machine was to halt during execution of a task, it is possible to determine, from the ladder logic, which output coils are energised and thus their respective places in the control net. From the control net, the sub-nets that were executing when the halt occurred can be identified. In such a manner, the current state of the machine can be traced down through the control hierarchy to the output nets. At this point, the reason for the halt can be determined.

As far as the Petri net is concerned, the only reason for the system to halt is if one or more transitions are waiting for an input token before they can fire. The reason for the token being unavailable may be that a sensor has not been activated because, either it is itself faulty, or an actuator has failed to activate it in the correct manner.

2.6 Limitations of the Current Method

The method for controller design and implementation presented in this chapter is adequate for relatively small systems with low numbers of actuators. However, its limitations become apparent when applied to larger systems, with more actuators and concurrent processes.

2.6.1 The design method

The design approach used for the control software has little structure, aside from the fact that top down development is used. Also there are no guidelines as to what constitutes a 'good' design decision, other than those gained by the experience of the designer. Both of these points cause few problems where small systems are concerned, but when larger systems are to be controlled, some structure to the design approach is necessary. A more structured approach to the software design would allow relatively inexperienced designers to create well constructed code, and provide more consistency in the approaches taken for different types of controller.

One advantage of using Petri nets for the design and implementation of control software is that they allow structural analysis of the system in order to detect any adverse properties such as deadlocks and conflicts. However, standard methods for analysis (see (Peterson, 1981) or (Murata, 1989)) are not possible with the structure presented. This is because the many nets in the control structure interact with each other changing the behaviour of individual nets. For any such analysis to be possible, it is necessary to determine the particular class of Petri nets to which those described here belong. A single net may be taken in isolation and analysed using standard techniques, but it is necessary to determine what effect hardware places, and link places from other sub-nets have on the behaviour of that net.

With the structure as described in the current chapter, there are few rules governing the manner in which nets are linked. For a better modular structure, these links must be clearly defined and there are criteria for doing so.

2.6.2 Implementation

The implementation method so far described also produces some problems. When used to control a complex machine, the initial implementation method was found to be introducing additional tokens into the system. With no means of detecting the origin of such tokens, the real cause of the problem could not be isolated. Suffice to say that the implementation of the Petri nets into ladder logic by hand was likely to be an error prone one anyway. Certainly, the design of the Petri net and the machinery were not at fault, which left the possible cause as the implementation. The dynamic nature of the problem meant that the precise moment at which an error occurred could not be captured. The initial reaction was to develop a new method of implementation that took into account the way in which the scan cycle of the PLC worked. It was believed that the order in which the logic was solved affected the order in which tokens were generated, and that currently tokens where being generated at the wrong time. A later reaction was to consider how such an error might be trapped and therefore isolated.

2.7 Important Properties

The approach to design and implementation described in this chapter possesses a number of beneficial properties. If alterations are to be made to either the design approach or implementation approach then these properties must be either preserved or improved upon. These desirable properties are:

- The size of the control code
- The complexity of the method
- The fault diagnosis capability

2.7.1 Size of the control code

Any translation from Petri nets into an executable language should not result in an unnecessary increase in the size of the control code. This becomes particularly important where small, low cost controllers are used, which have a limited memory capacity. In the particular case of PLC's, the size of the control code will affect the reaction time of the controller. Therefore, if the size of the code can be kept to a minimum, then the application domain of the method can be expanded to include high-speed applications.

2.7.2 Complexity

Ladder logic diagrams are well known for their complexity even for relatively small applications (Venkatesh et al, 1994). A comparison has been made between the complexity of Petri nets and ladder Logic programs for applications of varying size. The complexity measure was limited to the number of nodes (i.e. places and transitions in the Petri net compared with contacts and coils in the ladder logic). This is not necessarily the best measure of complexity as the number of arcs in a Petri net can cause an increase in its visual complexity. An over complex representation of the system will act to make its use undesirable and if it is used will affect the maintainability of the system.

2.7.3 Fault diagnosis

The current level of fault diagnosis provided by the use of the Petri net structure should be preserved. It should not be more difficult to detect faults if the design and implementation methods are altered. This does not exclude the possibility that the fault diagnosis method may become more complex. However, the user must be shielded from such increases in complexity.

2.8 Areas of Improvement

The areas in which it is necessary to improve the design and implementation methods are as follows.

2.8.1 Modelling and control of complex concurrent systems

As stated previously, the method as described in this chapter handles relatively small applications where there is only a minor degree of concurrency. This is clearly not sufficient if the control structure is to be expanded to higher levels of the manufacturing environment, or to more complex manufacturing systems such as Flexible Manufacturing Systems.

2.8.2 Structural analysis

The ability to analyse system models will greatly reduce the amount of time spent on removing design faults in the system before implementation.

2.8.3 Enhance fault diagnosis

The fault diagnosis method at present relies on the operator searching through the ladder logic program with the Petri net graph for guidance. Currently faults can only be detected if the machine halts. Can the process of fault diagnosis be automated and extended to situations where the system does not stop? Could it facilitate the detection of transient faults?

2.8.4 Implementation on other types of controller.

There is a clear need for such a method to be applicable not only to other types of PLC but to any other controller present in a modern manufacturing facility. It should also be extensible to general-purpose computer systems, such as PC's.

2.9 Chapter Summary

This chapter has described the starting point of the research work described in the rest of this thesis. It has presented a method for the design and implementation of manufacturing system control software, which uses a Petri net based representation. This method has a number of weaknesses, but also some valuable properties, which must be preserved if any attempt is made to remove those weaknesses. The attempt to enhance these properties has been the motivation for the remainder of the work presented in this thesis.

2.10 References

Murata, T., 1989, "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE,* **77**, pp. 541-581.

Peterson, J. L., 1981, *Petri net theory and the modeling of systems.* Englewood Cliffs, NJ, USA: Prentice Hall.

Stanton, M. J., 1996, Arnold, W. F. and Buck, A. A., "Modelling and control of manufacturing systems using Petri nets." In *Proc. 13th IFAC World Congress*, San Francisco, USA, vol. J, pp. 324-329.

Venkatesh, K., Zhou, M. and Caudill, R. J., 1994, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system." *IEEE Transactions on Industrial Electronics,* **41**, pp. 611-619.

Chapter 3
Structured Petri Nets

3.1 Introduction

The Petri nets used in Chapter 2 offer a hierarchical approach to the development of control code for manufacturing systems. Given the problems associated with the use of these nets and the desire to further automate the software development process, there is a need to offer a more formal definition. In their current state the nets serve the purpose of offering a state/transition representation for control code development but do not offer any means of analysis. Analysis is required in order to ensure that the code controlling the system does include structural errors that might result in deadlocks or overflows. In order to develop any analysis techniques, the position of these nets within the Petri net literature needs first to be established.

This chapter starts by reiterating the Petri net definition given in Chapter 2. It then goes on to examine a number of other Petri net classes that bear some resemblance to those described in Chapter 2. By finding such similarities, it is hoped that they will point to analysis techniques that may be applied to these Structured Petri nets.

3.2 A Standard Petri Net Definition

A definition for a safe Petri net is presented here which is essentially the same as that given in (Peterson, 1981). There are other similar definitions presented throughout the Petri net literature. This represents the definition for the class of ordinary Petri nets, and other such classes are described later in this chapter and also in (Murata, 1989) and (David and Alla, 1992).

3.2.1 Petri nets

A safe Petri net is a 5-tuple, $PN = \{P, T, I, O, \mu_0\}$ where:

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

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

and $P \cap T = \varnothing$.

$I : T \rightarrow P$ is the input function mapping from transitions to places,

$O : T \rightarrow P$ is the output function mapping from transitions to places,

$\mu_0 : P \rightarrow \{0,1\}$ is the initial marking.

For the Petri net definition given here, the graphical representation uses a circle to represent a place and a bar (or sometimes a box (Desel and Esparza, 1995)) to represent a transition. The input and output mappings are represented by directed arcs.

3.3 Important Properties

There are many properties of Petri nets some relating to those in Graph Theory (see (Murata, 1989) for a comprehensive discussion). However as far as the control of manufacturing systems is concerned there are a few important properties that have a specific meaning. Those that are considered relevant to this work are presented here. A few additional properties are described in (Zhou and DiCesare, 1989) and (Zurawski and Zhou, 1994).

3.3.1 Safeness

A place is said to be *k*-bounded when the maximum number of tokens it may contain from an initial marking is *k*. Safeness is a special case of the boundedness property. If a place may only contain at most one token, then it is 1-bounded, also called safe.

Thus a safe place is one which is 1-bounded. A Petri net is safe if all of its places are safe. Safeness is dependent both on the Petri net structure and the initial marking.

Boundedness can be used to indicate if any buffers within a system will overflow given certain operating conditions. The boundedness of systems is not a consideration in this work as all systems are assumed to be safe. This is because the nets described here are used to indicate the availability of resources but do not explicitly model the resources themselves.

3.3.2 Liveness

A transition, t_j, is live if for every reachable marking from the initial marking, μ_0, there exists a firing sequence, σ, such that transition t_j is enabled. A Petri net is live if every transition in the net is live.

Liveness indicates that the net (and therefore the system under consideration) is free from deadlocks. It is also, therefore, an indication of the repeatability of system processes.

3.3.3 Conflicts

Conflict occurs when two processes are competing for the same resources. In a Petri net conflict is represented by two or more transitions being enabled by shared input places, such that if any one of the transitions fires, the remainder will no longer be enabled.

A class of Petri nets called free-choice Petri nets (Desel and Esparza, 1995) permits conflicts only where there is a single shared input place, with the class of extended free-choice nets allowing more than one shared input place which must be shared by all the enable transitions.

The Petri net formalism used for structured Petri nets must allow the representation of conflict since these occur naturally in manufacturing systems, and in particular in the Petri net structure described in Chapter 2.

Example:

Consider a machining station that is capable of more than one type of operation (three in the example of Figure 3.1), but may only carry out one of those operations at any given time. When the machine is in its ready state it will be able to accept a request for one of those actions. Figure 3.1 shows that the transitions labelled t_1, t_2, and t_3 are all enabled when the system is ready and therefore there is structural conflict in the net. The conflict must be resolved in some way in order to make the system operational.

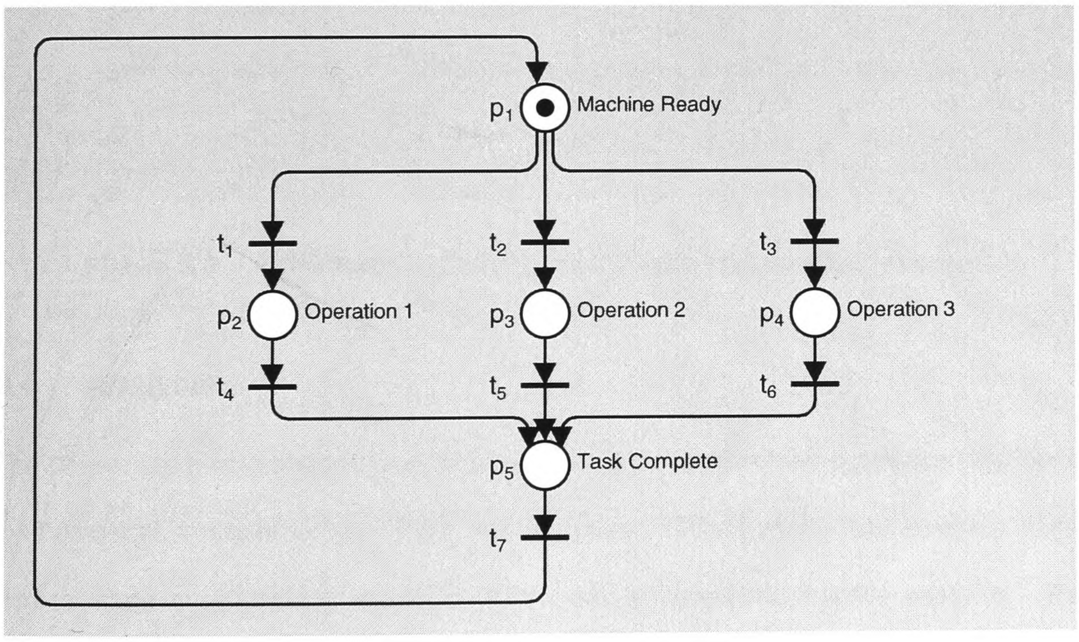

Figure 3.1 A Petri net showing structural conflict when there is a choice of operations

Adding mutually exclusive controls as inputs to t_1, t_2, and t_3, effectively removes this conflict. These controls are shown in Figure 3.2, and they represent requests for the particular operations to start from a supervisory controller. Note that there is also an additional place at the output of transition t_7. This acts as feedback to the controller to indicate that the requested task has been completed

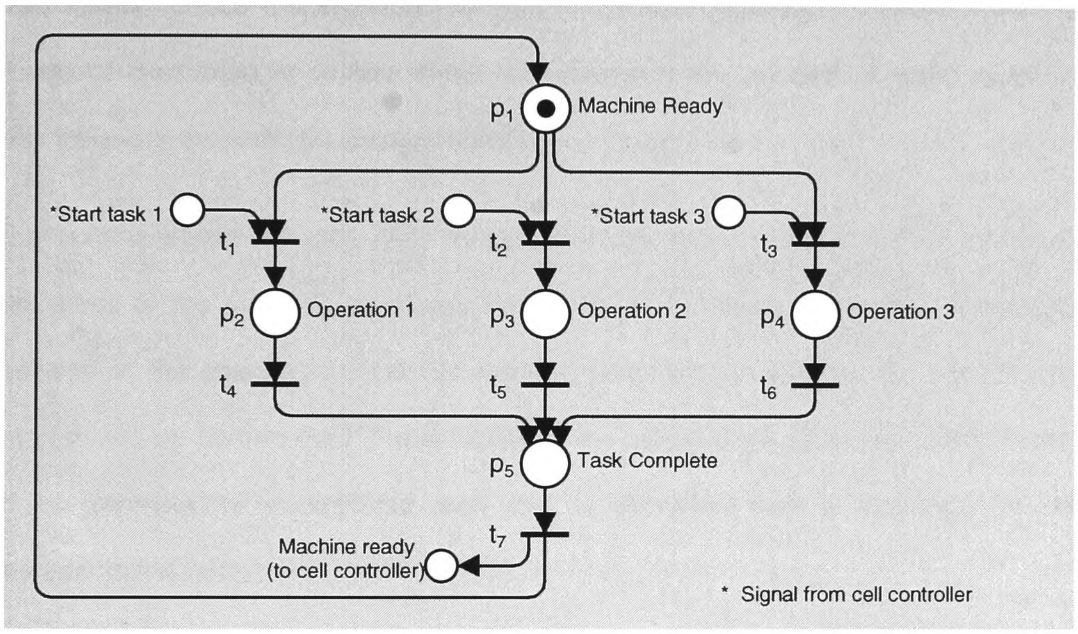

Figure 3.2 The Petri net of Figure 3.1 with the conflict resolved.

3.4 *Analysis*

The properties discussed in Section 3.3 can only be determined by performing some mathematical analysis on the Petri net structure. Traditionally this involves some form of state enumeration, which for Petri nets is called reachability analysis. The problem with state enumeration techniques is that, the larger the system, the more states it can generate, and the number of possible states increases exponentially. This is due to the distributed state representation of Petri nets, where the addition of *m* places to a Petri net, with *n* places, will increase the number of possible states by 2^m giving a total of 2^{n+m} states.

It has been claimed in (Stanton et al, 1996) that the modular structure of Structured Petri nets allows analysis of each module independently of the rest of the system. Since the modules contain a subset of the total number of places in the net, it may dramatically reduce the effect of the state explosion problem on analysis. Some questions then arise as to how those elements that are peculiar to Structured Petri nets influence the analysis being carried out.

The term analysis is used here to describe the means by which the structural properties of the net may be determined. Since no timing information is explicitly included in the structured Petri net representation of the system, no performance analysis will be carried out. The flexibility of the model does allow timing information to be represented if required and this is therefore not a limitation on the representation but an enforced condition.

The main difficulty with the analysis of Structured Petri nets is how to deal with the places that act as exogenous inputs and outputs. Although these places act in the same way as ordinary places, the arrival of tokens at an input place and the removal of tokens from an output place has not been dealt with in the definition given above. A number of questions can be posed concerning these exogenous inputs and outputs.

- Can these inputs and outputs be excluded from structural analysis?
- If they are excluded, then what class of Petri net is the underlying model?

- How does their later inclusion effect the structural properties of the underlying net?

The next few sections of this chapter examine a number of Petri net classes that have features similar to those of Structured Petri nets. These are considered in order to uncover the relationship between such Petri net classes and Structured Petri nets. It will be shown that structured Petri nets do not in fact belong to any one of these classes but contain features common to many of them.

It should be noted that there are a great many classes and extensions of ordinary Petri nets, which are usually tailored to an authors particular application. Structured Petri nets have little in common with such extensions and their simple nature has been a primary motivation throughout their development.

The classes of net considered are as follows:

- Marked graphs
- Decision free Petri nets
- Free Choice Petri nets
- Petri nets with external inputs and outputs (including Controlled Petri nets)

Each of these is discussed in the following sections highlighting the aspects of each class that are relevant to the definition of structured Petri nets.

3.5 Marked Graphs

Marked graphs are a sub class of Ordinary Petri nets. The definition given here is the same as that presented in (Murata, 1989).

3.5.1 Definition

A Marked Graph is an ordinary Petri net such that each place *p* has exactly one input transition and one output transition. Using the dot notation of (Hack, 1972):

$$|\bullet p| = |p \bullet| = 1 \quad \text{for all } p \in P$$

Marked graphs are decision and conflict free, since each place has only a single transition and therefore there is no decision to be made as to which output transition will fire.

As discussed previously, for control purposes it is necessary that all system conflicts be represented by the model in order that they may be addressed and resolved (i.e. that the controller is decision free). The marked graph is a restricted class of Petri net that is unable to model conflicts. Therefore, despite their decision free nature, they are of little use in the definition of structured Petri nets. This was recognised by (Krogh and Sreenivas, 1987).

3.6 Decision Free Petri Nets

Decision free Petri nets are introduced in (Dubois and Stecke, 1983). The definition given was the same as that of marked graphs, as defined in (Peterson, 1981) or (Murata, 1989) and shown in the previous section. Structured Petri nets need to be less restrictive than marked graphs, since they must allow the direct representation of conflicts within the system model. In (Krogh and Sreenivas, 1987) the notion of Essentially Decision Free (EDF) Places is introduced, in the context of a class of nets called Operation/Resource Nets. These use a slightly modified graphical notation to distinguish between operation and resource places, which makes the net appear more complex. They define a procedure for identifying non-EDF places, which is similar to that proposed in Chapter 5 for decision free places in Structured Petri nets. The method proposed for resolving any conflicts incorporates the use of 'NOT' arcs, which are a similar concept to inhibitor arcs.

3.6.1 Operation and resource places

The distinction between operation and resource places has also been made in (Zhou and DiCesare, 1995). Operation places are generally regarded as safe (1-bounded) places, whereas resource places can contain as many tokens as there are units of that resource available. In structured Petri nets, such a clear distinction between resource places and operation places is not made. At the highest level of control developed so far, the nets respond to:

- Direct commands from a controller (be that man or machine)
- The state of the system hardware in terms of its sensory output

It has already been stated that structured Petri nets model the availability of resources but not the resources themselves (Section 3.3.1). In Structured Petri nets, where a sensor is used to indicate the presence of a unit resource in the physical system, e.g. an item of raw material in a store, then a hardware place which represents that sensor is used to indicate when that resource becomes available.

Example:

Consider a raw materials store that may contain up to six items. The presence of an item in the store is indicated by a single sensor, which detects the next available item. It is tempting to use a place containing up to six tokens to represent the number of items in the store. This is fine if the behaviour of the system is being modelled, but for implementation purposes the controller should only respond to the sensory input indicating the availability of a single item.

The common implementation for a place with multiple tokens would be to use a counting device, which decrements every time an item is removed from the store. This can be problematical if the counter, through error, contains the wrong value, and potentially dangerous if the counter is erroneously informing the system that the store is empty. This is one reason why the Structured Petri net approach tries to avoid such representations and instead depicts the hardware through its sensory output.

3.7 Free-Choice Petri Nets

A Free-Choice Petri net is a sub class of ordinary Petri nets. They have been extensively covered in (Desel and Esparza, 1995). A Free choice net is a Petri net such that every arc from a place is either a unique outgoing arc or a unique incoming arc to a transition (see (Murata, 1989)).

A free choice Petri net contains structural conflicts, but the set of input places to each transition in the conflict is the same. It therefore allows any of the conflicting transitions to fire i.e. there is a 'free choice' in which transition can fire. The net shown in Figure 3.1 is a free choice net because firing one of the transitions in the conflict will disable all others within the conflict. This is a suitable interpretation for the uncontrolled structured Petri nets, which may be free choice nets.

The necessary and sufficient conditions for liveness in a free choice Petri net have been described by (Hack, 1972). If the underlying net of a structured Petri net can be shown to be a free choice net then there are proofs for properties such as liveness and safeness.

In structured Petri nets, the 'uncontrolled' Petri net has structural conflict. This means that a place may have two (or more) output transitions to which it is the only input. For example, when a machine is ready to accept work, it may be able to carry out one of any number of tasks. This represents some form of conflict, and in terms of the machines behaviour, it doesn't matter which of its actions is requested, since it is capable of carrying out any one of its tasks. However, since the structured Petri net is actually being used to control a machine, this conflict must be solved, and this is done by introducing a set of mutually exclusive control places as inputs to all the conflicting transitions.

Once a particular task has been started, then the machine should experience no further conflict until the task has been completed. However, if the machines' sub-systems are capable of more than one action, then these will naturally contain their own conflicts, which must be resolved at the level above which they occur. This leads the definition onto a class of nets called decision free Petri nets.

3.8 Petri Nets with External Inputs and Outputs

Petri nets with external inputs and outputs were defined in (Ichikawa, et al, 1985) and later redefined in (Ichikawa and Hiraishi, 1988). A set of places was used as inputs, to control certain transitions in the system and another set of places was used as outputs. The basis for this development was that in real systems not all the transitions of the system would be controllable and not all states of the system would be observable. They where introduced to allow the control of discrete event systems. They have a similar function to, and were a motivating factor behind the controlled Petri nets introduced in (Krogh, 1987).

3.8.1 Controlled Petri nets

Controlled Petri nets were introduced in order to analyse a number of control policies on a plant model. This is done by describing the states of the plant as a net, and then introducing a control policy to govern the firing of certain transitions within the plant in order to achieve the desired behaviour.

A control policy is defined as a sequence of markings on a set of control places. Such control places give rise to the concept of *controlled transitions*. A controlled transition is one that has a control place as one of its input places. Controlled Petri nets define a control feedback which is simply a function mapping a marking onto the next control.

Controlled Petri nets have some relation to structured Petri nets as they both have exogenous inputs. One difference between the controlled Petri nets of (Krogh, 1987) and those described in (Ichikawa and Hiraishi, 1988) is that controlled Petri nets assume a certain set of observable places, which is a subset of the set of all places in the net, whereas the nets of (Ichikawa and Hiraishi, 1988) explicitly define a set of observable places. It is not clear whether the output places of (Ichikawa and Hiraishi, 1988) are able to consume tokens in the same way as output places do in structured Petri nets. It is clear that this is not the case for controlled Petri nets. However both the nets proposed in (Krogh, 1987) and (Ichikawa, et al, 1985) are described as controlled Petri nets in the survey paper (Holloway, et al, 1997). This would indicate that their definitions are equivalent.

3.9 A Definition for Structured Petri nets

The definition for structured Petri nets was first presented in (Stanton and Arnold, 1997) and is similar to that for ordinary Petri nets except that a new distinction is made between control/feedback places and state/action places.

3.9.1 Definition

A Petri net with external inputs and outputs is a 5-tuple, $PNIO = \{P,T,I,O,\mu_0\}$ where:

$P = S \cup C^{in} \cup C^{out}$,

$S = \{s_1, s_2, ..., s_i\}$ is a finite set of state places,

$C^{in} = \{c_1^{in}, c_2^{in}, ..., c_j^{in}\}$ is a finite set of input places,

$C^{out} = \{c_1^{out}, c_2^{out}, ..., c_k^{out}\}$ is a finite set of output places,

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

$P \cup T = \varnothing$,

$I : T \rightarrow P$ is the input function mapping from transitions to places,

$O : T \rightarrow P$ is the output function mapping from transitions to places,

$\mu_0 : P \rightarrow \{0,1\}$ is the initial marking.

Notes:

- Only state places can be initially marked, thus the initial marking of the control places is always zero.
- Output places are never inputs to transitions of the same net and are only outputs to transitions of one net.

- Input places are never outputs to transitions of the same net and are only inputs to the transitions of one net.
- There is no distinction between state and action places. If there were it would be necessary to indicate that only state places may be initially marked. However, there are actually no initially marked places, except for hardware places.

The definition of Section 3.9 differs to that given by (Ichikawa and Hiraishi, 1988) where external outputs are represented as a subset of transitions (and therefore event signals) rather than explicitly by places (and therefore condition signals). The use of places as external outputs as well as inputs is the key to the modular structure presented here and is favoured for its simplicity and because it provides a uniform method of communication between nets.

Example

Figure 3.3 shows a Petri net with external inputs and outputs.

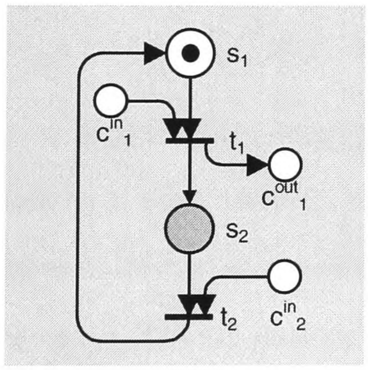

Figure 3.3 Example Petri net with external inputs and outputs

The net can also be described in terms of the definition given in section 3.9. This is presented as follows:

$S = \{s_1, s_2\}$, $C^{in} = \{c_1^{in}, c_2^{in}\}$, $C^{out} = \{c_1^{out}\}$,

$T = \{t_1, t_2, \ldots, t_n\}$,

$$\begin{array}{ll} I(t_1) = \{s_1, c_1^{in}\} & O(t_1) = \{s_2, c_1^{out}\} \\ I(t_2) = \{s_2, c_2^{in}\} & O(t_2) = \{s_1\} \end{array}$$

$\mu_0 = \{1,0,0,0,0\}$

3.10 Implications on Properties

The properties of liveness and safeness have a particular interpretation for manufacturing systems (Beck and Krogh, 1986). Safeness of an operation place indicates that there will not be a request for an operation that is already in progress. Thus there is no conflict in the enabling logic for that operation. Liveness of an input transition to an operation place indicates that there is no deadlock in the system, and liveness of output transitions to an operation place indicates that the operation will always finish.

3.10.1 Liveness

The property of liveness can be applied to individual transitions and to a complete net. Varying levels of liveness have been defined (see (Murata, 1989)), the work here requires that the nets are *L4 - live* i.e. all transitions are infinitely fireable from any marking of the net. The set of possible marking is restricted because of the method by which the initial marking is determined.

With the addition of control places, liveness becomes the responsibility of the controls as well as that of the net structure. The degree of independence of the net structure must be determined and the effect of adding controls to preserve liveness must be examined.

For every allowable path through a net:

- The initial transition of the path must be live
- The path itself must be live

For the initial transition of every path to be live, the enabling state of that path must always be reachable from any position in a path, and the enabling transitions in the control net must also be live.

Every transition in a path must be live including the terminating transition.

3.10.2 Safeness

Safeness becomes an important property when places are used to represent tasks. If a task place contains a token, then that task is currently taking place. If a task place contains more than one token, there is no sensible meaning. Some meaning could be ascribed to such a condition, such as the task is taking place and will be carried out again immediately it has finished. However this increases the complexity of the implementation and the complexity of the meaning of simple elements of the net and is thus disallowed in this net structure.

Safeness can be a structural property, but is also closely tied with the initial marking of a net. By careful control of the initial marking, safeness of action places can be ensured.

Multiple tokens in resource places are used to represent multiple resources, such as multiple parts ready for processing. However in the control structure described here, such resource places are not used. This removes the need for any places with multiple tokens.

3.10.3 Conflicts

There are conflicts within a net that could allow a choice of transitions, which can fire for a particular marking. If the nets were free choice nets then strictly speaking there would be a free choice as to which transition can fire. However, the nets used here are at least simple Petri nets.

When a machine is in its ready state there should be a free-choice as to which action the machine can carry out. However once a choice has been made, the remainder of the net should be 'decision free' i.e. there should be only one path through the net and no choice as to which transitions can fire.

The only point at which there may be a choice is where some part of the system is required to make a decision and the resulting path depends on the outcome of that decision.

It therefore seems that the nets used here are a hybrid, requiring the properties of controlled free choice nets in some instances, and decision free Petri nets in other instances.

Decision free Petri nets are described in (Krogh and Sreenivas, 1987) and (Krogh and Beck, 1986)

3.11 Chapter Summary

This chapter has looked at a number of Petri net classes, which appear to have a similar definition to those described in Chapter 2. The particular classes focussed on are Marked Graphs, Decision Free Petri nets, Free Choice Petri nets, and Petri net classes with external inputs and outputs. These all bear some similarities to what are now described as structured Petri nets (as they can be used to develop a variety of control structures). The chapter also presents a more formal definition for structured Petri nets, and discusses the properties of safeness, liveness and conflict-freeness, in relation to these nets and to the control of manufacturing systems. It appears that the uncontrolled structured Petri net (that with the exogenous inputs and outputs removed) is a free-choice Petri net. If this is the case then there are proven results concerning liveness and safeness of such systems (Hack, 1972).

3.12 References

Beck, C. L. and Krogh, B. H., 1986, "Models for simulation and discrete control of manufacturing systems." In *Proc. IEEE International Conference on Robotics and Automation,* San Francisco, CA, USA, pp. 305-310.

David R. and Alla, H., 1992, *Petri nets and Grafcet: Tools for modelling discrete event systems.* London, England: Prentice Hall.

Desel, J. and Esparza, J., 1995, *Free choice Petri nets.* Cambridge, England: Cambridge University Press.

Dubois, D. and Stecke, K. E., 1983, "Using Petri nets to represent production processes." In *Proc. IEEE Conference on Decision and Control,* San Antonio, TX, USA, pp. 1062-1067.

Hack, M., 1972, "Analysis of production schemata by Petri nets." Masters Thesis, Massachusetts Institute of Technology.

Holloway, L. E., Krogh, B. H., and Giua, A., 1997, "A survey of Petri net methods for controlled discrete event systems." *Discrete Event Dynamic Systems: Theory and Applications*, **7**, pp. 151-190.

Ichikawa, A., Yokoyama, K., and Kurogi, S., 1985, "Reachability and control of discrete event systems represented by conflict-free Petri nets" In *Proc. IEEE International Symposium on Circuits and Systems*, Kyoto, Japan, pp. 487-490.

Ichikawa, A. and Hiraishi, K., 1988, "Analysis and control of discrete event systems represented by Petri nets." In *Discrete Event Systems: Models and Applications*, Berlin, Germany: Springer-Verlag, pp. 115-134.

Krogh, B. H., 1987, "Controlled Petri nets and maximally permissive feedback logic." In *Proc. 25th Annual Allerton Conference.* University of Illinois, USA, pp. 317-326.

Krogh, B. H. and Beck, C. L., 1986, "Synthesis of Place/Transition nets for simulation and control of manufacturing systems." In *Proc. IFIP Symposium on Large Scale Systems: Theory and Applications*, Zurich, Switzerland, pp. 583-588.

Krogh, B. H. and Sreenivas, R. S., 1987, "Essentially decision free Petri nets for real-time resource allocation." In *Proc. IEEE International Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1005-1011.

Murata, T., 1989, "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE*, **77**, pp. 541-581.

Peterson, J. L., 1981, *Petri net theory and the modeling of systems.* Englewood Cliffs, NJ, USA: Prentice Hall.

Stanton, M. J. and Arnold, W. F., 1997, "Extension of structured Petri nets for the control of a conveyor system." In *Proc. Factory 2000: IEE 5th International Conference*, Cambridge, England.

Stanton, M. J., Arnold, W. F. and Buck, A. A., 1996, "Modelling and control of manufacturing systems using Petri nets." In *Proc. 13th IFAC World Congress*, San Francisco, USA, vol. J, pp. 324-329.

Zhou, M., and DiCesare, F., 1989, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems." *IEEE Transactions on Systems, Man, and Cybernetics*, **19**, pp. 963-973.

Zhou, M. and DiCesare, F., 1993, *Petri net synthesis for discrete event control of manufacturing systems*, USA: Kluwer Academic Publishers.

Zurawski, R and Zhou, M., 1994, "Petri nets and industrial applications: A tutorial." *IEEE Transactions on Industrial Electronics*. **41**, pp. 567-583.

Chapter 4

Petri Net Modules

This Chapter takes the structured Petri nets defined in Chapter 3 and from them creates a Petri net module describing a single element of a manufacturing system. This element can be combined with other such elements to form a larger manufacturing sub-system. Likewise subsystems are combined to form entire systems. Once the creation of a module has been described, the elements of structured Petri nets are described, along with their interpretations, in some detail.

The first part of the chapter uses a Petri net model of a pneumatic cylinder as an example to introduce the main elements of each net in the control structure. This model is then expanded on to show the interaction between a number of such nets.

4.1 Modelling a Pneumatic Cylinder

A Petri net may be used to describe the function of a particular system or sub-system. This description may be self-contained describing fully the possible changes of state of the system, and the conditions under which those state changes are possible. The level of detail used to describe the change of state will vary, depending on the particular application.

Consider the example of a pneumatic cylinder that moves up and down. A Petri net describing the possible states of the cylinder and the transitions between those states is shown in Figure 4.1.

Figure 4.1 shows that the cylinder must be in the *up* position before it can move to the *down* position and vice versa and is thus a complete functional description of the cylinder. The transitions labelled t_1 and t_2 represent the transitions between the *up* and the *down* positions.

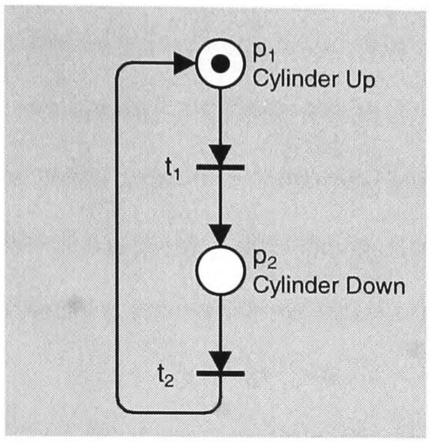

Figure 4.1 A simple Petri net description of a pneumatic cylinder

Transition t_1 represents movement of the cylinder from the up state to the down state and thus takes a finite amount of time to occur. During such time the cylinder is in neither the *up* state nor the *down* state. Therefore the presence of a token in either place would be a misrepresentation of the true state of the system. The same argument can be applied to transition t_2. A more informative net is shown in Figure 4.2.

The Petri net of Figure 4.2 represents a more complete description of the pneumatic cylinder, which may now be either up, down or moving in a particular direction. Note that a single intermediate state could have been used to indicate that the cylinder is in transition, but by using two additional states we gain more information concerning the state of the cylinder, namely information concerning the direction of its travel. Such a description is usually satisfactory for the purposes of modelling and is thus as far as many Petri net based methods will go in terms of systems descriptions.

A point to note is that having a token in place p_1 indicates that the initial state of the model is with the pneumatic cylinder in the up position. This should, of course, match with initial state of the real system. It will be seen later that it is possible to set the initial marking of the model to reflect the actual initial state of the system even if the initial system state is not known until the system is powered up. In fact it is desirable to check the state of the system on power up and from its initial state make certain checks to ensure that the system is ready and in a known state before it can start its operations. This is, in part, the role of the Safety net described in Chapter 2.

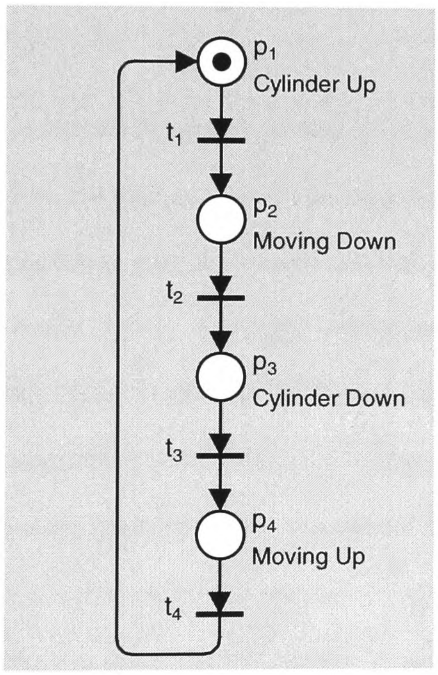

Figure 4.2 An extended Petri net description of the pneumatic cylinder

4.2 *Monitoring and Control*

There are two main reasons for using Petri nets in this work:

1) To monitor the state of real systems

2) To control the action of real systems

The net of Figure 4.2 does not allow for either monitoring or control, as there is no means by which it can be connected to a real system. The net description thus far used does not allow for such connections.

It is possible to assign properties or conditions to transitions such as those used in Grafcet (David and Alla, 1992), however the problem with such extensions is that they add to the complexity of the system and cause analysis of the system to become difficult. Also any such extensions will not be explicit in the net representation and will therefore increase the graphical complexity of the formalism.

4.2.1 Monitoring

In order for the Petri net description shown above to monitor the state of the real system it must be attached to the real system in some way. With applications where a PLC is used to control systems, the links between the controlling device and the machine hardware are made using memory addresses. In particular these addresses are described as input or output addresses, depending on whether the controller is receiving or transmitting information. It is these memory addresses that act as the interface between the machine hardware and the software that is controlling it. Therefore the same concept will be used here to link the software (described by a Petri net) to the machine hardware. It will be assumed here that a Petri net can be implemented on such a controller, leaving a full discussion of implementation issues until Chapter 7.

A pneumatic cylinder will usually include limit switches to indicate to the controller that it as reached the end of its actuation. Therefore once the limit switch is activated, the controller knows that the actuator has finished moving and has reached one of its fixed states. These limit switches can be shown on the Petri net description as in Figure 4.3.

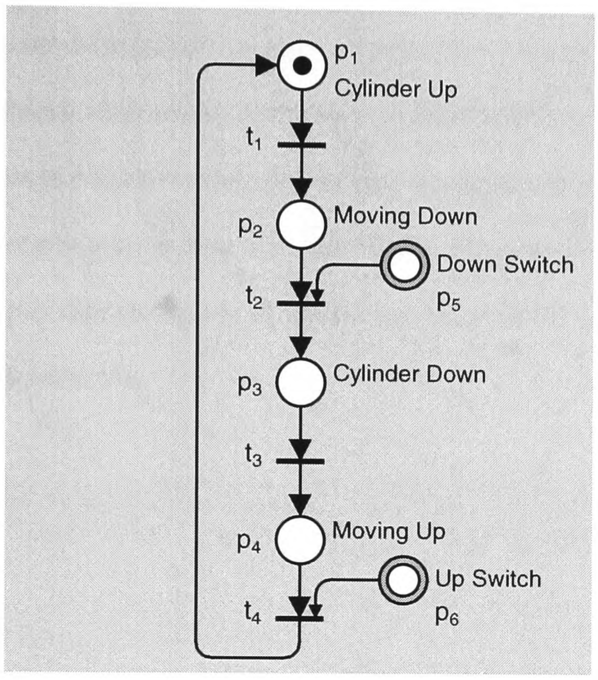

Figure 4.3 A Petri net with external inputs

Now that the hardware places p_5 and p_6 have been added the Petri net will be able to reflect the state of the system that it is monitoring. The description will now, using the original modelling formalism, permit the monitoring of systems and provide feedback as to their current state. It is also now possible for the system to reflect the state of the real system, assuming the limit switches are functioning correctly.

4.2.2 Controlling the hardware

A pneumatic cylinder, as described in the preceding section, will usually be actuated by one or two solenoids, depending on whether the device is single acting or double acting. For double acting devices, the control logic must ensure that the solenoid to move the cylinder down is only actuated when the cylinder is up, and conversely that the solenoid to move the cylinder up is only activated when the cylinder is down. The solenoids are here termed output devices, and such devices must be controlled from the Petri net. To do this control places are connected to the output devices to start actuation. As discussed previously in this chapter, the control places are actually attached to the memory addresses that are in linked to the output devices. The resulting net is shown in Figure 4.4 which has both external inputs and outputs which allow it to communicate with a real system. This net will allow actuation of the pneumatic device, and also detection of the actual state of the pneumatic device as indicated by the limit switches.

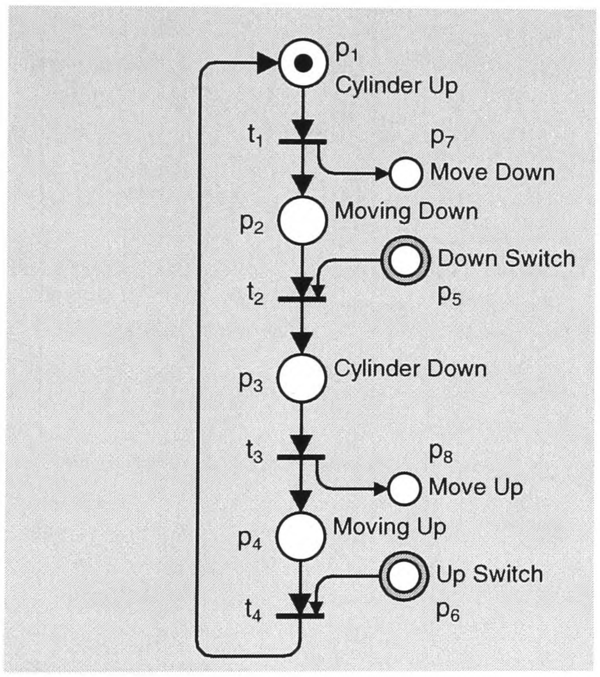

Figure 4.4 A Petri net with both external inputs and outputs.

4.2.3 Controlling the software

It is unlikely that any machine will consist of only a single pneumatic cylinder that works in isolation. Therefore it is necessary to introduce some mechanism for co-ordinating the activities of this device with other such devices attached to system.

The Petri net description of Figure 4.4 can be described as a software module that is controlling the pneumatic cylinder subsystem. A higher level of control is needed to instruct the cylinder control module when it is to move up or down in relation to the actions of other system devices. Such instructions will be sending control signals to the cylinder module and will in turn require feedback to indicate that the module has completed its required function. In more complex systems, the function required of a subsystem will include more actions than a single movement up or down. For the current example control and feedback places are added to the Petri net module as shown in Figure 4.5.

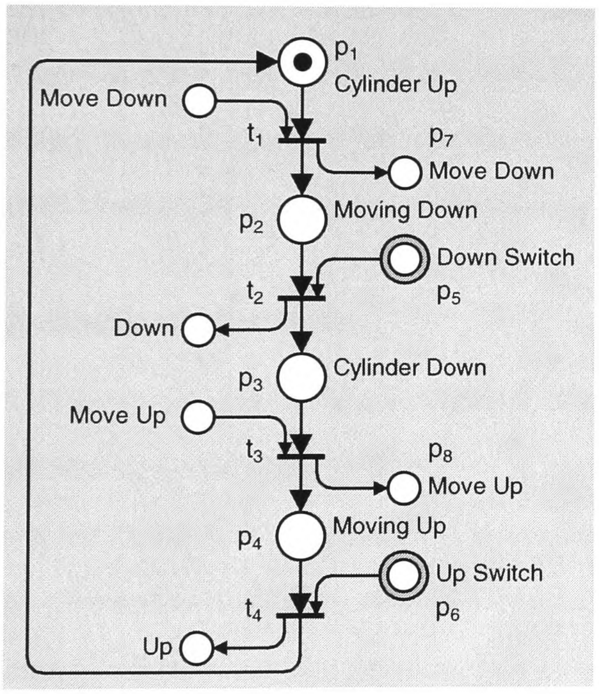

Figure 4.5 A full Petri net description of the pneumatic cylinder controller

The description of the pneumatic cylinder shown in Figure 4.5 now has all of the required elements for the cylinder module. It clearly shows the possible states of the system and the internal conditions necessary to allow transition between states. It also incorporates the necessary external conditions necessary for a transition to occur, which are described using control and feedback places. Control and feedback is both from the machine hardware, keeping the control software informed of the state of the real system and updating it as necessary, and from other software elements, co-ordinating the actions of this module with any others present in the system.

A net such as this will usually be drawn with the inputs and outputs to the right of the Petri net representing the interface with the machine hardware. The inputs and outputs to the left of the net represent the interface between the pneumatic cylinder control software and some higher-level control software. An important fact concerning the net representation shown is that despite the slightly different symbols used for the different types of place, they behave in the same way, there is no hidden meaning to each different representation. In the next few sections the elements of these nets are described in some detail along with their graphical representations.

4.3 Elements of Structured Petri Nets

An important aspect of the current work is the practical application of structured nets and consideration has been given primarily to the control of manufacturing systems. For the theoretical nets of Chapter 3 to be applied to a practical control problem, the net elements must be interpreted in a clearly defined and consistent manner. The graphical symbols used to represent all the elements of a structured Petri net are shown in Figure 4.6. *This figure may be compared to the symbols used in the original* net definition presented in Chapter 2.

4.4 Interpreting Net Elements

Each of the elements shown in Figure 4.6 has a specific interpretation. The interpretations do not modify the behaviour of the net elements, but instead modify the meaning of their behaviour. The addition of textual descriptions to places will provide further interpretations on the meaning of their behaviour. The structured Petri nets described here are actually a simple form of Petri net, but the representation gains modelling power by the use of interpretations rather than by the use of extensions to the formalism, and therefore the modelling power is increased without increasing the complexity of the formalism.

The implementations of the net elements are described in the following sections.

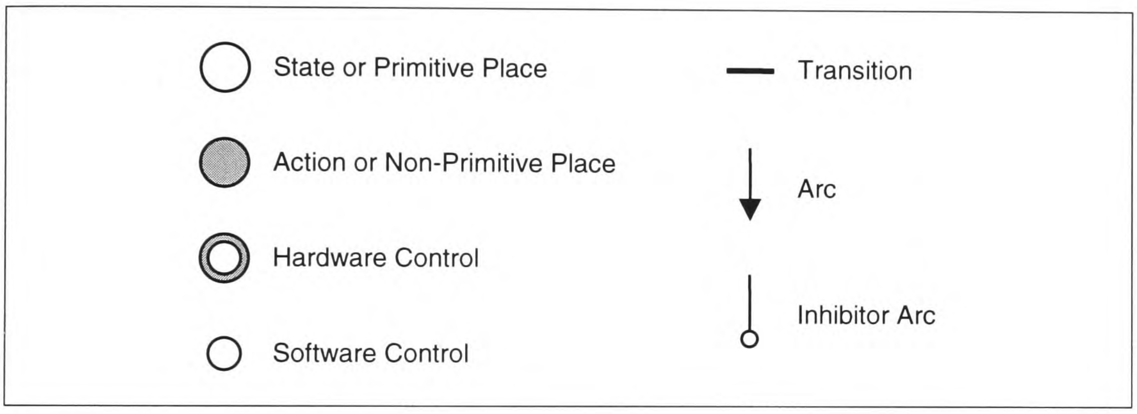

Figure 4.6 Symbols used in the graphical representation of Structured Petri nets

4.4.1 Transitions

Transitions, as their name implies, represent the transition between system states. They also delimit the start and the finish of non-primitive events. As stated in section 4.1 any event that takes a finite amount of time can be represented as a state (or sequence of states), so here transitions are interpreted as primitive events, events that are considered to be instantaneous, therefore taking zero time to occur. The firing of a transition will thus take zero time, with all input tokens being consumed and all output tokens being produced simultaneously. The issues arising when implementing such instantaneous transitions are discussed in Chapter 7, with particular reference to implementation on a PLC.

4.4.2 Places

Places are subject to a number of interpretations depending on their context. In most cases they behave in the same way although the meaning ascribed to the presence of a token in each type of place is different. The various interpretations for places are listed as follows:

- Primitive Places (also called State Places)
- Non-Primitive Places (also called Action Places)
- Control Places, which consist of three subtypes:

  Software Controls

  Hardware Controls

  Feedback Places

Each of the place types is now described in turn.

Primitive Places (State Places)

Primitive places, or state places, are used to represent conditions or states of the system. If a primitive place is marked then the state represented by that place holds (is true). If a primitive place is not marked then the state represented by that place does not hold (is false). In Figure 4.7 places p_1 and p_5 are primitive places or state places. The token in place p_1 indicates that the machine is in its ready state. A token in place p_5 would indicate that the machines' task is complete. These are both States of the system and do not represent any non-primitive action. By its very nature, a state place may only contain a single token, since it may only have one of two possible states (true or false).

Non-Primitive Places (Action Places)

Non-primitive places, or action places, are used to represent the actions, or tasks, that occur during the operation of the system. They are described as non-primitive places as they represent sequences of non-primitive events occurring in the system (a non-primitive event being one that does not take zero time, such as a robot placing a part in a milling machine). These can be likened to the non-primitive transitions described in (Peterson, 1981). If a non-primitive place is marked it indicates that the action represented by that place is currently being carried out. If a non-primitive place is not marked then this indicates that the action represented by that place is not being carried out. This will be either because the action has not been requested or the action has been completed. Non-primitive places may contain, at most, one token, as they may only have one of two possible states (the action is being carried out, or the action is not being carried out). In Figure 4.7 places p_2, p_3 and p_4 are non-primitive places.

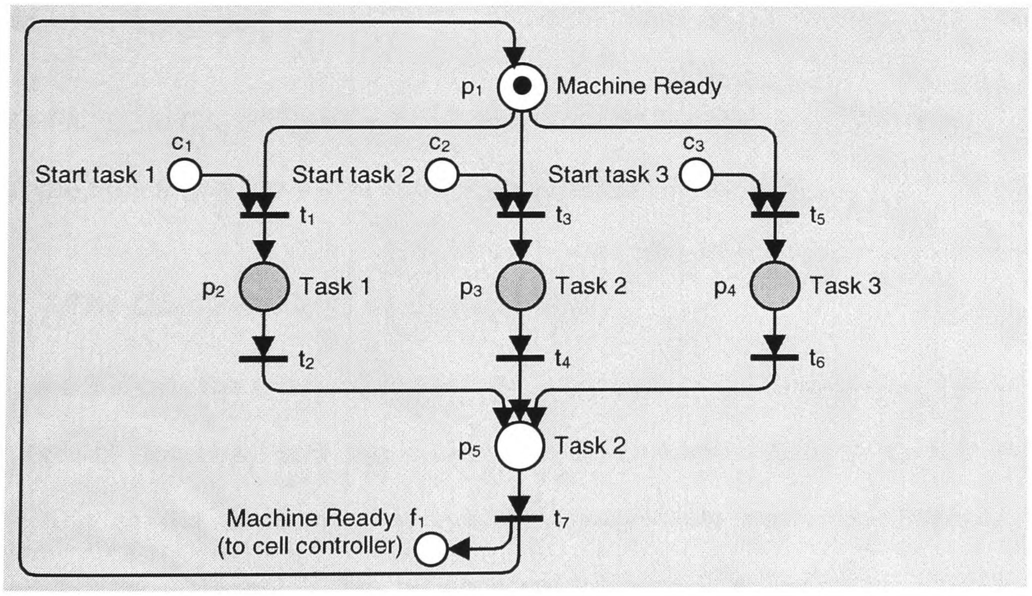

Figure 4.7 A primitive place, p_1, is used to indicate that the system is ready

Each non-primitive place is associated with a subnet, describing the action that the place represents. These are discussed in section 4.6.

4.4.3 Control places

Control places are used to control the occurrence of actions and changes of state within a system. They are the means by which external inputs (hardware and software controls) and outputs (feedback places) can be added to each Petri net module. External inputs may be received from a number of sources and thus both control and feedback places will be linked to a number of different system elements. These elements include:

- Supervisory controllers, or any other automated system.
- A human operator - via a software interface or through hardware switches and contacts in the form of a control panel.
- From the hardware of the system under control (as inputs from sensors).
- A safety subsystem, which takes over operation of the system when an unsafe situation is detected.

Places c_1, c_2, and c_3 in Figure 4.7 are control places, the control signals of which may originate from any of the above mentioned sources.

4.5 The Controller and its Environment

Figure 4.8 shows the relationship between a manufacturing workstation controller (or the control software) and the other system elements listed in section 4.4.3. According to the terminology of system theory these elements constitute the controller's environment. The direction of the arrows indicates the direction of information flow between the system elements. The controller may be any kind of manufacturing controller such as a PLC or a general-purpose computer. In the latter case, the controller may include output devices such as a printer or monitor and input devices such as a keyboard or mouse.

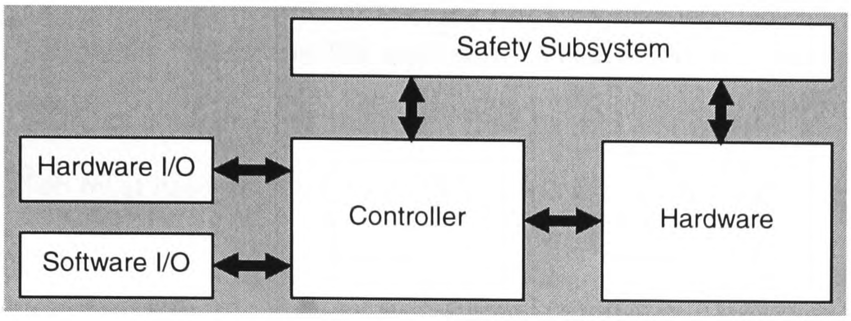

Figure 4.8 The relationship between a controller and its immediate environment

4.5.1 Hardware

This represents the physical machine, or workstation that is to be controlled. As already described, it contains a variety of output devices which are actuated by the control software and a number of input devices that are monitored by the control software.

4.5.2 Safety subsystem

The safety subsystem monitors system hardware in order to detect unsafe conditions. If such a condition arises it will take control of the system in order to restore the system to a safe state. This subsystem will include hardware interlocks and any software routines that may be required to implement safe shutdown. If the safety subsystem is to take control of the system, then the main controller must be informed of this occurrence. It is possible that some of the routines carried out by the safety subsystem are actually carried out through the main controller.

4.5.3 Hardware and software I/O

Hardware I/O represents any devices attached to the system via a hardware control panel, except for emergency stops and safety interlocks, which are directed through the safety subsystem. Software I/O represents either I/O from software systems such as SCADA, or a supervisory controller. This covers any communication over a network, which must itself originate from some other device or controller attached to the system.

The variety of sources of control software means that there is some requirement for variety in the representation of control places.

4.5.4 Hardware controls

Hardware controls were introduced in section 4.2.1 and are linked to the physical input devices that are attached to the system under control. These control places provide feedback from the system hardware to indicate the current state of the system and also whether a requested action has been completed. The input devices attached to hardware controls will typically be sensors and switches (either closed or open contacts), which provide a discrete state feedback. Thus when the physical device is on (either closed or open depending on the type of contact) the hardware control is marked. Conversely when the physical device is off then the hardware control is unmarked. The hardware control is the only element presented here which does not follow the formal Petri net definition. The hardware control is completely controlled by the device to which it is logically attached. Thus if a hardware control is the input to a transition which at some instant fires, the token may remain in the hardware control after firing if the physical device is still on.

Hardware controls are also attached to the controller from hardware I/O systems that might be present on a control panel (see Figure 4.8). Such devices act in the same way as switches connected to the system under control, except here they are used to initiate actions rather that provide feedback as to the state of the system.

In summary, hardware controls both provide feedback on the current state of the physical system, and are used for the purpose of synchronisation. For example, a limit switch attached to an actuator provides feedback as to the state of that actuator. A proximity switch will indicate that there is an item present, which may then permit another process to start, hence providing synchronisation.

4.5.5 Software controls

A software control represents the transmission of information either by the control software itself, or by the software I/O described earlier (Section 4.5.3). The information transmitted by the software control is usually a request for the system or one of its subsystems to perform an action. As such, software controls are more commonly used for communication, although in some cases they will be used for synchronisation between separate systems. Synchronisation within a particular system is usually carried out by primitive places. If software controls are implemented in the same way as hardware controls then any mechanism for detecting errors in the execution of a system by the use of hardware places can also be applied to the software places (and vice versa). This line of reasoning is expanded upon in Chapter 8.

4.5.6 Feedback places

In a complex system, which is required to carry out many functions using the same actuators, simple state feedback is insufficient to provide information on the progression of the system through a complex sequence of events. Therefore additional information concerning this progress is required. The advantage of using a modular scheme to design control software is that once a module has finished operation, we know that a particular part of the sequence has been carried out. The mechanism used to indicate that a module has finished its operation is the feedback place.

In order to monitor the state of the system and to implement handshaking signals, it is necessary to provide some form of feedback from the controlled system to the controller. This feedback is also implemented in the form of feedback places.

As indicated in the discussion of hardware controls (section 4.5.4), there is a close correspondence between feedback places and control places. This is discussed in more detail in section 4.7.

4.6 Subnets

A non-primitive place will have a subnet associated with it. Each subnet may be implemented by either a Petri net, or by some other formalism. The term subnet will be used whatever the actual method of implementation. The link between subnets and non-primitive places has already been discussed briefly in Chapter 2.

Assuming the method of implementation reflects the behaviour of the Petri net, a subnet is always running (or, an action is currently being carried out) if there is an occurrence of a token in its associated non-primitive place. In a hierarchical structure, this non-primitive place appears in the parent of the subnet (often the control net). The non-primitive place receives the token by the firing of one of its input transitions. Therefore the non-primitive event itself is triggered by the input transition to its associated non-primitive place. This is the case since, as stated in section 4.4.1, transitions fire instantaneously.

4.7 Creating a Hierarchical Structure

A system can be broken down into its constituent subsystems and each subsystem can in turn be broken down, until smaller manageable units are obtained. At each level there is control and co-ordination of the subsystems at the level below. Thus there is obtained a hierarchical structure starting with the overall co-ordination and control of the machine at the top level and ending with the co-ordination of the physical output devices at the bottom level of the structure. The number of levels in the system is dependant on the system itself and often the particular subsystem divisions preferred by the designer. An example of a three level control structure is shown in Figure 4.9 with the control net residing at the top level, various subnets at the intermediate levels and at the bottom, the output nets, which may be used to model the output devices.

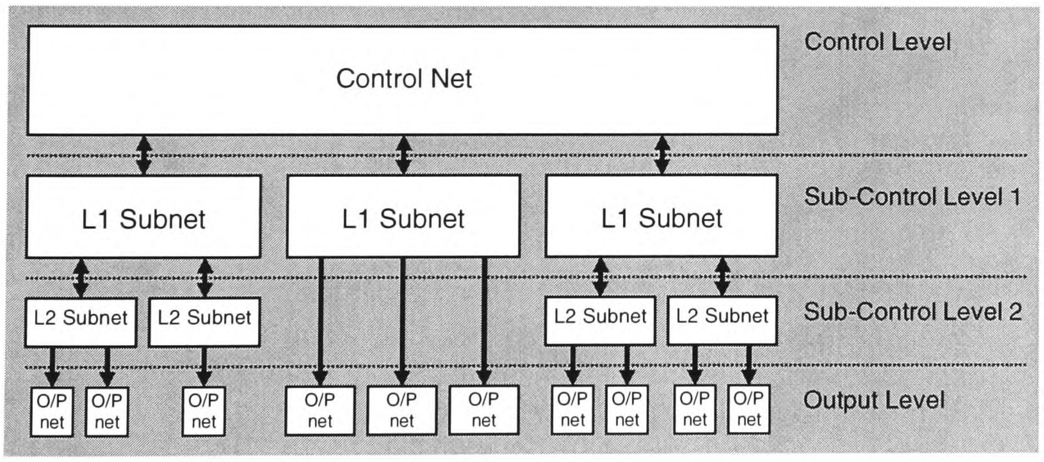

Figure 4.9 A control structure with three levels of control

The structure described in Figure 4.9 is based on that described in Chapter 2, although a point to note is that there is technically no limit to the number of sub-control levels that can be used, and not all subsystems must contain the same number of levels. Guidelines for the development of a control structure are given in Chapter 5.

4.8 Joining Petri Net Modules

A token appearing in a control place is used to trigger an event or action in a particular subsystem. The control place appears in the subsystem that describes the event or action, yet the token that triggers the event originates from another part of the system. A key factor to the success of the method presented here is the means by which these modules are joined and the way in which the communication is represented. A control place receives a token from another part of the system. If the other part of the system is a Petri net, then how do we transmit the token? The answer is the use of a shared place such as places p_c and p_f shown in Figure 4.10.

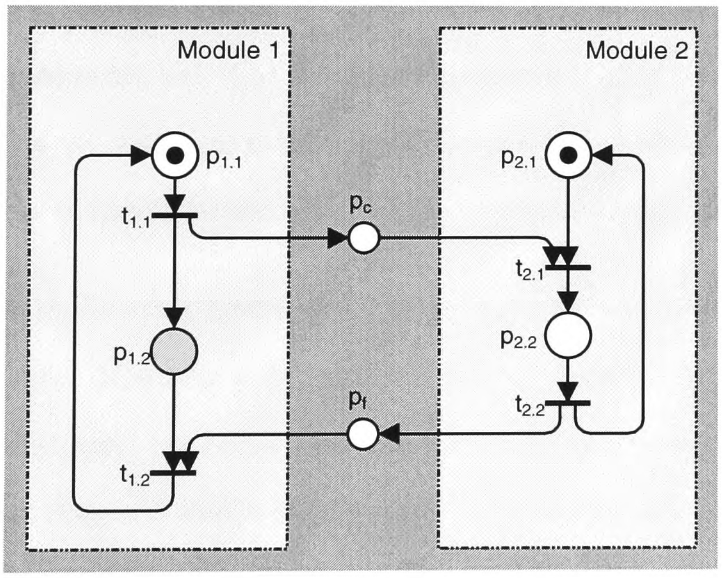

Figure 4.10 A communicating pair of Petri net modules

Figure 4.10 shows a pair of communicating Petri net modules, with Module-1 acting as a controller and Module-2 being the controlled subsystem. The figure shows that places p_c and p_f actually belong to both nets, and that the function of each is altered depending on which net is being studied. From the context of the net in Module-2, place p_c is a control place controlling the firing of transition $t_{2.1}$, and p_f is a feedback place indicating that the action represented by the net of Module-2 has been completed. However from the context of the net in Module-1, p_f is in fact a control place controlling the firing of transition $t_{1.2}$, and in a more complex system, p_c could easily represent a feedback place allowing perhaps a further action to take place in Module-2. Any feedback place must act as a return signal to the controlling element that initiated the action or state change.

This relaxes somewhat the roles of controller and controlled system as described in (Holloway et al, 1997) since a sub-system may spend some of its time acting as a controller to another sub-system and the remainder of its time being controlled by other sub-systems. If two nets with external inputs and outputs are defined, with the outputs of the first acting as the inputs to the second, and the outputs of the second acting as the inputs to the first, then the result is a communicating pair, such as those shown in Figure 4.10. Each net in the pair can be termed a module (in terms of the modular design of software), as there is a well-defined interface between them.

Each of the nets shown in the control structure of Figure 4.9 would be constructed as a Petri net module, providing a stronger method for development of the control structure than that used previously (see Chapter 2). The modular approach also allows more flexibility in the design of control code for manufacturing systems. There is a move towards more distributed control in manufacturing systems, and such a modular representation allows the construction of distributed controllers, without requiring any modification to the formalism.

The Petri net modules presented in this chapter provide a useful tool for creating manufacturing control code, but without any method for applying them, they may easily become as complex and difficult to maintain as an unstructured Ladder Logic program. Not only is a modular structure needed to provide flexibility, but also a method and guidelines for software design are necessary to enable those without vast experience to benefit to some extent from the method. This method is the subject of the next chapter.

4.9 Chapter Summary

This Chapter described how the structured Petri nets defined in the previous chapter are used to describe elements of a manufacturing system, which can be considered as Petri net modules. Each module is then able to communicate with other similar modules by the use of its control and feedback places. The chapter also more formally presents the graphical elements of structured Petri nets and describes in some detail the interpretation of such elements. It is the ability to create self-contained, communicating modules from structured Petri nets that allows a control structure to be developed. This control structure is described in the next chapter.

4.10 References

David R. and Alla, H., 1992, *Petri nets and Grafcet: Tools for modelling discrete event systems*. London, England: Prentice Hall.

Holloway, L. E., Krogh, B. H., and Giua, A., 1997, "A survey of Petri net methods for controlled discrete event systems." *Discrete Event Dynamic Systems: Theory and Applications*, **7**, pp. 151-190.

Ichikawa, A. and Hiraishi, K., 1988, "Analysis and control of discrete event systems represented by Petri nets." In *Discrete Event Systems: Models and Applications*, Berlin, Germany: Springer-Verlag, pp. 115-134.

Krogh, B. H., 1987, "Controlled Petri nets and maximally permissive feedback logic." In *Proc. 25th Annual Allerton Conference.* University of Illinois, USA, pp. 317-326.

Peterson, J. L., 1981, *Petri net theory and the modeling of systems.* Englewood Cliffs, NJ, USA: Prentice Hall.

Chapter 5

Developing a Control Structure

There are a number of methods described in the literature for the synthesis of Petri net controllers for manufacturing systems. Many of these rely on the experience of the designer, both in terms of the manufacturing system itself, and the actual use of a Petri net model. This chapter presents a new method for developing control software for a manufacturing system based on the Petri net modules that were described in Chapter 4. The new approach, which is loosely based on structured methods, is introduced first, and is then compared with approaches based on Petri nets found in the literature.

5.1 The Aims of the Method

The aims of introducing the design approach were described in Chapter 1, namely to facilitate the automation of manufacturing control software development. Given an initial specification, the software designer requires a development system that will check the specifications, and ultimately produce working, verifiable code. The control software produced should also allow error detection, diagnosis and recovery.

In order to produce such a development system the method used for software production must be structured in some way, with each stage in the development process feeding the next. Even so the structure of the method should not prevent some feedback and iteration between different stages.

The method proposed here has used a modular programming approach. This allows each subsystem and its interface with the other parts of the system to be clearly defined. Once the interface is defined, the internal behaviour of the subsystems can be modified, without the need for modifying the rest of the system.

5.2 The Method

Using the experience gained from the development of control software for two different systems a series of steps has been developed to aid development of future systems using the structured Petri net modules already discussed in earlier chapters. These steps are discussed fully in the next few sections and are summarised in Figure 5.1.

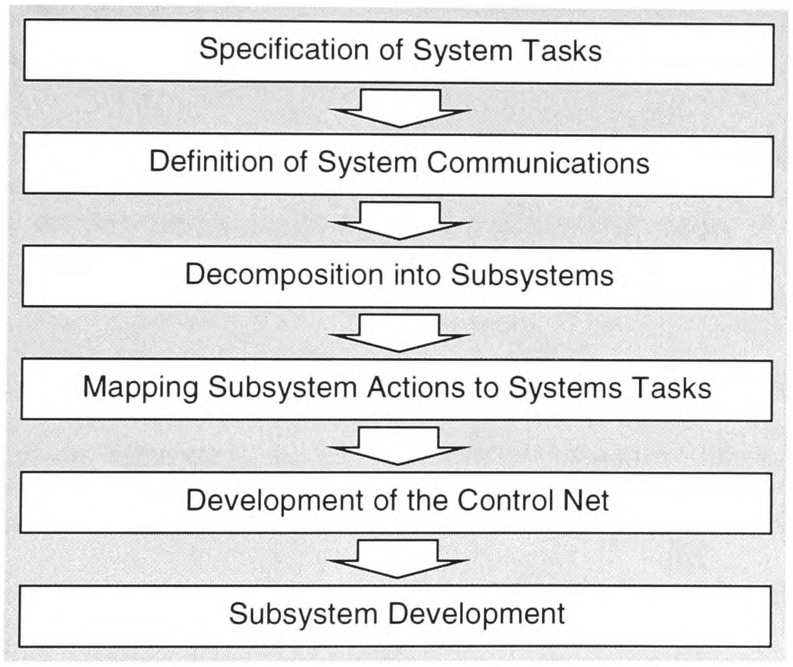
Specification of System Tasks
Definition of System Communications
Decomposition into Subsystems
Mapping Subsystem Actions to Systems Tasks
Development of the Control Net
Subsystem Development

Figure 5.1 Sequence of steps for the Petri net controller development

These steps do not represent a full life-cycle model for the system; instead they represent a design stage for the software. As previously noted there will be a degree of iteration between the steps. It should also be noted that the design process is recursive because each subsystem will be designed in the same way as the system itself until the are no further subsystems to be developed.

The advantages of modular design are most apparent when software is being designed and maintained. These steps represent some of the analysis stages of systems development but mainly concentrate on the design. A discussion of how the Petri nets may be implemented will be provided in Chapter 7. The following sections deal with each of these steps in turn.

5.3 Specification of Systems Tasks

System tasks are those that the system under consideration is required to carry out. Initially these tasks must be defined from the viewpoint of those environmental entities that will be requesting them (see Section 5.4.1). These tasks represent an answer to the question *'What is the system going to do?'*.

Each system task is listed and may be given an appropriate abbreviation that will allow a more concise description of the system during later stages of the software development.

This may seem an obvious task but it is essential that careful thought be given to these tasks at an early stage so that the view of the system is not clouded by implementation detail.

5.3.1 The initialisation task

One system task, which has been used throughout this development and is considered common to any system, is the Initialisation Task. This task is run when the system is powered up, possibly being initiated by an instruction from its supervisor. There are two stages to the Initialisation Task, Software Initialisation and Hardware Initialisation. Only when the software initialisation has been completed will the hardware initialisation be carried out.

Software Initialisation

Software initialisation is carried out to ensure that the software elements of the system, such as timers and counters, have been set up correctly. The software initialisation also initialises each of the Petri net modules, placing tokens in their appropriate places. This means that the initial state of the system can be predicted, as the system can be placed into any desired state before it becomes available for operation.

Hardware Initialisation

Hardware initialisation is carried out to ensure that the state of the system hardware is known and is ready to start any requested actions. The hardware initialisation stage will require checks on sensors to indicate the current state of the system hardware and will also include some short sequences that check more complex parts of the system (e.g. those whose state is not easily detected by feedback from sensors). These sequences will also drive the hardware of the system to the desired initial state if it is not found to be in the desired state on start-up.

Once the initialisation task has been completed the system can enter a ready state indicating that it is ready to receive commands. On entering the ready state the system will inform its supervisor, or other users that it is able to accept requests for actions.

5.4 Defining the Communications

The second step in the design process represents one of the key features of the development method proposed in this thesis. It requires that the communications signals between the system and its environmental elements be clearly and precisely defined.

These communications signals represent the interface between the system and its environment. This interface will be described without regard to how it will eventually be implemented, whether it is between hardware elements, software elements or both hardware and software. Once this interface has been established and agreed, the internal design of the system can be developed, tested and modified with no effect on the further development of the wider system. It is therefore essential that this interface is defined and agreed early in the development process. Clearly this step cannot be completed to a satisfactory degree until the previous step has been completed.

Early development of the interface between systems and their wider environment allows the environmental elements to be developed either concurrently, or by staged development (Sommerville, 1996). This argument will also apply to the development of subsystems.

It should be noted that at this stage the implementation details have not been considered. The first two steps of the process represent the specification of the system's functional requirements.

5.4.1 Describing the interface

The description of the interface between the system and its environment should include all the functions that the system is expected to carry out, along with all the feedback signals that the system will send back out to its environment. It is useful at this stage to draw a diagram to help visualise these control and feedback signals between the system and its environment. An example of such a diagram is shown in Figure 5.2.

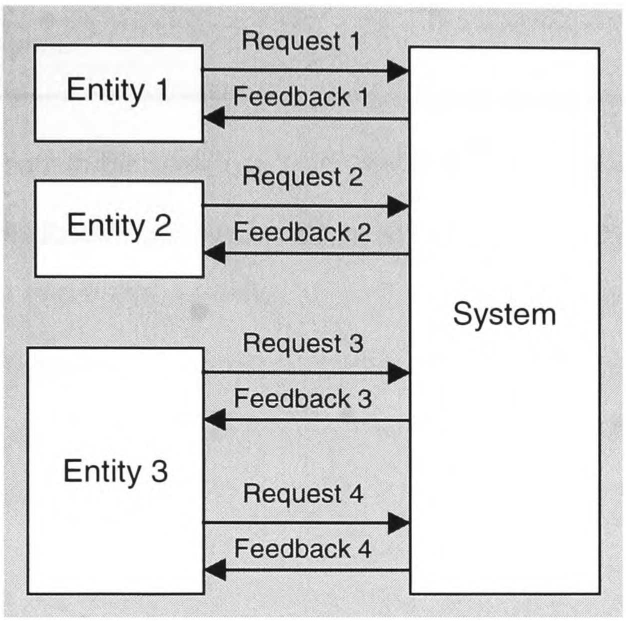

Figure 5.2 Control/Feedback Diagram for a system with many environmental entities

It is likely that there will be more than one potential source for these control and feedback signals and these should be indicated on the diagram. Such a diagram can be compared with the use of the context diagram in SSADM (Structured Systems Analysis and Design Method) (Eva, 1995) which is used to describe the flow of information between an information system and its environmental elements.

5.4.2 Control/feedback pairs

The communications between the system and its environmental entities are defined by establishing a set of control/feedback pairs. At this point a rule is introduced for establishing these control/feedback pairs. This is stated in Rule 5.1 as follows:

Rule 5.1

For each control signal from a user of the system there must also be a feedback signal.

The control signal sent to the system can be interpreted as a request for some action or service. The feedback signal will indicate either that the action or service has been completed, or that the request has been processed and that the system is able to receive another request. The feedback signals at this level provide an indication of the status or state of the system, and may be received by a supervisory controller or user of the system.

Rule 5.2 describes the relationship between the control and feedback signals that make up a control/feedback pair.

Rule 5.2

A feedback signal must return to the originator of its associated control signal.

It is possible that a number of control signals will be paired with the same feedback signal. For example, consider a system that is able to perform a number of actions and a different control signal is used to request each action. A single feedback signal may be used to indicate that the system is ready to receive another request for action. In order for Rule 5.2 to apply in such a case there would need to be some broadcast mechanism so that all potential requestors know when the system has become available. This is again an implementation detail and as such will not affect the design of the system at this stage.

5.5 Decomposing the System into its Constituent Subsystems

The system under consideration will usually be made up of a number of subsystems, each with a particular function of its own. This stage in the software development identifies those subsystems and the actions that each subsystem will carry out.

The number of subsystems into which the system is decomposed is generally based on the logical arrangement of system hardware, such as groupings of pneumatic actuators to form a more complex manipulator. However, there will often be elements of the system that can be seen as being part of more than one subsystem or single elements that do not seem to belong in any subsystem. The manner in which these are dealt with will be based purely on the experience and preference of the software designer. A few guidelines can be borne in mind based on experiences gained in developing the test systems presented in Chapter 6 and on relevant software development literature (Sommerville, 1997), (Pressman, 1998), (Meyer, 1998).

Cost

The number of subsystems chosen will affect the number of modules required for the software architecture. The cost of the software development and implementation (certainly in terms of time) increases with the number of modules used in the software architecture. As the number of modules increases there is a point at which further modularity becomes uneconomical (Pressman, 1998).

Size

As already mention in Chapter 2, the size of the control code becomes important when using low cost PLC's. If a system is broken down into a large number of subsystems, then a large number of modules will be required to represent those subsystems. Each module requires state and communication places, and the more modules, the greater will be the overhead represented by these places. Therefore to minimise the size of the final code, the number of modules used should be kept to a sensible minimum.

Complexity

If the graphical nature of the Petri net is to be used to its full potential, then it should act as an aid to communication. One of the aims of the development method is to minimise the complexity of the control net thus capitalising on the Petri net's strengths as a graphical formalism. Therefore, it is not just the control net that must be kept simple but the whole of the net structure. When considering the number of levels to be used in the design of the control software hierarchy, some guidance can be taken from the graphical technique of data flow diagramming as used in SSADM (Structured Systems Analysis and Design Method). The rule in SSADM is to have no more than three levels of diagrams representing the system (Eva, 1995). At the fourth level Data Flow Diagramming uses a textual description of the process, called an Elementary Process Description. Such a description could be described in the graphical notation of a Petri net (as could the other levels in a Data Flow Diagram) see (Eva, 1995). Data flow diagramming also uses a Level 0 diagram, called a Context Diagram. This is equivalent to the Communications Diagram as described in section 5.4.

One problem with representing the control structure as a hierarchy, is that the complexity of the control net increases with the number of concurrent elements in the system. This is because the control net deals with the co-ordination between the subsystems, and if the control net is to provide adequate information as to the current state of the system, it must contain a non-primitive place for each concurrent operation. Two or more concurrent processes could be modelled by a single non-primitive place. But this increases the complexity of analysis when a fault occurs in the system (see Chapter 8).

5.5.1 Subsystem actions

Once the system has been divided into its constituent subsystems, the actions that each subsystem will carry out can be described. These actions are listed in the same way as the System Tasks in the initial stage of the development (see section 5.3), and a descriptive code is given for each. Each subsystem should perform a set of related actions, and this can be used as a test to ensure a good choice of subsystems has been made.

5.6 Mapping Subsystem Actions to System Tasks

Having defined the subsystem actions (section 5.5.1), these must now be used to describe the system tasks (section 5.3). This requires that a sequence of subsystem actions be used to describe each system task. Each sequence will start with a state place (representing a control signal from the system's environment) and terminate with a state place (representing a feedback signal returning to the system's environment). A number of sequences may share the same subsystem actions, and the same state places. At this stage any concurrent actions will be identified.

At the end of this step there is now a description of what tasks the system will perform in terms of its subsystem actions. This now leads directly into the development of the Petri net model.

5.7 *Constructing the Control Net*

The construction of the control net relies on the concept of a path. This is a different definition to that given in (Krogh and Sreenivas, 1987), which has been used specifically for the approach presented in this chapter.

Paths

A path is a sequence of alternating places and transitions joined by arcs. The concept of a path is used in the design process to represent each system task in terms of the subsystem actions. Thus each net in a control structure will consist of one or more paths depending on the number of actions described at that level of abstraction.

A path starts with a controlled transition (one that has at least one control place in its set of input places), and ends with a feedback transition (one that has at least one feedback place in its set of output places). The control on the initiating transition and the feedback from the terminating transition must be linked to the same control net (i.e. feedback must go to the source of the control). The path may also contain concurrent activities, these concurrent activities must be initiated by the same transition (i.e. they do not contain any decisions).

Each path can be represented by a Petri net starting and finishing with a state place. The initial marking of a path will always cause a token to appear in the starting state place, and from this initial marking the path will remain live until a token arrives in its terminating state place.

5.7.1 Describing the paths

Having described each of the system tasks as a sequence of subsystem actions, non-primitive places can now be used to represent each action, and primitive places can be used to represent each state. A path net can therefore be drawn for each of the system tasks showing the sequence of actions, and any concurrent actions that occur.

At this point it is possible to identify all the transitions that will appear in the control net directly from the specifications of path sequences. However the graphical nature of the Petri nets for each path makes the positions of the transitions relative to the actions clearer.

An important task here is to identify those transitions that are shared between paths and those that belong to a single path. Applying Rule 5.3 to each path will identify these transitions.

This rule works on the basis that when a system moves between two states, then that movement will be represented by a single transition. If the sets of inputs and outputs of two transitions are the same then the states represented by those inputs and outputs are the same, and therefore the transitions are the same.

Rule 5.3

When the pre- and post- places for two or more transitions in different paths are identical, the transitions are given the same label. This is in all cases except when the transition is the initial transition in a path.

5.7.2 Merging the paths

To provide a single graphical representation of the control net (which may be necessary to construct the control code) the paths must be merged. However, it may be that the resulting control net is too complex graphically to be of any value as a systems description, it proving easier to analyse the paths individually. A mathematical representation of the paths may be equivalent to that of the merged control net. If the control code can be generated automatically from the mathematical representation (see (Hanisch et al, 1996a)) then it may not be necessary to use the control net in its graphical form at all, other than for descriptive purposes.

One advantage of using the control net is that it clearly highlights the points at which there are conflicts. It may be possible to determine these from the mathematical structure of the net, but in that case they cannot be visualised.

5.7.3 Ensuring the net is conflict free

If a single structured Petri net is taken without any of its communications signals then the net will be seen to contain many conflicts. The communication signals resolve many of these conflicts, but it is possible that some still remain. Graphically, the remaining conflicts are highlighted by merging the paths into a control net. These conflicts may be resolved by the addition of redundant state places in parallel with those places that are in conflict. These redundant places can be viewed as memory as they allow the system to remember an earlier firing of a transition which is used later on in the sequence.

The following algorithm can be used to locate the points in the control net at which redundant state places are required.

Algorithm 5.1

```
For each place with more than one output arc:
   For each output arc:
         Is there a transition to which this place is the only input?
         If yes add a state place in parallel with the current place.
         If no then go to next place:
End
```

Algorithm 5.1 searches for any places that have more than a single output transition. It is only these places that will be potentially involved in any conflicts. Once this set of places has been identified, then each output transition of the place is examined to see if it has only this place as its input. If it does then there is a need for an additional place in parallel with the place that is currently being examined. An example of such a conflict is shown in Figure 5.3.

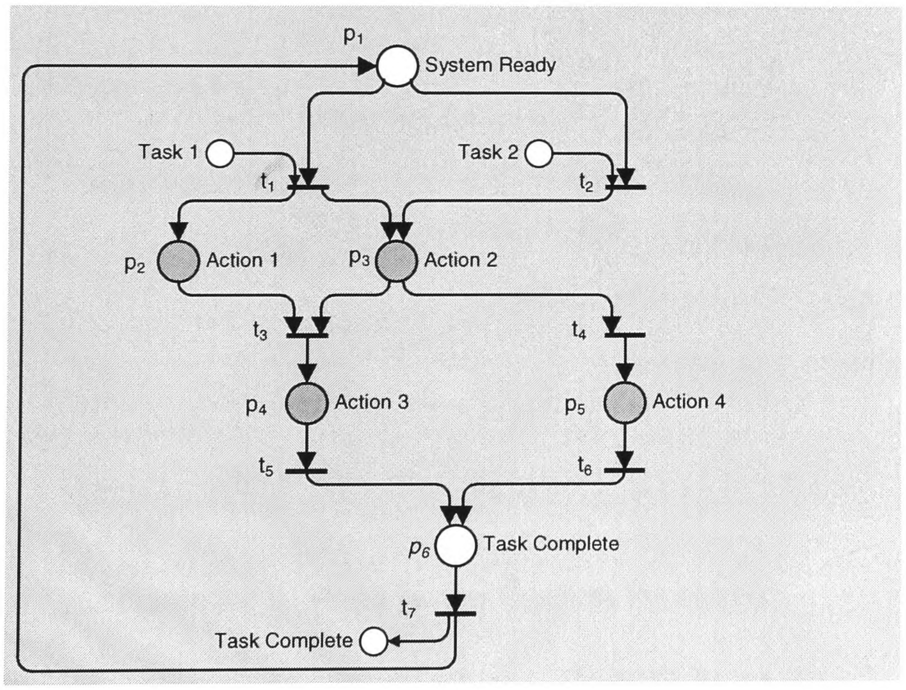

Figure 5.3 Action 2 is shared by both tasks and therefore causes a conflict between transitions t_5 and t_6.

The Petri net shown in Figure 5.3 details part of a subsystem that may carry out one of two tasks. The net shown is the result of merging the paths for each of the two tasks. Task 1 incorporates Action 1, Action 2, and Action 3. Task 2 incorporates Action 2 and Action 4. It can be seen that if Task 2 is requested, Action 2 is carried out followed by Action 4. However, when Task 1 is requested, Actions 1 and 2 are carried out concurrently, but when they are complete there are two enabled transitions (namely t_3 and t_4). To complete Task 1 according to the system specification, transition t_5 should be the next to fire, but this is not the only possible outcome especially in the light of implementation methods (see Chapter 7). The only way to ensure that the correct path is followed is to resolve the conflict. This is done using Algorithm 5.1.

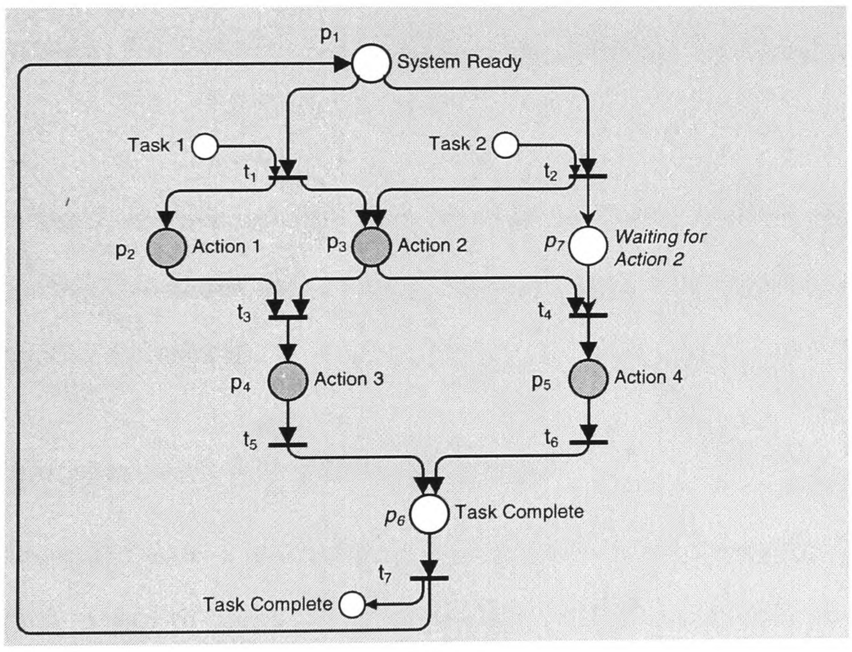

Figure 5.4 Place p_7 now resolves the conflict.

The result of applying Algorithm 5.1 to the net of Figure 5.3 results in a new net, which is shown in Figure 5.4. It can be seen that place p_7 now prevents transition from being enabled when Task 1 is requested, thus removing the conflict between transitions t_5 and t_6.

5.8 Subsystem Development

The development method proposed in this work is a recursive process. This is because once the system itself has been developed according to the method described above, then the subsystems are developed using the same method. This is repeated until the required depth has been reached. The lowest level of description presented here describes the output devices that are attached to the system. In some cases it may be that a particular subnet will not be described as a Petri net. In this case there will be no further development of that subsystem using this approach. However the subsystem is still useable with this method because the interface between the non-Petri net subsystem and the Petri net description will be the same.

System elements that are not Petri nets must be taken into account earlier in the design process. This allows their existing inputs and outputs to be used by the Petri net description of the system.

5.9 Other approaches to controller design

In the literature there are a number of proposed methods for the design of Petri net models. Only a few of these are actually used to implement controllers for real systems. Also there are some other controllers that are implemented for real systems but there is no design approach other than that using the intuition of an expert. The next few sections describe these approaches and make some comparison between them and that proposed here.

5.9.1 Venkatesh

The method for developing a control structure can be compared directly with that given in (Venkatesh et al, 1994). The algorithm presented in (Venkatesh et al, 1994)) is based on the design procedure described in (Pessen, 1989) for the design of Ladder logic programs. This was used despite the apparent lack of any intention to realise the control code in Ladder Logic.

The design approach here may still be based on the same specifications but that is where the similarity ends. It is felt that the design of Petri net controllers based on a method for ladder logic design will unnecessarily restrict the Petri net model and ties the designer to the more restricted approach required for straight ladder logic design.

5.9.2 Net condition/event systems

Net Condition/Event Systems have been proposed by Hanisch and Rausch (see (Hanisch et al, 1996a), (Hanisch et al, 1996b), (Rausch et al, 1996), (Rausch and Hanisch, 1995), (Hanisch and Rausch, 1995)) for the synthesis of controllers for manufacturing systems. The work is based on the solution of forbidden state problems, which take the full state representation of the system and synthesise the controller that will prevent the occurrence of the events, which cause certain unwanted states, to hold.

The Net Condition/Event Systems is a Petri net like representation, which makes use of modules to describe different parts of the system. Communication between these systems parts is carried out using event signals, which force transitions in other parts of the system to fire, and condition signals, which enable transitions in the same way as the control places used here.

Event signals are used to synchronise the occurrence of different events in the system, something which is achieved here using control places. The authors have also managed to develop an automatic code generator based on their nets, which produces code for limited systems, which conforms to the IEE 1131 standard.

The main drawback of these systems is the complexity of the model, along with the complexity of the graphical notation. The need for both condition and event signals is questionable since the occurrence of an event can be indicated by a condition signal.

Another limitation of the Net Condition/Event systems is the types of system to which they have been applied. The examples provided are all of small parts of a larger system, with limited complexity. Even though some idea of possible sequences of events is provided, there is, as yet, no information as to how these systems are linked with larger super-systems, and no information on the initialisation of such systems.

Forbidden state problems rely on state feedback from the plant model and this is obtained using structured Petri nets by the state feedback of the hardware places. It would be interesting to develop an example of a forbidden state problem using the notation presented here. It is believed that analysis of the system would be a lot simpler, due to the presence of only one type of communication signal, which is the same as the formalism for the state representation.

5.9.3 Zhou and DiCesare

The synthesis method proposed in (Zhou and DiCesare, 1993) uses both a top down and a bottom up approach. The top down approach is used to define the Petri net model of the system without regard to any shared resources. The bottom up approach is used to define the interaction between the Petri net model and the shared resources whilst ensuring that the system model is live, bounded and reversible.

The development of the model is started in a top down manner using stepwise refinement and standard Petri net 'modules' to construct a 'free-choice' Petri net model of the system. The modules described are really simple Petri net structures representing sequence, choice, decision-free choice, and parallelism. The refined places are replaced with these structures, increasing the size of the net, but preserving the free-choice property of the net. The operation places, which are used to describe the top level net, are also termed A-Places. Once the Petri net structure has been laid down, the non-shared resources are added (also called B-Places). Non-shared resources are those that are used by a single operation place. Buffer places are also added at this stage (whose definition is similar to the non-shared resource places). Finally shared resources are modelled using parallel and sequential mutual exclusions. Using the mutual exclusion theories presented in (Zhou and DiCesare, 1993) ensures that the properties of liveness boundedness and reversibility are preserved.

Once again the problem of graphical complexity of the final net is not adequately handled. An example provided in the work of a FMS is very complex and this makes the use of the net as a graphical tool virtually useless. Also the method itself has a high degree of complexity which may eventually be hidden by a suitable automation tool, but relies heavily on the engineer having knowledge of both the complexities of Petri nets and how they may be applied to a particular problem. The method also separates the issues of modelling, implementation and fault monitoring, which need to be handled simultaneously.

As shown in the method proposed in this work, the development of control software needs to be related to the system being developed at an early stage. Otherwise it will be seen as too complex and will not be used. Any complexity in the development method must be hidden from the user of the method.

5.9.4 Resource Control Nets

A more recent synthesis method has been proposed by Mu Der Jeng in (Jeng, 1997a) and (Jeng, 1997b). This is a modular composition method using a bottom up approach. The modules are named Resource Control Nets (RCN) and are actually state machines, although they are described by Petri nets. The generation of a Petri net model is achieved by merging the RCN's along common transition subnets (those paths that share completely a set of transitions).

Again a problem with the approach is its formality and the complexity of the graphical description of the system. There is no indication of how each RCN is modelled, it is just assumed that the engineer will be able to carry out such a task.

5.10 Chapter Summary

At each phase in the design procedure the most important aspect is the communications between the different levels of the system controller. All the signals that pass between the different levels of the system must be defined early in the process. Once these have been defined the rest of the details can then be included and, as long as the communications are unaffected, they can also be modified with no effect on the rest of the system. In this way the communication signals play an important role in the modularity of the system, but they also provide the means by which fault monitoring can be implemented.

Another important aspect of the method shown here is its lack of complexity. This may cause the loss of some of the benefits of more formal approaches (Zhou and DiCesare, 1993) (Jeng, 1997a), but it does provide a solid foundation upon which more formal approaches can be built. The next chapter describes an application of the design method to a manufacturing workstation of reasonable complexity, which is part of a larger manufacturing system. This will clarify the design steps as outlined in this chapter.

5.11 References

Eva, M., 1995, *SSADM Version 4: A User's Guide (2e)*, McGraw Hill.

Hanisch, H.-M., Kölbel, S. and Rausch, M., 1996, "A modular modelling, controller synthesis and automatic control code generation framework." In *Proc. IFAC World Congress*, San Francisco, CA, USA, pp. 495-500.

Hanisch, H.-M., Lüder, A. and Rausch, M., 1996, "Controller synthesis for Net Condition/Event systems with incomplete state observation." In *Proc. Computer Integrated Manufacturing and Automation Technology (CIMAT)*, Grenoble, France.

Hanisch, H.-M. and Rausch, M., 1995, "Synthesis of supervisory controllers based on a novel representation of Condition/Event Systems." In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, British Columbia, Canada.

Jeng, M. D., 1997, "A Petri net synthesis theory for modeling Flexible Manufacturing Systems." *IEEE Transactions on Systems, Man, and Cybernetics - Part B Cybernetics*, **27**, pp. 169-183.

Jeng, M. D., 1997, "Petri nets for modeling automated manufacturing systems with error recovery." *IEEE Transactions on Robotics and Automation*. **13**, pp. 752-760.

Krogh, B. H. and Sreenivas, R. S., 1987, "Essentially decision free Petri nets for real-time resource allocation." In *Proc. IEEE International Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1005-1011.

Meyer, B., 1998, *Object Oriented Software Construction*, Upper Saddle River, NJ, USA: Prentice Hall PTH.

Pessen, D. W., 1989, "Ladder diagram design for programmable controllers." *Automatica*, **25**, pp. 407-412.

Pressman, R. S., 1998, *Software Engineering: A Practitioners Approach*, New York, USA: McGraw Hill.

Rausch, M. and Hanisch, H.-M., 1995, "Net condition/event systems with multiple condition outputs." In *Proc. INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, pp. 592-600.

Rausch, M., Lüder, A. and Hanisch, H.-M., 1996, "Combined synthesis of locking and sequential controllers." In *Proc. IEE International Workshop on Discrete Event Systems*; Edinburgh, UK, pp. 133-138.

Sommerville, I., 1996, *Software Engineering*. Addison-Wesley.

Venkatesh, K., Zhou, M. and Caudill, R. J., 1994, "Automatic generation of Petri net models from logic control specifications" In *Proc. International Conference on*

Computer Integrated Manufacturing and Automation Technology. Rensselaer Polytechnic Institute, Troy, NY USA, pp. 242-247.

Zhou, M., McDermot, K. and Patel, P. A., 1993, "Petri net synthesis and analysis of a flexible manufacturing system." *IEEE Transactions on Systems, Man and Cybernetics*, **23**, pp. 523-531.

Zhou, M. and DiCesare, F., 1993, *Petri net synthesis for discrete event control of manufacturing systems*, USA: Kluwer Academic Publishers.

Chapter 6

Application of the Control Structure

The Petri net modules described in Chapter 4, and the method of controller design described in Chapter 5 are, in this chapter, applied to a real system. The system presented here was used as a basis for development of both the control structure and the design method. The end result would be a software control structure that enabled fault monitoring at a variety of levels, both in hardware and software. Details of the fault monitoring work are presented in Chapter 8.

6.1 Example: A Raw Materials Station

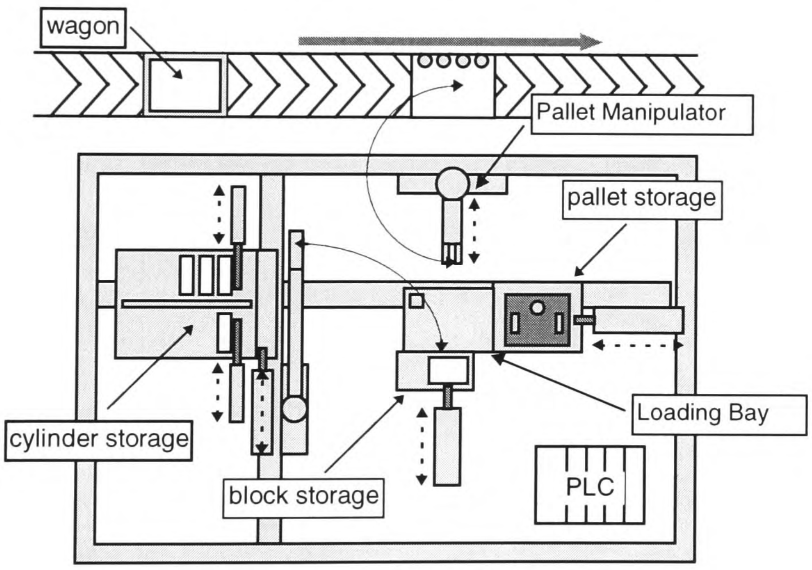

Figure 6.1 Layout of raw materials station

A Raw Materials station consisting of in total 15 pneumatic actuators will be used as an example of how the control structure for a single machine is developed. This particular system was used as a test bed throughout the development of the design method and the fault monitoring method described later. The layout of the raw materials station is shown in Figure 6.1.

6.1.1 Function of the raw materials station

The station provides, on demand, requested items of raw material to a larger manufacturing system. The particular types of raw material provided are a Perspex block, of fixed dimensions, and aluminium cylinders with one of two different diameters. Each of these three raw materials has its own storage position on the station. The raw materials are transported around the system on pallets, which are designed to take any of the three raw materials present. These pallets are also stored on the raw materials station, and each raw material is loaded onto a pallet before being despatched onto the conveyor. Pallets may also be despatched with no raw material on them.

Empty pallets may also be removed from the conveyor and loaded with raw materials, before being replaced on the conveyor again. Such pallets cannot be placed in the pallet storage area, and must be used before any other operation is carried out.

6.2 System Requirements of the Raw Materials Station

Description of System Tasks	Abbreviation
1. Provide a Perspex block and a pallet from store.	Get block (+ store)
2. Provide a cylinder from slope 1 and a pallet from store.	Get cyl1 (+ store)
3. Provide a cylinder from slope 2 and a pallet from store.	Get cyl2 (+ store)
4. Provide an empty pallet from store.	Get pallet (store)
5. Provide a Perspex block and a pallet from the conveyor.	Get block (+ conv)
6. Provide a cylinder from slope 1 and a pallet from the conveyor.	Get cyl1 (+ conv)
7. Provide a cylinder from slope 2 and a pallet from the conveyor.	Get cyl2 (+ conv)

Table 6.1 System tasks of raw materials station

Initially, a top down approach is taken to the station controller design thus ensuring that all the functions of the station are designed to fulfil the goals of the system as a whole. At the highest level of the software design the main tasks of the raw materials station are described. These descriptions must initially be stated in terms of the requirements of the super-system, that is the manufacturing system as a whole. Each of the tasks is assigned an abbreviation that will be used as its identifier throughout the remainder of the design process. The descriptions of the system tasks and their associated abbreviations are shown in Table 6.1.

6.3 Defining the Communications

The next step in the design process is to clearly define the communications signals that pass between the raw materials station and its environment. The environment of the raw materials station includes the safety subsystem as well as a supervisory controller, and a hardware control panel. For this example only the communications with the supervisory control unit will be examined and these are shown in Figure 6.2.

The supervisory controller informs the raw materials station of the type of part that is required by sending a part code. This code is interpreted by the controller of the raw material station and initiates a particular action. Other communication signals are used to inform the raw materials station that an item can be placed on the conveyor or removed from the conveyor. Each communication from the supervisory controller to the raw materials station initiates some action. A feedback signal is sent to the supervisory controller indicating the successful completion of each requested task. Also signals indicating the station is ready to start an action, and is ready to place an item on the conveyor are present.

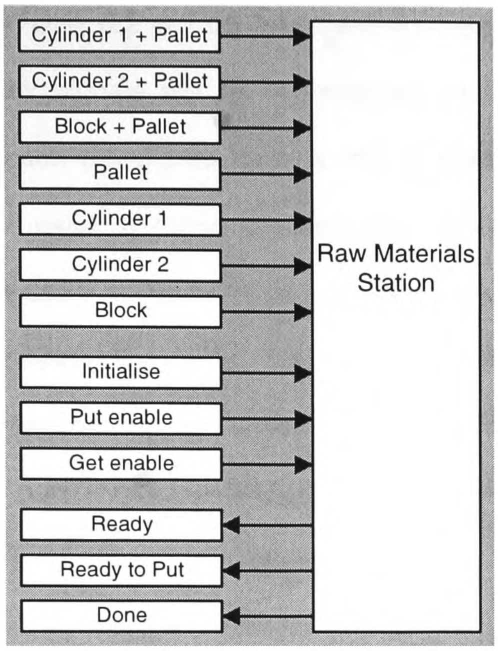

Figure 6.2 Communications between the raw materials station and its supervisory controller

It can be seen here that the number of control signals from the supervisory controller is different to the number of feedback signals from the raw materials station. This is because some of the actions share the same feedback signal. Such cases are shown in Table 6.2 where the control/feedback pairs for the raw materials station are presented.

Control	Feedback
1. Initialise	Ready
2. Get Pallet	Ready to Put
3. Get Cylinder 1 + Pallet	Ready to Put
4. Get Cylinder 2 + Pallet	Ready to Put
5. Get Block + Pallet	Ready to Put
6. Get Cylinder 1 + Get Enable	Ready to Put
7. Get Cylinder 2 + Get Enable	Ready to Put
8. Get Block + Get Enable	Ready to Put
9. Put Enable	Done

Table 6.2 Control and feedback signals for the raw materials station

The controls shown in Table 6.2 will be represented in the control net as control places and the feedback signals will be represented as feedback places. One advantage of the approach developed here is that it doesn't matter whether the supervisory controller is used to make requests for action via these controls, or whether some other system is used to make the requests (e.g. hardware or software), the interface remains the same. This means that the software constructed for the raw materials station can be tested without the aid of the supervisory controller, or any other part of the real manufacturing system. This approach was used to great effect in developing the control code for another system (to be described somewhere else). The software for this new system was designed and tested on a standalone PLC using SCADA (Supervisory Control And Data Acquisition) software to act as controller and to respond to signals sent to the manufacturing hardware.

6.4 Breakdown of Subsystem Components

The raw materials station is made up of a number of subsystems, each of which will be responsible for carrying out a part of the different functions described in Table 6.1. Again using a top down approach, these subsystems are identified along with the tasks each subsystem can perform. The number of subsystems identified will depend on the system under consideration, and on the preferences of the individual designing the controller.

6.4.1 Effect of number of subsystems on the control net

The discussion in Chapter 5 states that the control net needs to be kept as simple as possible and that the more subsystems used the larger will be the resulting control code. The number of subsystems into which the system breaks down and the number of actions each component can perform will dictate the total number of non-primitive places in the control net. For this example the station is broken down into three subsystems. These subsystems and the tasks that each will carry out are described in Table 6.3. A descriptive code is used for each of the tasks allowing a more concise representation for the later stages of the design process. These codes should be unique for each task so as to avoid any confusion.

The hierarchical structure of the control code for the raw materials station is shown in Figure 6.3. It can be seen that below the control net there are three subnets, each representing one of the subsystems (or axes). The subsystems respond directly to sensory data received from the hardware of the workstation, and produce actions by sending signals to the output nets shown below them.

Sub-system	**Function**	**Code**
A) Pallet Manipulator	1. Get pallet from conveyor	PM_get_pallet
	2. Put pallet on conveyor	PM_put_pallet
B) Loading Bay	3. Get pallet from storage	LB_get_pallet
	4. Put block onto pallet	LB_put_block
C) Cylinder Storage	5. Get cylinder from slope 1	CS_get_cyl1
	6. Get cylinder from slope 2	CS_get_cyl2
	7. Put cylinder onto pallet	CS_put_cyl

Table 6.3 The subsystems of the raw materials station and their functions.

It can be seen that in this case there is no 'multiple ownership' of the output nets by subsystems. That is, each subsystem sends signals to a unique set of output nets. This will not necessarily be the case for all systems and depends on the manner in which the system is hierarchically decomposed.

The seven tasks shown in Table 6.3 will be represented in the raw material station control net as non-primitive places.

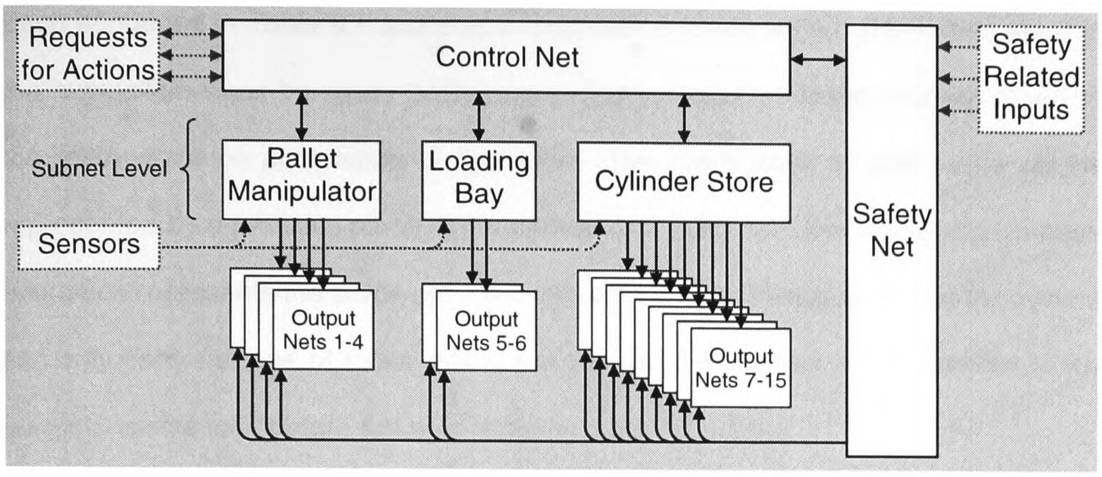

Figure 6.3 Petri net structure for the Raw Materials Station.

6.5 Mapping System and Subsystem Tasks

The system tasks developed in Section 6.2 must now be described in terms of the subsystem actions of Table 6.3. This requires the development of a sequence of actions for each of the system tasks, which may include intermediate states. The system tasks and their associated sequences are now shown in Table 6.4.

System Task	**Sequence**
1. Get block (+ store)	ready → LB_get_pallet → LB_put_block → ready to put → PM_put_pallet → ready
2. Get cyl1 (+ store)	ready → (LB_get_pallet // CS_get_cyl1) → CS_put_cyl → ready to put → PM_put_pallet → ready
3. Get cyl2 (+ store)	ready → (LB_get_pallet // CS_get_cyl2) → CS_put_cyl → ready to put → PM_put_pallet → ready
4. Get pallet (store)	ready → LB_get_pallet → ready to put → PM_put_pallet → ready
5. Get block (+ conv)	ready → PM_get_pallet → LB_put_block → ready to put → PM_put_pallet → ready
6. Get cyl1 (+ conv)	ready →(PM_get_pallet // CS_get_cyl1) → CS_put_cyl → ready to put → PM_put_pallet → ready
7. Get cyl2 (+ conv)	ready →(PM_get_pallet // CS_get_cyl2) → CS_put_cyl → ready to put → PM_put_pallet → ready

a // b indicates that tasks *a* and *b* occur concurrently
a → *b* represents a transition from task *a* to task *b* and indicates that task *a* occurs before task *b*

Table 6.4 Task sequences for the raw materials station

Each sequence in Table 6.4 starts and ends with a ready state. These indicate that the subsystem must be ready before an operation can be started and will return to the ready state on completion of that task. The ready state in each case will be represented by a primitive place in the control net. Note that the ready state in each sequence represents the same state (i.e. system ready). This means that the system can only carry out one of these tasks at a time, and must wait for completion of the currently active task before the next requested task can start.

This represents an initial specification of the top-level system tasks and provides the full sequences for each possible task of the system. In the next few stages these tasks will be broken down into sub-sequences that are each initiated by a control signal.

The actions PM_get_pallet and CS_get_cyl1 have been designed as concurrent actions in the design of the task sequences of Table 6.4. This is means that the process of getting a cylinder and placing it on a pallet in the loading bay has been split into two actions. This is an example of where the use of concurrent operations increases the complexity of the control net, since now both parts of the single process of placing a cylinder on a pallet must be represented as non-primitive places. An argument can be made for the actions PM_get_pallet and CS_get_cyl2. Part of the rationale for dividing such actions is to prevent the repetition of non-primitive places in the control net.

6.6 Creating the Petri net

6.6.1 Describing the paths

The initial stage of Petri net construction involves creating paths for each subsystem. As described in Chapter 5, each path is initiated by a control signal and ends with a feedback signal. Table 6.2 can now be altered to describe the paths of the raw materials station's control net. The resulting set of paths is shown in Table 6.5.

Station Task (Path)	**Sequence**
1. Initialise Station	Ready_to_init → Init_software → Init_hardware → RM_ready
2. Get block (+ store)	RM_ready → LB_get_pallet → LB_put_block → Ready_to_put
3. Get cyl1 (+ store)	RM_ready → (LB_get_pallet // CS_get_cyl1) → CS_put_cyl → Ready_to_put
4. Get cyl2 (+ store)	RM_ready → (LB_get_pallet // CS_get_cyl2) → CS_put_cyl → Ready_to_put
5. Get pallet (store)	RM_ready → LB_get_pallet → Ready_to_put
6. Get block (+ conv)	RM_ready → PM_get_pallet → LB_put_block → Ready_to_put
7. Get cyl1 (+ conv)	RM_ready → (PM_get_pallet // CS_get_cyl1) → CS_put_cyl → Ready_to_put
8. Get cyl2 (+ conv)	RM_ready → (PM_get_pallet // CS_get_cyl2) → CS_put_cyl → Ready_to_put
9. Put Pallet	Ready_to_put → PM_put_pallet → RM_ready

Table 6.5 Paths for the raw materials station

This set of paths differs from that given in Table 6.4 in that there is a completely new task included to initialise the station (Initialise Station) and that the path Put Pallet has now been separated from the other tasks. This is because it starts with a control place and ends with a feedback place, which elevates its status to that of a complete path. This will also reduce the complexity of the final net since the single path can be followed once all the previous tasks have been completed.

6.6.2 Mapping tasks to places

The tasks shown in Table 6.4 can now be mapped onto a set of non-primitive places for the control net. There are also three primitive places indicated by the tasks of Table 6.4. These are described in Table 6.6.

All the places and their descriptions are shown in Table 6.7, along with a descriptive caption that will be used on the Petri net graph.

Name	Meaning
Ready_to_init	Indicates that the raw materials station is ready to accept an initialisation order from its supervisory controller.
RM_Ready	Indicates that the raw materials station is ready to accept a task from the supervisory controller.
Ready_to_put	Indicates that the station is ready and waiting to place its loaded (or empty) pallet onto the conveyor.

Table 6.6 Full descriptions of the non-primitive places from Table 6.4

Place label	Caption	Description
p_1	Ready_to_init	Ready to initialise station
p_2	Init_SW	Initialise the station software
p_3	Init_HW	Initialise the station hardware
p_4	RM_Ready	Raw materials station ready
p_5	PM_get_pallet	Get pallet from conveyor
p_6	LB_get_pallet	Get pallet from storage
p_7	LB_put_block	Put block onto pallet
p_8	CS_get_cyl1	Get cylinder from slope 1
p_9	CS_get_cyl2	Get cylinder from slope 2
p_{10}	CS_put_cyl	Put cylinder onto pallet
p_{11}	Ready_to_Put	Station ready to place pallet on conveyor
p_{12}	PM_put_pallet	Put pallet on conveyor

Table 6.7 Place descriptions for the raw materials station control net

6.6.3 Describing the transitions for the Petri net

The transitions for the Petri net should all be identifiable from the sequences described in Table 6.5. Each unique transition is identified using the rules defined in Chapter 5. Each path described in Table 6.5 will start and end with a transition. In such cases the transition will be described in terms of the place controlling it. There will also be a transition for each arrow shown in the table indicating the start and completion of the subsystem actions. Often the transition will represent both the completion of one action and the start of the next action. Note that any concurrent operations require a single transition to initiate them and a single transition to synchronise their completion.

The transitions for the control net and their descriptions are shown in Table 6.8. Rules concerning the merging of transitions when separate paths are merged are described in Chapter 5. These rules are applied later on in the design process. The table also provides a brief description that is used in the Petri net. Using the sequence information of Table 6.4, the paths of the control net can be generated. These are shown in Figure 6.4 to Figure 6.8.

Transition label	Description
t_1	Ready to initialise station
t_2	Software initialisation complete
t_3	Hardware initialisation complete
t_4	Get block + pallet from storage
t_5	Get block + pallet from conveyor
t_6	Get empty pallet
t_7	Get cylinder 1 + pallet from storage
t_8	Get cylinder 1 + pallet from conveyor
t_9	Get cylinder 2 + pallet from storage
t_{10}	Get cylinder 2 + pallet from conveyor
t_{11}	Got pallet from store
t_{12}	Got pallet from store (block)
t_{13}	Got pallet from store + cylinder 1
t_{14}	Got pallet from store + cylinder 2
t_{15}	Got pallet from conveyor (block)
t_{16}	Got pallet from conveyor + cylinder 1
t_{17}	Got pallet from conveyor + cylinder 2
t_{18}	Block put
t_{19}	Cylinder put
t_{20}	Ready to put
t_{21}	Pallet put

Table 6.8 Transition labels for control net

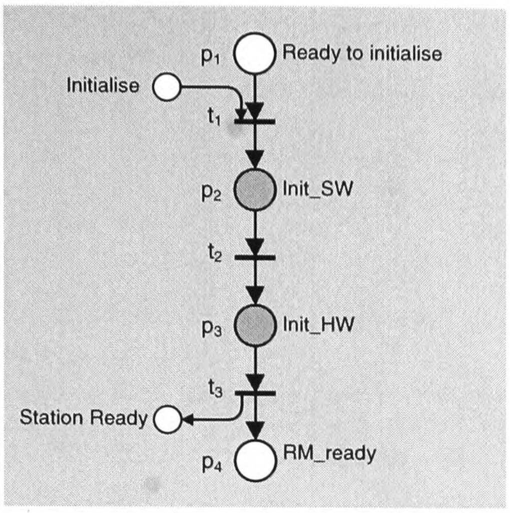

Figure 6.4 Initialisation path for the raw materials station

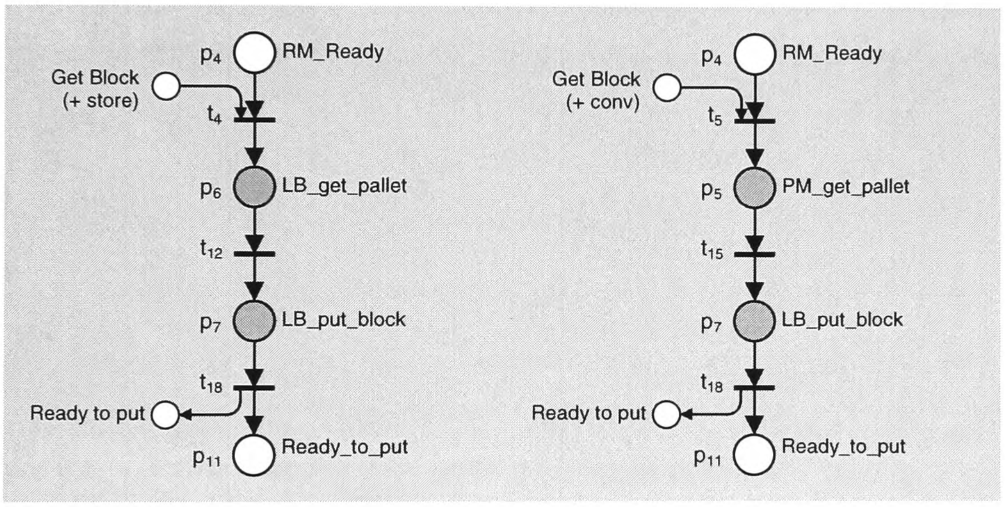

Figure 6.5 Paths describing the function of getting a block and pallet

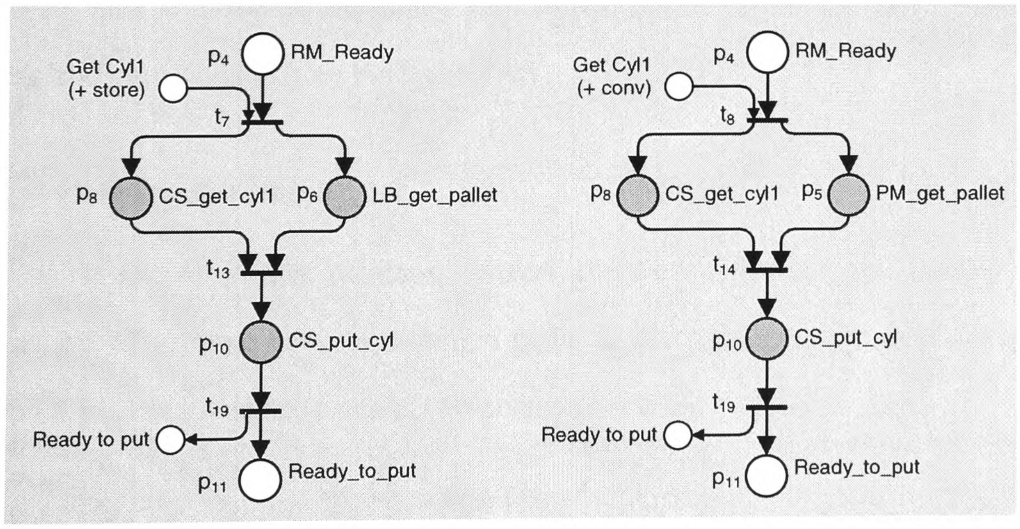

Figure 6.6 Paths describing placing a cylinder from slope 1 and a pallet

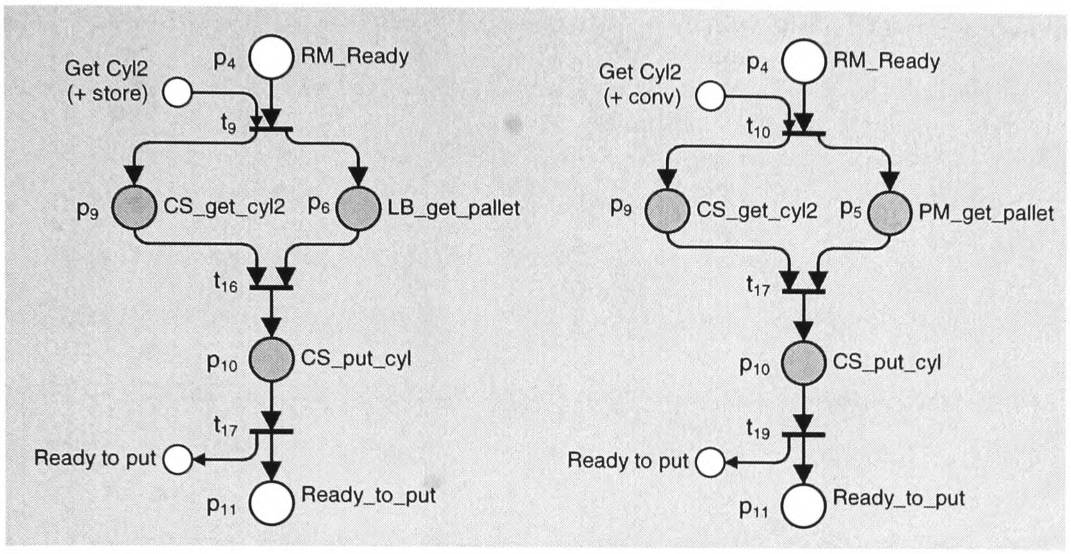

Figure 6.7 The paths describing placing a cylinder from slope 2 and a pallet

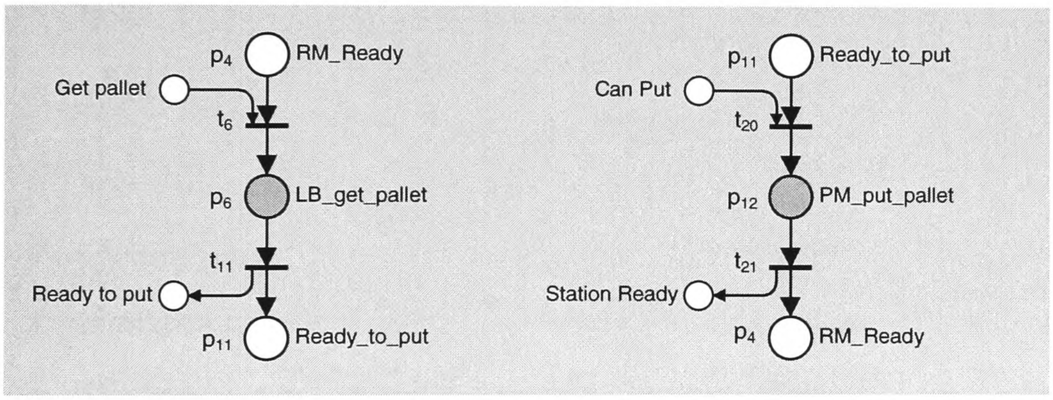

Figure 6.8 Paths that describe getting an empty pallet and placing a loaded (or empty) pallet onto the conveyor

The next task is to merge the paths from Figure 6.4 to Figure 6.8 along common places and transitions to form the control net.

6.7 Merging the Paths

The paths shown in the previous section contain many common places and transitions. The most obvious common place is that indicating the raw materials station is in its ready state (place p_4) which appears in the majority of paths.

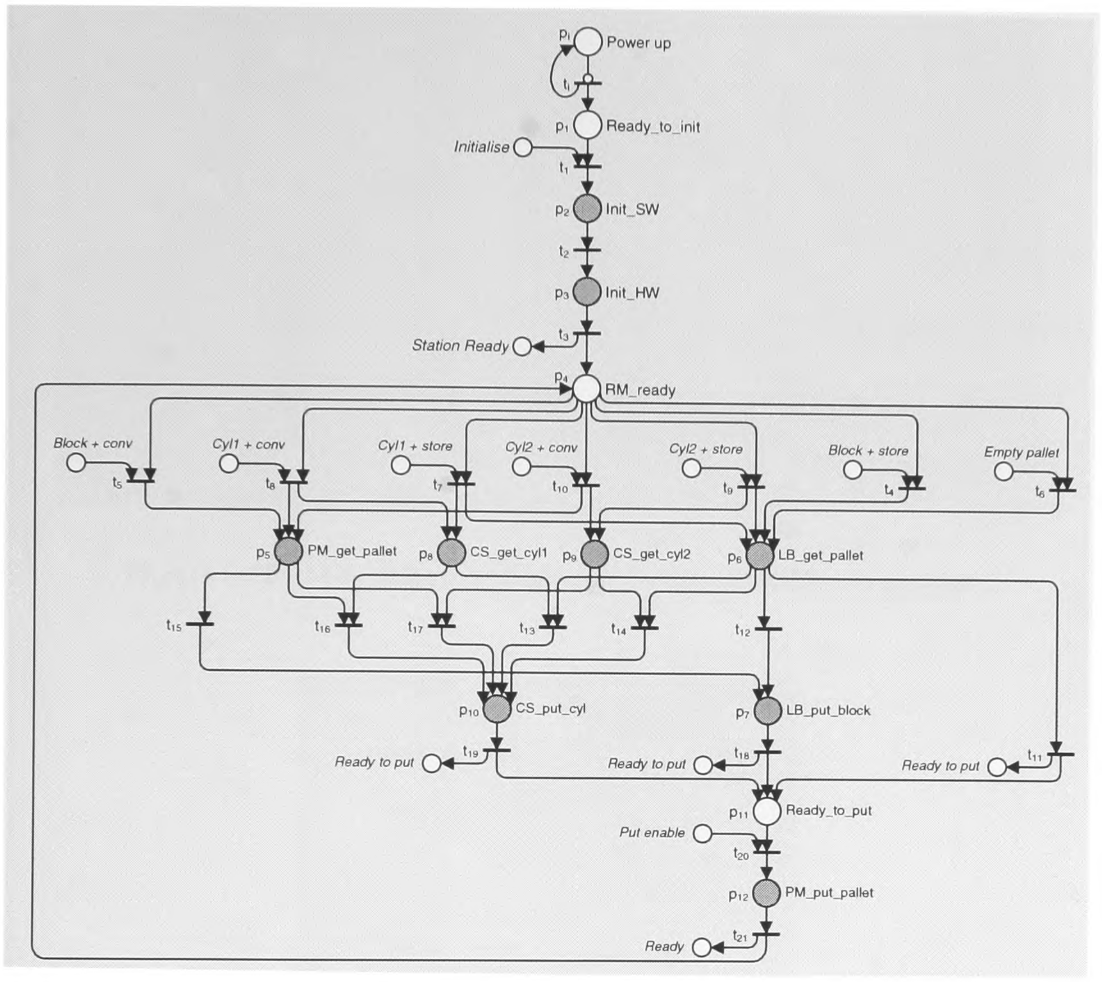

Figure 6.9 Control net for the raw materials station.

Rules for merging places and transitions were described in Chapter 5. By applying these rules, the control net is obtained and this is shown in Figure 6.9.

6.7.1 Removing decisions from the net

During the process of merging the subnets, the transitions need to be analysed to ensure that the control net is decision-free. This is done by applying Algorithm 5.1 from Chapter 5, it can be seen that transitions t_{12} and t_{15} have the same set of input places. This is also the case for transitions t_{17} and t_{20}. The paths produced by removing these decisions are shown in Figure 6.10 and Figure 6.11.

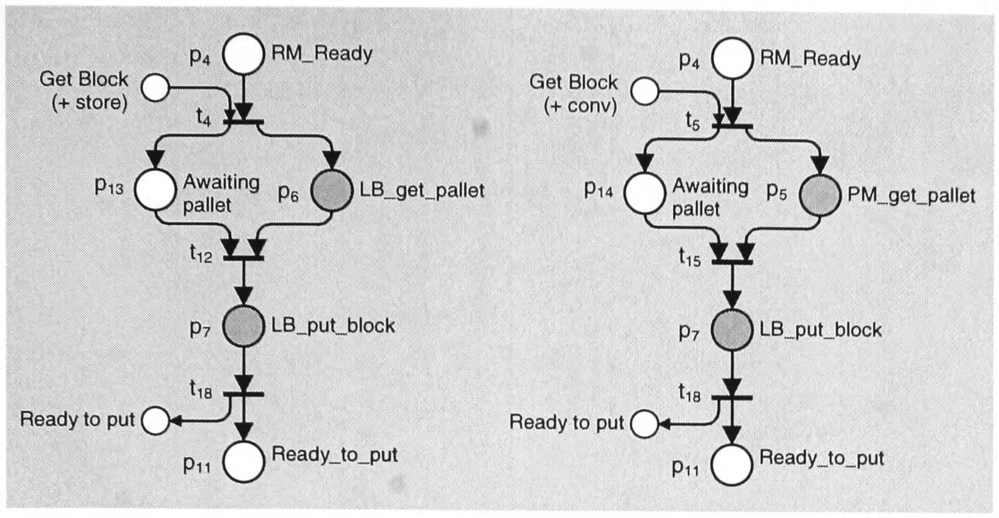

Figure 6.10 The paths of Figure 6.5 with added state places

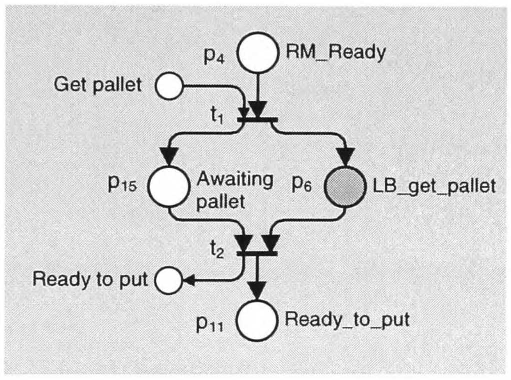

Figure 6.11 The path for getting an empty pallet with additional state place

The final control net, including the additional state places is shown in Figure 6.12.

6.8 Generating the Subnets

The interface between the control net and the subnets has been developed during the preceding stages of the software development. These subnets can now be developed in a similar manner to that of the control net development process. The majority of the subnets carry out simple sequences and generally there will be little concurrent activity carried out in the subnets as this will mostly be covered by the actions at the control net level.

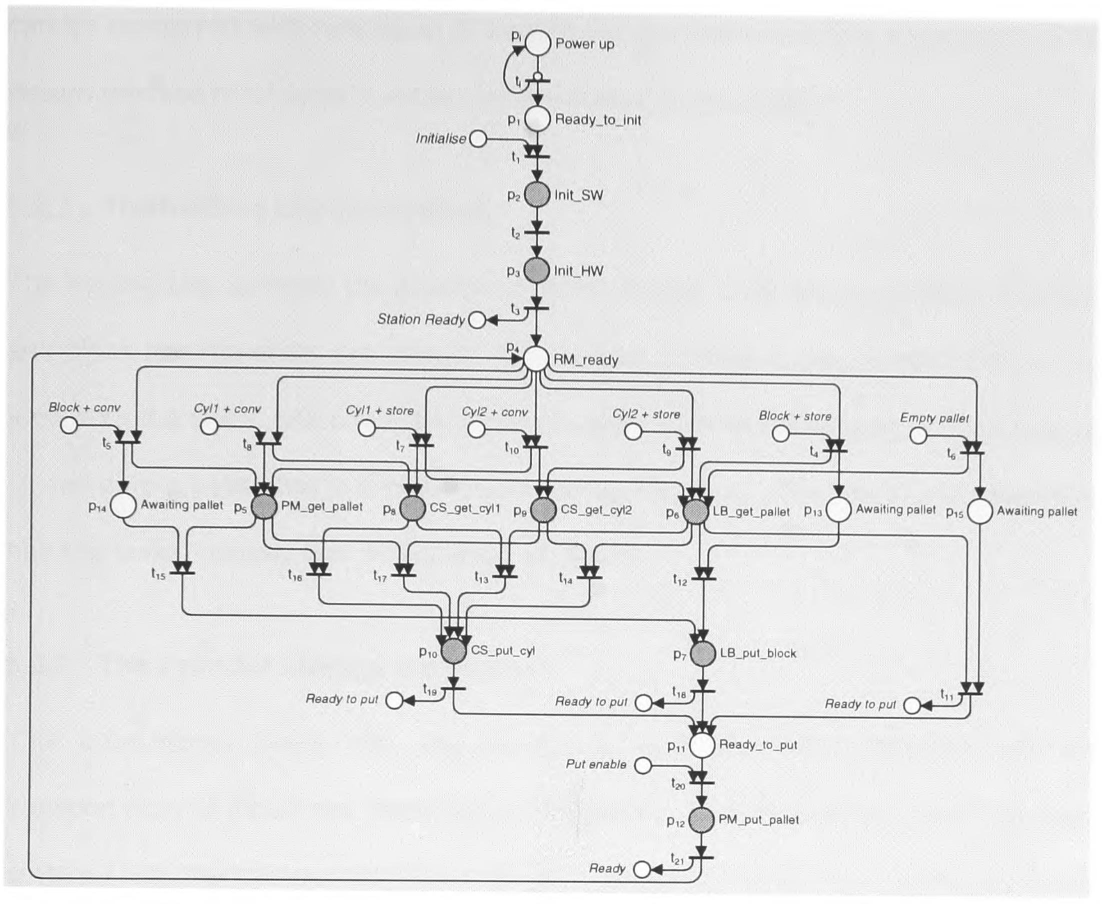

Figure 6.12 Control net including additional state places

6.8.1 The pallet manipulator subsystem

The pallet manipulator is used to transfer loaded (or unloaded) pallets from the loading bay and place them onto the conveyor. It may also remove empty pallets from the loading bay and replace them on the conveyor once they have been loaded. The actions of the pallet manipulator can therefore be described in terms of two sequences, namely **Get_Pallet** and **Put_Pallet**. These sequences will be initiated by signals from the control net. **Get_Pallet** is carried out when transitions t_5, t_8, and t_{10} fire, and **Put_Pallet** will be carried out when transition t_{20} fires.

Brief descriptions of the remaining subsystems are provided in this section. Subnets describing the operation of the pallet manipulator can be found in Appendix 2. This can be compared with nets used to control the system before the application of the design process of Chapter 5, which are presented in Appendix 1.

6.8.2 The loading bay subsystem

The loading bay contains the pallet and block storage units where supplies of pallets and block raw materials are stored. On request a pallet is placed onto the loading area from the store, which is then ready for loading. Also on request a block may be placed onto a pallet that is in position on the loading area. The loading bay therefore has two tasks, namely Get_Pallet and Get_Block.

6.8.3 The cylinder storage subsystem

This subsystems deals with the storage of cylindrical raw materials and the transportation of those raw materials to the pallet. This subsystem may have been split into two, the cylinder store and the Raw materials manipulator. However it was decided that this would give no added advantage to the operation of the system and would increase the complexity of the control software. In order to gain the fastest possible throughput of the raw materials station the further division of the system might be necessary – but this is not a significant factor in the requirements of the current development. The Cylinder storage subsystem has three possible operations, Get_Cylinder1, Get_Cylinder2 and Put_Cylinder.

6.9 Designing the Output Nets

The raw materials station has 15 pneumatic cylinders in all – each of which is actuated by a solenoid. The state of each solenoid is described by an output net which contains two possible states – On, or Off. Output net examples are also shown in Appendix 2. It should be noted that there is no direct feedback from the solenoids to their Petri net description as there are no sensors to perform such a task. The only feedback that the controller has from these devices is gained when their associated actuator meets one of its limit switches. Only then does the controller have any indication that the solenoid has responded as required by the control software. However if the actuator was to fail in reaching a limit switch, then there is no way of determining whether this is due to a solenoid fault, an actuator fault, or a limit switch fault. It is also possible that a controller error has caused the failure of the solenoid to actuate. Such faults may be detected using the process described in Chapter 8.

6.10 Software Design for a Manufacturing Cell

In the same way that a workstation such as the raw materials station can be broken down into subsystems and the software designed independently of the rest of the manufacturing system, so can a system be broken down into a number of stations. The control structure for a manufacturing cell may appear as that shown in Figure 6.13.

In Figure 6.13 each of the machines A, B and C could be described with a control structure such as that shown for the raw materials station in Figure 6.3.

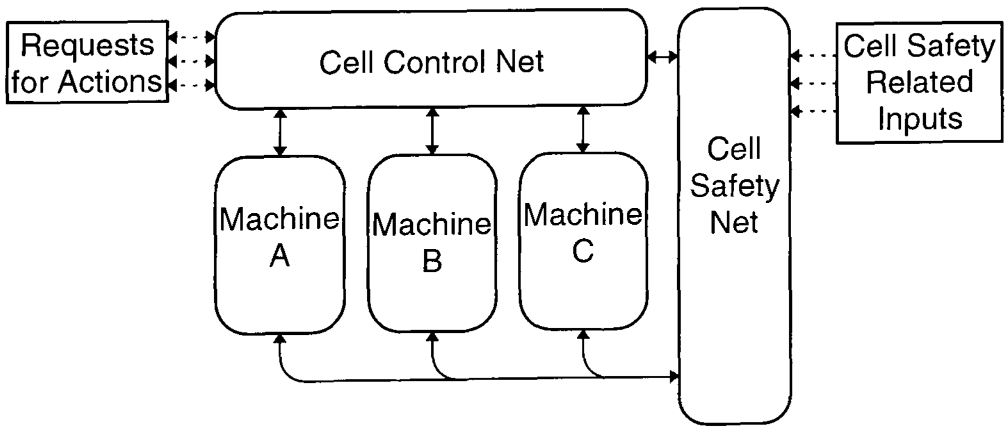

Figure 6.13 Control structure for a manufacturing cell.

One of the major advantages of the approach used in this work for controller design is that it doesn't rely on a purely Petri net approach to controller design. Other types of controller may be found in manufacturing systems whose control languages are not suitable for direct translation into a Petri net. An example would be a CNC machine tool. The sequence of events that occur in CNC operation can still be modelled as a Petri net but even this is not necessary for the control structure to be used.

The Petri net structure has been successfully applied to a manufacturing cell consisting of two industrial CNC machine tools and a robot, which is used to load the machines. The overall cell is supervised by a PLC. The layout of the cell is shown in Figure 6.14.

6.10.1 Development of the manufacturing cell control code

The code for the PLC, which sits at the top of the controller hierarchy, was developed using the Petri net structure described earlier in this work. The method used was less formal than that developed and presented in Chapter 5 and used for the raw materials station example in this chapter. The main point of interest for the manufacturing cell was the fact that it had a number of different controllers which all needed to be integrated before the cell could be automated. The controllers included were:

- Two CNC controllers (one for each machine tool)
- A robot controller
- A PLC, which was to co-ordinate the activities of the whole cell.

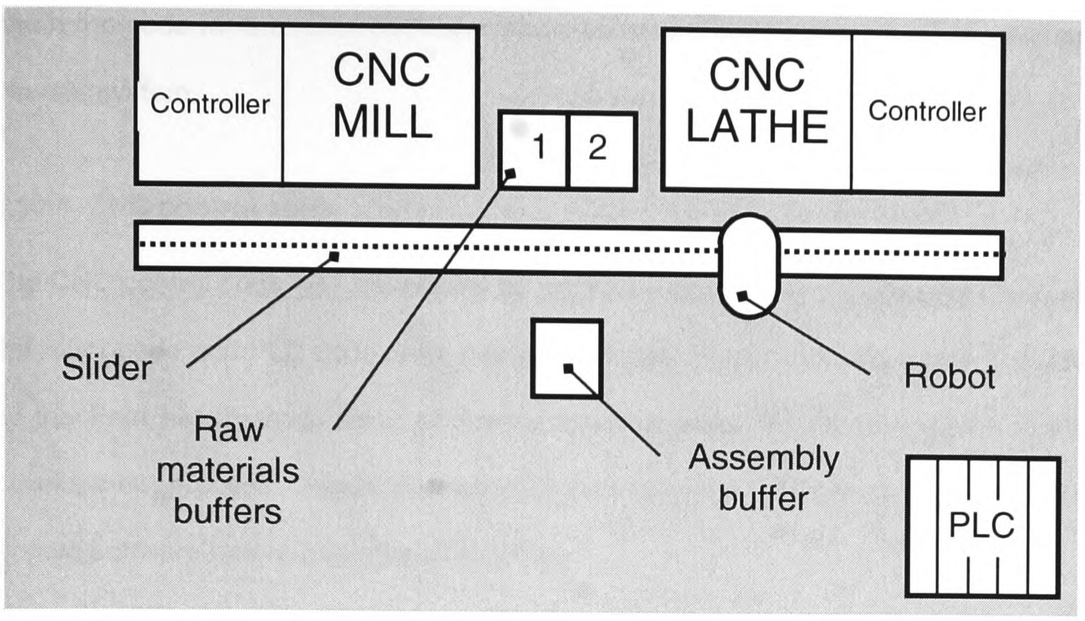

Figure 6.14 Layout of the Manufacturing Cell

6.10.2 Designing the cell control net

The cell control net was designed and implemented on the PLC using the Petri net structure described in Chapter 2 and the implementation method as described in Chapter 7. One of the advantages of the structured Petri net approach was that the code could be written and tested on a PLC that was completely unconnected to the Manufacturing Cell. In fact the code was written and tested at a completely different location.

6.10.3 Designing the robot control net

The part of the system with the most complex task was the robot. This was used to remove raw materials from the input buffers and assemble a finished product at the output buffer. The control language for the robot is very similar to BASIC, and is therefore unable to represent the Petri net structure directly. However, the Petri net could still be used for the design, with each subnet being implemented as a subroutine.

Again the code for the robot controller could be tested before being implemented on the real system.

6.10.4 CNC control code

The CNC control code was developed by another individual during the same time as the robot code and PLC code were being developed. Again this was made possible by the Petri net method, since all communications were described early on in the development process. Again this software was tested in isolation to the rest of the system software before putting it all together.

6.10.5 Implementation

The software was implemented in stages, with the PLC code and the Robot code being installed and tested together first. The signals from the CNC machines were simulated by forcing the inputs to the PLC. The only problems encountered at this stage were that the robot needed to be moved to a number of intermediate positions due to space limitations during certain operations, such as inserting raw materials into the CNC mill. It was a relatively simple exercise to implement these omissions, and it required no extensive rework of either the PLC code or the robot code.

6.10.6 Further developments and enhancements

The owners of the machine cell wish to develop the system further by adding additional stations. In particular they wish to add a visual inspection station, which examines the machined parts before assembly, and either accepts of reject them.

At present the system will only produce one product at a time, due to the physical layout of the system. It would be desirable for the system to be able to produce a number of parts in the fastest possible time.

There is also the desire to have the option of different programs and part types being developed.

6.11 *Further Developments for the Control Structure*

6.11.1 Shop floor controller

In the same way that separate machines have been grouped into a manufacturing cell, such cells along with other standalone machines can be grouped into complete flexible manufacturing systems. An example of such a system is shown in Figure 6.15 and more details of its possible development are given in (Stanton and Arnold, 1997).

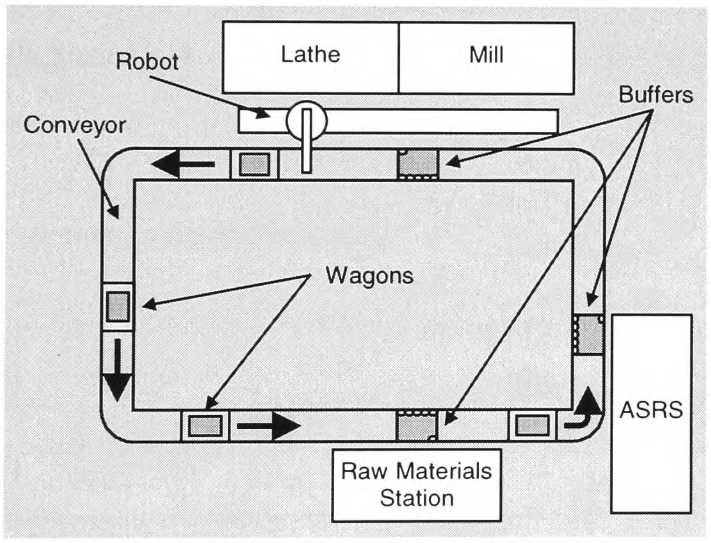

Figure 6.15 Example of a Conveyor System

Developing control code for such a complex system would be a challenge to the software development method proposed here. At present control of such a system would depend on clear specification of the desired sequences of operation. However, for the system to be truly flexible it would be desirable for previously unused sequences and processes to be used, perhaps for a single batch of products.

There is a similar need in the case of the manufacturing cell discussed in section 6.10.

There is also debate as to whether the control structure should have a hierarchical architecture, as described for the raw materials station, or a distributed architecture. Another advantage of the modular approach used in this work is that the modules can be used either in a hierarchical architecture or a distributed architecture. The key is in defining the communications between modules. Taking the raw materials station as an example, it does not matter where the requests for actions originate, they will all be treated in exactly the same way. Therefore a particular raw material can be requested by a supervisory controller, a manual request from an operator, or by another system workstation, such as a CNC machine tool. The internal working of the raw materials station will not need to be modified in any way to allow such a variety of requests.

6.11.2 Computer Integrated Manufacture

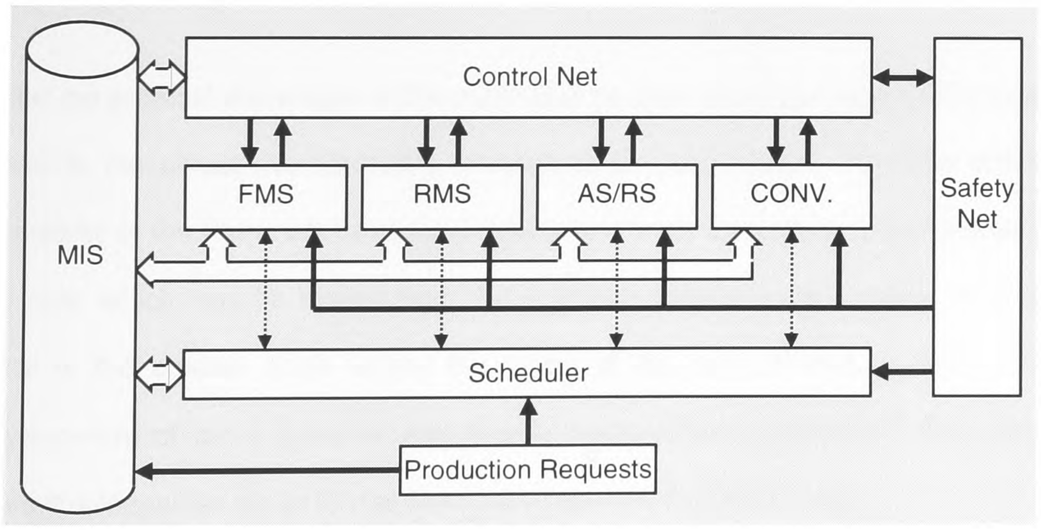

Figure 6.16 Proposed architecture for Computer Integrated Manufacture

The structure described in the preceding sections can be expanded further to include other functions often found in Computer Integrated Manufacturing (CIM). These include Manufacturing Information Systems, Scheduling software, and fault monitoring and diagnosis. A possible architecture for CIM is shown in Figure 6.16.

6.12 Chapter Summary

This chapter has presented a manufacturing example of the application of the design method described in Chapter 5. The raw materials station used to provide the example has been used during the development of the control structure and for the implementation methods described in Chapter 7. The Petri nets used to control the station are shown in the Appendices, and represent a number of different stages of development for the design method, culminating in the method presented in Chapter 5. Compared to other more formal approaches, the method presented here may lack some formal proofs of certain properties, such as liveness and boundedness – but with all the examples used so far, the preservation of such properties can be seen by examining the nets.

By far the greatest advantage of this method is its clear design process that is closely linked to the actual manufacturing process at an early stage. Coupled with the modularity of the Petri nets used this forms a firm basis for a more formal analysis of the nets, which may be hidden from the engineer designing the system. The later parts of the chapter show where the future of the method lies, such as in the development of more complex and flexible manufacturing controllers that provide more challenges as far as formal Petri net properties are concerned.

6.13 References

Stanton, M. J. and Arnold, W. F., 1997, “Extension of structured Petri nets for the control of a conveyor system.” In *Proc. Factory 2000*, Cambridge, England.

Chapter 7

Implementation

If Petri nets are to be used as controllers for the systems they model then an accurate method of implementation is vital. Also, in order to perform any analysis on even relatively small Petri nets they must be implemented on a computer. It is the ability to implement a Petri net on a number of different types of computer in a variety of different ways that enables the integration of separate systems into a single integrated manufacturing system. If Petri nets can be used for such integration then they may form the basis of the common distributed control language proposed in (Naylor and Volz, 1987).

A method of implementing Petri nets on a PLC has been discussed in Chapter 2. When this method was applied to the control of a relatively complex manufacturing workstation, problems arose which warranted a re-evaluation of the implementation method.

This chapter examines the earlier implementation method and looks at some alternative approaches, and their behaviour when running on a PLC. It also looks at how Petri net implementation has been dealt with on a robot controller, which uses a high level control language similar to BASIC, and then finally describes an implementation of a Petri net on a relational database.

7.1 Interpretation and Implementation

A Petri net is a theoretical structure exhibiting a high degree of parallelism in the form of transition firings and token movements. Implementation of such a structure requires translation into the programming language of the controlling device, which will usually be a sequential machine and will therefore not explicitly represent the parallelism in the net structure. Translation of the mathematical net structure is therefore a form of interpretation. This interpretation may be performed at different levels. For example, a low-level interpretation might attempt to implement the places and transitions of a net as a data structure, whereas a higher level interpretation might implement transitions or places as subroutines, with no explicit representation of the Petri net structure.

Interpretation is also related to the level of abstraction at which a system is described. Higher levels of abstraction deal with interpretations such as 'get pallet' or 'Move robot to lathe', whilst lower levels of abstraction deal with interpretations such as 'Switch on' or 'activate solenoid'. Interpretation in this sense was discussed in Chapter 4.

7.2 Methods of Implementation

Petri nets are generally implemented in one of two ways, both of which have been employed during the course of this work. One method is that of the 'token player', and the other is the implementation of the net in the form of a structured program such that the flow of events in the net is emulated by the logical structure of the program. Many authors concentrate on the implementation of Petri nets using high level languages. In (Colom et al, 1986) centralised, decentralised and hybrid schemes are presented for the implementation of Petri nets in ADA, or other similar concurrent programming languages, whereas in (Taubner, 1988) the chosen language is Occam.

In (Venkatesh et al, 1994) a comparison in made between Petri nets and ladder logic diagrams for sequence control in manufacturing systems. Such a comparison was based on the complexity of the ladder logic program compared with that of the Petri net. The main complexity measure was taken as the number of nodes in the graphical representation of the programs (places and transitions in the case of the Petri net, and coils and relays in the case of the ladder logic program). However, the ladder logic program is directly implemented whereas the Petri net must first be converted into some form of executable code. In fairness the comparison should have included some aspect of the implementation as well as taking the arcs into account for measuring the graphical complexity of a Petri net. The implementation of Petri nets in ladder logic is discussed in more detail in section 7.3.

A Petri net controller which operates mainly with a software interface (not directly interacting with the hardware signals) was proposed in (Crockett et al, 1987) and the work was extended to the use of coloured Petri nets in (Kasturia et al, 1988). This is another example of a system that relies on a complex general-purpose computer to act as a controller. The implementations described in this work concentrate on those controllers commonly found in automated manufacturing systems (e.g. PLC's, robot controllers, CNC). The languages in which the programs for these controllers are written do not contain the expressive power of high level languages, and it is therefore impractical to use such complex formalisms as coloured Petri nets for sequence control. Implementation methods can be divided into two classes, either employing the 'token player' or the structured program approach.

7.2.1 The 'Token Player'

The 'token player' is a program that uses data structures to represent the Petri net and its dynamic behaviour. An algorithm is used which carries out the firing of transitions according to the distribution of tokens amongst places (Taubner, 1988). In (Silva and Velilla, 1982) a number of implementation techniques are compared all of which are token players. Some of these implementations use matrices as the fundamental data structure, others use lists. For control purposes these are usually augmented with some mechanism for communicating the state of the net with the outside world.

7.2.2 Structured program approach

The structured program approach uses the Petri net model to describe the structure of the overall program, but the actual behaviour of the net is not implemented. Instead each action represented in the net is described by a procedure or function, which must be completed before those following in the net can be started. This method uses the Petri net as a control flow-charting tool rather than implementing a data structure. Therefore the structured program approach is likely to result in more compact control code.

The token player allows both the structure and the dynamic behaviour of the Petri net to be analysed. Once the behaviour of the program has been verified, it may then be implemented as a set of subroutines. Thus the two approaches can be used to complement each other.

7.3 Implementation on a PLC

When implementing a net on a PLC the token player method has been favoured. The token player works by scanning the transitions in the net to determine which are enabled. On a general-purpose controller, such as a personal computer, this scan is implemented as a software loop. A programmable logic controller operates by scanning its inputs, and solving the logic program in a cyclic manner. It is therefore unnecessary to implement this scan as a software loop because it occurs during the normal operation of the PLC.

The main choice to be made when implementing a Petri net on a PLC (in particular as a Ladder Logic Program) is how to represent the Petri net elements (places, transitions and arcs). In the implementation method of Chapter 2, places were implemented as output coils and transitions were implemented as the logical combination of their input places. This method was favoured over others in the literature (e.g. (Satoh et al, 1992)) as it produced a relatively compact program and facilitated simple manual fault diagnosis. One of the distinguishing features of this implementation method was the absence of any properties or logic associated with transitions other than that represented by their input places (this is not the case with Grafcet or Sequence Function Charts (David and Alla, 1992)).

The types of controllers present in modern manufacturing systems are relatively simple sequential devices (such as programmable logic controllers, CNC controllers, robot controllers) which are in many cases designed specifically for the machine to which they are attached. General-purpose controllers include PLC's and Microcomputers and are applicable to a variety of applications. Due to their popularity the work here has concentrated its efforts on implementing controllers on PLC's. There are a number of requirements for an implementation when using such controllers.

1. The control code must be as small as possible.

   There is usually a limit to the scan time of a PLC and thus there is a maximum length of the control program. Small memory sizes of many PLC's also limit the size of program that can be implemented.

2. The implementation must be an accurate representation of the Petri net

   Any inaccuracies in the conversion from Petri net to controller code will invalidate the use of Petri nets as a specification tool. If the behaviour of the net is to be taken as the desired behaviour of the controller then the behaviour of the theoretical net must be matched by the implementation.

If the controller cannot be programmed to behave in the same manner as the net then an approximation must be used. This raises the issue of how correct an implementation needs to be.

If either of these two factors is unachievable for a particular controller then the Petri net still finds some application as a design tool. In such a case the net can be used to provide the general structure of a program and to identify its communication channels with the other parts of the system. This was the case when using the net to describe the control program for a robot controller (see Chapter 6) and also to describe the communication between the PLC program and the CNC machine tools.

7.4 Representation of a Petri Net by Ladder Logic

The conversion of Petri nets to ladder logic has been dealt with previously in (Henry and Webb, 1988), (Cutts and Rattigan, 1992), and (Satoh et al, 1992). A comparison between the Petri nets and ladder logic programs was presented in (Venkatesh et al, 1994). More recently a method for representing timed and coloured Petri nets has been proposed for implementation on a Siemens PLC (see (Uzam and Jones, 1996) and (Jones and Uzam 1996)). This method relies heavily on function blocks that are specific to Siemens PLC's and therefore the method is not sufficiently general. The approach presented in Chapter 2 is similar to that used by (Henry, R. M. and Webb, M., (1988)) and was originally adopted as it preserves both the structure and the diagnostic capability of the Petri net.

7.4.1 Implementation using Ladder Logic

In (Stanton et al, 1996) a method is proposed for implementing Petri nets as a ladder logic program which gives no explicit representation for transitions. Instead all transitions are implicitly represented by their sets of input places. This representation is favoured because:

- The size of the program is dramatically reduced.
- The program becomes less complex for the purposes of fault diagnosis.

The implementation is expressed in Boolean algebra using Rule 7.1 and Rule 7.2.

Rule 7.1

A place, p, is marked if any of its input transitions, t_{in}, are enabled or it is marked and none of its output transitions, t_{out}, are enabled.

$$p = T_{in} \wedge \left(p \vee \overline{T_{out}}\right) \tag{1}$$

where:

$$T_{in} = \left(t_{in_1} \vee t_{in_2} \vee \ldots t_{in_m}\right)$$

$$T_{out} = \left(t_{out_1} \vee t_{out_2} \vee \ldots t_{out_n}\right)$$

Rule 7.2

A transition, t, is enabled if all of its input places, p_{in}, are marked

$$t = P_{in} \tag{2}$$

where:

$$P_{in} = \left(p_{in_1} \wedge p_{in_2} \wedge \ldots \wedge p_{in_i}\right)$$

Since all transitions are expressed in terms of their input places (equation (2)), there is no need for their explicit representation in the ladder logic program. The above rules can therefore be expressed in terms of places:

$$p = P_{T_{in}} \wedge \left(p \vee \overline{P_{T_{out}}}\right) \qquad (3)$$

7.5 The Need for a New Approach

Analysis of these equations reveals a major assumption of this approach. The assumption is that all of the output places of any one transition will never become marked except by the firing of that transition. This method is therefore only valid if it can be shown that such a condition will always hold for the system modelled. If this can be shown, then there is a structural property of the nets that may be used to define the class of nets.

An example is shown in Figure 7.1 and Figure 7.2 where equation (3) is used as the firing rule for the implemented Petri net.

In Figure 7.1 only transition t_2 is enabled. When transition t_2 fires it removes the token from place p_3 (its input place). However, it also removes the token from place p_1 because the token appearing in place p_4 is the only output place of transition t_1.

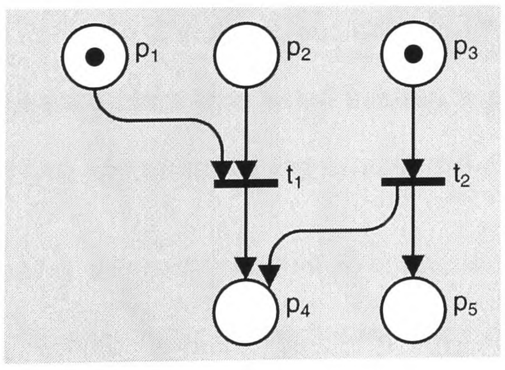

Figure 7.1 Example net before the firing of transition t_2

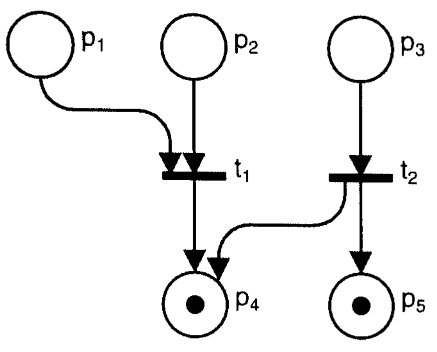

Figure 7.2 Example net after the firing of transition t_2

In the definition of a decision free Petri net as used here, there is only one possible path through a net from any single initial transition. At any point during the course of a path's execution, the only enabled transitions will be those that belong to that path. However, each path is not necessarily disjoint. Therefore each complete set of output places from each transition in the path may not be marked by only their preceding transitions within the path.

7.5.1 Defining a 'correct' implementation

The implementation can be viewed as being correct if the sequence of Petri net markings is matched by the sequence of implementation markings. For a more 'correct' representation of the net in Ladder Logic, it is necessary to represent both Places and Transitions in the program. If a flag is used to represent a transition, then when that flag is set the transition is enabled, and when the flag is reset it is disabled. If a flag is used to represent a place then when the flag is set, the place is marked (we assume a safe Petri net) and when the flag is reset, the place is unmarked.

This notion of correctness is different from that of functional correctness, where the behaviour of the implementation may not match exactly the behaviour of the net, but the external behaviour of the system matches that required by the system specification. The notion of a correct implementation has been discussed in

(Agerwala and Flynn, 1973) where the behaviour of the 'interpreted' Petri net implementation is used, rather than the 'standard' Petri net definition, to verify the correct behaviour of system.

There are two sides to the issue of correctness. One deals with the correctness of an implementation, which must match as closely as possible the mathematical behaviour of the Petri net. The other deals with adequate implementations and how they cause the system to behave according to its specification. The Petri net implementations used here are not correct in the first sense but need to be proven to be correct in the second sense. Therefore the criteria used for testing each implementation is whether or not the machine being controlled operates correctly to its specification. If the implementation doesn't work in exactly the same way as the Petri net would, but the end effect of its operation is the same as that which the Petri net would produce, then functionally there is no error. The machine is performing to its required specification, whether or not the Implementation of the Petri net is 'correct'. Surely, then, the two implementations (that of the theoretical Petri net and that of the Ladder Logic program) are equivalent under the circumstances in which the machine is to operate.

7.6 Testing the Implementation Methods

An example Petri net has been used to test the differences between the various ladder logic implementations of Petri nets that are described in this chapter. The tests were used to observe the behaviour of the PLC when running ladder logic to check whether the order in which the logic is solved has a significant effect on the behaviour of the program, and hence the Petri net implementation.

Figure 7.3 shows a control net with a single action place (non-primitive place p_3). This is linked to the action's corresponding subnet, which comprises p_5, p_6, p_8, and

p_9. The control net and the subnet share places p_4 and p_7. When transition t_1 fires, the subnet is initialised and the control net is placed in its 'ready' state (represented by place p_2). Place p_{10} represents an exogenous input from either another system, or a control panel switch indicating that a request has been made to start the action of place p_3. Place p_8 represents a call to either a lower level subnet, or an output net. Place p_9 is a software link used to indicate that the action represented by the lower level subnet has finished. It may also model the behaviour of a sensor, which would be present if the subnet was connected to a hardware system, via an output net.

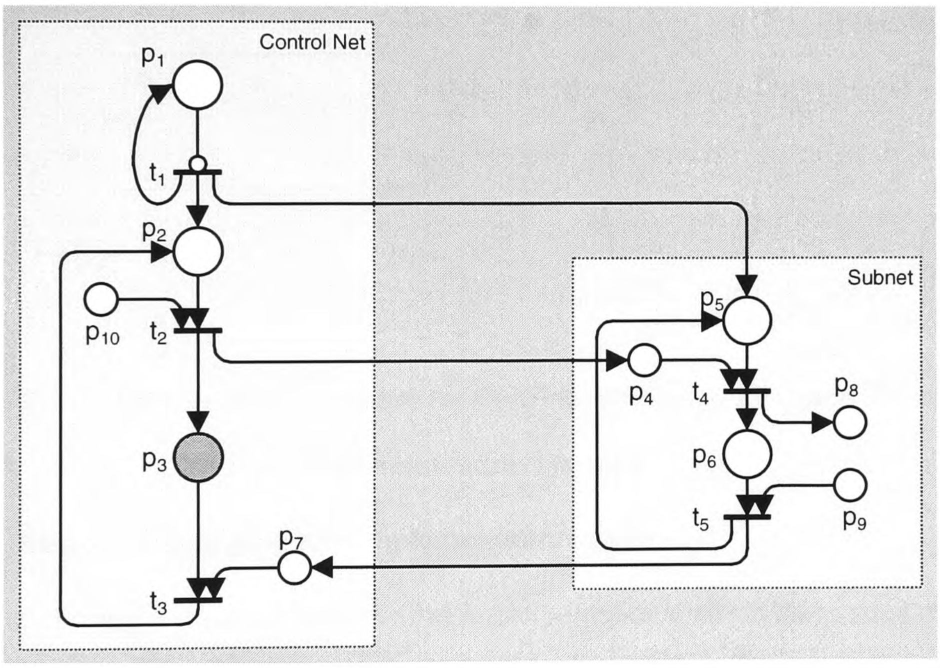

Figure 7.3 Sample net used to test implementation methods

Figure 7.4 shows the ladder logic program used to represent the Petri net of Figure 7.3. This program was written using the original implementation method as described in Rules 7.1 and 7.2 and in section 7.4.1. In addition to allowing an examination of the behaviour of various implementations, the experiment also demonstrates how the Petri net can be tested before it is attached to the hardware.

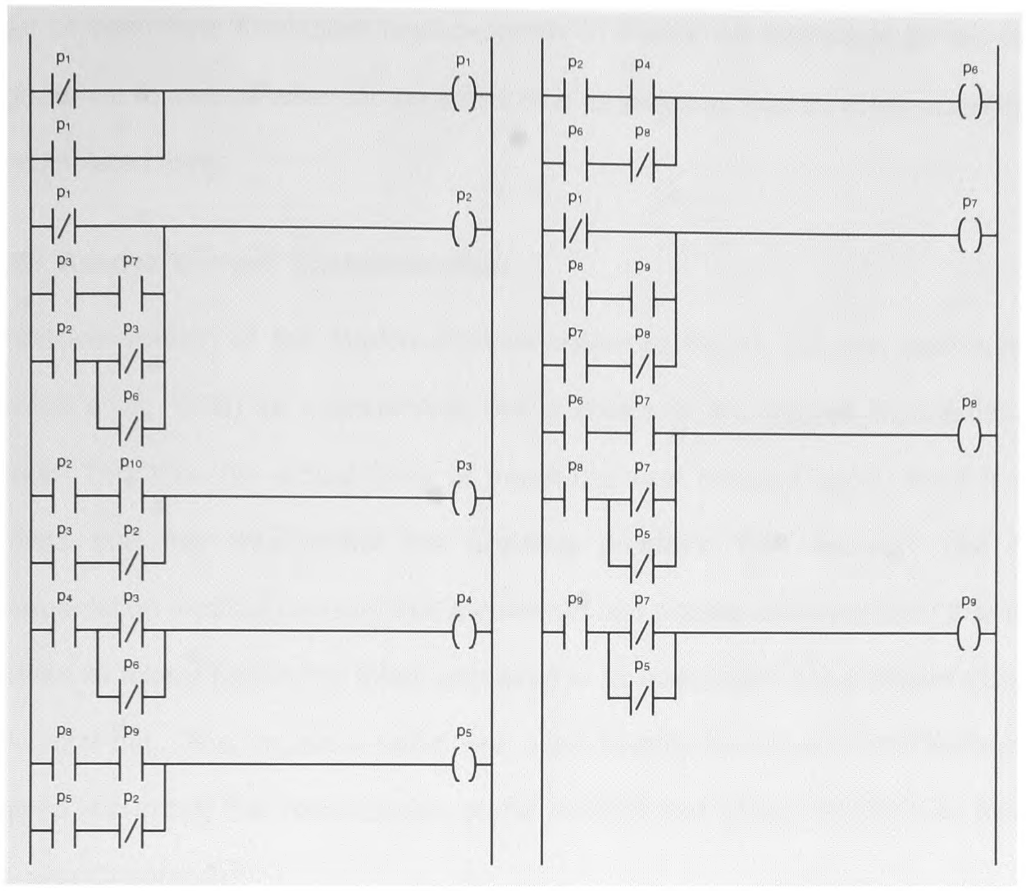

Figure 7.4 Ladder logic program for Petri net of Figure 7.3 using the original implementation method

7.6.1 Results of applying this implementation method

When controlling a large system with this implementation method there seemed to be some spurious generation of tokens during the operation of the system (the particular system being controlled was the raw materials station described in Chapter 6). It was initially believed that these additional tokens arose when there was no feedback from the controlled sub-net (such as when the software initialisation stage was run). Also there seemed to be a problem with the subnet/control net links where transitions took more than one PLC cycle to complete firing.

It can be seen from the ladder logic program, in Figure 7.4 that place p_8 has been programmed to lose its token at the same time as place p_9, that is, when transition t_5 has completed firing.

7.6.2 A more 'correct' implementation

A modified version of the implementation shown in Figure 7.4 was proposed in (Stanton et al, 1996) as a solution to the problems of the original implementation method. This time the actual firing of transitions was focused upon, since it was believed that this was where the previous problems had arisen. The new implementation method ensured that the subnet had started operation (and therefore received its token) before the token appeared in its associated non-primitive place in the control net. This implementation was considered to be more correct because it followed accurately the interpretation of the non-primitive places which is as follows (see also Chapter 2):

If a non-primitive place is marked then it indicates that its associated subnet is in operation.

Therefore, if the distribution of tokens after a transition firing was to be spread over a number of PLC cycles, then it should be ensured that this definition holds by causing the subnet to receive its token before its associated non-primitive place. The ladder logic program for the second method of implementation is shown in Figure 7.5.

The method of implementation shown in Figure 7.5 was used to develop a working controller for the raw materials station described in Chapter 6. It was also used for the development of the PLC code or the manufacturing cell, also described in Chapter 6. These systems were tested and found to operate without error, thus proving that the method for control code design was satisfactory. However, a case was discovered for which neither this method, nor the previous one, would work.

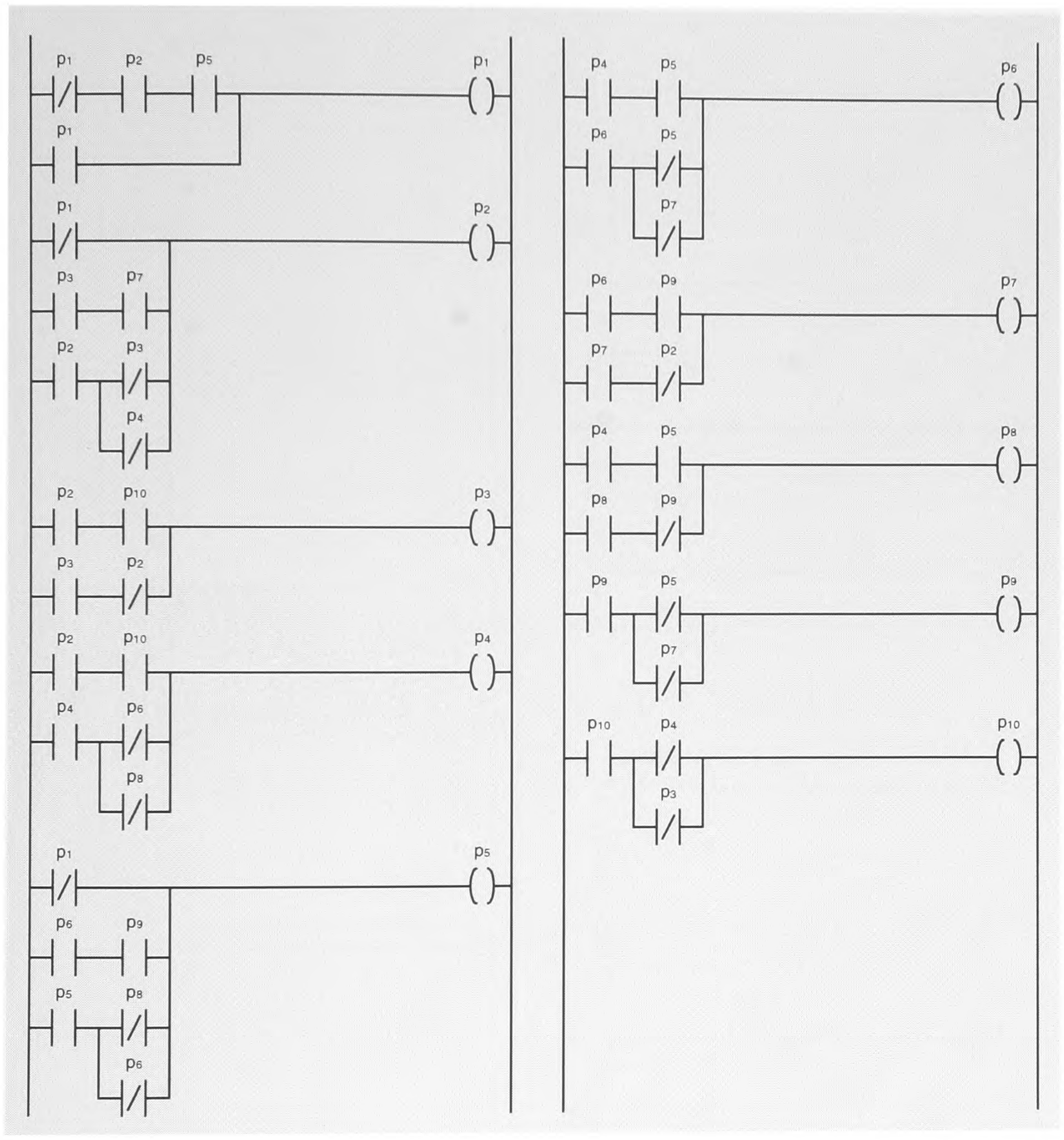

Figure 7.5 Second ladder logic implementation of the Petri net of Figure 7.3

If place p_9 were removed from the Petri net of Figure 7.3, the subnet would model the situation that arises in an output net (i.e. there is no feedback place to prevent transition t_5 from firing immediately). This problem prompted a re-evaluation of the logic for the Petri net implementation, and resulted in the findings shown earlier in section 7.5. In turn these findings resulted in a third implementation method which is shown in Figure 7.6.

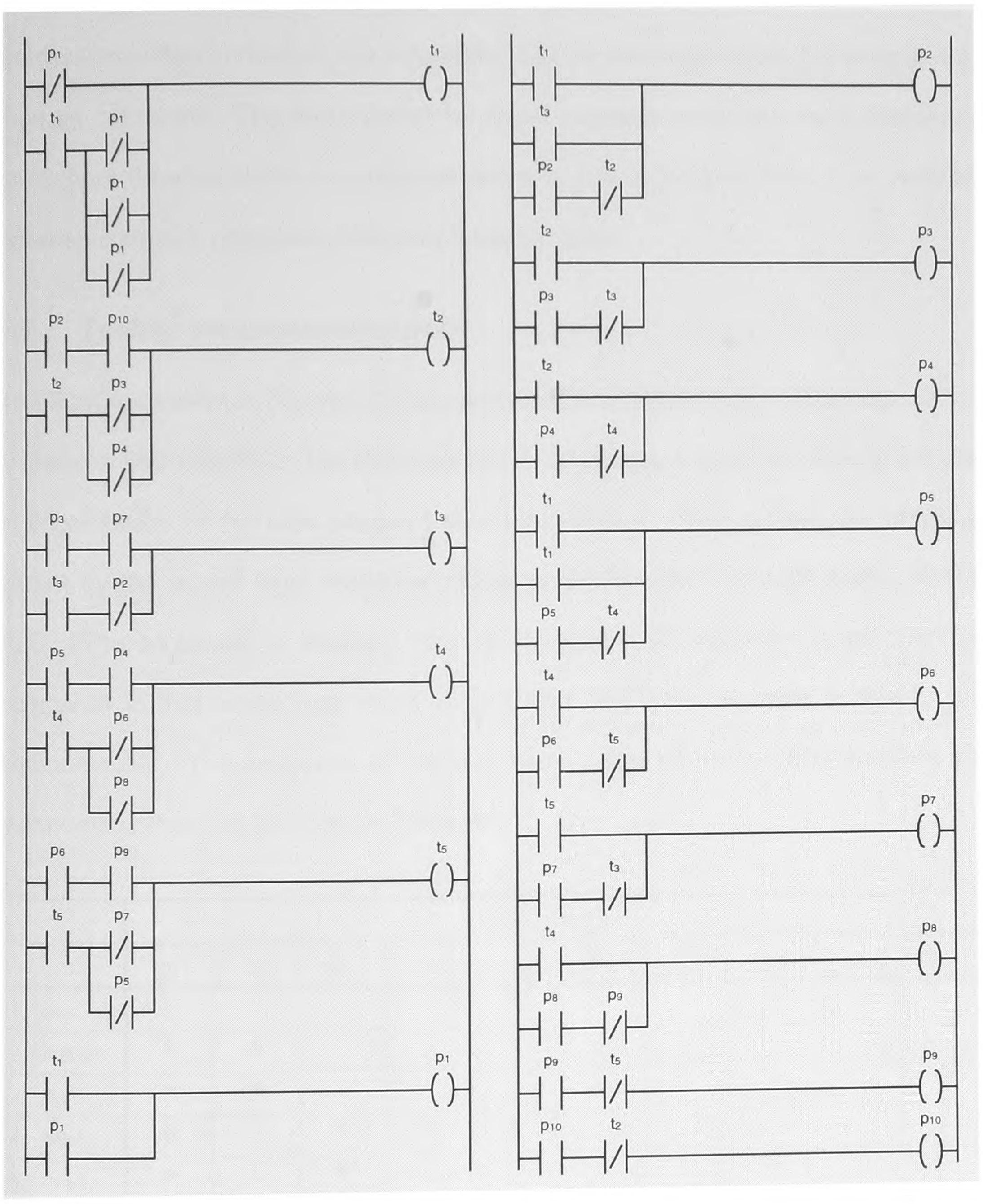

Figure 7.6 Third ladder logic implementation of the Petri net of Figure 7.3

This implementation method includes transitions as well as places, using an output coil to represent both. This is a similar approach to that adopted in (Satoh et al, 1992), and suffers from the same problems as their implementation method (as described in (Stanton et al, 1996)), namely the size of the control code generated. This is a problem highlighted in (Ferrarini, 1994) where it is proposed that once the Petri net has been converted into a suitable form for implementation, the code should then be optimised. The requirement for more compact code has been expressed throughout the discussion on implementation in this work and there is a trade-off between compact, optimised code and readable code.

7.7 Testing the Implementations

The Petri net shown in Figure 7.3 was converted into ladder logic, which was run on a Modicon 985-145 PLC. The PLC was run in single step mode, and therefore made a single sweep of the logic program and then paused. This allowed the 'marking' shown by the ladder logic implementations to be recorded for each sweep of the PLC. The sequence of marking vectors for each implementation could then be compared to that which one would expect from the Petri net were it able to run autonomously. The sequence of marking vectors that would be obtained from the autonomous Petri net is shown in Table 7.1.

	Place Number									
	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
1. m_0	-	-	-	-	-	-	-	-	-	-
2. fire t_1	◆	◆	-	-	◆	-	-	-	-	-
3. p_{10}*	◆	◆	-	-	◆	-	-	-	-	◆
4. fire t_2	◆	-	◆	◆	◆	-	-	-	-	-
5. fire t_4	◆	-	◆	-	-	◆	-	◆	-	-
6. p_9*	◆	-	◆	-	-	◆	-	-	◆	-
7. fire t_5	◆	-	◆	-	◆	-	◆	-	-	-
8. fire t_3	◆	◆	-	-	◆	-	-	-	-	-

Table 7.1 Distribution of tokens in a single cycle of the Petri net

The '*' by places p_9 and p_{10} indicate that they are marked by exogenous inputs. It is assumed that the token in place p_8 is consumed by the time place p_9 receives a token. This reflects the behaviour of a net running a real system where p_8 would start a task and its token would be removed before the task is complete, i.e. before p_9 becomes marked. The marking m_0 is the initial marking, and is always assumed to be zero (or the vector whose elements are all zero).

Time is not represented explicitly in the net but it is clear that the net will wait an arbitrary length of time for tokens to arrive in places p_9 and p_{10}. It should also be noted that when t_3 fires (row 8) the marking returns to the start of a repeatable sequence. The marking of rows 2 and 8 can therefore be viewed as the 'home state' of the system (Murata, 1989).

7.7.1 Experiment 1

The Petri net of Figure 7.3 was converted to ladder logic using the updated method of (Stanton et al, 1996). The resulting ladder logic program is that shown in Figure 7.5. Again only the places are explicitly represented and the order in which they appear in the ladder logic program is related to their order in the Petri net. This ordering of places within the Ladder (and also the net) is representative of the flow of events in the system over time.

The marking vectors, recorded at each sweep of the PLC, are shown in Table 7.2 below. The state of the system after sweep no. 2 is the ready state of the system and it is assumed that once the program returns to this state, no further new states will be entered. The test therefore represents a single cycle of the control net of Figure 7.3.

	Place Number									
Sweep	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
1.		■			■					
2.	■	■			■					
3.	■	■			■					
4.	■	■			■					■
5.	■	■	■	■	■	■				
6.	■		■			■		■		
7.	■		■			■		■		
8.	■		■			■		■	■	
9.	■		■		■	■	■			
10.	■	■			■					

Table 7.2 Distribution of tokens for the first implementation

	Place Number									
Sweep	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
1.		▲			▲					
2.	▲	▲			▲					
3.	▲	▲			▲					
4.	▲	▲			▲					▲
5.	▲	▲	▲	▲	▲	▲		▲		
6.	▲		▲			▲		▲		
7.	▲		▲			▲		▲		
8.	▲		▲			▲		▲	▲	
9.	▲		▲		▲		▲			
10.	▲	▲			▲					

Table 7.3 Distribution of tokens for the second implementation

	Place Number									
Sweep	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
1.	●	●			●					
2.	●	●			●					
3.	●	●			●					●
4.	●		●	●	●					
5.	●		●			●		●		
6.	●		●			●		●		
7.	●		●			●		●	●	
8.	●		●		●		●			
9.	●	●			●					
10.	●	●			●					

Table 7.4 Distribution of tokens for the third implementation

The state of the PLC after the second sweep is that of the home state identified in the autonomous net. It can be seen that there is now an intermediate state before the home state is reached.

7.7.2 Experiment 2

Experiment 2 was used to examine the relationship between the marking produced by the ladder logic implementation of the Petri net and the position in which the places are written in the logic program.

The program used for the experiment was the same as that in Figure 7.5, except that place p_7 was shifted to an earlier position in the logic program. This shift represented the inclusion of place p_7 as part of the control net rather than the sub-net as in the previous example. The resulting markings were observed and are presented in Table 7.3.

By comparing Table 7.3 with Table 7.2, it can be seen that by moving a single place to a different position in the ladder logic program, a different set of markings will be obtained. This has implications for deciding which of the control/feedback places belong to which nets in the control structure.

7.7.3 Experiment 3

In this experiment a different implementation method was adopted. As a result of the analysis of the logical behaviour of the Petri net, it was decided that to truly represent the dynamic behaviour of the net, both places and transitions would have to be represented. The resulting program for the Petri net of Figure 7.3 is shown in Figure 7.6. Here place p_7 is returned to its original position and the logic for places and transitions are grouped. This was done to reduce difficulties in finding the rung for individual places.

The resulting markings for the net were observed as shown in Table 7.4. The experiment was repeated with place p_7 moved to an earlier position in the net and the resulting markings were found to be unchanged.

7.8 Comparing the Results

The markings generated in the three experiments can be compared with each other and with the desired behaviour of the system, which is represented by the autonomous behaviour of the Petri net (see Table 7.1 on page 7-16).

A method developed to provide a graphical comparison is shown in Figure 7.7. The graph is generated by assigning a numerical value to each system state. Since the places may contain, at most, one token, the total state of the system can be represented by a binary value, which in turn can be converted into its decimal equivalent. Each separate state of the system will therefore have a unique value, which can be represented on the y-axis of a graph.

The initial observations show that all the implementations lag behind the desired behaviour, with the closest being the implementation of experiment 3. This is because a single transition takes a number of PLC sweeps to complete firing. Another consequence of this is that there are intermediate states generated by most of the implementations. These intermediate states will be influenced by the order in which the places appear in the rungs of the ladder logic program, hence the difference between the implementations of experiments 1 and 2, which used the same implementation method but with different a rung position for place p_7.

Experiment three is considered to be the closest to the desired behaviour having two intermediate states, which do not appear in the desired behaviour. The first occurs during the firing of the power-up transition t_1, and is the result of the number of sweeps taken by the firing of transition t_1. The second occurs whilst waiting for the simulated 'hardware place', p_9 to become marked. The marking of place p_9 and the removal of the token from p_8 were considered to be simultaneous in the autonomous Petri net, but this behaviour was not possible in any of the implementations.

The graph also shows that experiments one and two contain states that do not appear anywhere in the desired behaviour, and omit some of the desired states. It was initially assumed, when using these implementation methods, that all the desired states would eventually be achieved despite the presence of some intermediate states. This is clearly not the case.

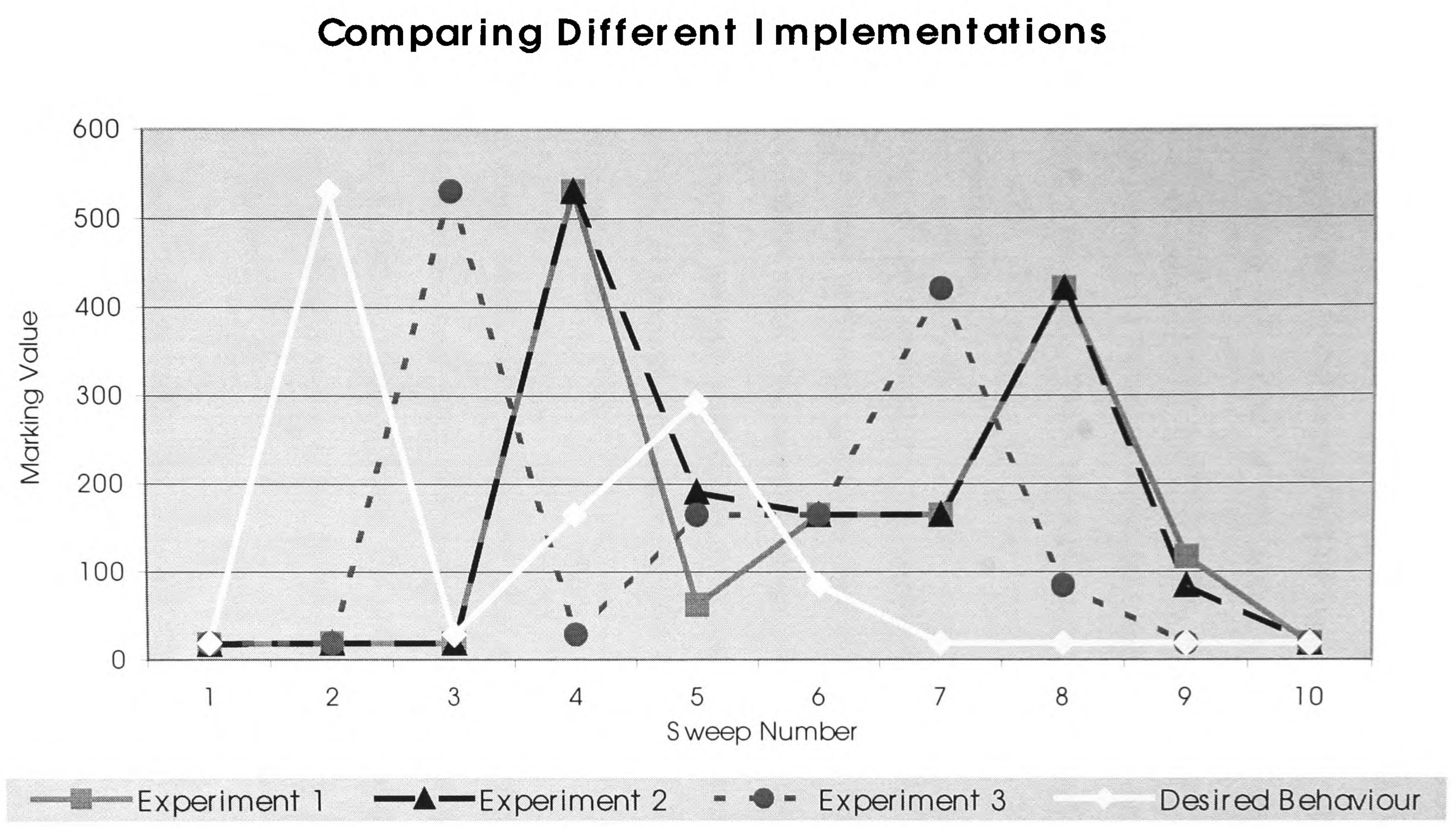

Figure 7.7 Comparison between markings of different implementations

7.9 Implementation in Higher Level Languages

Integrated manufacturing systems usually employ a variety of controllers, which must communicate over some form of channel. The implementation of Petri nets on programmable controllers in the form of ladder logic has been dealt with above. They have also during the course of this work been implemented on a robot controller, whose control language is similar to the BASIC programming language, and as a 'token player' on a Relational Database using Structured Query Language to effect the firing of transitions. These two implementations are discussed below.

7.9.1 Robot controller

The development of the manufacturing cell, described in Chapter 6, required the integration of a number of different controllers. One of these was the controller for a Mitsubishi Movemaster robot. The control language used by the robot is very similar to the BASIC general purpose programming language, with respect to the control structures provided. For example, the implementation of subroutines using GOSUB and RETURN constructions.

In addition to these BASIC like features, the robot controller hides much of the complexity associated with controlling multiple robot joints from the user by providing commands such as MOVE, X which moves the arm to specified position X. The position X must be defined previous to the command being issued by the use of a handheld teach pendant.

Due to its simplicity, the control language is not able to describe complex data structures and therefore it is not possible to implement a token player on such a controller. Instead, the Petri net is used as a control flow diagram with a main routine calling various subroutines, depending on the control signals received.

This has been termed the structured program approach and it allows a uniform design process to be carried out and is easily integrated with token player based controllers.

7.9.2 Relational database

As part of the development work on the fault monitoring system (Chapter 8), there was a requirement to implement a Petri net that would communicate with software running on a Personal Computer running Microsoft Windows. As an experiment a relational database system was used for the following reasons.

- To investigate the possibility of setting up a database to represent Petri net elements as a group of relations.
- To develop an application quickly that could respond to signals from a PLC

The first of the above was achieved by creating a relation for each of the four main elements of the net as follows:

- Places – Describing each place in the net and its marking
- Transitions - describing each transition in the net
- Input Arcs – describing the arcs from places to transitions
- Output Arcs – describing the arcs from transitions to places

The dynamic behaviour of the net is implemented by executing a series of SQL queries, which run Algorithm 7.1.

This algorithm assumes that there will be at most a single arc between a particular place and transition. It may be generalised by providing a weighting to each arc, and this would require modifications to the queries that test for enabled transitions and remove and update the tokens in input and output places.

Algorithm 7.1

```
1. Find all the marked Places
2. Test the transitions linked to them to see which are enabled
3. Set the enabled field in those transitions that are enabled
4. Add tokens to all the output places of those transitions that are enabled
5. Remove a token from all the input places of the enabled transitions
6. Reset the enabled field for all transitions
```

This implementation is a version of the token player as there is a direct representation of the net and its dynamic behaviour. The use of a relational database to implement Petri nets opens up new avenues for the development of automated manufacturing control software. It is envisaged that this will be a useful tool for the further development of the design approach described in this work, as it should be possible to automatically generate the net from sequence specifications if they are used to create the net as a set of tables. There is further work to be carried out in this area and this is described in more detail in Chapter 9.

The ability to implement a Petri net in a relational database highlights the relationship between Petri nets and relational databases. This relationship is seen as a valuable topic for future research.

7.10 Chapter Summary

This chapter has discussed the work carried out on the implementation of structured Petri nets on a number of controllers. In particular the implementation on PLC's has been considered because of their wide usage for sequence control in manufacturing systems. A number of implementations methods have been proposed in the literature but these have either been implemented in high level languages, or have not taken into consideration the differences in behaviour between a Petri net and its implementation. The results presented show that there are differences in behaviour with only slight changes in the implementation, and point to the implementation incorporating places and transitions as being the most accurate method. One reason for this is that, unlike the others shown, the implementation may be forced to contain all the states present in the desired system behaviour.

Another issue considered here has been the size of the code generated. This will be larger for the implementation with places and transitions than it would for those containing just places. However it may be possible to use the earlier implementations as a basis for a reduced Petri net representation which may be applicable in certain cases. The structured program approach to implementation is also a means of reducing the size of a program.

The chapter ends with a description of a new implementation using a relational database to describe the Petri net structure and a set of SQL queries to execute the dynamic behaviour of the net. This implementation may prove useful for the rapid development of Petri net controllers and, through a well designed user interface, may facilitate the automatic generation of the net structure from the process plans and criteria input by the user.

7.11 References

Agerwala, T. and Flynn, M., 1973, "Comments on capabilities, limitations and 'correctness' of Petri nets." In *Proc. 1st ACM Annual Symposium on Computer Architecture*, New York, USA, pp. 81-90.

Colom, J. M., Silva, M., and Villarroel, J. L., 1986, "On software implementation of Petri nets and coloured Petri nets using high-level concurrent languages." In *Proc. 7th European Workshop on the Application and Theory of Petri Nets*, Oxford, UK, pp. 207-241.

Crockett, D., Desrochers, A. A., DiCesare, F., and Ward, T., 1987, "Implementation of a Petri net controller for a machining workstation." In *Proc. IEEE International Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1861-1867.

Cutts, G. and Rattigan, S., 1992, "Using Petri nets to develop programs for PLC systems." In *LNCS 616, Proc. 13th International Conference on the Application and Theory of Petri Nets*, Sheffield, UK, pp. 368-372.

David R. and Alla, H., 1992, *Petri nets and Grafcet: Tools for modelling discrete event systems.* London, England: Prentice Hall.

Ferrarini, L., Maffezzoni, C., and Giua, A., 1994, "Design and implementation issues in the control of Discrete Event Systems." In *Proc. IEEE International Conference on Industrial Electronics, Control and Instrumentation.* Bologna, Italy, pp. 1515-1520.

Henry, R. M. and Webb, M., 1988, "Ladder logic for sequence generation - A methodology." *Measurement and Control,* **21**, pp. 11-13.

Jones, A. and Uzam, M., 1996, "Towards a unified method for converting coloured Petri net controllers into ladder logic using TPL: Part 2 – An Application." In *Proc. IEE International Workshop on Discrete Event Systems*, Edinburgh, Scotland, UK, August, pp. 314-319.

Kasturia, E., DiCesare, F., and Desrochers, A. A., 1988, "Real time control of multilevel manufacturing systems using coloured Petri nets." In *Proc. IEEE International Conference on Robotics and Automation,* Philadelphia, PA, USA, pp. 1114-1119.

Murata, T., 1989, "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE,* **77**, pp. 541-581.

Naylor, A. W. and Volz, R. A., 1987, "Design of integrated manufacturing control software." *IEEE Transactions on Systems, Man, and Cybernetics,* **SMC-17**, pp. 881-897.

Satoh, T., Oshima, H., Nose, K., and Kumagai, S., 1992, "Automatic generation system of ladder list program by Petri net." In *Proc. IEEE International Workshop on Emerging Technologies and Factory Automation,* pp. 128-133.

Silva, M. and Velilla, S., 1982, "Programmable logic controllers and Petri nets: A comparative study." In *Proc. IFAC Conference on Software for Computer Control.* Madrid, Spain, pp. 83-88.

Stanton, M. J., Arnold, W. F. and Buck, A. A., 1996, "Modelling and control of manufacturing systems using Petri nets." In *Proc. 13th IFAC World Congress,* San Francisco, USA, vol. J, pp. 324-329.

Taubner, D., 1988, "On the implementation of Petri nets." *In:* Rozenberg, G., ed. *Advances in Petri nets 1988 (Lecture Notes in Computer Science 340),* Berlin, Germany: Springer-Verlag, pp. 418-439.

Uzam, M. and Jones, A., 1996, "Towards a unified method for converting coloured Petri net controllers into ladder logic using TPL: Part 1 – Methodology." In

Proc. IEE International Workshop on Discrete Event Systems, Edinburgh, Scotland, UK, pp. 178-183.

Venkatesh, K., Zhou, M. and Caudill, R. J., 1994, "Comparing Ladder Logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system." *IEEE Transactions on Industrial Electronics*, **41**, pp. 611-619.

Chapter 8

Fault Monitoring

The Petri net modules discussed in Chapter 4, along with the control structure which may be applied to them, provide the opportunity for a fault monitoring scheme that will allow not only detection of faults in the machine hardware, but also any errors that occur in the control software. The consistent use of Petri net elements to represent both the modules and the communication between those modules, and for representing both hardware and software communications, also allows for the monitoring of those communications.

This chapter will initially provide definitions for several classes of system failure, which will be based on the apparent source of the failure. It will then discuss the main considerations to be taken into account in a fault monitoring system and then finally describe some development work on the fault-monitoring scheme based on Structured Petri nets and the control structure described in Chapter 4.

8.1 Faults and Failures

The words fault and failure are often used interchangeably. In (Lala, 1985) the distinction between faults and failures in digital circuits is stated in as follows:

> *'A failure is said to have occurred in a circuit or system if it deviates from its specified behaviour. A fault on the other hand is a physical defect which may or may not cause a failure.'*

A software failure occurs when the software is executing (Sommerville, 1996). Software faults are programming or design errors that prevent the delivered program from conforming to the system specification. They can also be specification or documentation errors. Software faults are discovered by either static testing or by software failures occurring when the system is running.

In both of the above cases, faults can therefore be described as defects, either in system hardware or system software, that may or may not result in a failure.

8.2 Hard and Soft Faults

The notion of hard and soft faults has also been considered. These are described as follows:

Hard faults are considered to be those that prevent a system element from carrying out its predefined action.

Soft faults are those that inhibit the action of system components but do not prevent them from carrying out their predefined actions.

An example of a soft fault could be a dirty or worn actuator whose action is slowed down by the fault. Such a soft fault does not prevent actuation but inhibits actuation to a limited extent. Such a soft fault, if left untended, could eventually give rise to a hard fault - e.g. the actuator ceases functioning altogether.

According to the definition of faults and failures given in section 8.1, a fault may or may not cause a system to fail. Thus a hard fault is one which causes the system to fail. A soft fault on the other hand may not actually cause the system to fail unless, referring to the earlier example, there is a specified time limit for actuation. In this case the fault has again caused a failure, as the system is unable to perform according to its specification.

The distinction between hard and soft faults is therefore dependent on the degree to which the performance of a system has been specified. It is a useful distinction since the detection of a soft fault may indicate the potential failure of a device before its actual occurrence.

8.3 Classifying Failures

The common failures occurring in manufacturing systems can be grouped into three main classes:

- Hardware failures
- Software failures
- Product failures

It will be seen that there is some interdependence between these classes, which causes difficulties in their diagnosis.

8.3.1 Hardware failures

Hardware failures exhibit themselves as failures in the manufacturing system hardware. There are a number of potential causes of hardware failures and these are categorised as follows:

Sensor fault

A sensor is damaged in some way preventing it from carrying out its required task. The sensor may have failed completely and thus have a 'stuck at' type fault (either stuck on or stuck off) or, in the case of more complex sensors, may be giving an erroneous output.

Actuator fault

An actuator is damaged in some way preventing it either from acting at all, or from acting within a prescribed period of time. Again the actuator fault could be described as a 'stuck at' fault if it is either permanently actuated or permanently not actuated. Alternatively there may be a gradual degradation of actuator performance caused by wear or dirt, preventing actuation within the required period of time, as such the actuator would be said to have a soft fault.

Software failure

As well as being a class of failures in their own right, software failures may be the cause of an apparent hardware failure. If a failure occurs in the manufacturing system software then this will only be recognised when the manufacturing hardware fails to act in the expected manner. Thus any failure in the system software will result in a hardware fault of some description. Software failures are described in more detail in section 8.3.2.

Using purely sensory data, it is very difficult, and often impossible to distinguish between any of these three types of fault.

8.3.2 Software failures

A software failure results from a fault that exhibits itself in the system control software. Software failures arise due to the complexity of most modern software, and are caused by faults in various stages of software development. These are described as follows:

Design fault

A design fault occurs when the software design does not properly conform to the specification. Deadlock or overflow may occur in manufacturing systems using shared resources if there are errors in the design of the control code. Deadlocks and overflows would, of course, not form part of the specification, although it may be assumed by a specification that these would not be desirable and therefore overlooked during the design. This would then be a specification error.

A design error in manufacturing control software will often exhibit itself as a system or component failure. For example an actuator may not actuate at the correct time, or the system may not initialise.

Implementation fault

An implementation fault occurs when there is an error translating the system design into control code. An error made during implementation of the control software results in an apparent system or component failure. Such implementation errors will be common when using languages such as ladder logic where there is no static testing software available and where the syntax is limited and the symbols used in that syntax are very similar.

Controller failure

A controller failure occurs when part of the controller hardware, such as the microprocessor, or memory, fails. This could result in the control unit failing altogether, or will result in the corruption of software or data. With industrial programmable logic controllers the design is robust and such failures will rarely be allowed to propagate down to the working manufacturing system. However with the increasing interest in Personal Computers as general-purpose controllers this will become more of an issue. If a controller fails it will adversely affect the control software, which will result in an apparent system or component failure in the manufacturing system under control.

Network failure

A network failure is the result of an error in the communication of data between different controllers. When the data/computer network in place in the manufacturing system fails or produces errors, the result is a data error. Such a data error could result in an apparent software fault, which again will exhibit itself as a hardware failure. A more general case is described in (Adlemo and Andréasson, 1993) which may be classed as communication failures rather than network failures. These point to failures in the data network and the materials network of the system.

8.3.3 Product faults

A product fault exhibits itself in the actual product being manufactured. Such a fault could be caused by low quality in materials/components, but may also be caused by a hardware fault in the manufacturing system. (Hardy et al, 1989) discuss a class of faults called task faults, which are the same as product faults. These task faults include problems such as the arrival of a faulty component or a component being dropped.

One class of product faults is that of product quality. If the product quality at some stage is not to the specified standard (assuming that there is some specified standard of quality in the product specification) then, although a fully functioning product has been produced, there is still a fault that effects the quality of the product. This would be regarded as a soft fault in the definitions given above.

8.3.4 The need to reduce software faults

The more complex the control software of the system the increased likelihood of a software error in the design/implementation stage. Some of the design faults may be detected if the system is run on a simulation, but this will not detect implementation errors. The only way to prevent implementation errors in this case is to use the same formalism/language for both simulation and control.

In general, all the software faults described above will exhibit themselves as hardware or product faults. This can cause many failures in manufacturing systems to be mistakenly diagnosed as hardware failures when in fact the software has caused them. This situation is hinted at in (Adlemo and Andréasson, 1995) but is not expanded and there are currently no statistics indicating the level of misdiagnosis.

In order to diagnose faults in manufacturing systems more accurately, there must be some means of distinguishing between the types of fault that have been described thus far. The rest of this chapter looks at some considerations when carrying out fault diagnosis and then goes on to describe a method of diagnosis based on the Petri net controller design method described in Chapter 5.

8.4 Fault Prevention

There are a number of methods for preventing errors from being introduced into software (see (Adlemo and Andréasson, 1995)). Some of these are detailed here with reference to the Petri net method applied in this thesis.

8.4.1 Structured Programming

Structured programming languages are used in general software development because they are written in some form of sequential manner that is easier to read and therefore errors are more easily detected. Most sequential programming languages incorporate a set of control structures that clearly indicate the flow of control through the software. Structured programming encourages the use of subroutines. These allow the control flow of the software to jump to other parts of the program and return to the initial point once the subroutine has completed. Therefore the use of constructs such as the GOTO statement in BASIC are discouraged because they do not allow the return of the control flow to the point at which the jump occurred. This causes the program to take the form of 'spaghetti code' which is difficult to read and therefore to debug and maintain.

Ladder Logic is a far from structured approach to software development, yet, some form of logic programming is necessary in manufacturing systems to define the sequences and conditions under which events can occur. The introduction of the IEC1131 standard is intended to encourage a more structured approach to manufacturing software design, using languages such as structured text (see (Juer and Oliver, 1993)).

Chapter 7 described how the Petri net is implemented in Ladder Logic and described the order in which the places are implemented within the Ladder logic program. By structuring the program in this way, it is easy to follow the program and to locate a section of code relating to a particular net. Therefore the problems of readability have been greatly reduced.

8.4.2 Parallelism

Parallelism is one construct that is likely to introduce errors into system software because of its complexity. Yet the use of parallel processes is an important aspect of manufacturing systems. The use of a Petri net method can reduce the complexity of parallel systems. It allows analysis of such systems and therefore may prevent many of the complexities associated with such a construct.

8.4.3 Modularity

The structured approach presented here provides many of the advantages of modular design. Each subsystem is described in terms of its interface with other elements of the system and thus there is a well-defined means by which access is granted to that subsystem. This prevents the multiple access of many resources in the system and thus prevents many possible errors. The requirements for a modular approach are described in (Meyer, 1988).

8.5 Considerations for Monitoring

In (Holloway and Chand, 1994) a fault monitoring method is presented that will 'trace' entities through the system. They present a list of requirements for such a system, which has been used as a benchmark for the system described here. The list of requirements they give is as follows:

- Low Processor requirements
- Easily distributed
- Applicable to modelled and observed behaviours (i.e. the system will run on real and modelled systems)
- Functional under unknown start-up conditions and improper observations
- Suitable for highly concurrent systems

The example system used in (Chand. 1993) and (Holloway and Chand, 1994), looks at a conveyor system with a number of elements moving along the conveyor in order. Each element is given a time signature and is traced through a number of stages in the manufacturing process. The authors concentrate on the ability to distribute the control code for the monitoring system over a number of processors.

The requirement given for unknown start up conditions is necessary in the case of conveyor systems because there may be occurrences during production of faults that may not be immediately detectable using sensors. This would particularly be the case where the quality of a product is the source of a failure.

8.5.1 Initial state

Where the hardware or software faults in a machine are being monitored, the initial state of the machine must be known (in both hardware and control software terms).

Many Petri net methods ignore the initial state of the system as an issue assuming the state is known. However when controlling a real device, its initial state must be set, for safety as well as operational purposes – and certainly with CNC machines there are a variety of initial states that can be used depending on the particular job.

Using the Petri net method described in this thesis the initial state of the system can be determined according to the sensory data available. The hardware and software initialisation stages described in Chapter 6 are used specifically for this purpose. The hardware initialisation stage checks the sensory data and if necessary actuates parts of the system to ensure that any areas undetectable by sensors are clear. The software initialisation stage sets any counters and timers and modules to their required initial states.

Thus the initial state of the system is always known after power up since it has been designed into the control logic (the initial state of the system is not known at power up, but the system is placed into the required state before processing can start).

8.5.2 Timing information

Any method for fault monitoring must accurately characterise the behaviour of the system, both in the sequencing of events and in the timing relationships between events. Any discrepancy in the sequence of events will indicate the presence of a hardware failure of some kind, or perhaps a controller failure. A discrepancy in timing will indicate a less serious failure - perhaps a faulty but not failed actuator, perhaps a sensor that has moved slightly, or perhaps a problem with the product itself being not arriving when it is supposed to.

In fault monitoring a great deal of interest is placed on the timing characteristics of system events.

Timing becomes important initially when there are no sensors in place to detect an event, for such cases it is common to insert a timer to indicate that such an event has occurred. In fact even with a timer in place there is still a lack of information concerning the event in question. When the controller receives a time-out, the only real information conveyed is that the timer has timed out. There is no direct information concerning the state of the system hardware. The potential danger of such a use of timers is quite clear. Therefore, any timer used in place of a sensor cannot be relied upon, certainly for safety reasons, but also for the purposes of fault monitoring.

Where timing is useful for fault monitoring is in timing the actions of events and making some comparison between the actual event time and the expected event time. This will indicate the presence of wear or some other fault on an actuator or sensor. Such timing information has also been used to indicate the loss of a product during conveyance (Holloway and Chand, 1996).

The initial work on the fault monitoring scheme presented in the next section has concentrated on sequence information in order to look at the initial feasibility of the system.

8.6 A Fault Monitoring Architecture

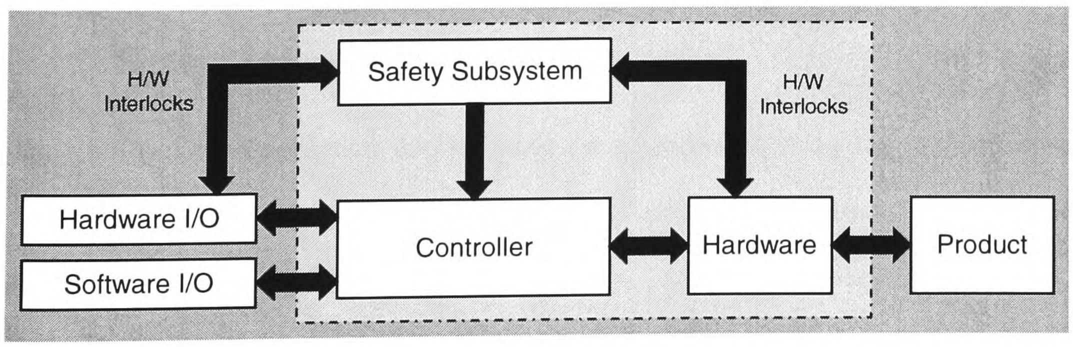

Figure 8.1 Relationship between a manufacturing station and the elements in its environment

The work here is interested in detecting faults in manufacturing software and preventing the effects of such faults from propagating through to the hardware and the product. Software faults will originate in the controller for a variety of reasons (see section 8.3.2). Figure 8.1 is a slightly modified version of that shown in Chapter 4 (Figure 4.8). It not only shows the controller and its immediate environment but also details the environment of the hardware of the system. The lighter grey box indicates the workstation, which consists of the safety subsystem (relating only to this particular workstation) the controller, and the machine hardware (consisting of actuators and sensors). Note that the product itself is not considered part of the workstation, even though in some cases it may be a raw material that is stored in a part of the machine hardware.

It can be seen from this figure that any error occurring in the controller may propagate through the hardware to the product itself. It is therefore proposed that if a software fault can be trapped before it has a chance to propagate then its effects on the hardware and the product can be prevented, or at least reduced.

The top level design for such a fault monitoring architecture is described in Figure 8.2. The figure shows separate computers monitoring the control signals and the feedback signals passing between the machine's hardware and its controller. These tasks may in fact be carried out by a single device.

The logic behind the fault monitoring mechanism detailed in Figure 8.2 is as follows:

If the Petri net representation can be used as a specification for the working of the manufacturing system then, providing the current state of the system is known, then the next states or possible states are also known. Therefore the actual next state of the system can be compared with the desired next state (which is taken from the specification) and any discrepancy between the two will indicate the presence of a fault.

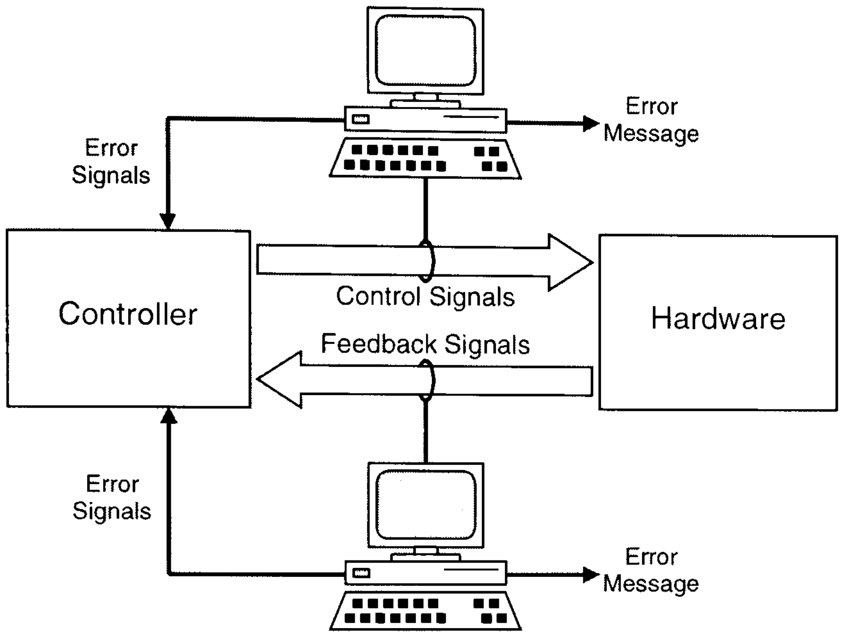

Figure 8.2 Architecture of the proposed fault monitoring system

This is the standard approach (if there is any standard approach) for a 'state follower' type fault monitoring system, as it simply compares the current state with the expected state of the system. The point at which this arrangement differs is that it not only monitors feedback in terms of the state of the system, but also monitors the control sequences being sent to the system. Thus any errors caused by failures within the controller (be they software or hardware) can be detected as they are sent to the machinery that is under control. If they can be detected, then they can be trapped and their propagation down through the manufacturing hardware to the product can be prevented.

The ability to trap software error does not stop at the interface between the controller and machine hardware. The modular structure of the software design and the use of places as a communication medium between modules will allow errors to be trapped earlier on in the software process (at higher levels in the software hierarchy). Figure 8.3 shows, in a simplified form, the communication messages passing between software modules in the control structure.

A software error in the control net will propagate down to the subnet level, which in turn will propagate down to the output net level. It is from this level that the communication signals are passed to the system hardware. Therefore when an error occurs, the output net must first be analysed and then the subnet that controls the output net (there may be a number of these) and then finally the control net must be examined. If the error can be detected passing between the layers in the control hierarchy, much time and effort can be saved in detecting and correcting the error.

Again, since the means of communication is the same in all cases, the monitoring of individual software modules can be carried out in the same way if the software modules are implemented on the same controller as it would if they were distributed over a number of controllers.

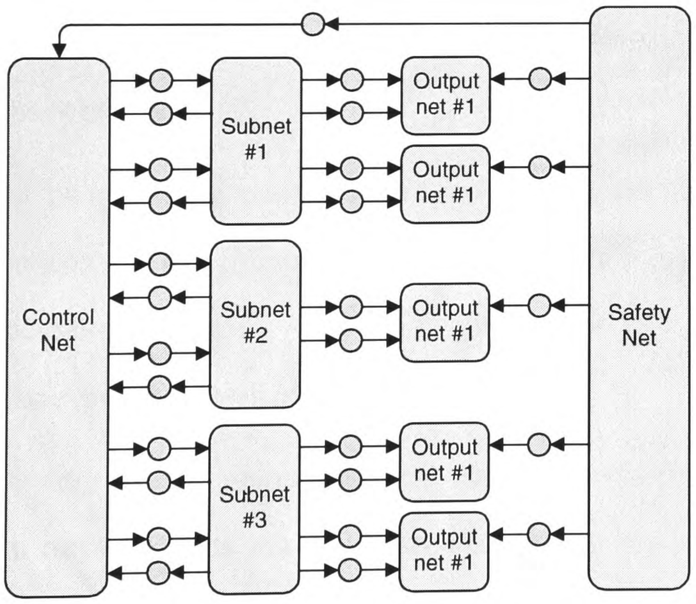

Figure 8.3 General Petri net structure used for the control of manufacturing systems including communications signals

8.7 Monitoring a Sequence

In Chapter 5, the control net was constructed using a set of paths. Given such a control net, the paths can be described as the set of possible sequences that might occur, given a particular request from the system's environment. Therefore the arrival of a request can be used to trigger the monitoring of a particular subset of all the possible control signals generated by the system. Once a sequence starts, its sub-operations (represented by the subnets) will be called upon. Each of these sub-operations will also be represented by a sequence and monitoring of these sequences will be triggered by their initiating control place being asserted. Once a sequence has been completed it will cease to be monitored until its initiating place is asserted once more.

8.8 Space Cost of the Method

The proposed fault monitoring method will not monitor every possible state of the system. It simply monitors which sequences are triggered and in which order. Therefore the only places that need to be included in the monitoring exercise are the control and feedback places.

The space cost of the monitoring method can be estimated by calculating the number of possible sequences that can occur. There will be a control place and a feedback place for each possible sequence. Added to this will be the control and feedback pairs for each subnet that is present in the control structure.

This systems therefore offers a reduced size compared to a standard state follower as there is not a requirement to monitor every state, rather the requirement is to monitor every change of state necessitating communication.

A reduction in the size of the data being monitored has an important implication on the speed of the system. The comparison between the current state and the desired state can be achieved more quickly with less data involved in the comparison.

8.9 Fault Monitoring Mechanism

The test bed for the fault monitoring system was the raw materials station described in Chapter 6. The Station is controlled by an AEG Modicon 984 PLC which is attached by a serial port to a Microcomputer, which is running the Microsoft Windows 3.1 operating system.

Initial investigations into the feasibility of the approach to fault monitoring were carried out using the Modicon Programming Panel software (Modsoft) to monitor the addresses of the PLC.

8.10 Implementation Issues

The Petri net design method provides a clear description of the sequence of events through which the system will pass. However it has been noted in Chapter 7, that the method of implementation although it achieves the same result as the Petri net description, may not go through exactly the same sequence of steps. Therefore a number of questions need to be answered.

1. What are the differences between the implementation and the expected behaviour of the system, in terms of PLC addresses being set or reset?
2. Are the differences predictable between different runs of the same portion of code?
3. Are the differences predictable between different PLC's?

Once these questions have been answered, the sequence of events that should occur in the system, as seen by the PLC, can be determined. The experiment carried out in Chapter 7 on various implementations show that for a small Petri net the sequences generated by the PLC are repeatable. The method used to enumerate the sequences may be used to create a fast numerical comparison between the desired behaviour of the system and its actual behaviour.

8.11 Dealing With Choices

Often there will be a choice of possible events that could occur within the system. For example where a test is being carried out and the system is awaiting one of a number of outcomes, or where another subsystem is instructing a station to produce one of a range of possible parts. In these cases the sequence of events is not known in advance. However once the request for a particular part has occurred the sequence becomes known and therefore can be followed. Here the result of the test is used as the trigger for the remainder of the sequence.

This has implications for the design of control nets and subnets, as a question arises concerning the subsystem that is actually determining the outcome of such a choice. Can it be treated in the same way as an external agent making a request of the system?

8.11.1 Concurrency

When concurrent operations are being carried out, it does not matter which one finishes first. In fact either of two concurrent operations may finish first on different occasions. Any fault monitoring system must be able to handle such concurrent operations and the uncertainty associated with them.

8.12 Monitoring

Now that the sequences of events can be determined they need to be compared to the real events that occur in the system during run time. The events being monitored are the occurrences of the control signals that pass between the different modules in the control structure (and also the signals that pass to the hardware). These signals are not associated directly with any action by the hardware and therefore they will often occur during only a single cycle of the PLC, which has at most a 25μs duration for the PLC used. Therefore some automatic mechanism for capturing the events must be used.

8.13 Diagnosis and Containment

As already mentioned the control signals are transient and may occur in a very short period of time. If an error is detected, then its effect must be prevented before it has the opportunity to propagate too far down the control structure. Therefore the diagnosis approach must be completed in a short period of time – ideally in a single cycle of the PLC. It may be necessary to suspend the action of the PLC or the control code in some way so that the error can be diagnosed. This will not usually adversely effect the operation of the manufacturing hardware except to slow down the process that is being carried out at the time of detection. If the time taken to diagnose the error is sufficiently small, then the process will not suffer any noticeable adverse effects. This is an option described in (Hardy et al, 1989) where the time critical element is described as

data collection → detection → catastrophe avoidance

and also in (Hasegawa et al, 1990) where a 'layered' Petri net is used for exception handling.

8.14 Using standard software

The software used to capture the events as they occur in the PLC is a specialised piece of software that will allow information from the PLC to be handled by any

Windows application. The use of off the shelf software aids more rapid development of the system, but it is expected that software would be able to react faster if it is written specifically for the purpose.

8.15 Chapter Summary

This chapter has presented a new taxonomy for manufacturing system faults and failures in which they are classified by their apparent source. It points out that there will usually be a cause and effect relationship between classes of failure and proposes that a number of hardware failures in manufacturing systems may actually be caused by failures in the software.

A method is presented for the detection and diagnosis of faults that is based on the Petri net design method of Chapter 5. The ability to automatically detect and diagnose faults in manufacturing systems and to attempt automatic recovery not only saves manufacturing system down-time but will also prevent a number of accidents involving manufacturing systems operators or maintainers (Järvinen and Karwowski, 1995). This method allows the distinction between a fault caused by faulty hardware and one caused by an error in the software. It should also be possible to detect which part of the software caused the error. In addition, this diagnostic ability is a product of the software development process and is not appended to the software. This means that there is close integration between the control software and the monitoring software, and that fault diagnosis is not merely considered as an afterthought in the software development process.

8.16 References

Adlemo, A. and Andréasson, S., 1995, "A dependability taxonomy for flexible manufacturing systems." *International Journal of Computer Integrated Manufacturing*, **8**, pp. 189-196.

Adlemo, A. and Andréasson, S., 1993, "Failure Semantics in Intelligent Manufacturing Systems." In *Proc. IEEE International Conference on Robotics and Automation*, Atlanta, USA, vol. 2, pp. 166-173.

Chand, S., 1993, "Discrete-event based monitoring and diagnosis of manufacturing processes." In *Proc. American Control Conference*, San Francisco, CA, USA, pp. 1508-1512.

Hardy, N., Barnes, D., and Lee, M., 1989, "Automatic Diagnosis of task faults in flexible manufacturing systems." *Robotica*, **7**, pp. 25-35.

Hasegawa, M., Takata, M., Temmyo, T, and Matsuka, H., 1990, "Modelling of exception handling in manufacturing cell control and its application to PLC programming." In *Proc. IEEE International Conference on Robotics and Automation*, pp. 514-519.

Holloway, L. E. and Chand, S., 1994, "Fault monitoring in manufacturing systems using concurrent discrete event observations." In *Proc. AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, Stanford University, CA, USA, pp. 65-69.

Holloway, L. E. and Chand, S., 1996, "Distributed fault monitoring in manufacturing systems using discrete event observations." *Integrated Computer-Aided Engineering*, **3**.

Järvinen, J. and Karwowski, W., 1995, "Analysis of self-reported accidents attributed to advanced manufacturing systems." *International Journal of Human Factors in Manufacturing*, **5**, pp. 251-266.

Juer, J. and Oliver, J., 1993, "Building and using graphical programming tools for the IEC 1131-3 standard." In *Proc. IEE Colloquium on Advances in Software Engineering for PLC Systems.* Savoy Place, London.

Lala, P. K., 1985, *Fault Tolerant and Fault Testable Hardware Design.* London, England: Prentice Hall International.

Meyer, B., 1998, *Object Oriented Software Construction*, Upper Saddle River, NJ, USA: Prentice Hall.

Sommerville, I., 1996, *Software Engineering,* Addison-Wesley.

Chapter 9
Conclusions and Further Research

This thesis details how a Petri net design method has been taken and formalised with the aim of automating the development of manufacturing control code. The original Petri net definition has been used to provide a modified interpretation, which has been called structured Petri nets. The elements upon which the structure is built are Petri net modules, which differ from those found in the literature as they use places for inter-module communication. This can be seen to simplify the complexity of the control structure, and as such will allow the use of standard Petri net analysis techniques that can be applied to individual modules, or even to the whole system.

The thesis also deals with important issues of implementation, which seem to be missing from much of the literature, especially that dealing with implementation on Programmable Logic Controllers. It also presents a new method of implementation on a Relational Database.

The thesis then goes on to develop a fault monitoring method which is made possible by the modular structure and the use of places as communication 'agents'. This fault monitoring method has the potential to distinguish between faults that occur in the hardware of a manufacturing system and the faults that occur in the manufacturing control software. It is also, by the same reasoning, possible to locate the particular software module where the error occurred before it propagates down to the manufacturing hardware.

9.1 Structured Petri Nets

Chapter 3 dealt with the development of structured Petri nets as a more formal modelling approach to that presented in Chapter 2. Each module in the structure is built up from these Petri nets with external inputs and outputs, which are represented by places. This more formal approach also attempts to classify Structured Petri nets in relation to other classes present in the literature. The chapter shows that they are very closely related to controlled Petri nets, a Petri net variant that is commonly used for solving forbidden state problems (Krogh, 1987). They may also be compared with decision free nets, as proposed by (Krogh and Sreenivas, 1987).

9.1.1 Relationship with Free Choice Petri nets

There also appears to be a relationship between the general structure of each Petri net module and Free-Choice Petri nets. All the systems to which the nets have been applied can be described by Free-Choice Petri nets, and this is ether due to the class of system to which the method is being applied or due to restrictions placed on the system description by the modelling formalism. If it is due to restrictions of the formalism then such restrictions are acceptable and do not affect the performance of the systems being controlled. If it is due to the class of systems being modelled then these systems can be classified as free choice systems, and Structured Petri nets as controlled Free-Choice Petri nets. Such a classification would give many advantages, as free Free-Choice nets are more general than other more complex classes of Petri net. Such a relationship appears to be borne out by other related work (Proth et al, 1997).

9.2 Petri Net Modules

The idea of Petri net modules, as developed in Chapter 4, is not in itself a new approach. However, the explicit nature of each module as a stand-alone entity is not found in other supposedly 'modular' methods. For example, in (Proth et al, 1997) the modules used are not self-contained and are more like the paths described in Chapter 5. It is this modular cohesion that allows the reuse of modules, the extent of which will depend on the flexibility of the controller being programmed. At the very least, the design and implementation of a module may be reused.

It is also the true modular nature of the Petri nets and the simple communications between the modules that allows the fault monitoring method of Chapter 8 to be possible. In (Hansich and Rausch, 1995) a more complicated communication mechanism uses both condition and event signals between modules. Also the modular construction provides more consistency than that described by (Hansich and Rausch, 1995) since when the modules are constructed into a full system the system is still described by a Petri net and can conceivably be analysed using standard Petri net analysis methods.

The advantages of the modular approach adopted in this work are as follows.

1. Once the external interface has been finalised, the internal behaviour of the module and any sub-modules can be altered without the need to modify the whole system.
2. A module can be tested in isolation of the real system and then be combined with the remainder of the system after it has been shown to behave correctly. The testing does not have to take place on line on the intended system – it can be carried out at a completely separate location. This is essential for manufacturing applications.
3. The modular structure of the system can potentially reduce the complexity of any analysis that needs to be done on the system. Each module can be analysed

and its own properties recorded. In this way a set of standard modules may be developed with pre-defined properties and behaviour.

4. The modular structure also reduces the perceived graphical complexity of the system to the user/developer/manager. It is believed that the complexity of most Petri net methods prevents their being adopted by practitioners, and that a less complex appearance would encourage more widespread adoption.

9.2.1 Centralised and distributed systems

The work carried out here concentrates on a centralised control structure and describes a controller hierarchy from a control net down to output nets. Another advantage of a modular approach to software design is that it enables a more distributed approach to controller design. Manufacturing systems require a hybrid of both centralised and distributed design because an individual workstation's hardware is broken down in a stepwise hierarchical manner and its 'dumb' elements need to be controlled centrally. However, to achieve greater flexibility it may be more desirable to give greater autonomy to individual workstations in the system allowing them to make their own decisions based on knowledge of their current state.

Using a modular Petri net approach allows both centralised and distributed control code to be developed without the need for multiple formalisms or complex extensions to existing formalisms. It is conceivable that the same module may be used in both a centralised control system and a distributed control system.

9.2.2 Systems integration

The work of Chapter 6, on the manufacturing cell shows that systems programmed using structured Petri nets can be easily combined by systems programmed using other methods. The example given shows that the Petri net implemented on a PLC can communicate effectively with a structured program (which used the Petri net as a basis for its structure) running on a robot controller. The Petri net was also able to communicate with programs designed without the use of Petri nets running on the CNC machine tools. The combination of Petri nets with other programming techniques stems from the simple communication mechanism, and enables the integration of a variety of manufacturing controllers, possibly over communications networks. This is something rarely considered in the development of Petri net controllers.

9.3 The Design Method

One of the objectives set out in the introduction (Chapter 1) was to automate the development of manufacturing control code. The design method developed is structured in such a manner that it is amenable to automation, and in fact would greatly benefit from automation.

There is however an issue of designing not purely software components but hardware/software components as described in (Naylor and Volz, 1987). This is where the hardware behaviour and software signals from a component are both used to describe the interface between a component and its environment. This ideal has still not been obtained and requires a language in which to express this interface. This work has shown that structured Petri nets are capable of describing the interface for components, such as pneumatic actuators, and subsystems made up of a number of such components.

There is also a related issue in the design of manufacturing hardware. Many hardware systems are developed without regard to the control software that will be running on them. In some cases this is due to the general-purpose nature of many of the controllers. It is the view of this work that hardware and software should be developed in tandem to produce the most efficient combination of the two. If a hardware component has a particular fixed set of tasks then there is no need for a general-purpose controller, but if it is required that the hardware is flexible and may perform a variety of tasks, then its incorporated software should also be capable of such flexibility.

9.4 Implementation

Work on the implementation of Petri nets on PLC's has shown that there is no agreed method, and that many authors do not treat the issue with the necessary caution. It is not enough to simply create a rough parallel to the Petri net in the language of the particular controller. If the Petri net is to be used to compare that actual behaviour of the system to a specification, then the implementation must be predictable and execute in the same manner as the Petri net 'model'. If there is a difference between the manner in which the implemented Petri net runs and the way in which the Petri net model behaves then this should be predictable and should be accounted for in any automated monitoring system.

The implementation method used for the third experiment in Chapter 7 follows closely the behaviour of the Petri net, and can be used as a satisfactory model for Petri net behaviour for the example given. The markings given by the implementation have been shown to be repeatable, and therefore may be used as a template for monitoring sequences of events in a manufacturing system. The method used for enumerating the states of the system may also be beneficial to that purpose.

In (Hanisch and Rausch, 1995), the control code has actually been implemented automatically on a test system. In this research, however, the implementation has been carried out manually. It is suspected that further detailed investigation would show a greater similarity between the algorithm used by (Hanisch and Rausch, 1995) and that used here.

9.4.1 Implementation on a relational database

The implementation on a relational database is new and offers many possibilities as far as rapid development of systems is concerned. The necessary information to generate each path may be requested by the system and once a set of paths has been completed the associated Petri net can be produced using the rules presented in Chapter 5.

At present only a small, single path example has been programmed into the database, and it is expected that if another path is to be entered, then this will require the generation of a completely new database. With a number of paths per control net, and one database per path, the resulting set of databases will become quite large. It is therefore expected that this would not be an ideal final solution, but would certainly aid further development of the automatic control code generation. The drawbacks of the database approach are therefore the size of the final system and that the system incorporating a database might be slower than one using a specially designed file system for the storage of the Petri net structure. Its advantages are that it can be used as an inexpensive development system to test ideas, and generate prototypes, from which more compact representations may be developed.

9.5 Fault Monitoring

The fault monitoring system proposed in Chapter 8 appears to be a promising development. It differs form other monitoring systems proposed as it captures a failure at its origin, be it in software or hardware. The main contribution of the method is on the software side where the module in which the error occurred can be isolated. This will lead to a more focussed diagnosis of the fault, which in turn will increase the speed of diagnosis.

Chapter 8 also presents a new taxonomy for failures in manufacturing systems, based on the apparent origins of the failures. Failures will propagate, and therefore it is difficult to know whether a hardware failure was due to a hardware fault or was perhaps caused by a software fault. The monitoring method proposed will remove some of this doubt at the start of diagnosis by indicating that the failure was either initiated in the hardware or the software of the system.

The work carried out so far on fault monitoring indicates the potential of structured Petri nets in this area. There is still more work required to show that the approach will work on a real system and to what extent the problems highlighted in Chapter 8 can be overcome. One major concern over the work carried out to date is that there is no published information on failures in manufacturing systems and their origins, and so it is difficult to evaluate the usefulness of the fault monitoring system. There is a need for a system that distinguishes between the failures arising from hardware faults and failures arising from software faults. Recognition of this need comes from the well documented problems of generating error free software (Jagdev et al, 1995) and, more specifically, the lack of formal approaches in manufacturing control software development. Also specific cases from the manufacturing literature point to the need for such a system (see (Adlemo and Andréasson, 1995) and (Järvinen and Karwowski, 1995)).

9.6 Further Research

Further work needs to be carried out in order to develop the work of this thesis into a commercially viable solution. In addition the work carried out thus far points to new areas of research that have not yet been specifically addressed here.

9.6.1 Structured Petri nets

Analysis

One aspect of the ultimate objectives of this work (see Chapter 1) is the ability to analyse the control code to ensure that certain properties are maintained. This becomes even more important when looking at the co-ordination level of manufacturing systems. Further work needs to be carried out on the analysis of structured Petri net models. Analysis of individual nets may be done using reachability techniques, however this is only suitable for small problems. So far in this work all modules have been small enough for such analysis. If a distributed approach is adopted then it is feasible that all modules will be relatively small and therefore reachability analysis is a viable means of analysing the properties of the nets. However, with a centralised system the control nets become relatively complex and therefore other analysis techniques may be required. Some enumeration of the amount of time saved by analysing individual nets over that used for analysing the whole model would give some indication of the advantage of a distributed model over a centralised model.

Further work also needs to be carried out on the suitability of structured Petri nets for the co-ordination level of manufacturing systems. This would require extending the work to other workstations in the Mechatronics Research Centre (as described in (Stanton and Arnold, 1997)) in addition to that of the raw materials station. This could lead to the use of structured Petri nets for performance analysis by incorporating time into the formalism.

9.6.2 Forbidden state problems

The work carried out in this thesis does not deal specifically with forbidden state problems although there is some relationship between the nets proposed here and those developed by authors such as (Hanisch and Rausch, 1995) and (Holloway and Krogh, 1990). As a separate piece of research it would be interesting to apply the structured Petri nets to a forbidden state problem, and make comparisons between the structured Petri net approach to that of controlled Petri nets and Net Condition/Event systems. It would appear that the nets used here are less complex than Net Condition/Event systems, but this needs to be shown more formally.

9.7 The Design Method

Some further research needs to be carried out on the specification of manufacturing processes and how they are broken down into sub-processes, and sequences of events. It is believed that there is currently no universal standard for such representations and this is borne out by the variety of such representations used in the Petri net literature (for example, (Proth et al, 1997) and (Hanisch and Rausch, 1995). If these sequences can be entered into a computer in such a manner that their sequence and parallelism can be described, then the Petri nets could be generated automatically. Once the Petri nets have been generated, the control code can then be generated, using either the algorithms presented in Chapter 7 or something similar to that used in (Hanisch and Rausch, 1995). The advantage of the work in (Hanisch and Rausch, 1995) is that the control code has actually been implemented automatically on a test system, whereas in this research the implementation has been carried out manually. It is suspected that further detailed investigation would show a greater similarity between the algorithm used by (Hanisch and Rausch, 1995) and that used here, than is currently understood.

9.8 Implementation

The implementation method used for the third experiment in Chapter 7 follows closely the behaviour of the Petri net, and can be used as a satisfactory model for

Petri net behaviour for the small example given. Work needs to be carried out to ensure that this will still be the case for a larger system, such as the raw materials station. Also the state enumeration technique of Chapter 7 needs to be examined further, and used on a larger system to see whether it will provide any advantages for fault monitoring.

9.8.1 Analysis of a PLC

Another avenue of further research is to analyse the behaviour of PLC's to see if they fall into a particular class of computing machinery. If they do, then the task of modelling this class of machine with a Petri net can be attempted. This is an issue for formal languages, checking whether a Petri net can generate the formal language that describes the behaviour of a PLC. This would give a better insight into the behaviour of programs that run on a PLC and make them more predictable.

9.8.2 Relational Database and CASE

At present only a small, single path example has been programmed into the database. The use of the database needs to be extended to allow the user to describe the manufacturing process for which code is to be generated. Also an automatic code generator needs to be produced to ensure that there are no implementation errors in the system.

This leads to the notion of CASE tools for manufacturing software and systems design. The existence of manufacturing specific CASE tools would make an interesting study, along with comparisons between this for manufacturing systems and those used for purely software systems. Since there is considerable overlap between the two it may be possible to exploit similarities, and transfer techniques from one area to the other.

9.8.3 Fault monitoring

The fault monitoring system requires a large amount of work to become a fully implemented system. The ideas of the system seem very promising, but certain problems will only arise when the system is closer to running on a complex system. Initially work will be carried out on monitoring simple sequences, which will then be followed by investigation of the behaviour of concurrent processes. The fault monitoring system is the culmination of all the other ideas in this work, and thus the further work suggested for some of these ideas will also need to be carried out before the fault monitoring system can be completed.

Systems described by high level languages

Some development work needs to be done on representing more complex message passing between Petri net modules. If more complex messages can be represented in a relatively simple manner then there is scope for the fault monitoring system to be developed for systems described using high level programming languages. This would provide application areas wider than manufacturing systems. It may be that higher level Petri nets are required for this and there is probably a suitable class in existence, however the communications is central to the ability to monitor different parts of the system, and so care needs to be taken over any possible extensions.

9.9 Work on Safety Systems

The Petri net structure first described in Chapter 2 and then extended in Chapter 3, includes a safety subsystem, called the safety net. Initially the role of the safety net has been to monitor the system for an unsafe condition to arise and then to perform a safe shutdown of the system. This was implemented in the initial control code design for the raw materials station of Chapter 6.

A more complete role of the safety subsystem could include fault diagnosis software and also failure recovery software. It has been shown in (Liu and Chiou, 1997) and (Yang and Liu, 1997) that fault trees can be produced using Petri nets, and indeed they show that there are some advantages to doing so. This would be an ideal way of appending the task of the safety subsystem to not only detect failures but also to diagnose them, and produce the appropriate error signals indicating such failures. In this role the safety subsystem is not purely acting as a safety system but also a general failure diagnosis system.

It is also planned that the safety subsystem may incorporate the necessary code for recovery from certain failures. Petri nets have been used in many cases for error recovery (e.g. (Zhou and DiCesare, 1989)) but a satisfactory method has yet to be devised.

9.10 Chapter Summary

This chapter summarises the work presented in the body of the thesis. It highlights the contribution made by the work and makes a number of conclusions based on the experiences gained whilst carrying out the work. A summary of the areas of contribution is as follows.

- The extension of a Petri net formalism to introduce true modularity and introducing a more formal definition of the nets. This formalism has been called Structured Petri nets.
- The development of a more formal design method for sequence controllers, which is currently lacking in the Petri net literature. Most synthesis methods currently involve an ad hoc approach to system design.

- A new appraisal of Petri net implementation methods, with particular attention being paid to ladder logic programs. Petri nets have until now been used for ladder logic code generation, but not with a eye on the accuracy and behaviour of the implementation.
- A new implementation of a Petri net on a relational database, using SQL queries to carry out transition firings. This potentially leads to many applications for the analysis of Petri nets, the automatic generation of control code, and manufacturing CASE tools.
- A new method of enumerating the possible states in a system which may have application in the fault monitoring scheme.
- A new taxonomy of manufacturing failures, which highlights the chain of failures that may exist.
- A new approach to fault monitoring based on Structured Petri nets, which allows the distinction between hardware and software faults and even potentially traces a software fault to a particular module.

The chapter also raises a number of issues for further examination in order to achieve the objectives set out in Chapter 1, and highlights a number of possible extensions to the work. The most important of these are as follows:

- Automating the generation of the Petri net and then the control code.
- Further understanding of the structured Petri net to other classes of nets, with some attempt to solve different classes of problem.
- Developing the net structure to incorporate planning and scheduling systems.
- Developing the safety subsystem to incorporate fault monitoring and diagnosis.

The objectives set out in Chapter 1 are an ideal that many authors aspire to. Although, along with others, this work is yet to achieve those objectives, it has certainly taken a number of significant steps that make the achievement of those objectives more of a reality. With each of these steps comes a new area of work which, it is hoped, will be pursued to completion.

9.11 References

Adlemo, A. and Andréasson, S., 1995, "A dependability taxonomy for flexible manufacturing systems." *International Journal of Computer Integrated Manufacturing*, **8**, pp. 189-196.

Hanisch, H.-M. and Rausch, M., 1995, "Synthesis of supervisory controllers based on a novel representation of Condition/Event Systems." In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, British Columbia, Canada.

Holloway, L. E. and Krogh, B. H., 1990, "Synthesis of feedback control logic for a class of controlled Petri nets". *IEEE Transactions on Automatic Control,* **35**, pp. 514-523.

Jagdev, H., Browne, J., and Jordan, P.,, 1995 "Verification and validation issues in manufacturing models." *Computers in Industry*, **25**, pp. 331-353.

Järvinen, J. and Karwowski, W., 1995, "Analysis of self-reported accidents attributed to advanced manufacturing systems." *International Journal of Human Factors in Manufacturing,* **5**, pp. 251-266.

Krogh, B. H., 1987, "Controlled Petri nets and maximally permissive feedback logic." In *Proc. 25th Annual Allerton Conference.* University of Illinois, USA, pp. 317-326.

Krogh, B. H. and Sreenivas, R. S., 1987, "Essentially decision free Petri nets for real-time resource allocation." In *Proc. IEEE International Conference on Robotics and Automation*, Raleigh, NC, USA, pp. 1005-1011.

Liu, T. S., and Chiou, S. B., 1997, "The application of Petri nets to failure analysis." *Reliability Engineering and System Safety*, **57** pp. 129-142

Naylor, A. W. and Volz, R. A., 1987, "Design of integrated manufacturing control software." *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-17**, pp. 881-897.

Proth, J. M., Wang, L. and Xie, X., 1997, "A class of Petri nets for manufacturing system integration." *IEEE Transaction on Robotics and Automation*, **13**, pp. 317-326.

Stanton, M. J. and Arnold, W. F., 1997, "Extension of structured Petri nets for the control of a conveyor system." In *Proc. Factory 2000*, Cambridge, England.

Yang, S. K. and Liu, T. S., 1997, "Failure analysis for an airbag inflator by Petri nets." *Quality and Reliability International*, **13**, pp. 139-151

Zhou, M., and DiCesare, F., 1989, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems." *IEEE Transactions on Systems, Man, and Cybernetics*, **19**, pp. 963-973.

Appendix 1

Petri Nets for the Raw Materials Station

The Petri nets shown in Appendix 1 are those that were originally developed for the raw materials station. They include a control net, much simpler than that given in Chapter 6, but with less functionality, and containing a larger number of places. There was no method to the design process, and little consideration of the other functions that the system might be asked to perform. It did however provide a good platform on which to build a more structured design process.

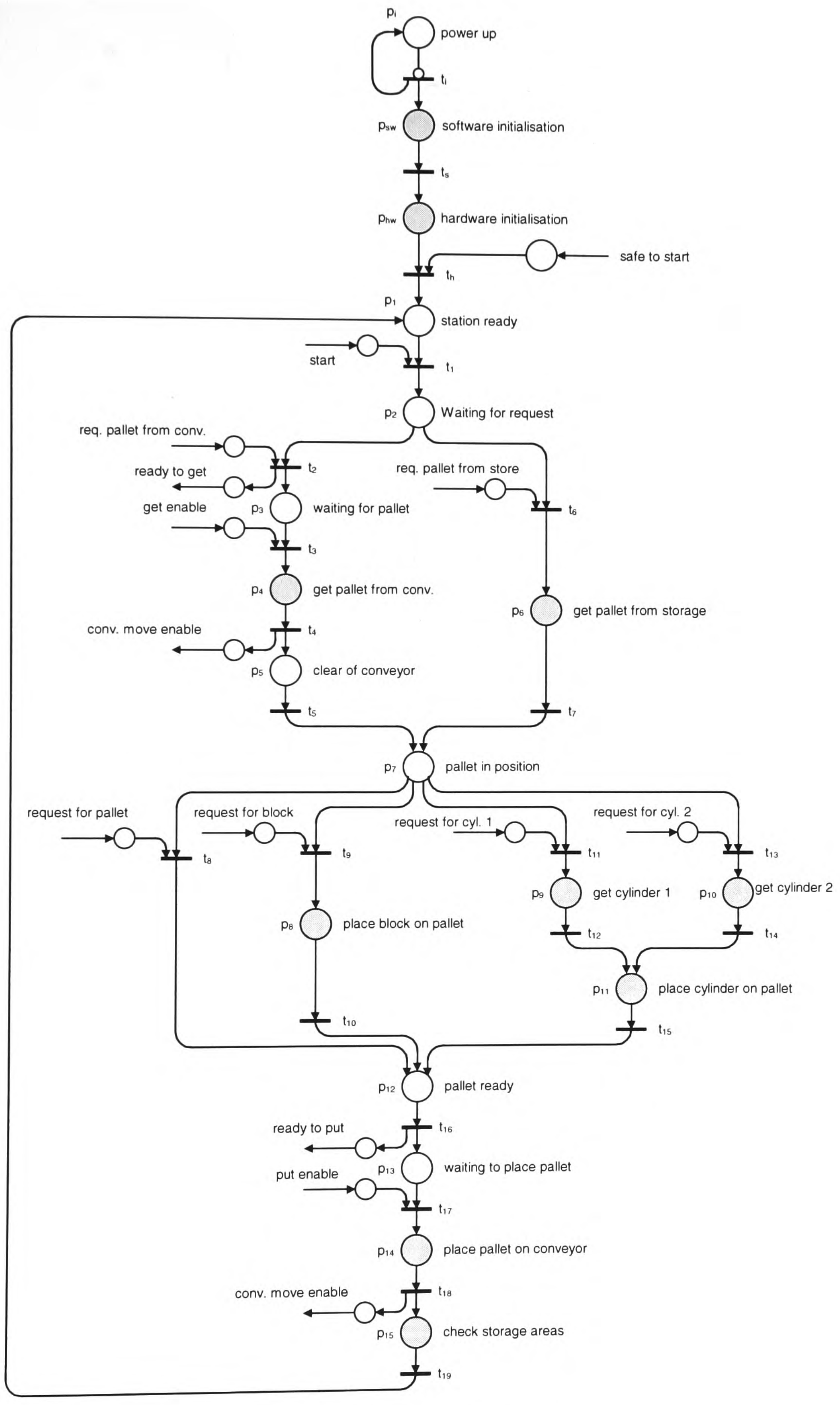

Figure A1.1 Control net

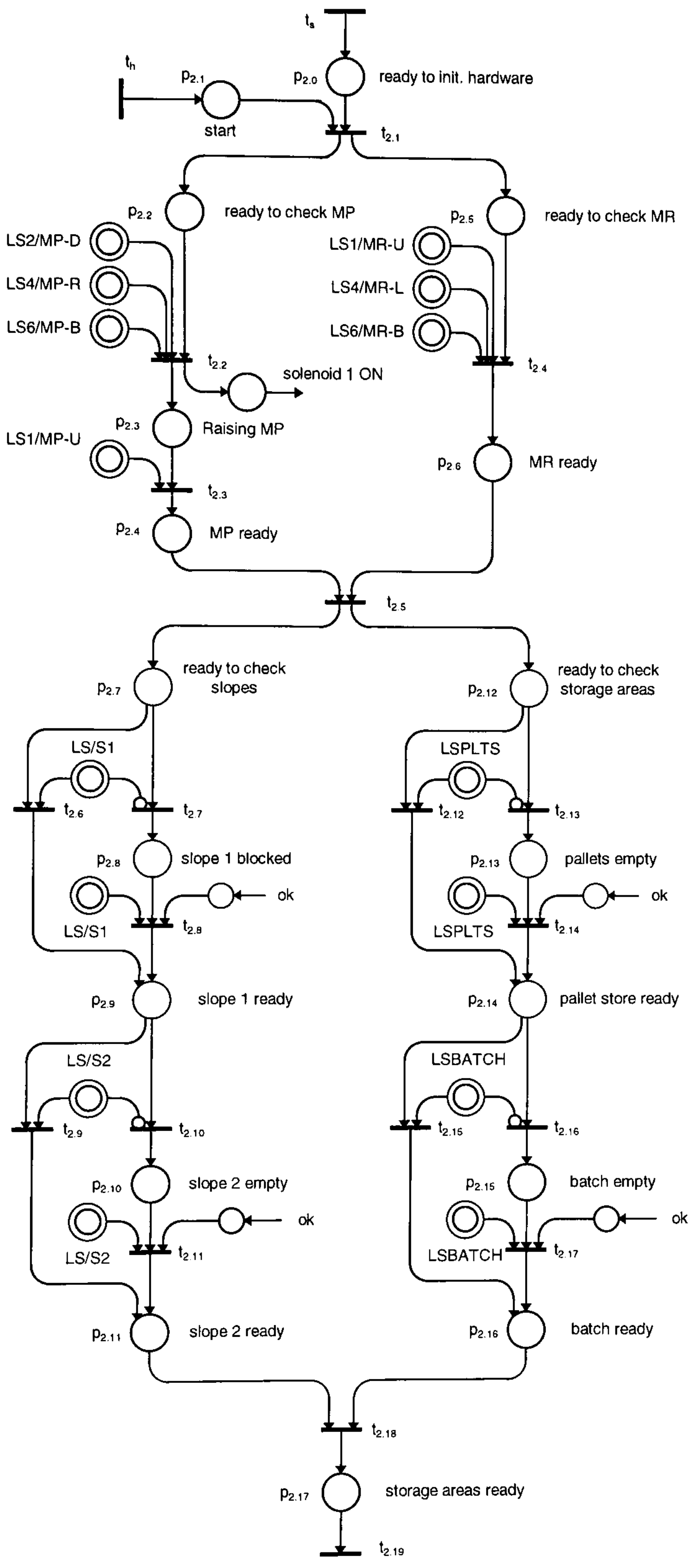

Figure A1.2 Hardware initialisation

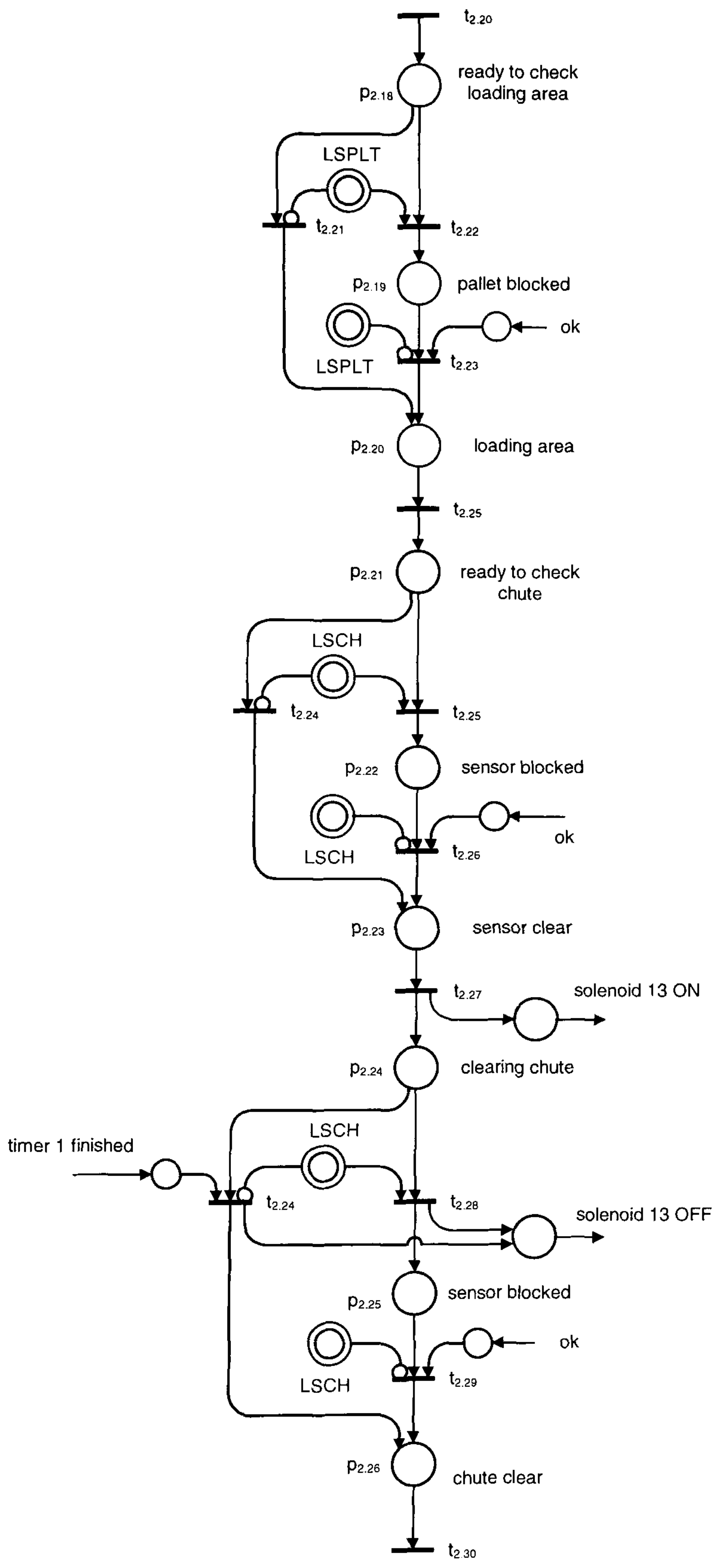

Figure A1.3 Hardware initialisation continued...

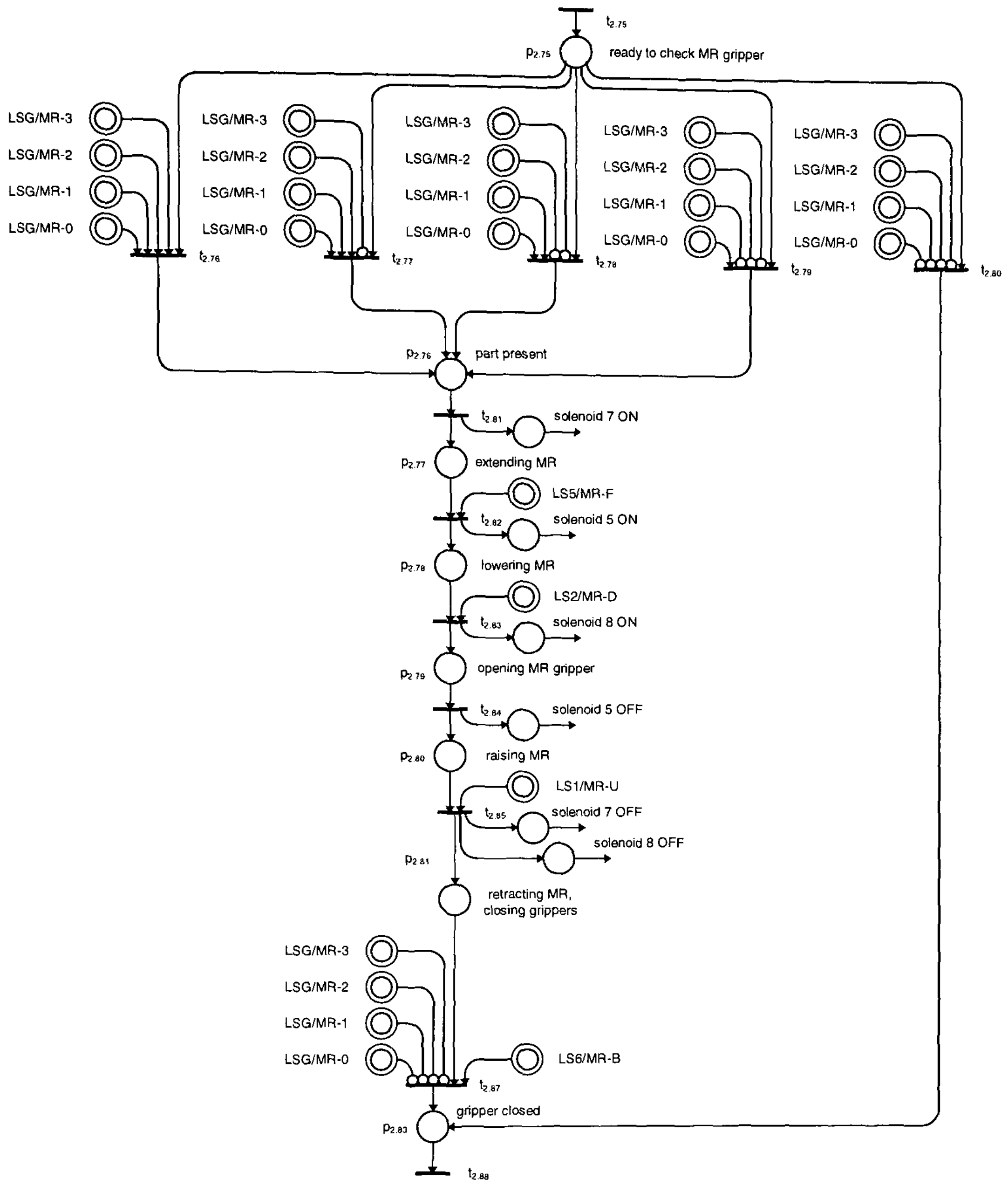

Figure A1.4 Hardware initialisation continued

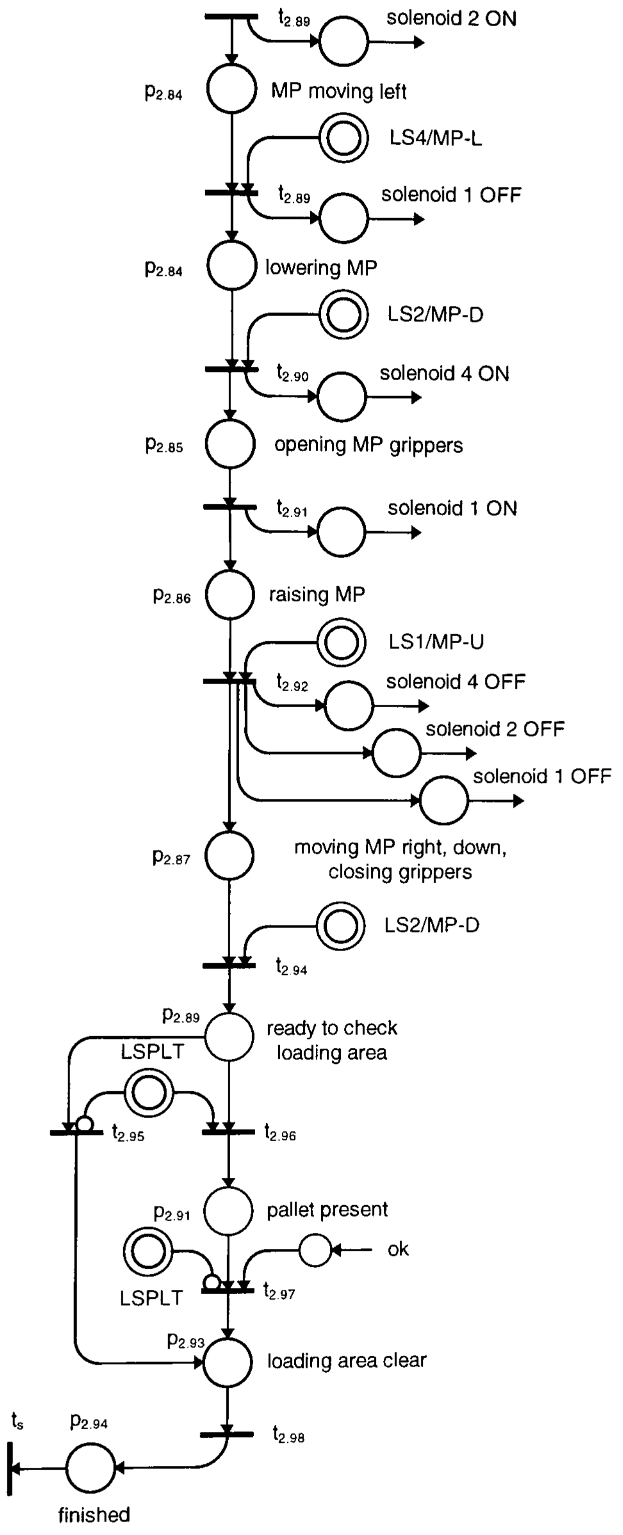

Figure A1.5 Hardware initialisation continued

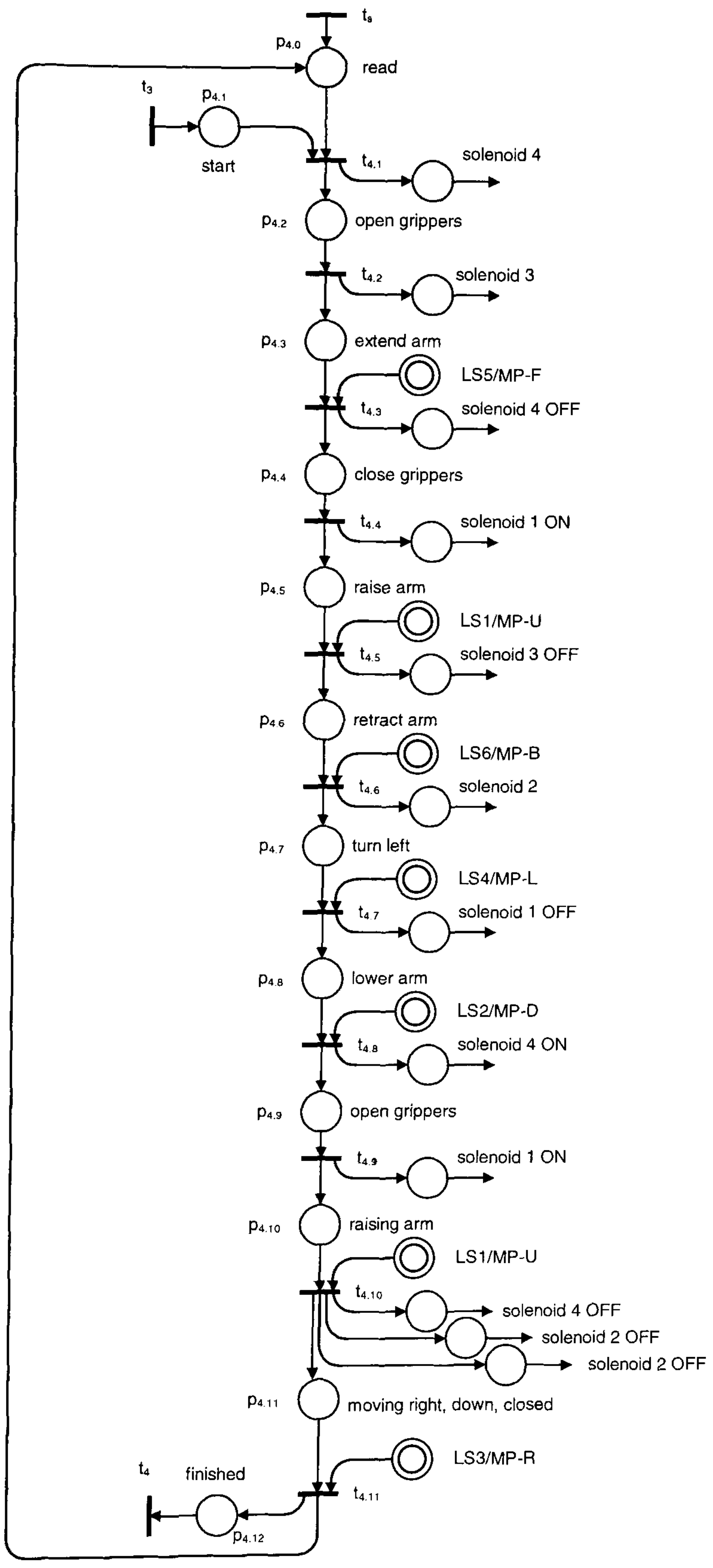

Figure A1.6 Get pallet from conveyor

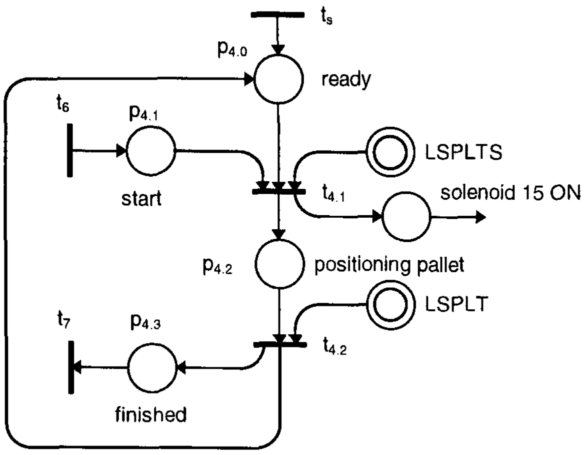

Figure A1.7 Get pallet form storage

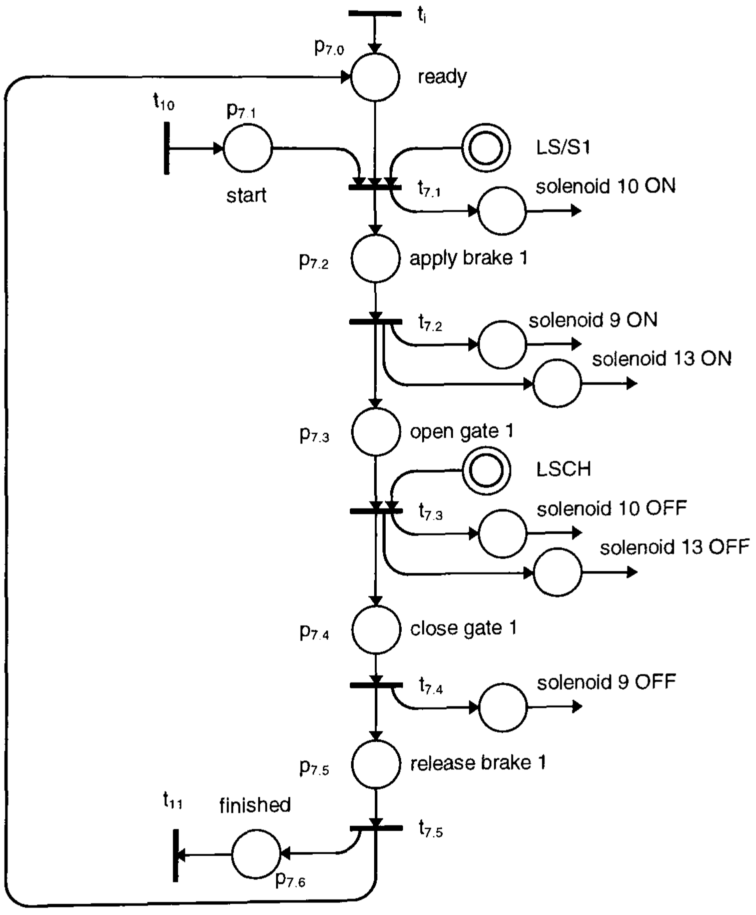

Figure A1.8 Get cylinder 1

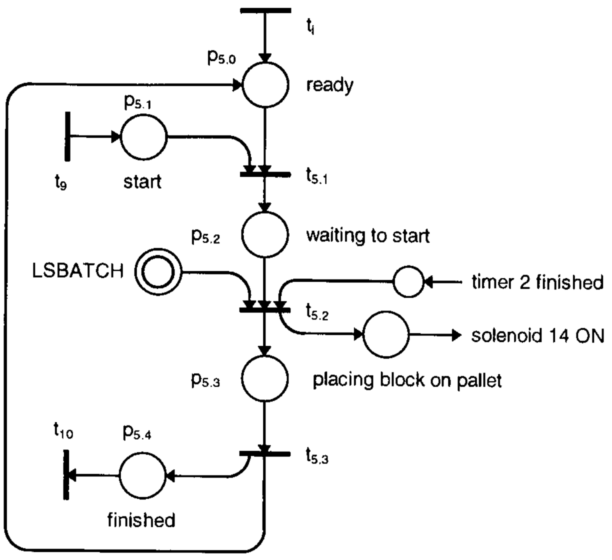

Figure A1.9 Place block on pallet

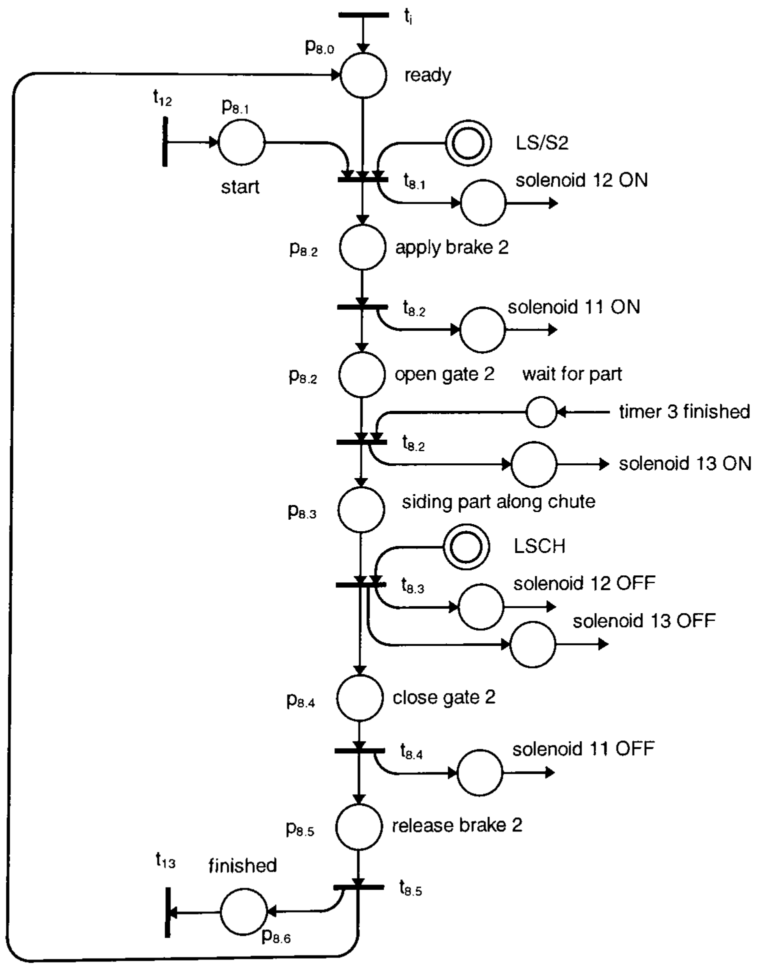

Figure A1.10 Get cylinder 2

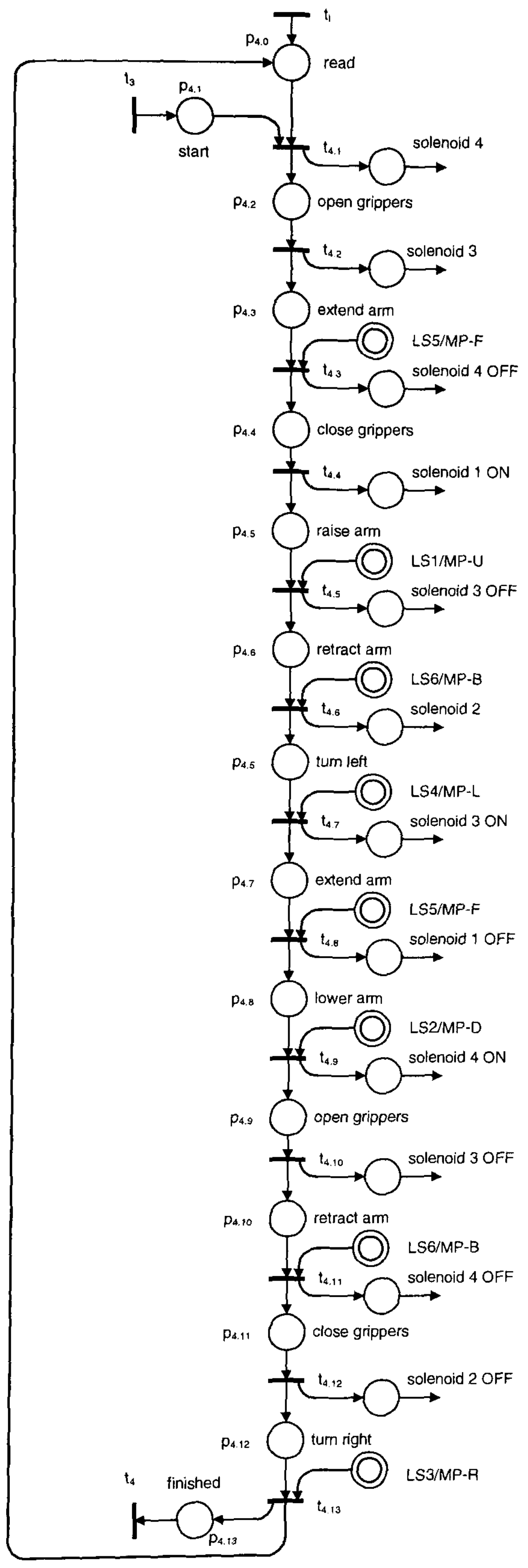

Figure A1.11 Place cylinder on pallet

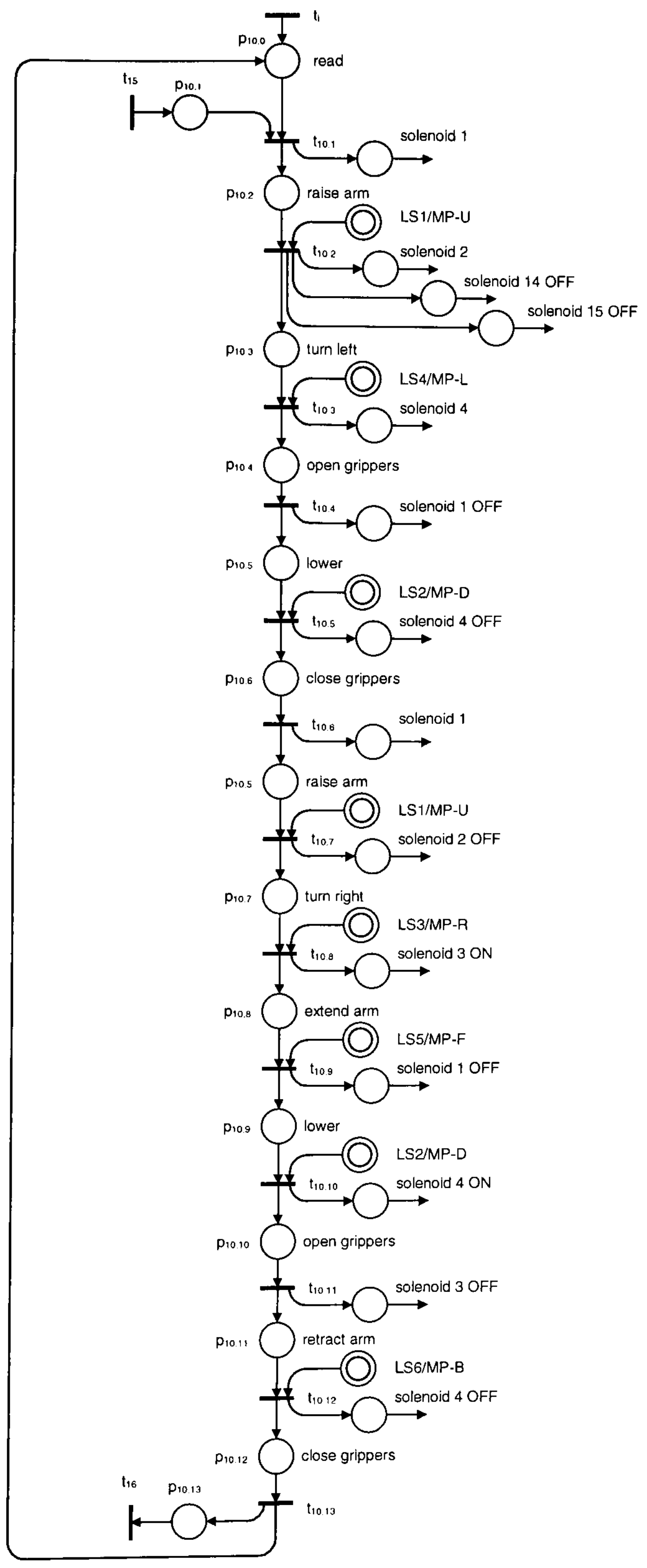

Figure A1.12 Place pallet on conveyor

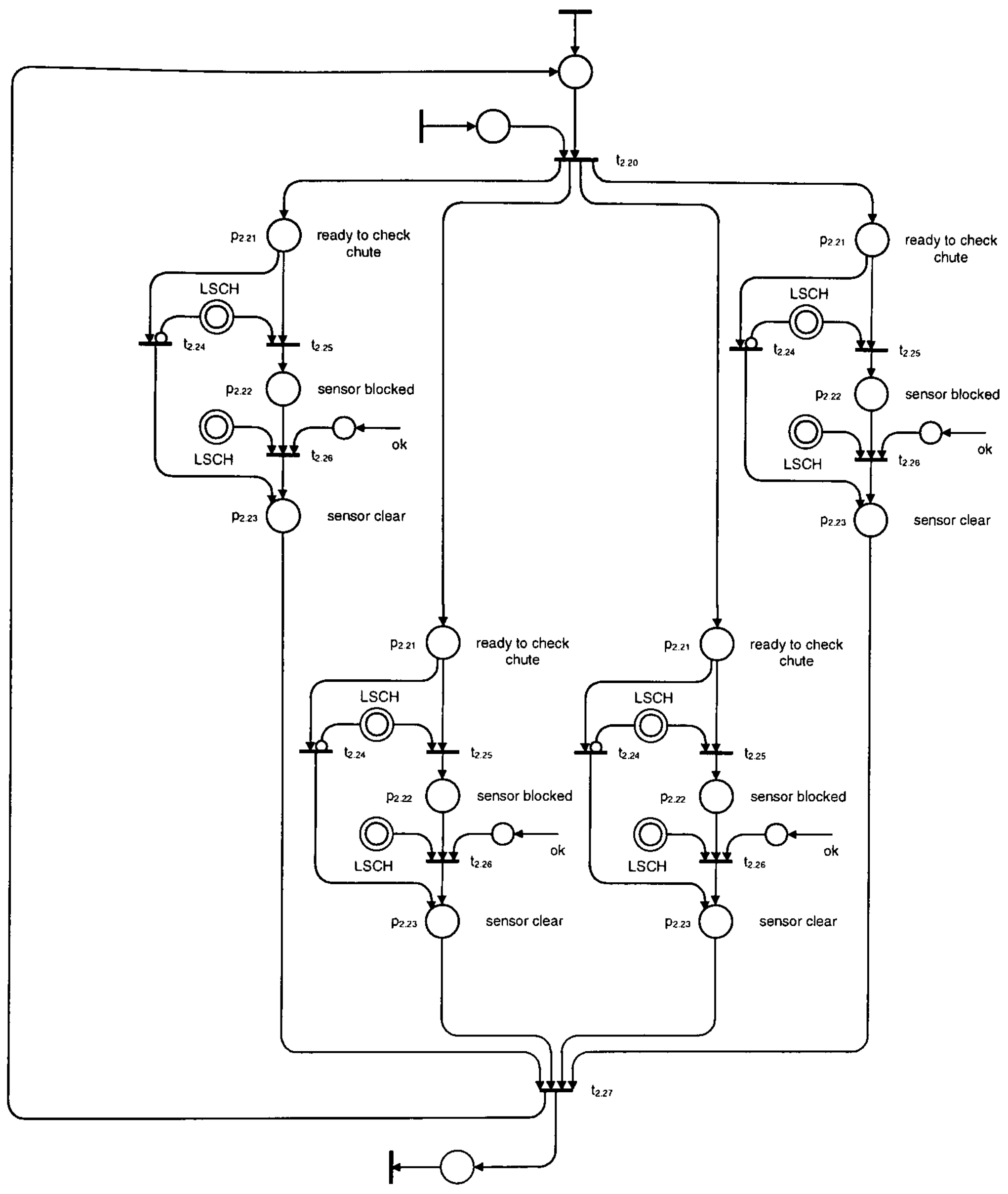

Figure A1.13 Check storage areas

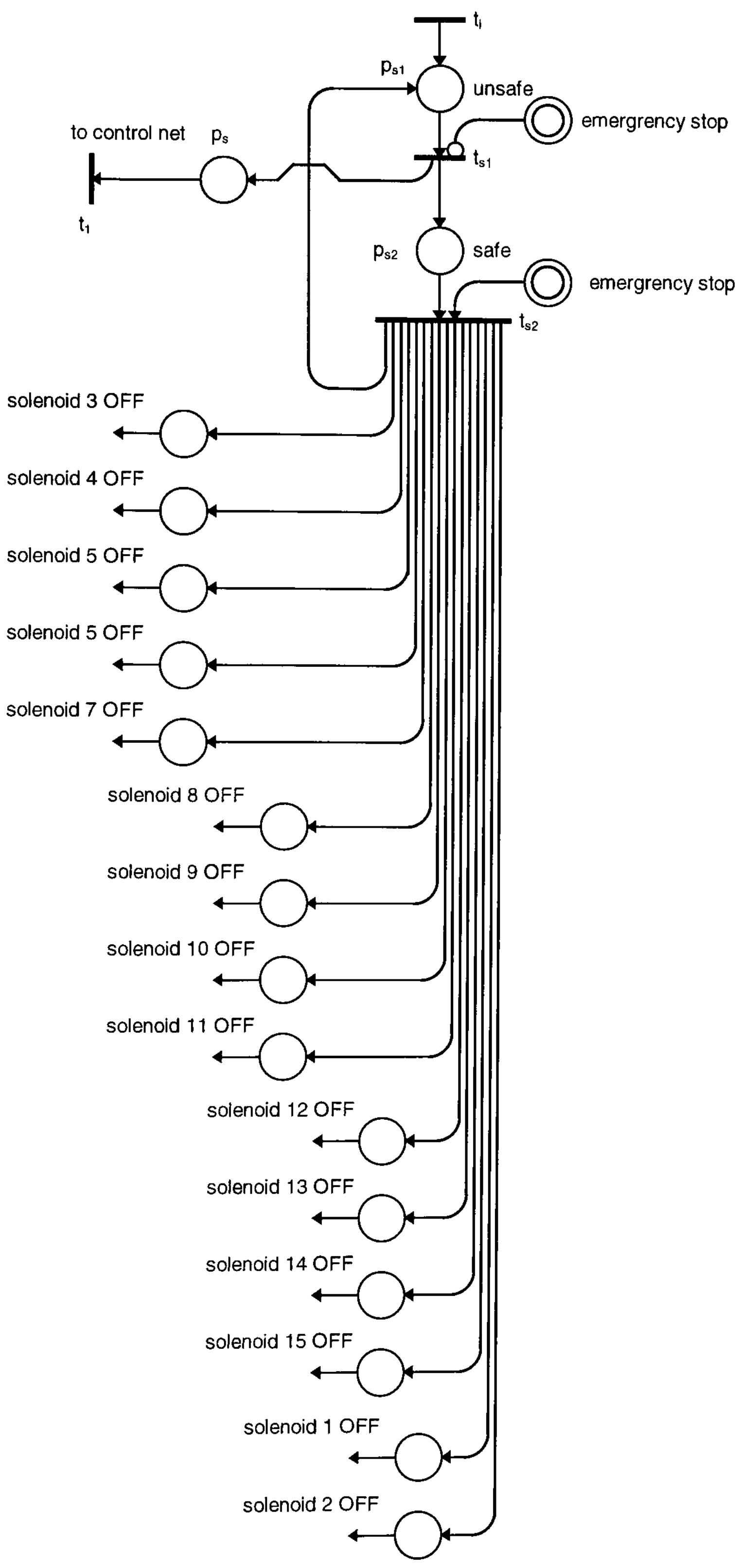

Figure A1.14 Safety net

Appendix 2

Updated Net for the Pallet Manipulator

The nets presented in Appendix 1 are those that have been developed for the raw materials station described in Chapter 6. Appendix 2 describes the Petri net module for the pallet manipulator and the output nets designed to work with it.

Gets a pallet from the conveyor or puts a pallet to the conveyor. Consists of 4 single acting pnuematic cylinders each working in a different degree of freedom. Thus the manipulators two tasks consist of the actions described in .

Task	**Sequence**
Get_Pallet	Open_Grip → Move_Out → Grip_Pallet → Raise_Pallet → Move_In → Swing_Left → Lower_Pallet → Release_Pallet → Move_Up → Swing_Right → (Move_Down//Close_Grip)
Put_Pallet	Move_Up → (Swing_Left//Open_Grip) → Move_Down → Grip_Pallet → Raise_Pallet → Swing_Right → Move_Out → Lower_Pallet → Release_Pallet → Move_In → Close_Grip

a // *b* indicates that tasks *a* and *b* occur concurrently

a → *b* represents a transition from task *a* to task *b* and indicates that task *a* occurs before task *b*

Table A2.1 Task sequences for the pallet manipulator

These task sequences have been used to create the Petri net of Figure A2.1. Also included here are subnets created for each pneumatic cylinder and output nets for each of the solenoids used to activate them.

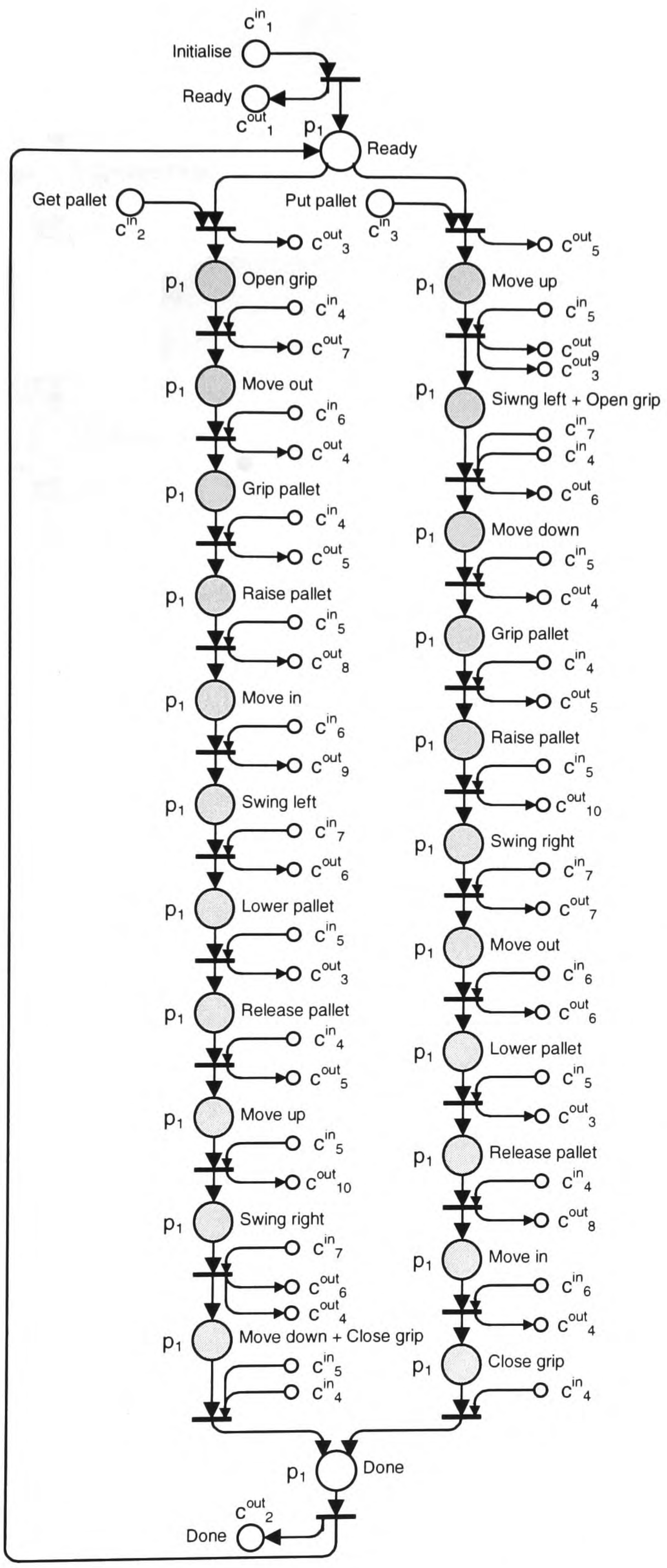

Figure A2.1 Subnet showing sequence for Pallet Manipulator

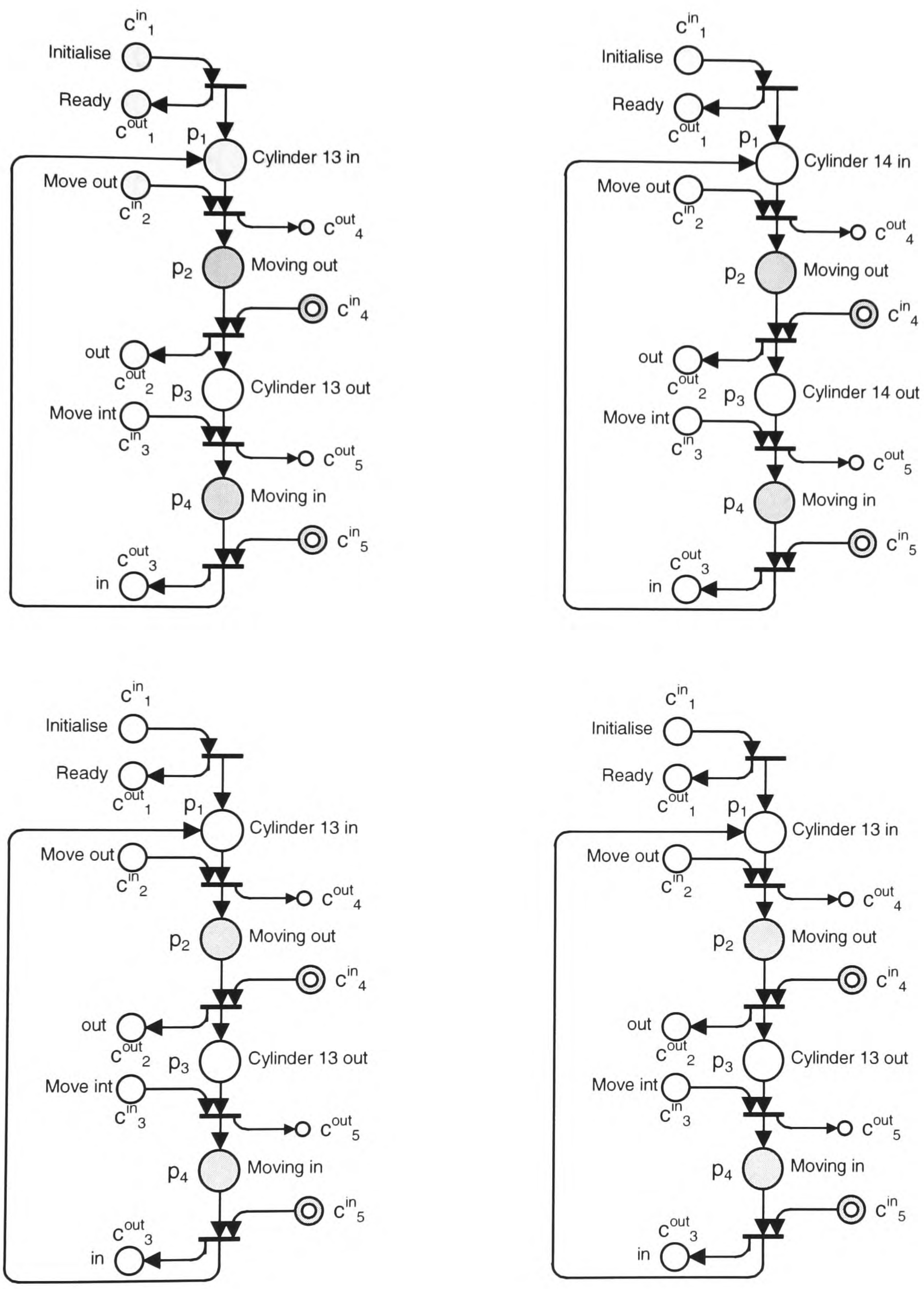

Figure A2.2 Subnets for pneumatic cylinders making up the pallet manipulator

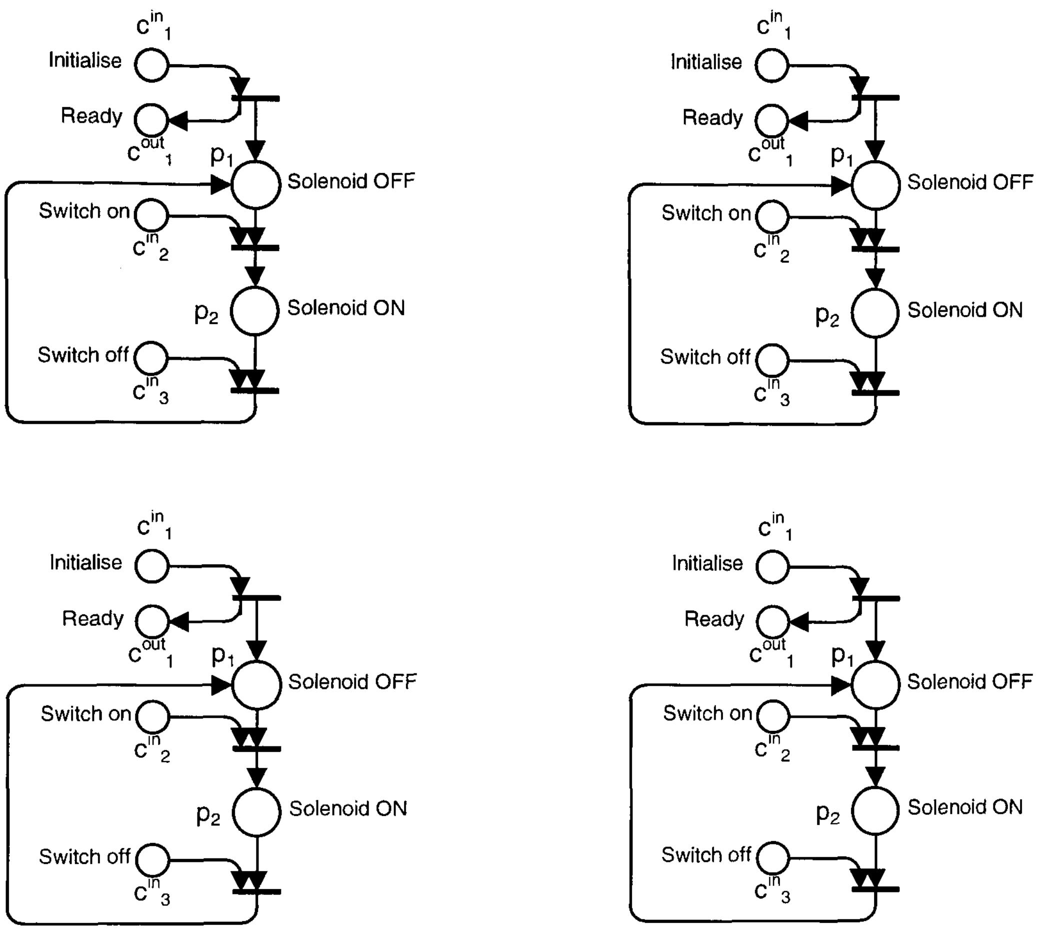

Figure A2.3 Output nets for the pallet manipulator

Appendix 3
Documentation for the Manufacturing Cell

The documentation presented here contains details of the manufacturing cell described in Chapter 6, and its operation. It includes the Petri nets used to describe the control software for the PLC, the robot and the CNC machine tools. It is not in precisely the same form as that presented with the working version of the software as the ladder logic program and some other elements have not been included here.

The nets were designed using the design approach described in Chapter 2, and the experience gained during this project were in part the motivation behind development of the new, more formal design approach.

System Description

System Layout

The layout of the Flexible Manufacturing System is shown in Figure A3.1. A robot is situated on a sliding track, which enables it to feed both a CNC mill and a CNC lathe from the two input buffers placed between the CNC machines. When parts have been processed they can be assembled in the assembly buffer situated opposite the input buffers.

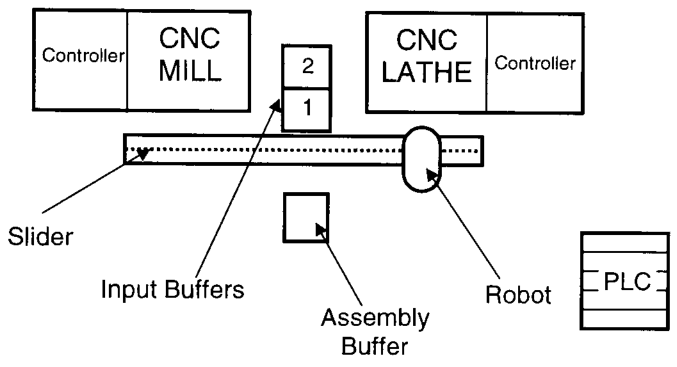

Figure A3.1 Layout of Flexible Manufacturing System

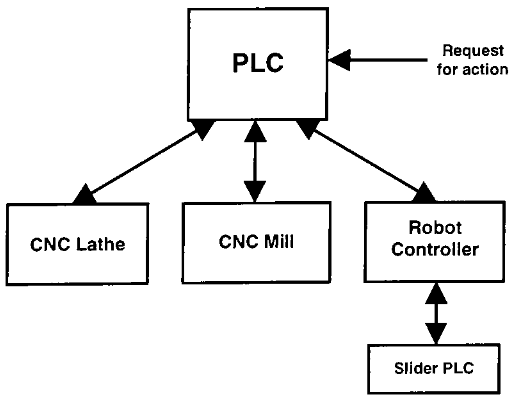

Figure A3.2 Cell controller hierarchy

Controller Hierarchy

The hierarchy of the controllers in the cell is shown in Figure A3.2. The PLC synchronises the machines within the cell, whose actions are described by their own controllers. The robot controller must also synchronise its actions with those of the slide controller. The cell is instructed to carry out tasks via a Personal Computer, which is linked to the PLC.

System Operation

Powering up the system

There are two steps in the powering up procedure:

1. Power up the main control system using the switch situated on the wall box.

2. Power up the mill, lathe, and robot using their respective power switches.

Running the Software

PLC Software

Once the system has been powered up, the operator should ensure that the PLC program is running. This can be determined by checking that the LED's on the PLC are lit and the YELLOW run light is on. If the program is not running it has not been downloaded or the PLC has stopped. In both these cases the operator should refer to the MODSOFT instruction manual for instructions on how to download and start the PLC.

Robot Software

The robot software is run by pressing the GREEN start button on the front of the robot controller. The program should run from the EPROM situated inside the door on the side of the controller. Before starting the program the operator should ensure that the correct EPROM has been placed in the controller and that the switches in the controller are in the correct position (refer to the Robot Manual for the correct settings).

VUNIQ Software

The operator should switch on the attached PC and run the VUNIQ software accompanying the system. It is Important that this is done before starting the system initialisation because this software is used to prompt the operator during the initialisation procedure and provide any necessary error messages.

Initialisation

When the system is powered up, the PC software will indicate that the system is ready to start initialisation. In order to increase the flexibility of the system, four different initialisation options are provided, these correspond to the machining options described in see section 0.

1. *Full Initialisation* - Initialises the mill, lathe and robot, thus allowing all of the machining options to be carried out.

2. *Initialisation for Cylinder Production* - Initialises only the robot and lathe, and therefore only allows the cylinder production or assembly tasks to be carried out.

3. *Initialisation for Block Production* - Initialises only the robot and mill, and therefore only allows the block production or assembly tasks to be carried out.

4. *Initialisation for Assembly* - Initialises the robot only. This allows only the assembly task to be carried out.

Having different options for initialisation allows the system to still be useable (although not for full production) when one or more of the system elements is faulty.

However if the robot is not functioning then none of the systems tasks can be completed, except by manual intervention, in which case initialisation is also carried out manually. The initialisation procedure for each machine is now described in detail:

Robot Initialisation

Before the robot can start automatic initialisation the operator is prompted to ensure it is clear of any machinery or object with which it might collide during nesting (see Robot Manual for details). If it is unable to nest freely it must be manually jogged to a safe position using the Teach Pendant. The system will wait for a signal from the operator indicating that the robot is clear to nest before beginning its automatic initialisation sequence.

Mill and Lathe Initialisation

Both the mill and the lathe must be placed in their reference positions, and their chip guards and chucks must be in the OPEN position before they are initialised fully. The PC software will prompt the operator to ensure that this has been carried out and wait for a signal from the operator on completion of these settings.

Running the system

Once the system is initialised, it will wait for a request from the operator to indicate that a particular operation is required. It should be noted that if an operation is selected for which the required machines have not been initialised, then the cell will be unable to carry out that operation correctly. Once a particular cycle has been completed, the system will wait for the operator to input another instruction, telling it to perform another cycle.

Machining Operations

The system is able to carry out one of four machining options, which are described as follows:

1. *Full Production Cycle* - A block and a cylinder are taken from the input buffers, machined concurrently and then assembled at the output buffer.

2. *Cylinder Production Cycle* - A cylinder is taken from the input buffer, machined and then placed in the output buffer.

3. *Block Production Cycle* - A block is taken from the input buffer, machined and then placed in the output buffer.

4. *Assembly Cycle* - A machined block and cylinder are taken from the input buffers and assembled at the output buffer.

Each of these cycles is described in the following sections in more detail.

Full Production Cycle

If the full production cycle is selected, the robot will first take the block from the input buffer (buffer 1) and place it in the mill. The mill will start processing the block as soon as the robot has moved clear. The robot will then proceed to the cylinder input buffer (buffer 2) and move the cylinder to the lathe. The lathe will also start processing as soon as the robot has moved clear. The robot will wait until the block has finished being processed, at which point it will collect it from the mill and place it in the assembly buffer. Once the cylinder processing has completed then the robot will the proceed to collect the finished cylinder from the lathe and place it in the appropriate position in the assembly buffer.

Block Production Cycle

For this cycle the robot simply takes an unmachined block from buffer 1, and places it in the mill. The mill starts processing the block as soon as the robot has moved clear. Once the block is completed the robot collects it from the mill and places it in the output buffer.

Cylinder Production Cycle

This cycle is similar to the block production cycle except that a cylinder is collected from buffer 2. The operator should ensure that a suitable receptacle is place in the output buffer to receive the completed cylinder (e.g. a machined block).

Assembly Cycle

In order to successfully perform the assembly cycle, the operator should ensure that components suitable for assembly are placed in the input buffers. The robot places the components in the assembly buffer starting with that from input buffer 1, followed by that from buffer 2.

Software Description

Lathe Cycle

The cycle for the lathe is described in the following paragraph with reference to the Petri net in Figure A3.3.

Lathe initialisation is started by the main control software. Once initialised, the lathe waits for a part to be delivered, and when this is done the chuck is closed. The lathe then awaits a signal form the main control net indicating that the robot has moved clear of the door. Once the robot has moved clear, the lathe door is then closed and the machining process is started. The end of the machining process is signalled by the door opening.

WARNING - *if the machining cycle is interrupted and the door opened by an operator, the control software will think that the cycle has finished normally and will instruct the robot to fetch the part. In order to prevent this from happening, both the robot controller AND the PLC should be stopped if there is any interruption to the machining process.*

When the door has opened, lathe waits until it is told to release the part (i.e. when the robot has gripped the part) and then opens the chuck. Once the chuck is open, and the part has been removed the lathe cycle is complete.

Mill Cycle

This is similar to that described above for the lathe. The Petri net description of this cycle is shown in Figure A3.4.

WARNING - *if the machining cycle is interrupted and the door opened by an operator, the control software will think that the cycle has finished normally and will instruct the robot to fetch the part. In order to prevent this from happening, both the robot controller AND the PLC should be stopped if there is any interruption to the machining process.*

Robot Cycle

The robot cycle is described by the Petri net in Figure A3.5. There are a number of tasks that the robot must perform and each of these is shown. Combinations of these tasks are used to carry out different actions within the system cycles.

Petri net descriptions of software

Lathe Control net

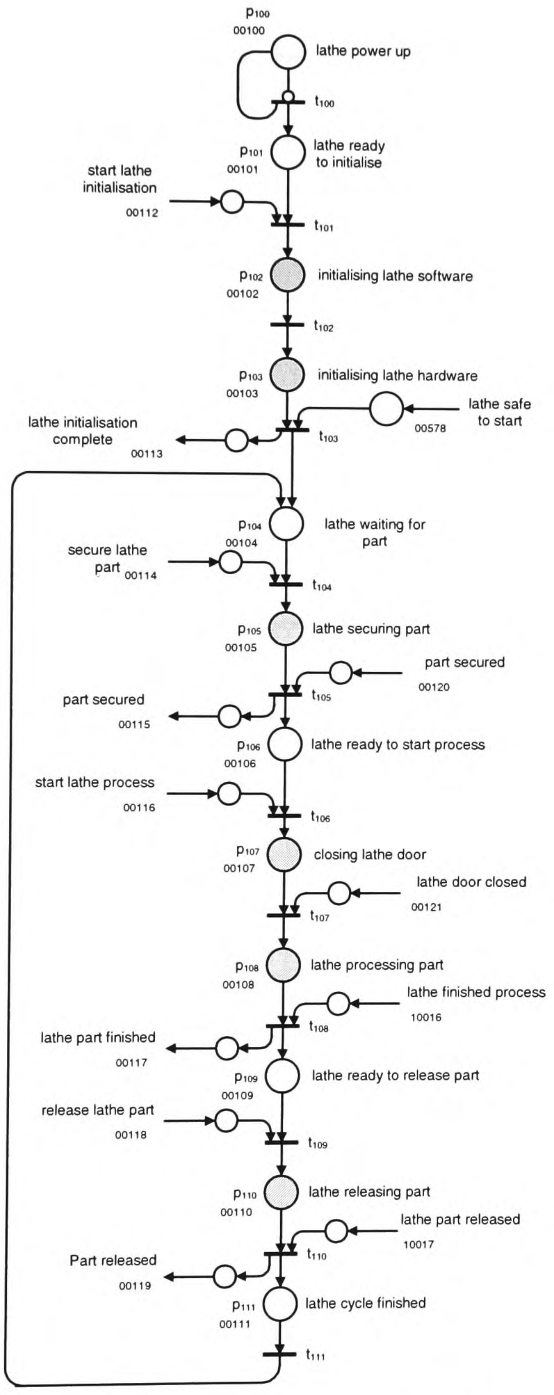

Figure A3.3 Lathe Control net

Mill Control net

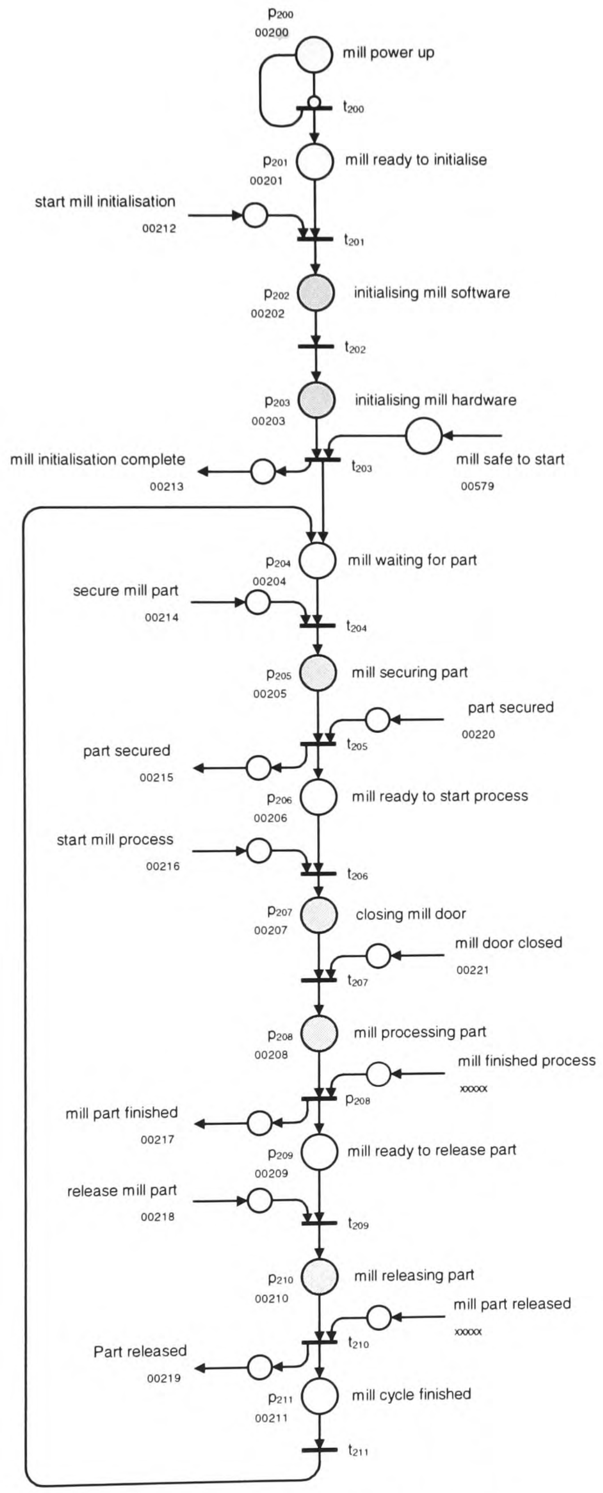

Figure A3.4 Mill Control net

Robot Control net

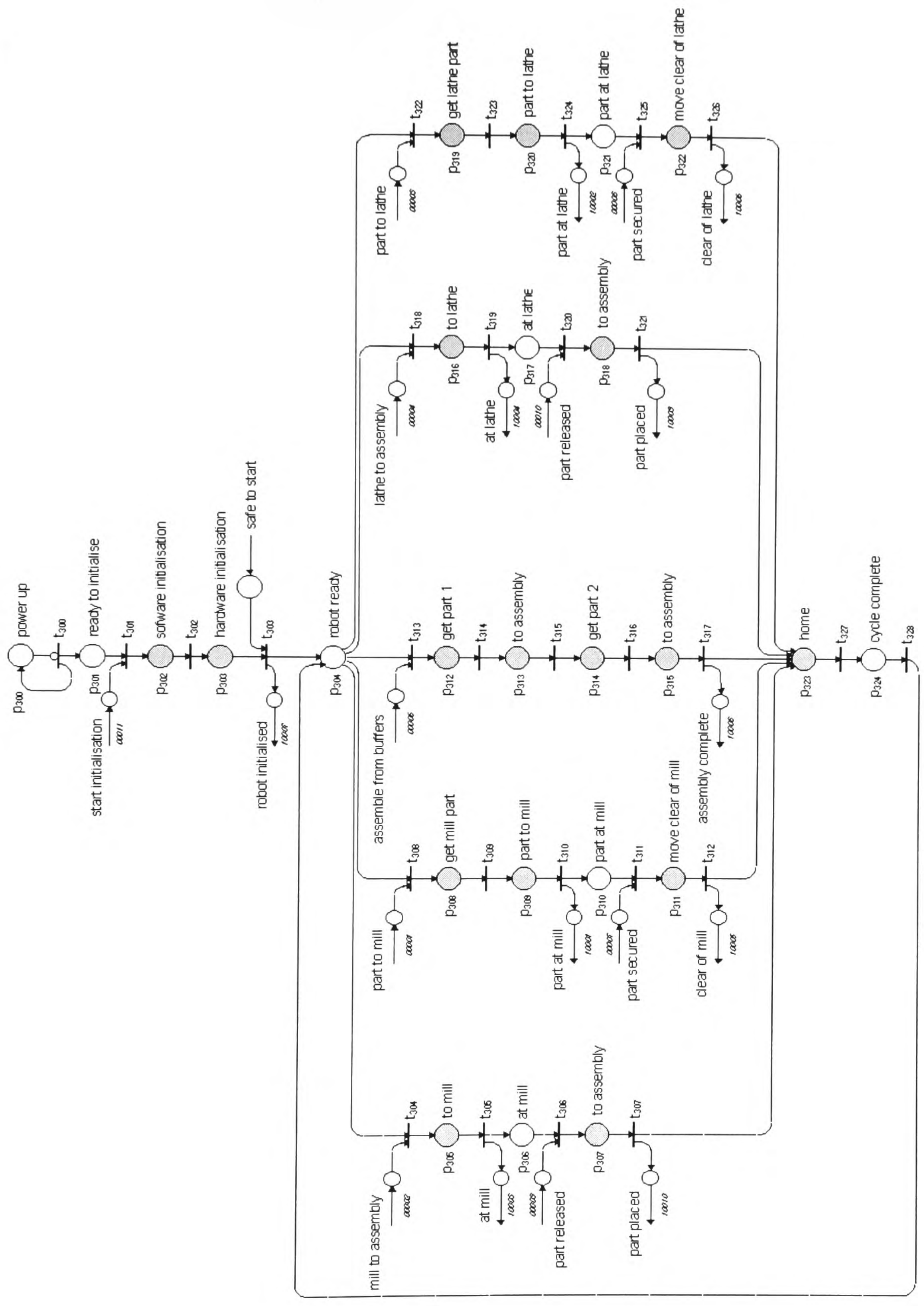

Figure A3.5 Robot Control net

Cell control net

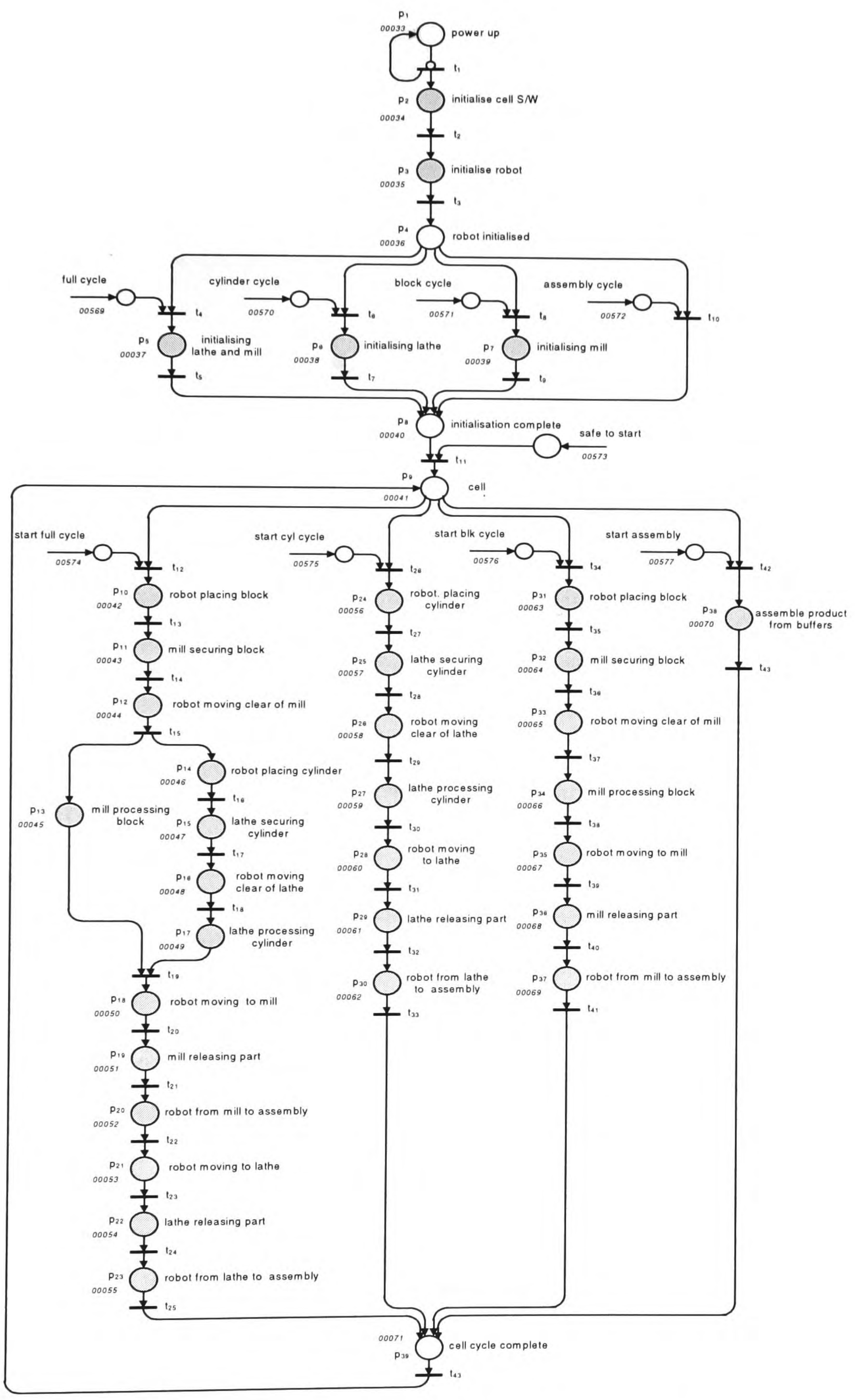
p1 00033 power up
t1
p2 initialise cell S/W 00034
t2
p3 initialise robot 00035
t3
p4 00036 robot initialised
full cycle 00569
cylinder cycle 00570
block cycle 00571
assembly cycle 00572
t4
t6
t8
t10
p5 00037 initialising lathe and mill
p6 00038 initialising lathe
p7 00039 initialising mill
t5
t7
t9
p8 00040 initialisation complete
safe to start 00573
t11
p9 00041 cell
start full cycle 00574
start cyl cycle 00575
start blk cycle 00576
start assembly 00577
t12
t26
t34
t42
p10 00042 robot placing block
t13
p11 00043 mill securing block
t14
p12 00044 robot moving clear of mill
t15
p13 00045 mill processing block
p14 00046 robot placing cylinder
t16
p15 00047 lathe securing cylinder
t17
p16 00048 robot moving clear of lathe
t18
p17 00049 lathe processing cylinder
t19
p18 00050 robot moving to mill
t20
p19 00051 mill releasing part
t21
p20 00052 robot from mill to assembly
t22
p21 00053 robot moving to lathe
t23
p22 00054 lathe releasing part
t24
p23 00055 robot from lathe to assembly
t25
p24 00056 robot. placing cylinder
t27
p25 00057 lathe securing cylinder
t28
p26 00058 robot moving clear of lathe
t29
p27 00059 lathe processing cylinder
t30
p28 00060 robot moving to lathe
t31
p29 00061 lathe releasing part
t32
p30 00062 robot from lathe to assembly
t33
p31 00063 robot placing block
t35
p32 00064 mill securing block
t36
p33 00065 robot moving clear of mill
t37
p34 00066 mill processing block
t38
p35 00067 robot moving to mill
t39
p36 00068 mill releasing part
t40
p37 00069 robot from mill to assembly
t41
p38 00070 assemble product from buffers
t43
00071 p39 cell cycle complete
t43

Figure A3.6 Cell control net for the FMS

PLC I/O LISTS

Outputs

Address	Description	
00001	TB-ROB/33 (IN 0)	START “BLOCK FROM BUFFER TO MILL”
00002	TB-ROB/17 (IN 1)	START “BLOCK FROM MILL TO ASSEMBLY”
00003	TB-ROB/32 (IN 2)	START “CYL FROM BUFFER TO LATHE”
00004	TB-ROB/16 (IN 3)	START “CYL FROM LATHE TO ASSEMBLY”
00005	TB-ROB/31 (IN 4)	START “ASSEMBLY FROM BUFFERS”
00006	TB-ROB/15 (IN 5)	
00007	TB-ROB/30 (IN 6)	MILL PART SECURED
00008	TB-ROB/29 (IN 8)	LATHE PART SECURED
00009	TB-ROB/13 (IN 9)	MILL PART RELEASED
00010	TB-ROB/28 (IN 10)	LATHE PART RELEASED
00011	TB-ROB/12 (IN 11)	START ROBOT INITIALISATION
00012	TB-ROB/27 (IN 12)	CLEAR TO NST
00013	TB-ROB/11 (IN 13)	
00014	TB-ROB/26 (IN 14)	
00015	TB-ROB/10 (IN 15)	
00016	OPEN CHUCK (LATHE)	
00017	CLOSE CHUCK (LATHE)	
00018	OPEN TAILSTOCK (LATHE)	
00019	CLOSE TAILSTOCK (LATHE)	
00020	START CYCLE (LATHE)	
00021	CLOSE DOOR/OPEN DOOR (LATHE)	
00022	READ IN PROGRAM (LATHE)	
00023	ALARM (LATHE)	
00024	REF/AUTO FUNCTION (LATHE)	
00025		
00026	OPEN VICE (MILL)	
00027	CLOSE VICE (MILL)	
00028	CYCLE START (MILL)	
00029	CLOSE DOOR/OPEN DOOR (MILL)	
00030	READ IN PROGRAM (MILL)	
00031	ALARM (MILL)	
00032	REF/AUTO FUNCTION (MILL)	

Inputs

Address	Description	
10001	TB-ROB/2 (OUT 0)	REQ MILL TO SECURE PART
10002	TB-ROB/3 (OUT 2)	REQ LATHE TO SECURE PART
10003	TB-ROB/4 (OUT 4)	REQ MILL TO RELEASE PART
10004	TB-ROB/5 (OUT 6)	REQ LATHE RELEASE PART
10005	TB-ROB/21 (OUT 7)	ROBOT CLEAR OF MILL
10006	TB-ROB/19 (OUT 1)	ROBOT CLEAR OF LATHE
10007	TB-ROB/8 (OUT 12)	ROBOT INITIALISED
10008	TB-ROB/9 (OUT 14)	ASSEMBLY COMPLETE
10009	TB-ROB/25 (OUT 15)	CYLINDER CYCLE COMPLETE
10010	TB-ROB/34 (OUT 3)	BLOCK CYCLE COMPLETE
10011	TB-ROB/20 (OUT 5)	REQ MOVE ROBOT CLEAR OF MACHINERY
10012	TB-ROB/24 (OUT 13)	
10013		
10014		
10015	CYCLE START STATUS (LATHE)	
10016	DOOR OPEN (LATHE)	
10017	CHUCK OPEN (LATHE)	
10018	TAILSTOCK OPEN (LATHE)	
10019	DRIVE STOPPED (LATHE)	
10020	ALARM (LATHE)	
10021		
10022	CYCLE START STATUS (MILL)	
10023	DOOR OPEN (MILL)	
10024	DRIVE STOPPED (MILL)	
10025	ALARM (MILL)	
10026		
10027		
10028		
10029		
10030	SENSOR 1 (OUTPUT BUFFER)	
10031	SENSOR 2 (LATHE INPUT BUFFER)	
10032	SENSOR 3 (MILL INPUT BUFFER)	

Appendix 4
Papers

Appendix 4 contains copies of all published papers taken from this research. They are summarised as follows:

Stanton, M. J., Arnold, W. F. and Buck, A. A., "Modelling and control of manufacturing systems using Petri nets." In *Proc. 13th IFAC World Congress*, San Francisco, USA, 1996, vol. J, pp. 324-329.

Stanton, M. J., and Arnold, W. W. "Implementation of Petri nets for the control of manufacturing systems." 5th UK Mechatronics Forum International Conference, University of Minho, Portugal, 1996, vol. 1, pp. 373-378.

Stanton, M. J. and Arnold, W. F. "Extension of structured Petri nets for the control of a conveyor system." In Proc. Factory 2000: IEE 5th International Conference, Cambridge, England, 1997, pp 472-478.

MODELLING AND CONTROL OF MANUFACTURING SYSTEMS USING PETRI NETS

M. J. Stanton, W. F. Arnold, A. A. Buck

Mechatronics Development Centre,
Faculty of Technology,
University of Wales College Newport,
Newport, Gwent, Wales, U.K.
e-mail: mstanton@gwent.ac.uk

Abstract: The Petri net graph is a powerful tool for the specification, control and analysis of discrete event systems. A well structured Petri net will provide a clear description of how a system functions. This paper describes the development of structured Petri nets for specification and design of control code for manufacturing systems, and discusses the implementation of such Petri nets on various controllers present in modern manufacturing systems. The usefulness of structured Petri nets for system modelling and analysis is also discussed with two practical examples where they have been used to design and implement control code on real systems.

Keywords: Petri-nets, programmable controllers, manufacturing systems, software specification, implementation.

1. INTRODUCTION

Petri nets are being used increasingly as tools for modelling and control in manufacturing. They have also proven to be ideal for specification, design and analysis of systems (Willson and Krogh, 1990; Ferrarini, 1992). Many authors propose extensions to Petri nets which provide increased modelling power but at the expense of clarity and simplicity of the analysis tasks (Jafari, 1990). If the extensions become too complex, it can result in the loss of the Petri nets properties that make their use attractive in the first place (David and Alla, 1992) and complicate the task of converting the Petri nets into control code.

Some authors have developed hardware controllers specifically designed to implement a form of Petri net (Murata, *et al.*, 1986; Dohi, *et al.*, 1992). However, if the Petri net is to be implemented on a variety of existing programmable controllers, some important factors should be considered:

i) There are a number of advantages in using the same formalism for specification, modelling and control of the system.

ii) If the Petri nets are to be converted into control code, the method of conversion should preserve the structure and properties of the Petri net.

To ensure that the control code will behave in accordance with the specification, the important properties of the Petri net must be clearly defined and must be preserved in the implementation. The Petri nets described in this paper are structured to simplify the task of control code design and to provide clear views of the system at different levels of abstraction. They also have other advantages to be described later on in the paper. The next section briefly describes the basic Petri net definition highlighting the important features, and is followed by a more detailed description of the structure imposed on them. This is followed by a discussion of the problems faced when converting the Petri nets into control code with reference to programmable logic controllers (PLCs) using ladder logic diagrams (LLDs) as their control language. Also discussed is the use of high level programming languages and other programming methods encountered in manufacturing systems. Finally some practical implementation examples will be described and the conclusions which have been drawn from these will be presented along with possibilities for future work.

2. PETRI NET DEFINITION

The Petri net used here has a similar definition to that of the ordinary Petri nets described in (Peterson, 1981) and (David and Alla, 1992). The use of inhibitor arcs has also been included and some authors refer to these as extended Petri nets.

An important aspect of the representation used here is the firing of transitions. Transitions fire instantaneously as soon as they are enabled and therefore unlike those in Timed Petri Nets or Stochastic Petri Nets (Murata, 1989) they have no time associated with them. Any time delays present in the net are associated with places rather than transitions. This distinction is important here because it is linked to the use of the nets as a diagnostic tool described later. It is also assumed that when a transition fires, all of it's output places receive their tokens simultaneously. When a transition has more than one output place this simultaneous generation of tokens represents a concurrent operation. This can pose some difficulties where the target controller is a sequential machine.

There are design issues which must be taken into account when using the Petri nets described here for the control of discrete manufacturing systems.

i) Care must be taken to avoid conflicts between transitions. This is done by ensuring that all concurrent operations originate from a transition rather than a place. The class of conflict-free nets is described in (David and Alla, 1992). However the nets used here do not fall into this class because places can have more than one output transition. The conflict is removed in this case by having another place attached to each of the output transitions acting as a guard by preventing more than one transition from being enabled at any time.

ii) The Petri net must be safe. This property (see (Peterson, 1981)) requires that each place in the Petri net may contain a maximum of one token. This is a structural property of the Petri net and is desirable as it again effects the usefulness of the Petri net as a diagnostic tool.

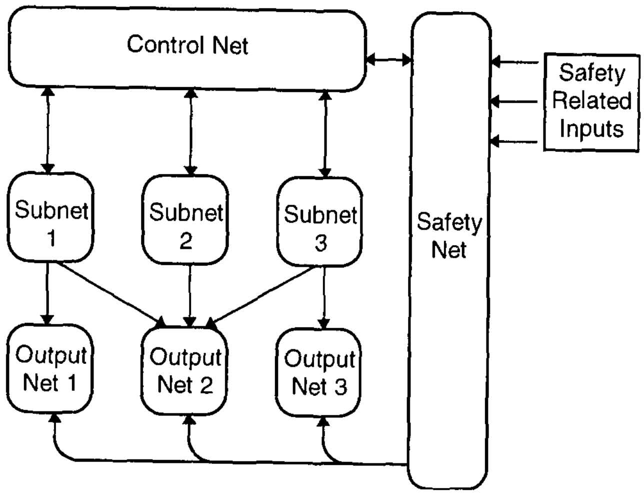

Fig. 1. Petri net structure for workstation.

3. PETRI NET STRUCTURE

In order to clearly specify a system using ordinary Petri nets, a rigid structure is applied. This structure has the following benefits:

- Gives a generic structure which can be used to describe any manufacturing system.
- Provides a clear graphical representation of the specified system.
- Gives a hierarchical structure to control code design.
- Ensures a modular structure allowing individual modules to be augmented without affecting the rest of the system.
- Provides enhanced diagnostic ability by leading the user to the point of error through the different levels in the hierarchy.

The term workstation is used here to describe the basic units of the manufacturing system. The Petri net describing the operation of each workstation is structured as shown in Fig. 1. The elements of this structure are described below:

3.1 Control net

The control net describes the basic actions of the workstation at the highest level of abstraction. Each action is shown as a place in the control net. The places of the control net are divided into primitive places and non-primitive places. Non-primitive places are simply the places which represent more complex actions at lower levels in the control code hierarchy. An example of a non-primitive place would be that representing the action "placing pallet on conveyor" Primitive places are those which do not represent complex actions. An example of a primitive place would be that representing the state "station ready".

3.2 Subnets

Each non-primitive place in the control net represents a more complex set of actions at lower levels in the hierarchy. Each of these actions is described in detail by a subnet. The subnets are initialised by the control net when the workstation is powered up. They are then ready to start their actions on request from the control net at the correct points in the machine cycle. Feedback from the sensors attached to the workstation indicate, within the subnets, whether the desired actions have taken place. When they have finished execution, the subnets return to their initial state and send a signal to the control net to indicate completion of their task. This behaviour can be seen in Fig. 2, where p_S starts the already initialised subnet and p_F represents the finish signal.

3.3 Output nets

In order to translate the actions described in the subnet into physical events, the subnets invoke output nets. These control the physical devices attached to the workstation such as solenoids or electric motors. An particular output net may be invoked by a number of different subnets at different points in the execution of the workstation's tasks. Communication between the subnets and the output nets is only in one direction. Any feedback from the execution of the output nets is monitored by the sensors attached to the system and signalled in the subnet requesting the action.

3.4 Safety net

The safety net sits at the same level in the hierarchy as the control net. It monitors the safety related inputs attached to the workstation, such as emergency stop conditions or machine guard status signals. If an unsafe state is detected the safety net invokes the output nets as required to either shut the machine down in a safe manner, or where possible, perform automatic recovery from such situations. There is also a link between the safety net and the control net which, on powering up, prevents the machine from carrying out any physical tasks until it is in a safe state.

The link between the control net and a subnet, detailed in Fig. 2, shows that when the transition t_1 fires, a token is placed both in the non-primitive place p_3 (shaded grey) and in the place p_S. Place p_S starts the subnet, from its initial state, and on completion of the its task returns to the initial

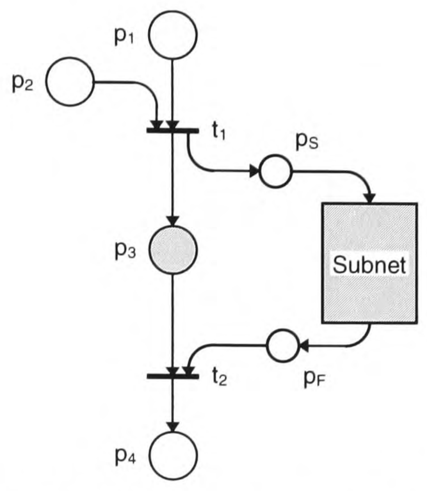

Fig. 2. Detail of link between control net and subnet.

state and generates a token in place p_F. This will enable transition t_2 to fire thus removing the tokens from place p_3 and p_F. Thus a token will remain in the place p_3 until its associated subnet has finished its task. Since the presence of a non-primitive place implies that there are links to a subnet, these links need only be shown on the Petri net graph of the subnet to which they belong. This increases the clarity of the control net, enhancing its use as a specification tool.

Where an output net is invoked by a subnet, there is only a single link requesting an action from the output net (e.g. switch solenoid on). There will be no return signal from the output net itself, but as mentioned previously feedback form sensors on the machine will indicate in the subnet whether the desired event has taken place. The signals from these sensors are implemented as places in the subnet from which the request originated.

4. IMPLEMENTATION

If the Petri net is to be implemented on programmable controllers, it is essential that all properties of the net are preserved in the translation into the languages used by those controllers. The problems faced in developing such a translation are discussed here with reference mainly to a PLC using ladder logic as its control language. These problems, however, are not exclusive to such an implementation. To aid the discussion of such problems, a more detailed description of how a Petri net is implemented as a Ladder Logic Diagram (LLD) is required.

4.1 Ladder representation of a Petri net

The conversion of Petri nets to ladder logic has been dealt with previously in (Henry and Webb, 1988; Cutts and Rattigan, 1992; Satoh, *et al.*, 1992) and a comparison between the two was presented in (Venkatesh, *et al.*, 1994).

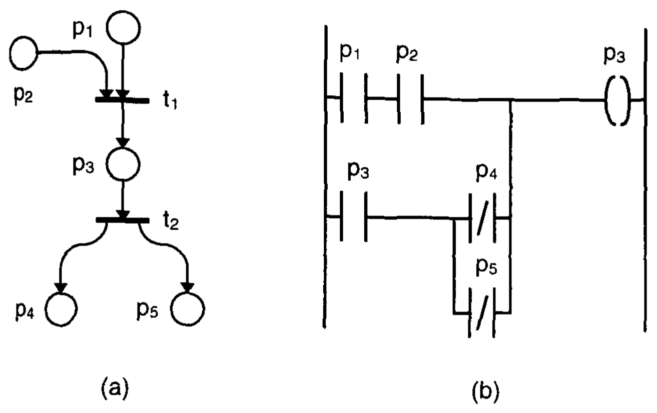

Fig. 3. (a) Petri net segment. (b) Ladder representation for place p_3.

An approach similar to that used by (Henry and Webb, 1988) has been adopted which preserves both the structure and the diagnostic capability of the Petri net. Each place in the Petri net is represented by an output coil in the LLD (see Fig. 3). When the output coil is set, it indicates that its corresponding place contains a token. Transistions are not directly implemented but each is represented by its set of input places. This is a different approach to that used in (Satoh, *et al.*, 1992) where transitions are also represented as outputs which are set when the transition is enabled. Their approach does reduce the number of relays used in each rung but the overall number of outputs required is greatly increased, as is the size of the LLD. This would be unacceptable in cases where small low-cost controllers are being used due to the memory requirements of large LLDs. For a discussion of efficient LLD design see (Pessen, 1989).

In Fig. 3(a), transition t_1 is enabled when places p_1 and p_2 both contain a token. When the transition fires, a token appears in place p_3. This token can only be removed when transition t_2 fires i.e. when places p_4 and p_5 have received their tokens. It can be seen that this behaviour is reflected in the LLD rung of Fig. 3(b) representing the set and reset logic for output p_3. The entire LLD representation of the Petri net is constructed in this way, with one rung for each place in the net.

The Petri net structure is preserved, by positioning the places of the control and the safety nets at the top of the LLD. These are followed by those of the subnets and then finally the output nets (see Fig. 4). The advantages of using Petri nets, as presented here, for design and maintenance of LLDs is clear. By preserving the net structure and representing only places as outputs in the LLD, structured control code is produced which is more easily maintained and provides a clearer diagnosis of fault conditions.

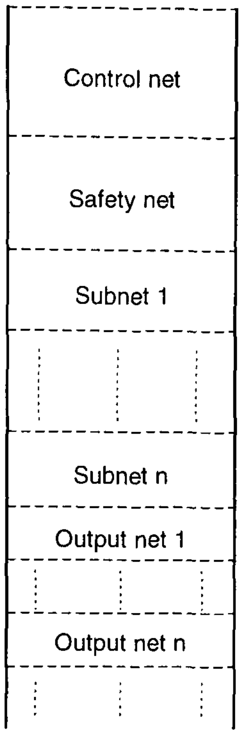

Fig. 4. Organisation of nets within LLD.

4.2 Token generation

A number of factors must be taken into account when implementing the net on a sequential machine. Concurrent behaviour such as the simultaneous generation of tokens cannot be directly implemented and the method used to simulate this behaviour is crucial to the correct operation of the Petri net. The way in which a program runs on a particular type of controller also bears heavily on how the net is implemented. In order to ensure correct operation of the Petri net when represented as an LLD, each transition must be kept enabled until all of its output places have received their tokens, i.e. for more than one scan cycle of the PLC. This requires that all the input places keep their tokens until the output places have received theirs. This is implemented as shown in Fig. 3(b), where the output coil representing place p_3 remains set until the relays representing places p_4 and p_5 are set (i.e. places p_4 and p_5 have received their tokens).

When implementing Petri nets in a high level programming language, such as C++, a method must be applied which allows all currently enabled transitions to be fired before checking the net for any further enabled transitions. This has been implemented as a list of currently enabled transitions, which is generated on each scan through the Petri net. All transitions in this list are fired before commencing the next scan of the Petri net.

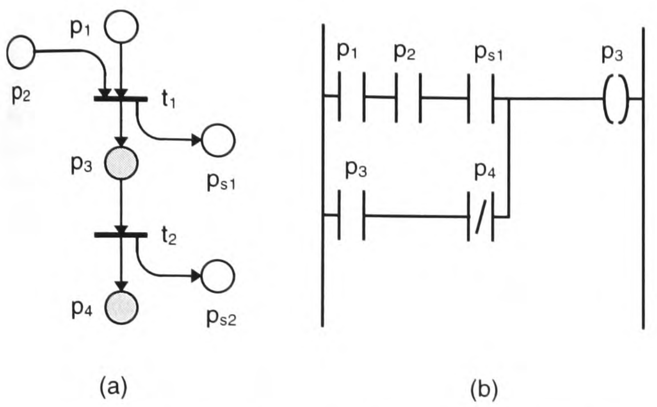

Fig. 5. Place p_3 receives its token after the subnet is started.

4.3 *Error diagnosis*

If it is to be useful as a diagnostic tool, the Petri net should be able to indicate where a machine has failed. If the machine stops, due to a fault during normal operation, the control net can be examined and the positions of any tokens present will indicate where in the machines cycle a problem has occurred. If these tokens are present in non-primitive places then the problem can be traced down through the Petri net structure to the appropriate subnets and output nets, and the cause of the stoppage can be located precisely. It is clearly important that in any translation of the Petri nets that this ability to trace faults in the machine cycle is preserved and it is for this reason that the timed place representation of the Petri nets was highlighted in section 2. The LLD generated from the Petri net specification in the manner shown here is equally ideal for locating the point at which the machine has stopped which is indicated by outputs being set.

4.4 *Limitations of current approach*

The method discussed previously for conversion to LLDs was found to pose a few problems when used to control complex systems with a large number of actuators. The problem exhibited itself as the multiple firing of a transition, causing multiple generation of tokens, and the non-removal of tokens from input places once a transition had fired. These errors were found to occur at the link between the control net and subnets where the control net place was receiving its token before the subnet places. An improved method was developed and is detailed in Fig. 5. Here the place p_3 now only receives a token after place p_{s1} has received its token and thus starts the subnet. This doesn't alter the accuracy of the Petri net representation in the LLD since it only affects the order in which the transition's output places receive their tokens. It can be seen in Fig. 5(b), that the reset logic now only needs to contain the control net place p_4. This is because it is known that p_4 can only be set if p_{s2} is already set. This gives us the added advantage of clearer LLDs and adds a certain amount of decoupling of the subnets from the control net. It also makes the LLD more deterministic because if a non-primitive place contains a token, then its associated subnet will have started its operation.

5. PRACTICAL EXAMPLES

The Petri nets shown here have been successfully used in the specification and design of control code for two separate systems. These systems are briefly described.

5.1 *Raw Materials Handling Station*

A raw materials handling station, which is part of a larger automated manufacturing system based in the Mechatronics Research Centre, has been used to develop the system of translation from Petri nets into LLDs. The station uses in total, fifteen single acting pneumatic cylinders to perform the task of loading pallets with different types of material and then placing these pallets on a conveyor. Any requests for raw materials are issued by a PLC which acts as the main controller for the whole system. It is also possible to use the workstation as a standalone unit with requests being generated by an attached PC. Structured Petri nets have been used to add greater functionality to the station and also allow increased flexibility.

5.2 *Flexible Manufacturing Cell*

Work has also been carried out to integrate two industrial CNC machines and a robot to form a flexible manufacturing cell. The cell was required to take two different types of raw material, process each according to a specified program and then assemble the processed parts. The main control of the cell is carried out by a PLC which co-ordinates the actions of a robot, a CNC lathe and a CNC mill. The robot is used for loading and unloading the CNC machines and for the simple assembly task. The main actions of the machines were specified using the structured Petri nets described in this paper (The specification was examined and finalised by the owners of the system, most of whom had no prior knowledge of Petri nets.). It was then converted into control code for the PLC. The control code for the robot was specified using Petri nets but the conversion into the language used by the robot controller, which is very similar to BASIC, was not quite as detailed as that carried out for PLCs. Since the actions of the robot were purely sequential it was sufficient to structure the code in the same way as the Petri nets are structured but there was no need to attempt to directly represent the Petri net in the control language. This allowed the production of well structure code for the robot controller which was found to be easily maintained and updated.

6. CONCLUSIONS

It has been shown that it is possible to use ordinary Petri nets, with a hierarchical structure to specify and design control code for discrete manufacturing systems. Using these nets gives the advantage of using the same formalism for the specification, design, modelling and control of the system. Directly implementing the Petri net specifications as control code enables well structured and maintainable control code to be produced. The need for careful analysis of the net structure, the way in which the net is represented within the controller, and also the way in which the controller operates have been highlighted. Current work has mainly concentrated on the implementation of Petri nets on PLCs using LLDs. Work is being carried out to develop their implementation in other languages and is continuing on an implementation in C++ for controlling workstations directly using personal computers. The safety net is to be expanded as the need for integrated safety systems becomes more important due to the complexity of modern manufacturing systems. If a variety of controllers are modelled and controlled using the same technique then integration of these controllers into a coherent, flexible system becomes possible.

REFERENCES

Cutts, G. and S. Rattigan (1992). Using Petri nets to develop programs for PLC systems. In: *Proc. 1992 International Conference on Application and Theory of Petri Nets*, Sheffield, UK.

David, R. and H. Alla (1992). *Petri nets and Grafcet: Tools for modelling discrete event systems,* Prentice Hall , London.

Dohi, Y., M. Sugiyama, H. Murakoshi, T. Sekiguchi, Y. Cai, I. Yomiya and D. Taufana (1992). Error detection scheme for Petri net sequence controller. In: *Proc. 1992 IEEE international Symposium on Industrial Electronics*, 45-48. Xian, China.

Ferrarini, L., (1992). An incremental approach to logic controller design with Petri nets. *IEEE Transactions on Systems, Man and Cybernetics*, **22**, 461-473.

Henry, R.M. and M. Webb, (1988). Ladder logic for sequence generation - A methodology. *Measurement and Control*, **21**, 11-13.

Jafari, M.A. (1990). Petri net based shop floor controller and recovery analysis. In: *Proc. 1990 IEEE International Conference on Robotics and Automation*, 532-537. Cincinnati, Ohio, USA.

Murata, T., N. Komoda, K. Matsumoto and K. Haruna, (1986). A Petri net-based controller for flexible and maintainable sequence control and its applications in factory automation. *IEEE Transactions on Industrial Electronics*, **IE-33**, 1-8.

Murata, T., (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, **77**, 541-581.

Pessen, D.W., (1989). Ladder diagram design for programmable controllers. *Automatica*, **25**, 407-412.

Peterson, J.L. (1981). *Petri Net Theory and the Modeling of Systems,* Prentice Hall, Englewood Cliffs, NJ, USA.

Satoh, T., H. Oshima, K. Nose and S. Kumagai (1992). Automatic generation system of ladder list program by Petri net. In: *Proc. 1992 IEEE International Workshop on Emerging Technologies and Factory Automation*, 128-133. Melbourne, Victoria, Australia.

Venkatesh, K., M. Zhou and R. J. Claudill, (1994). Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system. *IEEE Transactions on Industrial Electronics*, **41**, 611-619.

Willson, R.G. and B.H. Krogh, (1990). Petri net tools for the specification and analysis of discrete controllers. *IEEE Transactions on Software Engineering*, **16**, 39-50.

IMPLEMENTATION OF PETRI NETS FOR THE CONTROL OF MANUFACTURING SYSTEMS

M. J. Stanton, BSc, AMIEE.
Mechatronics Research Centre, University of Wales College, Newport, UK.
email: mstanton@newport.ac.uk

W. F. Arnold, BSc, MSc, CEng, MIEE.
University of Wales College, Newport, UK.

ABSTRACT

A simple Petri net definition is described which by applying appropriate design guidelines and a rigid structure is transformed into a powerful methodology for the design and implemtation of control code for discrete manufacturing systems. A description of how this methodology has been applied to real systems is presented along with a discussion of the issues arising from such an application and the advantages gained by its use.

1. INTRODUCTION

Petri nets provide a graphical and mathematical tool for the modelling, analysis and control of discrete event systems [1]. They have been found to decrease the development time of complex sequence controllers [2], and, graphically, provide an ideal communication tool for specification purposes.

There has been a great deal of interest in the use of Petri nets for the sequence control of discrete manufacturing systems and a number of different methods have been proposed. A hierarchical shop floor controller based on coloured Petri nets is presented in [3]. Here, the complexity of system analysis is reduced by the use of a modular structure, as each module can be analysed in isolation. Even so the analysis of coloured Petri nets is still complex and in this particular application important properties, such as liveness, are difficult to establish. There is little mention of how, or on what type of controller such a system would be implemented.

In [2] a Petri net based sequence controller is implemented on a number of real systems. This method uses an extension of Petri nets called C-net which provides a graphical representation for the system description which is interpreted to provide direct control of the system. The development time of the control software is reported to be shorter using this method than when compared with development times using Ladder Logic Diagrams (LLDs).

A more detailed comparison between Petri nets and LLDs can be found in [4]. The difficulties faced when designing and maintaining control code using LLDs are highlighted and it is suggested that Petri nets overcome such problems. A simple Petri net extension called Real Time Petri Nets is proposed. This extension is deliberately kept simple because it is intended that it should directly control their example system.

In [5] synthesis techniques for Petri net controllers are presented which guarantee certain structural properties of the net. The implementation of such nets is also discussed. This relies on a Petri net description language which is then compiled into the working code for a *token player*. The token player runs on a Personal Computer (PC) which acts as supervisory controller for the cell.

It has been noted in [6] that the use of complex extended Petri nets may result in the loss of some useful properties. This complexity will also affect the ease with which the Petri net can be implemented on the controllers present in modern manufacturing systems. Most of the methods described here for sequence control are implemented on PCs. Traditionally manufacturing systems are controlled using Programable Logic Controllers (PLCs) which are programmed using low level languages such as Ladder Logic Diagrams (LLDs) or Boolean Logic. A system for conversion of Petri nets into (LLDs) is described in [7]. In order to facilitate such conversion, the nets presented in this work are intentionally simple.

Their modelling power is increased, however, by the application of a hierarchical structure. This provides a number of advantages some of which are listed here:

- Gives a generic structure for describing any hierarchical manufacturing system.
- Provides a clear graphical representation of the specified system.
- Ensures a modular structure allowing individual modules to be augmented without effect on the remainder of the control code.
- Separates the sequence control code from the hardware model.
- Allows simple and effective diagnosis of system failiures.

The following section describes the Petri nets used in this work and highlights the important aspects of their definition. This is followed in Section 3 by a description of the structure applied to the Petri nets in order to increase their ability to accurately model hierarchical manufacturing systems. Section 4 describes some practical examples of real systems on which the Petri nets have been implemented. It describes first the implementation for a single machine, and then goes on to describe that for larger systems, consisting of a number of individual machines. Finally Section 5 provides details of continuing work using the nets system.

2. PETRI NET DEFINITION

The basic definition of Petri nets can be found in many texts (for example [6], [8]). The Petri nets used in this work are *ordinary* Petri nets but the use of inhibitor arcs is also permitted. The Petri nets must be safe (i.e. each place may only hold a single token at any time). This is particularly important when using the net for fault diagnosis, and where places are used to represent non-primitive events.

2.1 Primitive and non-primitive places

In [8] transitions are used to represent events and the notion of *primitive* and *non-primitive* events is discussed. A primitive event is one which occurs instantaneously whereas a non-primitive event takes time to occur. The non-primitive event can also be represented as two primitive events and a place. The primitive events can be described as "non-primitive event starting" and "non-primitive event finishing" with the place representing the condition "non-primitive event occurring". In this work, such a non-primitive event is used to describe complex actions which may be viewed as a sequence of primitive events. However, the place representing the condition "non-primitive event occurring" is referred to as a *non-primitive* place.

The places of a Petri net can therefore be divided into two separate classes.

i) *Primitive places* - ordinary places which represent states of the system (e.g. robot ready).

ii) *Non-primitive places* - representing system actions (e.g. lathe processing part). Non-primitive places usually have a subnet associated with them. A token in a non-primitive place indicates that its associated subnet or task is active. This is described in more detail in section 3.

Note that there is only one type of transition in the Petri net, which is that representing non-primitive events. This means that all transitions fire instantaneously as soon as they are enabled removing tokens from input places and generating tokens in output places as they do so. Thus, unlike the transitions in Timed Petri nets or Stochastic Petri nets they have no time associated with them.

2.2 Hardware places

For implementation purposes, another type of place has been included in the definition. These represent the physical sensors and switches attached to the system and are referred to as hardware places. A token in a hardware place indicates that its associated switch is on.

The operation of all places is exactly the same as that defined for places in ordinary Petri nets. The distinction between types of place becomes useful in the graphical representation of the system, clearly indicating the purpose of each place.

3. PETRI NET STRUCTURE

The structure imposed on the Petri net is shown in Figure 1. The nets used in each level of the hierarchy have the same definition as described in the previous section and all communication between the nets is carried out via shared places. The direction of communication is indicated by the arrows. The dotted lines represent communication with devices attached to the machine (sensors and safety related inputs) or with an attached user interface (e.g. a control panel or Personal Computer).

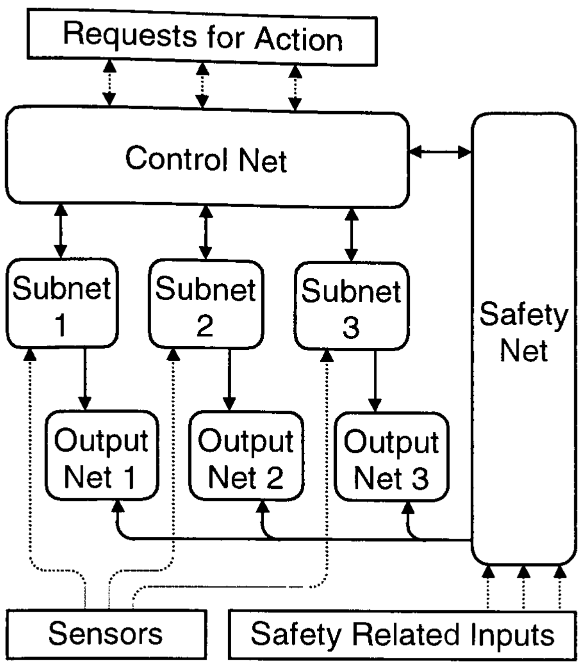

Figure 1. Petri net structure for a single machine.

3.1 Control net

The control net provides the sequence control for all specified machine actions. This includes details of any concurrency present in the system and any prioritisation specified by the system requirements. It must describe fully all requirements for correct system operation. As shown in Figure 1, the control net sits at the top of the control structure. It coordinates the actions taken by the subnets below it.

3.2 Subnets

The hardware of the system is represented by the subnets, with a single subnet describing each axis (or device). All states of the system must be described in the subnets. Subnets are invoked by the control net and will operate when invoked if able to do so. It is for this reason that the sequence information provided by the control net is complete and correct.

3.3 Output nets

Any output devices causing physical events in the system (such as solenoids or motors) are modelled using output nets. There is no direct feedback from the output nets to the subnets. Instead, feedback is provided by sensors attached to the system. These sensors are represented in the subnets as Hardware Places (see Section 2.2).

3.4 Safety net

The safety net sits at the same level in the control structure as the control net. When a machine is switched on, it is assumed to be in an unsafe state. Only when the safety net has verified that it is safe to do so will the machine be allowed to start operating. During operation, the safety net monitors both the state of the system and any safety related inputs (sensors attached to chip guards etc.). If an unsafe state is detected, it will take control of the output nets and initiate safe shutdown of the machine. There is also scope for implementing some automatic fault detection and recovery procedures within the safety net.

3.5 Initialisation

When a machine is powered up, its current state must be examined and, once known, the machine must be safely moved into a position where it is ready carry out its required operations. Such an initialisation is catered for by the inclusion of two additional places at the start of the control net. These places invoke subnets which carry out software and hardware initialisation of the system. Only when these subnets have finished executing will the machine be in a state where it may carry out operations. Ideally initialisation is carried out automatically, however, with the modularity of this control structure, it is possible, and perfectly reasonable to include steps which require feedback from a human operator.

4. IMPLEMENTATION

It is important for any control software to be implemented on existing systems as well as new installations. Existing systems may be adapted to increase flexibililty or augmented by the introduction of additional resources. This section decribes the implementation of the Petri net structure described in Section 3 on real systems. The first example is a single machine supplying raw materials to a larger working system and the second is a newly installed Flexible Manufacturing Cell.

4.1 Raw Materials Station

A raw materials handling station provides raw materials to a small manufacturing system based in the Mechatronics Research Centre. It consists of, in all 15 single acting pneumatic cylinders, which are controlled by a small PLC. The PLC is linked via a token ring network to other PLCs on the system and to a PC via an RS232 link. A diagram of the Raw Materials Station is shown in Figure 2. Pallets are stored on the station which are loaded with different raw materials and placed on the conveyor when a wagon is present to take it.

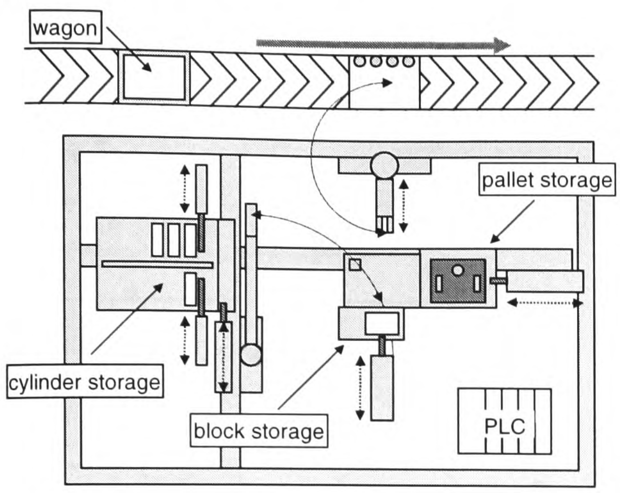

Figure 2. Layout of raw materials station

A Petri net controller was designed according to the structure presented in Section 3. The method used employs both a bottom up and top down approach. The top down approach is used to design the control net as follows:

1. The separate tasks required of the machine are identified.

2. The tasks are decomposed into a number of subtasks, each of which represents an action by an individual axis.

The bottom up approach is applied to the hardware model of the system. Each axis is modelled by a subnet, and the output devices are modelled by output nets. The actions required of these axes are initiated by the control net. The link between the control net and the subnets is shown in Figure 3.

Places p_S and p_3 both receive a token when transition t_1 fires. Place p_S starts the execution of the subnet, the details of which are not shown in the figure. The subnet completes its task by placing a token in place p_F. This allows transition t_2 to fire thus removing the tokens from places p_F and p_3. As stated previously, there is no time associated with the firing of transisitions, they fire instantaneously as soon as they are enabled. However, since the subnet, which is causing the movement of physical devices, will take time to complete, the token in place p_3 will remain in there until transition t_2 is enabled (by the arrival of a token in place p_F). Therefore place p_3 can be viewed as a timed place with time *T* (where *T* is the time taken for the subnet to execute).

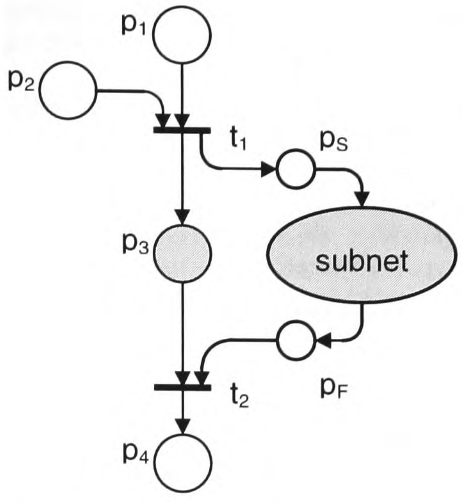

Figure 3. Detail of link between control net and subnet

In the particular case of the raw materials station, there are a large number of actuators which are grouped to form different devices. In order to simlify the control structure an additional control level was added to the system between the control net and the subnets. This means that the control net co-ordinates the different devices and the new "sub"-control nets co-ordinate the actuators comprising these devices. If each device is viewed as a machine in its own right, this then leads to a control structure for a group of individual machines. Such a control structure is discussed in the follwing section.

4.2 Flexible Manufacturing Cell

A Flexible Manufacturing Cell consisting of a CNC mill and a CNC lathe serviced by a robot which is mounted on a slide mechanism is shown in Figure 4. The robot and both CNC machines have their own controllers and the overall supervisory control of the cell is carried out by a PLC. Unlike the machine described in Section 4.1, this system was a new installation and so the control code was to designed from scratch.

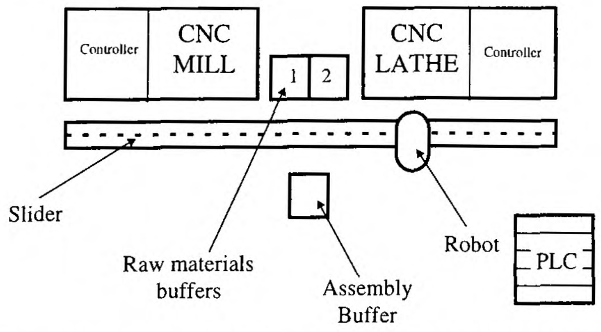

Figure 4. Layout of the Flexible Manufacturing Cell

One of the main challenges of such a system, is that it contains a variety of controllers on which the Petri net structure is to be implemented. Also, the control net is, in this case, co-ordinating a number of self contained machines rather than elements of a single machine. This resulted in a structure similar to that shown in Figure 5.

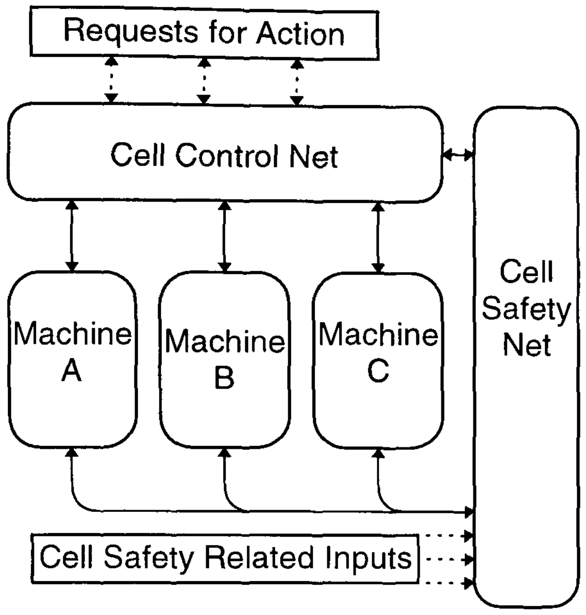

Figure 5. Petri net structure for a manufacturuing cell.

The cell control net co-ordinates the activities of the machines by requesting actions from them. This is done in much the same way as the control panel or PC was used to invoke the actions of a single machine (Figure 1). The blocks marked A, B, and C in the figure may each contain the full structure detailed in Figure 1. The cell control net communicates directly with the machine control nets and the cell safety net communicates directly with the machine safety nets.

4.3 Further Developments

A larger system consisting of a machining cell similar to that described in Section 4.2, the raw materials station discussed in section 4.1, an Automated Storage and Retrieval System (ASRS), and a conveyor system for the transportation of materials is currently housed in the Mechatronics Research Centre. Other workstations attached to the system may be incorporated in the project at a later date. Such a system presents an opportunity to develop the Petri net control structure further to;

1. Incorporate a higher level of control.
2. Describe a system with a greater amount of concurrency.
3. Investigate the modelling of a conveyor system.

At this level there are also other implications, such as interfacing with Management Information Systems and Databases for batch information of particular products. Some inclusion for a scheduling strategy must also be incorporated into the system. All these additions to the current system are made possible by the modular structure and as shown in section 4.2.

5. CONCLUSIONS

A method of generating Petri net controllers for discrete manufacturing systems has been presented and the benefits of such a method are detailed. The Petri nets have a simple definition but their power as a tool for modelling and control is increased by the structure imposed on them. For testing and maintaining control code they provide all the advantages of modular design techniques. This has the added bonus of enabling other types of controller, which are not described by Petri nets, to be included in the system and therefore this method is ideal for the integration of manufacturing systems. Further development of the safety subsystem, represented by the safety net (see Figure 1) is required to produce a fully integrated manufacturing safety system.

ACKNOWLEDGEMENTS

The authors would like to thank Arthur Buck for his continued support, advice and constructive criticism.

REFERENCES

[1] Zurawski, R., and Zhou, M., *Petri Nets and Industrial Applications: A Tutorial*, IEEE Transactions on Industrial Electronics Vol. 41 No. 6 pp 567-583, 1994.

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

[3] Jafari, M.A., *An Architecture for a Shop-Floor Controller Using Colored Petri Nets*, International Journal of Flexible Manufacturing Systems No. 4 pp 159-181, 1992.

[4] Venkatesh, K., Zhou, M., and Caudill. R.J., *Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System*, IEEE Transactions on Industrial Electronics Vol. 41 No. 6 pp 611-619, 1994.

[5] Zhou, M., and DiCesare F., *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer Academic Publishers, USA 1993.

[6] David, R., and Alla, H., *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems* Prentice Hall, London 1992.

[7] Stanton, M.J.,Arnold, W.F., and Buck., A.A., *Modelling and Control of Manufacturing Systems Using Petri Nets* to be presented at the 12th IFAC World Congress July 1996.

[8] Peterson, J.L. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, NJ, USA 1981.

EXTENSION OF STRUCTURED PETRI NETS FOR THE CONTROL OF A CONVEYOR SYSTEM

M. J. Stanton, W. F. Arnold

University of Wales College, Newport, Wales, UK

ABSTRACT

This paper describes the extension of a manufacturing control structure for the control of a closed loop conveyor which delivers parts and raw materials to a number of workstations. The control structure is based on ordinary Petri nets which have been extended to allow the addition of external inputs and outputs. These nets are designed as modules which can be linked, via condition signals, to form the control structure of more complex systems. The possibilities of further expansion of the control structure to include scheduling systems and manufacturing information systems is also described. Finally the proposed application of such controllers to fault monitoring and diagnosis is discussed.

INTRODUCTION

The application of Petri nets for modelling and control of manufacturing systems has been studied by many authors. They are favoured for their simple graphical representation and their ability to be used in the specification, design and implementation of systems, Zurawski and Zhou (1).

Many extensions to Petri nets have been proposed for the control of manufacturing systems, although not all give an indication of how they would be implemented. Those that do usually rely on a *token player* running on a Personal Computer (PC) which is used to control a manufacturing cell, Zhou and DiCesare (2). However it is more common to find Programmable Logic Controllers (PLCs), than Personal Computers used on the shop floor because of their rugged design and ability to handle a large amount of I/O. It is therefore necessary to design a Petri net controller that can be implemented on these and other common types of shop floor controller.

In order facilitate its implementation on controllers such as PLCs, the particular form of Petri net used should be as simple as possible. A robust method for the implementation of ordinary Petri nets in the form of a Ladder Logic Diagram has been developed in Stanton and Arnold (3). The modelling power of ordinary Petri nets may be greatly increased by the application of a modular structure to the nets. This is achieved by the addition of a simple extension to ordinary Petri nets which allows the inclusion of external inputs and outputs. The resulting structure provides the following among a number of benefits:

- It allows control modules to be developed and tested independently
- It allows simple and effective fault diagnosis during both development and operation of the system

These particular benefits are the subject of the work described in this paper

Use of Petri Nets for Manufacturing

In recent years a number of authors have published work on the specification, modelling, simulation and control of conveyor systems. Cruette et al, (4) describe a method for specifying the operating sequences of produced parts. Each operating sequence is described using an Object Petri net, which uses different tokens to represent each instance of the part being produced. By decomposing the system hierarchically they can make each transition in the operating sequence represent either a change of state or a change of position within the system. Although they apply their method to a conveyor system, the operating sequences they generate do not seem to depend on the type of materials handling system they are modelling, thus a conveyor belt, or an AGV system can be modelled in the same way.

In Lin and Lee (5) Timed Petri nets are used to model a zone-control conveyor which only allows certain numbers of wagons (loaded or unloaded) into each type of zone. The different zone types are modelled independently as transition bordered subnets which are analysed for desirable properties (boundedness, liveness, conservativeness, etc.). These zone modules are then selected to from the model of the conveyor system.

Cohen (6) describes the implementation of an Expert System to control a conveyor system. The Expert System acts as an event driven system which simply stores all possible states of the system in the knowledge base and uses rules to determine the possible next-state.

Moore and Gupta (7) describes an Stochastic Coloured Petri Net modelling method for flexible manufacturing systems. Subsystems are developed individually and merged at transitions. The preservation of properties of each subsystem net is used to ensure those same properties for the whole system. They present as an example the modelling of a conveyor belt which is divided into segments. This is a similar approach to that of (5) in that the conveyor itself is modelled by splitting it into a number of zones.

Yeung and Moore (8) introduces the concept of, and the requirement for, Flexible Conveyor Systems and presents an object oriented model for the control of such complex systems. To accommodate such a model, rather than using interconnected PLCs to control the system, the method proposed uses microcomputers interconnect using a version of fieldbus.

The contribution of this work is in the control of automated manufacturing systems. Many of the Petri net methods described in the literature present complex extensions to Petri nets which can only be implemented using high level languages on relatively powerful computers. This is also true for the methods described in (6) and (8). However the most common computer used for the control of manufacturing systems is the Programmable Logic Controller (PLC), favoured for its robust design and large amount of expandable I/O. Such controllers are usually programmed using Ladder Logic Diagrams (LLDs) or more recently Sequence Function Charts (SFCs). Other methods such as structured text are also in use but they are all very much low level programming languages and therefore do not allow the complexity of high level languages such as C/C++, or PASCAL.

It is with this in mind that the method of control code design presented here was developed. The Petri nets used are slightly modified, ordinary, safe Petri nets which are readily translated into LLDs or SFCs. Their modelling power is increased by imposing a structure on the nets which allows the development of system modules, and stepwise refinement of the control code for the various machines of the system. The next section describes ordinary safe Petri nets and then goes on to detail the extension to allow for external inputs and outputs. The general control structure is the defined, detailing the communications between the various modules. The application of this structure to the control of a conveyor system is then presented followed by description of the further extension of the structure. Finally a proposed method for fault monitoring is presented which uses the Petri net structure as a specification of expected system behaviour.

STRUCTURED PETRI NETS

A structured Petri net is defined here as an ordinary safe Petri net with external inputs and outputs, Ichikawa and Hiriashi (9). The external inputs and outputs are binary places, where the inputs act as binary control places, much like those introduced in Krogh (10), and the outputs act as binary feedback places giving the controller information about the state of the net. In this section a definition of ordinary safe Petri nets is presented followed by the extension required to describe Petri nets with external inputs and outputs.

Petri nets

A Safe Petri net is a 5-tuple, $PN = \{P, T, I, O, \mu_0\}$ where:

$P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions,
$P \cup T = \varnothing$,
$I : T \rightarrow P$ is the input function mapping from transitions to places,
$O : T \rightarrow P$ is the output function mapping from transitions to places,
$\mu_0 : P \rightarrow \{0,1\}$ is the initial marking.

Petri net with external inputs and outputs

A Petri net with external inputs and outputs is a 5-tuple, $PNIO = \{P, T, I, O, \mu_0\}$ where:

$P = S \cup C^{in} \cup C^{out}$,
$S = \{s_1, s_2, \dots, s_i\}$ is a finite set of state places,
$C^{in} = \{c_1^{in}, c_2^{in}, \dots, c_j^{in}\}$ is a finite set of input places,
$C^{out} = \{c_1^{out}, c_2^{out}, \dots, c_k^{out}\}$ is a finite set of output places,
$T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions,
$P \cup T = \varnothing$,
$I : T \rightarrow P$ is the input function mapping from transitions to places,
$O : T \rightarrow P$ is the output function mapping from transitions to places,
$\mu_0 : P \rightarrow \{0,1\}$ is the initial marking.

- Only state places can be initially marked, thus the initial marking of the control places is always zero.
- Output places are never inputs to transitions of the same net and are only outputs to transitions of one net.
- Input places are never outputs to transitions of the same net and are only inputs to the transitions of one net.

This definition differs to that given by (9) where external outputs are represented as a subset of transitions, (and therefore event signals) rather than explicitly by places (and therefore condition signals). The use of places as external outputs as well as inputs is the key to the modular structure presented here and is favoured for its simplicity and because it provides a uniform method of communication between nets.

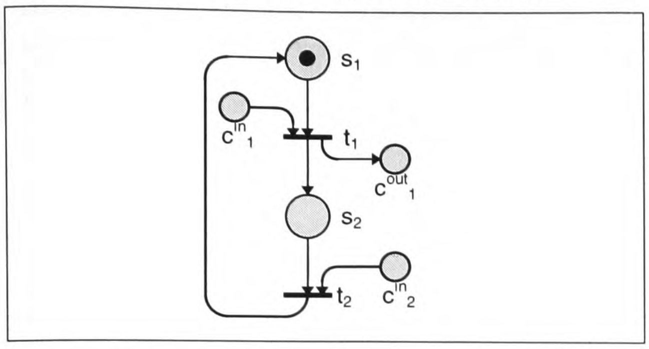

Figure 1. Example of a Petri net with external inputs and outputs

Example 1. Figure 1 shows a Petri net with external inputs and outputs with:

$$S = \{s_1, s_2\},\ C^{in} = \{c_1^{in}, c_2^{in}\},\ C^{out} = \{c_1^{out}\},$$
$$T = \{t_1, t_2, \ldots, t_n\},$$
$$I(t_1) = \{s_1, c_1^{in}\} \qquad O(t_1) = \{s_2, c_1^{out}\}$$
$$I(t_2) = \{s_2, c_2^{in}\} \qquad O(t_2) = \{s_1\}$$
$$\mu_0 = \{1,0,0,0,0\}$$

Petri net properties

There are several Petri net properties which have specific implications when applied to manufacturing systems. These are boundedness, liveness, and reversibility. Those most relevant to the work presented here is boundedness which is described as follows:

Boundedness. This property describes the maximum number of tokens present in the net for a given initial marking. A place is said to be *k*-bounded if the maximum number of tokens it will hold is *k*, where *k* is a positive integer. If all the places in the net are *k*-bounded then the Petri net is said to be *k*-bounded. In manufacturing terms, boundedness is used to ensure that there is no overflow of buffers of queues.

Safeness. The special case of boundedness is where the net is 1-bounded or *safe*. In structured Petri nets, safeness is used to indicate that a particular action is taking place, and to ensure that the same machine is not asked to carry out two tasks simultaneously.

CONTROL STRUCTURE

The basic control structure for simple systems is shown in Figure 2. The characteristic feature of this hierarchical structure is the control net which co-ordinates the activities of the subnets, and through which the subnets communicate. This type of centralised control structure has been applied to both a manufacturing cell and to a single machine. For a more detailed description see (3) and (11).

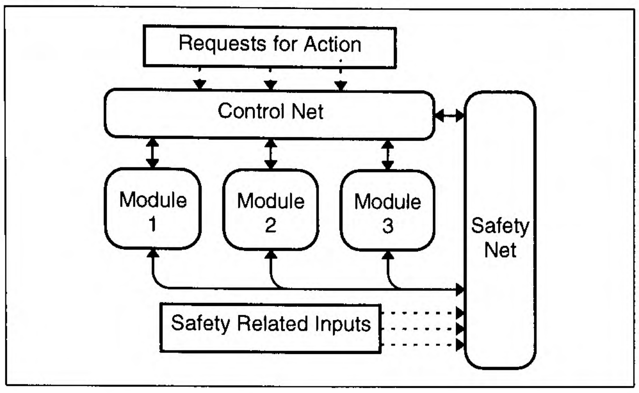

Figure 2: General Petri net structure used for the control of manufacturing systems.

Communication between nets

One of the important aspects of the Petri net structure is the communication between the nets. As described previously, communications are carried out using external inputs and outputs and along with the sequence control of the system define the structure and properties of the nets.

Example 2. Consider a machine which is capable of performing two separate tasks, but may only perform them one at a time. A controller must not request the machine to carry out these tasks simultaneously. The request signals are represented as output places from the controller and input places to the machine. The design of the controlling Petri net is thus constrained by the fact that it cannot allow the transitions that generate the request signals to be enabled simultaneously. This problem is similar to the mutual exclusion problem described in (2).

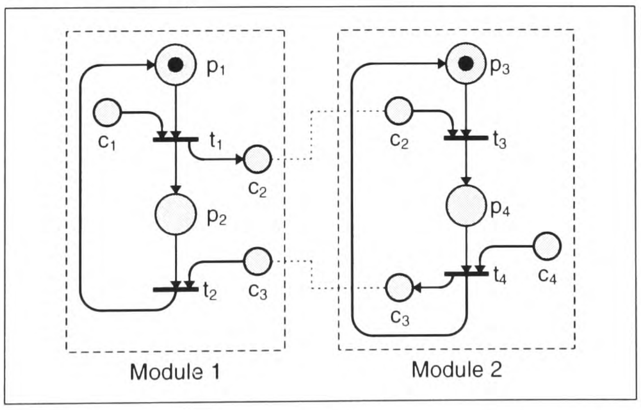

Figure 3. Example of two communicating Petri nets

Example 3. A simple example of two communicating Petri nets is shown in Figure 2. Transition $\mathbf{t_1}$ is *state enabled* (see (10)) by place $\mathbf{p_1}$ and will fire when control place $\mathbf{c_1}$ becomes marked. This control place may be linked to an external device, or be the output place of another Petri net. When transition $\mathbf{t_1}$ fires, tokens will be generated in state place $\mathbf{p_2}$ and control place $\mathbf{c_2}$. It can be seen that transition $\mathbf{t_3}$ is now enabled and will fire placing a token in place $\mathbf{p_4}$. When control place $\mathbf{c_4}$ becomes marked, again due to an external device or Petri net, the transition $\mathbf{t_4}$ will fire thus enabling transition $\mathbf{t_2}$. Once $\mathbf{t_2}$ fires the net will return back to its initial state.

It follows that a number of communicating nets such as these can be used to build a larger control structure of interacting systems.

CONTROL OF CONVEYOR SYSTEM

Previously, structured Petri nets have been used to control a single machine (11) and a Manufacturing Cell (3). They are now extended to provide a framework for the control of a more complex system.

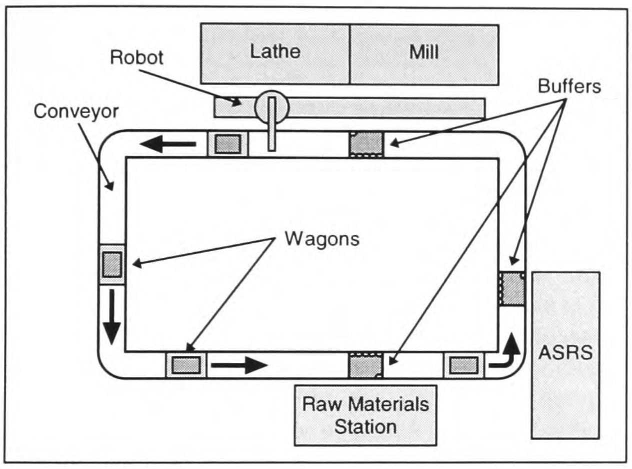

Figure 4: Layout of conveyor system

The system, shown in Figure 4, consists of a closed loop conveyor belt around which are positioned a number of workstations. The workstations included in the system are:

1. A Raw Materials Station (RMS) - provides two types of raw material for the manufacture of products.

2. An Automated Storage and Retrieval System (ASRS) - Storage area for completed products and as a temporary buffer for parts awaiting production.

3. Manufacturing Cell (MC) - consists of a milling machine and a lathe, each of which are loaded and unloaded by a single robot. The mill and lathe may operate simultaneously if there are parts available to be processed.

Parts are transported between the workstations on wagons which reside permanently on the conveyor belt. At each workstation, the conveyor has a buffer which stops the wagon and reads an identification (ID) number from it. This ID can be stored in a database with an entry describing the type of part, if any, the wagon is carrying. If the part matches that required by the workstation then the workstation will remove the part and update the state of the wagon.

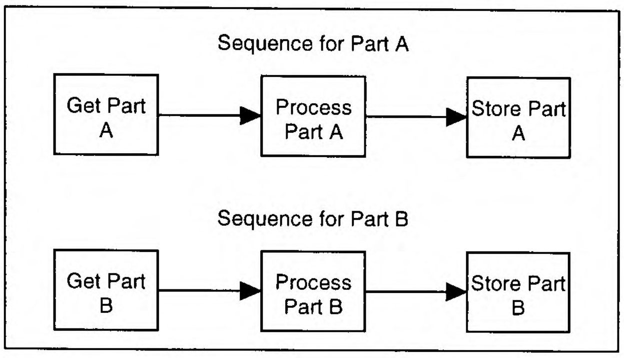

Figure 5: Production sequences for Part A and Part B

The system produces two types of part. Part A is processed in the milling machine, and Part B is turned in the lathe. The production sequence for each part is shown in Figure 5.

Each workstation on the system has it own controller, as does the conveyor system. In the case of the MC the robot controller is used as the cell controller. The other workstations and the conveyor are controlled by PLCs. The first stage in designing the control code for the system is to define the communications between each module (at this level of control, each controller is encapsulated in a single module). The communications for each module are shown in Table 1.

RMS. The RMS will indicate that it is ready to place materials on the conveyor and the type of material it is ready to put. When an empty wagon arrives, the conveyor controller informs the RMS that it can put, and

TABLE 1 - Communications between modules of the conveyor system

Station → Conveyor	**Conveyor → Station**
RMS	
Ready to Put Part A	Can Put/Get
Ready to Put Part B	
Ready to Put Pallet	
Wagon Move Enable	
MC	
Ready for Part A	Can Put
Ready for Part B	Can Get Part A
Ready to Put Part A	Can Get Part B
Ready to Put Part B	
Wagon Move Enable	
ASRS	
Ready to Put	Can Put/Get
Ready to Get	
Wagon Move Enable	

once it has done so, the RMS informs the conveyor that the wagon can move away (with the Wagon Move Enable signal) and the wagon state is updated.

MC. The MC informs the conveyor of its ready status (i.e. which type of part it is ready to process). Once the part is processed the MC informs the conveyor of the type of part it is ready to put. Once it has done so the wagon is allowed to move away and its state is updated.

ASRS. The ASRS waits for a finished item to arrive which is then removed from the system and the wagon state updated.

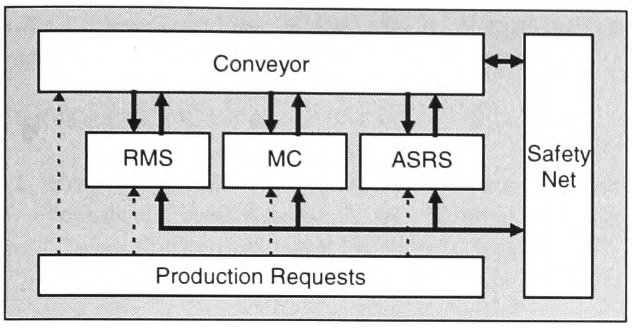

Figure 6. Control structure for conveyor system

The extended control structure for the conveyor system and workstations is shown in Figure 6. The major difference between this structure an the one shown in the previous section is that the conveyor controller seems to have taken over the role of control net. However this is not strictly the case because the conveyor is not co-ordinating the actions of the machines within the system. All the machines are instead autonomous units and their individual actions are prescribed by the Production Requests. The system now has a more distributed architecture rather than the centralised architecture of the previous section.

FURTHER EXTENSIONS TO THE CONTROL STRUCTURE

Due to the modular composition of the control structure presented here, it is possible to extend the existing structure to include such modules as scheduling systems and manufacturing information systems (see Figure 7 at the end of the paper). Such extensions are motivated by the need for automatic schedule generation and a database for *wagon status* and *work in progress* data. Also fault monitoring data as described in the next section can be stored in such a database. Such extensions represent a step towards a Computer Integrated Manufacturing (CIM) architecture.

FAULT MONITORING

This section will describe some development work on a fault monitoring scheme based on the Petri net control structure described previously.

Hardware faults

The Petri net control code provides information about the desired sequence of sensory information. By monitoring the *hardware places* (places in the Petri net which are linked directly to the system's sensors) and comparing them to the expected *sensory footprint* of the system at a given time, we can detect a hardware fault in the system. A hardware fault is defined here as one which exhibits itself in the system hardware (as distinguished from a production fault which exhibits itself as an error in the product). Such faults may be caused by one of the following:

- *Sensor failure* - A sensor is damaged in some way preventing it from carrying out its required task.
- *Actuator failure* - An actuator is damaged in some way preventing it either from acting all of acting within a prescribed period of time.
- *Software fault* - Defined as a fault which exhibits itself in the system software, possibly caused by a controller or network error.

Using purely sensory data, these failures are indistinguishable.

Software faults

The Petri net description of the control code also provides information about the control signals, which should be generated by the controller, given a particular sensory footprint. This *control footprint* can be provided by the output places of each Petri net module. Using this control footprint, any erroneous request for action by the controller can be detected and thus trapped before the system hardware acts on it allowing some form of recovery to take place. Thus allowing software faults can be removed from the list of indistinguishable faults described above.

CONCLUDING REMARKS

A Petri net structure has been described which has been applied to a single machine, a machining cell and now a more complex conveyor system. The modular representation of the system eliminates the state explosion problems usually faced when describing a system with ordinary Petri nets. Also the simplicity of the formalism used allows it to be faithfully reproduced on the types of low level controller found in modern manufacturing systems, such as PLCs. The use of modules allows the potential expansion of the structure towards a CIM architecture. Finally a proposed method of fault detection which allows the distinction between hardware and software faults is described. Such a distinction is made possible by the use of the communication places in the nets. Work is continuing on the implementation of such a fault monitoring system. Development work has also been started on a PC based software package to aid the design, testing and implementation of the net structure described in this paper.

ACKNOWLEDGEMENTS

The Authors would like to thank Mr A. A. Buck for his advice and continuing support.

REFERENCES

1. Zurawski, R. and Zhou, M.-C., 1994 "Petri Nets and Industrial Applications: A Tutorial", IEEE Transactions on Industrial Electronics, Vol. 41, 567-583.

2. Zhou, M.-C. and DiCesare, F, 1993 "Petri Net Synthesis for Discrete Event Control of Manufacturing Systems", Kluwer Academic Publishers, USA.

3. Stanton, M. J. and Arnold, W. F., 1996, "Implementation of Petri nets for the control of manufacturing systems", Proc. Mechatronics'96 with M^2VIP'96, Vol. 1, Guimarães, Portugal, 373-378.

4. Cruette, D., Bourey, J. P. and Gentina, J. C., 1991, "Hierarchical specification and validation of operating sequences in the context of FMSs", Computer Integrated Manufacturing Systems, Vol. 4, 140-156.

5. Lin, J. T. and Lee, C.-C., 1992, "A modular approach for the modelling of a class of zone-control conveyor system using timed Petri nets", International Journal of Computer Integrated Manufacturing, Vol. 5, 277-289.

6. Cohen, G., 1994, "Expert system to control and to design closed loop conveyor systems", Expert Systems With Applications, Vol. 7, 483-494.

7. Moore, K. E. and Gupta, S. M., 1995, "Stochastic coloured Petri net models of flexible manufacturing systems: Material handling systems and machining", Computers Ind. Engng, Vol. 29, 333-337.

8. Yeung, W. H. R. and Moore, P. R., 1996, "Object-oriented modelling and control of flexible conveyor systems for automated assembly", Mechatronics, Vol. 6, 799-815.

9. Ichikawa, A. and Hiriashi, K., 1987, "Analysis and control of discrete event systems represented by Petri nets", Discrete Event Systems: Models and Applications , Springer Verlag, Berlin, Germany.

10. Krogh, B. H., 1987, "Controlled Petri nets and maximally permissive feedback logic", Proc. 25th Allerton Conference, University of Illinois, 317-326.

11. Stanton, M. J., Arnold, W. F. and Buck A. A., "Modelling and Control of Manufacturing Systems Using Petri Nets", Proc. 12th IFAC World Congress San Francisco, USA.

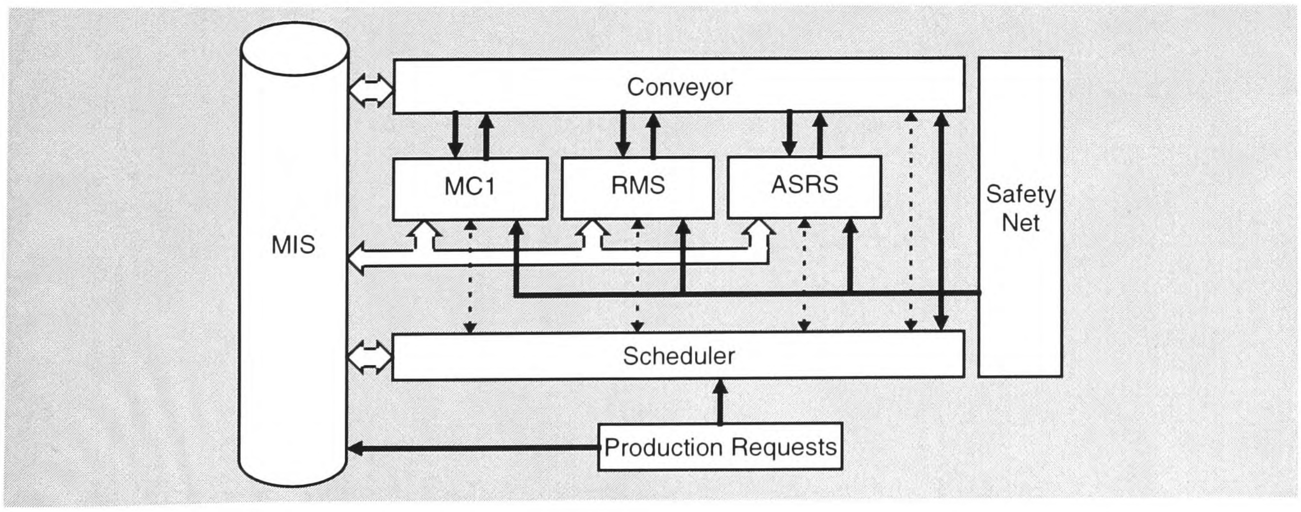

Figure 7. Proposed structure for Computer Integrated Manufacturing System