A Pragmatic Approach to the Development of Robust Real-Time Protocols

Darrell Joseph Smith

Submitted for the degree of Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

1998

The University of Aston in Birmingham

A Pragmatic Approach to the Development of Robust Real-Time Protocols

By Darrell Joseph Smith

Submitted for the degree of Doctor of Philosophy
1998

Summary

This research is concerned with the development of distributed real-time systems, in which software is used for the control of concurrent physical processes. These distributed control systems are required to periodically coordinate the operation of several autonomous physical processes, with the property of an atomic action. The implementation of this coordination must be fault-tolerant if the integrity of the system is to be maintained in the presence of processor or communication failures.

Commit protocols have been widely used to provide this type of atomicity and ensure consistency in distributed computer systems. The objective of this research is the development of a class of robust commit protocols, applicable to the coordination of distributed real-time control systems. Extended forms of the standard two phase commit protocol, that provides fault-tolerant and real-time behaviour, were developed.

Petri nets are used for the design of the distributed controllers, and to embed the commit protocol models within these controller designs. This composition of controller and protocol model allows the analysis of the complete system in a unified manner. A common problem for Petri net based techniques is that of state space explosion, a modular approach to both the design and analysis would help cope with this problem. Although extensions to Petri nets that allow module construction exist, generally the modularisation is restricted to the specification, and analysis must be performed on the (flat) detailed net.

The Petri net designs for the type of distributed systems considered in this research are both large and complex. The top down, bottom up and hybrid synthesis techniques that are used to model large systems in Petri nets are considered. A hybrid approach to Petri net design for a restricted class of communicating processes is developed. Designs produced using this hybrid approach are modular and allow re-use of verified modules.

In order to use this form of modular analysis, it is necessary to project an equivalent but reduced behaviour on the modules used. These projections conceal events local to modules that are not essential for the purpose of analysis. To generate the external behaviour, each firing sequence of the subnet is replaced by an atomic transition internal to the module, and the firing of these transitions transforms the input and output markings of the module. Thus local events are concealed through the projection of the external behaviour of modules.

This hybrid design approach preserves properties of interest, such as boundedness and liveness, while the systematic concealment of local events allows the management of state space. The approach presented in this research is particularly suited to distributed systems, as the underlying communication model is used as the basis for the interconnection of modules in the design procedure.

This hybrid approach is applied to Petri net based design and analysis of distributed controllers for two industrial applications that incorporate the robust, real-time commit protocols developed. Temporal Petri nets, which combine Petri nets and temporal logic, are used to capture and verify causal and temporal aspects of the designs in a unified manner.

Key Words: Real-Time Control Systems, Petri Nets, Atomic Commit Protocols, Temporal Petri Nets.

Acknowledgements

I wish to thank my supervisor Dr D.J.Holding for his guidance, support and encouragement of my research. I would also like to thank the other members of the IT research group at Aston, Dr Geof Carpenter, Jaspal, Daniel and Jingyue. Particular thanks must go to Dr Jaspal Sagoo for fostering a belief in myself and my work without which I would never have completed this Thesis (apologies to Mindy for taking so much of her husbands time). I must also thank Steve for keeping me sane and occasionally solvent during my student years. The ultimate thanks must go to my wife Maria for her patience, understanding and faith in me.

Finally, I wish to thank the Engineering and Physical Sciences Research Council (EPSRC) for funding my time as a research student.

This Thesis is dedicated to M.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Introduction

A major field of application for computers is in the control of physical processes. The computer system is often embedded within its controlled environment. Such systems are termed real-time computer systems. The essential requirement of these real-time systems is to provide:

i) a logically correct response to its environment, and

ii) a timely response to an input generated by its environment.

Thus the correctness of the systems response to its environment depends on both the logical correctness of the results it produces and the timing correctness. The consequences of a late response are used to classify real-time systems as either *hard* or *soft* [Krishna & Shin 97].

Soft real-time systems are required to meet deadlines, however failure to meet some deadlines will not result in some catastrophic event in the environment, but performance may be degraded below what is considered acceptable[Krishna & Shin 97], such as banks' automated cash dispensers or multimedia applications. While hard real-time systems are ones where failure to meet a deadline may result in a catastrophic system failure, these include embedded systems that control their environment such as aircraft, nuclear reactors and chemical plants. In this thesis the control of hard real-time systems will be considered.

Since hard real-time systems need to provide a service which is both timely and highly available in order to ensure a safe response to the environment, they must be developed using techniques which can provide a high degree of reliability.

Reliability has been defined as the ability of a system to perform its intended function for a specified period of time under a set of environmental conditions [Leveson 86]. However, the performance and reliability requirements of a system are often at cross purposes to one another [Levi & Agrawala 94], by making conflicting demands upon the system.

Hard real-time computer systems are often employed in the control applications, where the computer is interfaced directly to a process and is dedicated to the control of that process. A major use of such real-time computer controlled systems is in safety critical applications, due to this it is necessary to consider reliability and safety aspects in the design and development

of such systems. The type of fault likely to occur and their effects on the system must be considered. The type of fault can be classified as either:

i) Design faults, which are introduced into a system such as errors in design, poor component selection or software coding errors.

ii) Operational faults, which may arise during system operation due to permanent or transient component failures.

Since the development of computer systems for control purposes there has been a need to increase their reliability. Work in this area began in the 1950's and 1960's, and concentrated on improving the reliability of the hardware used through the duplication and triplication of components and sub-systems [Avizienis 85][Meyer & Pham 93]. However, as computer systems and their applications became more complex, this complexity in itself became an obstacle to reliability.

In the 1970's there began research into faults arising from the specifying, designing and implementation of the software used[Avizienis 85]. In response to this, increasing the reliability of a computer system can be approached in two main ways, through fault avoidance and through fault tolerance[Anderson & Lee 81]. Research into these two approaches has continued as computers are more widely used in safety-critical applications, increasing the need for high reliability [Levi & Agrawala 94].

Real-time control applications where failures can result in serious consequences include the control of nuclear power plants, air traffic control monitoring systems, patient monitoring in hospital intensive care units, military and defence systems[Leveson & Harvey 83]. In these type of applications software errors have led to costly failures, such as the Space Shuttle malfunction in 1982, the lethal doses of radiation given to patients under going cancer therapy in Canada in 1986 [Meyer & Pham 93] and the death of a worker due to an error in the software that controlled a production robot [Ostroff 92].

1.2 Research Background

Fault avoidance can be achieved through the use of structured and formal methods in the design and analysis of software controlled real-time systems. Structured methods have been applied to produce precise specifications and designs of hard real-time systems [Harel 87][Shaw 92] and are widely used in industry. Formal methods are used for the verification of the correctness of designs, and rely upon the use of an underlying mathematical model.

Fault avoidance is an approach involving methods to remove all known or anticipated faults in the specification, design and development of computer systems. This approach aims to provide failure free performance through an error free design, and will always involve extra effort and cost in the specification, design, testing and validation of a system [Meyer & Pham 93].

Fault avoidance entails the use of design methodologies and the selection of techniques in order to avoid the introduction of faults during the design of a system [Anderson & Lee 81]. These include the use of proven technology, the use of structured methods to aid the design of large, complex systems and provide the ability to verify the correctness of designs. The modelling technique used in design should be able to represent system properties precisely and unambiguously and should allow a range of properties about a system to be proved. The technique should provide a mathematical rigour that inspires confidence in the proofs achieved.

The use of fault avoidance entails increased costs, in [Heitmeyer 94] the application of formal methods to the specification and verification of real-time systems is discussed and the conclusion reached that "formal methods are still not robust enough for general use, but necessary for certain applications. e.g. it cost $40M to formally verify 2K lines of code at the Darlington nuclear facility (where a serious mistake was uncovered), but the cost of an accident would have been much higher".

The use of fault avoidance as the sole means for preventing software failures is inappropriate, as it will result in a rigid system with no provision for the occurrence of operational failures. These can lead to unpredictable delays in system availability, and are a factor especially important in real-time systems that cannot miss deadlines [Levi & Agrawala 94].

To increase reliability fault avoidance cannot be relied upon alone as no large system can be expected to be error free, no matter how thoroughly it has been tested, de-bugged, modularised and verified there will still be residual design faults in the software [Pham 92]. Thus designers of real time systems, especially safety critical systems, must also look for approaches to hardware and software fault-tolerance [Kim 89].

Fault tolerance involves techniques for the design of systems so that they have the ability to function in the presence of certain faults. This approach is based on the premise that due to the complexity of systems, careful design and validation will never remove all operational faults and systems will always contain some residual design faults [Anderson & Lee 81][Levi & Agrawala 94]. The ability to measure the affect of fault tolerance techniques on the

reliability of a system has been studied since the earliest hardware fault tolerance approaches [Bouricius 71] and have been continued with software reliability models [Littlewood 79].

The availability of redundancy in processing or storage forms the basis of mechanisms for fault tolerance, and as such distributed systems offer a natural framework in which to implement this redundancy [Levi & Agrawala 94], in fact the use of fault tolerance and distribution have been said to go 'hand in hand' [Powell *et al* 91]. Fault tolerant schemes rely upon redundancy:

i) Hardware redundancy: the use of active or passive replicated subsystems.

ii) Software redundancy: the use of active or passive software module replication, combined with validity checks, self-test programs and watchdog timers.

Techniques for fault-tolerance involve extra cost, due to the replication and redundancy in software modules and hardware components. However, this extra cost must be weighed up against the potential cost of the system's failure [Pham 92].

There is an increasing adoption of distributed computer control in embedded real-time applications. This approach offers increased efficiency through the parallelism of processing involved. However, the reliability of communication between the components of a distributed system must be considered as these are essential to its correct functioning. In real-time systems the key factor is the ability to deliver a response to the environment within a deadline [Krishna & Shin 97].

As the communication overhead involved in distributed systems directly affect the systems response time, the use of communication systems that provide bounded message transmission times are required for hard real-time systems. Hard real-time systems that are also fault-tolerant are termed *responsive* systems [Malek 90].

In this thesis the term distributed system is used to refer to an interconnected collection of autonomous computers or processors that communicate in order to achieve a common goal. For a distributed system, when a processing site or a communication system failure occurs, fault tolerant techniques should allow the operational sites to detect this occurrence and take appropriate action (continue processing, initiate safe shutdown). In a distributed database this action may be to proceed with an operation and to update the failed sites local data when the site recovers. For the distributed control of a processing plant the appropriate action may be to initiate an emergency procedure that provides a safe shutdown.

In this Thesis the coordination of distributed systems is considered with the intention of developing methods for the software control and coordination of distributed embedded controllers. These distributed controllers are used to control concurrent physical processes such as manufacturing machines and processing plants. The loss of coordination in such systems can lead to costly damage to the environment. Therefore, fault avoidance is required in the design and development of the protocols used for this coordination. These protocols will need to be fault tolerant and provide a real-time response.

A protocol is the formalised set of rules that govern communication between the processes of a distributed system. Various protocols exist for reaching agreement between processes, their use depends upon the application involved, the type of agreement required, the communication method used and the required response to failures. The type of agreement required will depend upon the application involved, such as:

(i) consensus protocols, where a minimum number of processes are required to agree to an action. These can be used to maintain consistency of replicated data in a distributed database [Tel 94],

(ii) Byzantine agreement, the agreement of several processes in the presence of malicious or arbitrary faults [Lamport *et al* 82],

(iii) approximate agreement, the distributed processes agree upon a real value within a specified tolerance [Lynch 96],

(iv) clock synchronisation [Lamport & Melliar-Smith 85][Kopetz & Ochsenreiter 87][Christian 89] , for synchronising the local clocks of a distributed system within a required accuracy,

(v) election protocols, where several processes need to reach agreement about which is to assume a particular role, such as coordinator of a subsequent protocol [Sharp 94],

(vi) commit protocols, the agreement of several processes to commit or abort an action, used in transaction processing [Bernstein *et al* 87][Skeen & Stonebraker 83] and control applications [Hill 90].

Transaction processing is used in database systems to maintain consistency, by controlling access to data. A transaction provides the properties of atomicity, consistency, isolation and durability, termed the *ACID* properties [Sharp 94]. A transaction is an example of an *atomic action*, which have the following properties:

i) indivisibility, either all steps in the atomic action complete, or none of them do,

ii) serialisability, all computation steps not contained within the atomic action either proceed or succeed the atomic action,

iii) recoverability, the external effects of all the steps of an atomic action either complete or have no effect.

1.3 Aims & Objectives of Research

The aim of this thesis is to investigate and develop techniques for the specification and verification of distributed real-time control systems. These techniques should provide fault avoidance. The use of fault-tolerance in designs for such systems is also investigated and developed.

In choosing an approach for the modelling and analysis of distributed hard real-time systems certain important features of these systems needs to be taken into account. The distributed systems involve independent concurrent processes, so the ability to model and reason about concurrency is required. The interprocess communications used is integral to the design, and so the ability to model and reason about this is required as the coordination of the system is dependent upon this. In order to provide mechanisms for fault tolerance it is necessary to analyse the effect of failures on the system, the ability to include the possible failures types in the model [Leveson & Stolzy 87] is required.

The use of design and analysis methods that allow the combination of formal and informal components in a unified manner is recommended. The informality of the graphical approach providing ease of understanding, and the formal method providing for precise specification and verification. The approach should include analysis methods which can be used to determine the consistency and behaviour of the system being modelled [Peterson 81][Harel 87]. The combination of a graphical formalism with a consistent formal logic will allow formal reasoning about a design.

Petri nets are a powerful technique for the modelling and analysis of distributed and concurrent systems, as well as offering the basis for a unified approach to the design of such systems [Ramaswamy & Valavanis 96]. The analysis of Petri nets is generally based on generating the reachability graph, which involves enumerating the complete state space of a model.

However, as Petri nets models of systems increase in size the generation of the associated reachability graph and its analysis can become a difficult task due to the problem of state space explosion [Scholefield 90]. There are various approaches to this problem, such as the use of stubborn sets and the projection of sub-nets behaviours [Valmari 91, 93, 94], reduced reachability graph through net reductions[Murata 89], or the composition of sub-net reachability graphs[Bucci & Vicario 95].

Petri nets, along with finite state machines, have previously been used in the design and analysis of commit protocols [Skeen & Stonebraker 83][Yuan & Jalote 89][Hill 90] and distributed control systems. In this Thesis the Petri nets [Peterson 81][Reisig 85][Murata 89], Time Petri nets [Merlin & Farber 76] and the Temporal Petri net analysis of Suzuki & Lu [89] are used for the design of both commit protocols and real-time distributed controllers. The combination of these techniques yields an approach that offers both an informal graphical and formal approach required for describing hard real-time systems.

The work presented in this thesis is specifically concerned with:
i) investigating modelling techniques that can be used for the design and analysis of protocols and distributed real-time control systems.

ii) the development of a commit protocol that provides both a real-time and fault-tolerant response.

iii) developing a modular approach to the design and analysis of the commit protocols, the analysis method should reduce the cost of state space enumeration.

iv) applying the commit protocol and the modular design and analysis method to two real world distributed control applications.

1.4 Thesis Organisation

Chapter 2 presents a survey of the various methods used for coordinating distributed systems, with particular emphasis being placed on the commit problem in distributed real-time control systems. The inter-process communications employed and the expected types of failure modes for these systems are also discussed. Several types of commit protocols are presented and discussed in detail due to their usefulness in these applications, as they can be extended with deadlines to provide both a real-time response and fault tolerance.

A survey of the techniques that are considered suitable for the modelling of both protocols and hard real-time systems is presented in Chapter 3. An approach that is applicable to the

design and analysis of the combination of distributed controller, commit protocol and communication primitive in a unified manner is required.

Petri nets are identified as a suitable approach to these applications, and their use in the development and analysis of such systems is described in detail. The combination of this technique with Temporal logic in Temporal Petri nets is described, as this allows the formal verification of Petri net designs. The existing approaches to Petri nets design are then discussed.

In Chapter 4 the commit protocols investigated in Chapter 2 are modelled and analysed using the Petri net techniques identified in Chapter 3. Petri net models of several commit protocols are developed and analysed, and existing methods for the application of fault-tolerant mechanism to these protocols are then investigated.

Fault-tolerant techniques, based on the use of timeout guarded communications, allow the development of robust real-time commit protocols that are inherently recoverable. The protocol design is then analysed using reachability based Petri net analysis [Murata 89] and the Time Petri nets of Merlin & Farber [76].

A commit protocol design (equivalent to that presented in Chapter 4) is developed using a hybrid approach in Chapter 5. This approach is based on the top-down refinement of the basic protocol structure, combined with the bottom-up composition of reusable Petri net templates. These templates model low level interprocess communications, and are designed and analysed in isolation using a minimal representation of each modules environment.

The modular templates are based on Valette's well-formed blocks [Valette 79] and their application shows how they offer both an intuitive approach to the design of the protocols and reduce the cost of reachability analysis and formal verification.

The composition procedure preserves certain verified properties of each of the templates. The separate analysis and formal verification of the templates are then used in the analysis of the commit protocol developed. The design procedure is based on interprocess communications and is shown to be a natural way of structuring distributed processes.

In Chapter 6 the modular approach to Petri net design and analysis developed in Chapter 5, is applied to the controllers for two real-time distributed control applications. The distributed controller designs are composed using Petri net templates of the commit protocols developed in Chapters 4 and 5.

This allows the application and evaluation of the developed protocols, along with the an evaluation of the modular design and analysis procedure. The reusable templates of the protocols offer a natural approach to the composition of distributed processes that coordinate their actions using message passing communications.

In Chapter 7 the achievements of this research effort are summarised and evaluated. The contribution made in this Thesis are assessed and suggestions for further work are made.

Chapter 2 Coordination of Distributed Systems

2.1 Introduction

This Chapter considers methods for coordinating distributed systems. Methods applicable to real-time control systems are emphasised, as the coordination and control of high-speed machinery is of prime concern. The coordination of such systems are required to be fault tolerant and so the failure behaviour of processing sites and inter-process communications are considered. Commit protocols have previously been used for such applications, and these are considered in detail, as they can provide both fault-tolerance and real-time response when extended with timeouts to guarantee deadlines.

Distributed systems and their communication and failure types are considered in the rest of section 2.1: distributed agreement in section 2.2 and commit protocols in section 2.3.

2.1.1 Distributed Systems

The term *distributed systems* is used to refer to a collection of physically distributed processing sites and the communication sub-system by which they exchange information. Although there is no agreed definition of what constitutes a distributed system the following definition from [Sloman 87] shall be used in this thesis:

"A distributed processing system is one in which several autonomous processors and data stores supporting processes and/or databases interact in order to achieve an overall goal. The processes co-ordinate their activities and exchange information by means of information transferred over a communication network."

Within this Thesis the autonomous members of a distributed computer system will be referred to as *sites*, and these exchange information by message passing via a common communication network only.

Closely coupled systems are those that communicate via shared memory or direct connection, and will not be considered as distributed systems within this thesis. On the other hand *Loosely coupled* systems execute asynchronously and communicate using message passing via some communication network. Loosely coupled distributed systems can be characterised by the following:

i) Modular Physical Architecture. The system should consist of interconnected processing elements, these may be physically distributed and vary in number during the systems lifetime.

ii) Communication. The system should communicate by message passing only using a shared communication system. This communication is by co-operation rather than in a master-slave manner.

iii) System-Wide Control. This integrates the distributed autonomous processing units into a coherent single system [Sloman 87].

Distributed systems are considered as being either static or dynamically reconfigurable. Static systems have a fixed number of processing sites and application processes. Dynamically reconfigurable systems are able to support site and application processes that increases and decrease in number arbitrarily (within limits), in order to provide system flexibility and availability. An example of the former would be a distributed control system designed specifically for embedding within the environment to be controlled, where changes are unlikely. An example of the latter would be a large distributed database system, where changes in the overall system capacity and number of remote sites is likely.

2.1.2 Communication Model

The Communication network that interconnects distributed sites can be classified according to how messages are transmitted, i.e. whether point-to-point or broadcast message transmission is used. In point-to-point transmission a message is sent from one site to its destination site, possibly via several intermediate sites. In broadcast communications messages are sent to all sites on the communication network. In this research the use of point-to-point message transmission via directly connected sites is considered for the target system. This is applicable to locally distributed processing sites employed in the coordination of real-time distributed control applications.

Two basic types of point-to-point communication need to be considered, these are synchronous and asynchronous communication. The difference between the two is in the sender process. In synchronous communication the sender process does not continue execution until the message is received by the receiving process, thus achieving process synchronisation. The sender process using synchronous communications can have knowledge of whether the receiver process actually *received* a message. Protocols using this type of communication can be considered immune to message loss. Chapter 4. considers protocol design in the presence of processor and communication failures.

In asynchronous communication, the sender transmits its message and continues its processing immediately, there being no synchronisation required between the two processes. This allows processes using this type of communication greater opportunity for concurrent operation. The type of communication used is important for real-time applications:
i) Synchronous communication networks allow active sites to communicate within a known bounded time.

ii) Asynchronous communication networks do not guarantee any bound on the time needed to communicate between two sites [Christian 93].

Interprocess communication for real-time distributed control applications should be stable and fault tolerant. The protocols used should provide robustness for the system, this is a combination of reliability and availability, the ability to provide an acceptable service in the presence of failures. The use of synchronous point-to-point communication, forms an excellent basis for such systems as messages cannot be lost or received out of order, as can happen using asynchronous communication and local networks.

2.1.3 Failure Modes

In this section the type of failures that can occur in distributed systems are considered, these include failures affecting processing sites and communication networks.

A failure is the possible result of an error in the system [Anderson & Lee 81] and the consequences, depending upon the application, may be costly in financial terms and injury to personnel [Leveson 86]. The following terms are defined as they are used in the rest of the research:
i) *failure* - a failure has occurred when the delivered service no longer complies with the system specification (an agreed description of the system's expected function), [Laprie 85, 92]. This corresponds to the manifestation of an error [Levi & Agrawala 94].
ii) *error* - an error is that part of a systems state that is liable to lead to a failure[Laprie 85].
iii) *fault* - a fault is the cause or origin of an error, [Laprie 85], or has the potential of generating an error [Levi & Agrawala 94].

Faults are often classified according to their duration, as either transient or permanent:
i) transient fault - this is present in a system for only a limited threshold duration, a recurring transient fault is termed an *intermittent fault* [Anderson & Lee 81]. The life span of a transient fault is significantly shorter than the systems recovery time [Laprie 90, 92][Levi & Agrawala 94].
ii) permanent fault - this is present in a system for longer than the intermittent fault threshold, and is a potential source of errors for the systems life span [Levi & Agrawala 94].

Transient and intermittent faults can be treated as noise in a system, while permanent faults require some maintenance procedures to remove them [Anderson & Lee 81][Levi & Agrawala 94]. Although a fault must first manifest itself as an error in order to be detected, a system may contain potential or residual faults which never lead to a failure of a system, but have the potential to do so depending on the system state reached.

Techniques for fault tolerance are commonly based on the use of redundancy in processing or storage. This includes the use of replicated information in distributed databases to increase data availability in the presence of site failure. The use of distributed architectures often provides the redundancy required to implement fault tolerant schemes [Powell *et al* 91][Levi & Agrawala 94].

Faults in a distributed system may originate in the processing sites or in the communication network, and may be due to either a hardware (H/W) or software (S/W) failure. Thus fault tolerance in distributed systems is more complex than for single processing site case. Due to the requirement for co-operation between sites there are two potential sources of errors, the communication network and the processing sites. Distributed fault tolerance requires further techniques to take account of the potential errors due to this need for communication and co-operation between sites.

Processing site failures are considered to be less frequent than communication failures, but for safe operation of distributed systems they must be considered. Typical communication links are subject to various forms of unreliability and error-free communication cannot be realistically assumed. Assumptions must be made about the kinds of faults that can occur in order to create a reasonable model of the proposed systems behaviour. A *fault model* is one which describes the kinds of failures that are anticipated, and is used in order to analyse the behaviour of a system in the presence of such failures. A fault tolerant distributed system is one which can operate in the presence of such faults.

2.1.3.1. site failure

The complete failure of a processing site, whether due to H/W or S/W faults, can result in system deadlock. The term *deadlock* refers to the blocking of shared resources in a distributed system. The detection of such situations, along with the avoidance of problems that may lead to them, is required for a fault tolerant distributed system [Levi & Agrawala 94].

In [Powell *et al* 91] the behaviour of failed sites is classified as either:

i) Fail Uncontrolled (FU). Processing sites do not posses any local error detection and thus can: (a) omit or delay sending messages, (b) send extra messages, (c) send messages with erroneous content, or (d) refuse to receive messages. This type of behaviour may involve arbitrary actions being performed by a failed site. This is also termed *Byzantine failure*, as dealing with these types of failures is similar to solving the Byzantine generals problem [Lamport *et al* 82]. The Byzantine generals problem concerns a distributed system with faulty processing sites which are unpredictable, that is entirely inconsistent and possibly malicious.

ii) Fail Silent (FS). Where a site stops processing completely. Sites that exhibit this type of failure possess extensive self-checking mechanisms, such that any messages sent are guaranteed to be correct. Thus sites within a distributed system detect errors locally and shut down before they are propagated to others. The replication of S/W components locally serves for error detection and recovery.

In Bernstein *et al* [87] the term *fail-stop* is used to refer to the same type of behaviour in the presence of site failure. A site is considered to be either operating correctly or not working at all. This type of site never performs incorrect actions as it fails only by stopping.

Most processing sites fall somewhere between these two classifications, by offering some level of fault tolerance without being fully fail silent.

In dynamically reconfigurable distributed systems, detection of failed sites is required so that the system wide control can re-allocate tasks/processes to available error free sites. While in a static distributed system design the detection of faults is required in order to guarantee the safe behaviour of the system through the correctly operating sites, this could involve the safe shutdown of machinery.

Where processing sites cannot be guaranteed to provide FS behaviour an interface to the communication network that provides FS behaviour can be used. FU site are not suitable for shared communication networks as a failed site may prevent communication between other sites by 'babbling' or overloading a network. To implement FS sites in the distributed operating system DELTA-4, each site is split into a computation component, and a communication component, termed a *Network attachment controllers* (NAC) [Powell *et al* 91]. These provide FS behaviour and can be used along with FU processing sites, in order to provide 'Fail Silent' sites.

The NAC communication component provides error checking of communications, so that errors cannot be spread to other sites. This is based on the use of hardware self-checking

techniques. As the communication component ensures a FS site, the computation component can be either FS or FU depending upon the available host hardware and the criticality of the task [Powell *et al* 91].

2.1.3.2. communication failure

The communication network which interconnects the sites of a distributed system is implemented as a logical link and constitute a major potential source of error. The failure modes of which, according to [Levi & Agrawala 94], are:

i) Message loss effects - These can be harmful to the system unless precautions are taken, e.g. a message which transfers control of a resource from one site to another may cause the whole system to become blocked if it is lost.

ii) Message duplication effects - In a completely interconnected network a message may be sent by several routes to decrease the chance of it being lost and increase its robustness to link failures. However, measures to deal with duplicate messages are required, for example duplicate messages may increase a counter twice and invalidate the process executing. The use of idempotent algorithms, where the execution of a sequence of actions result in the same effect as if it was executed only once, protect against the duplication of messages.

iii) Message de-sequencing effects - If multiple routes for messages are used then the order of arrival of messages may not be guaranteed.

iv) Modification of the message control - Dynamic network control may change the type of administration in use, e.g. a change of mode may change the priorities of pending messages and thus affect the previously predictable behaviour of the network.

v) Bounded transmission time - A process may assume a bound on the transmission time of a message. This assumption may be extremely important in hard real-time systems, where late arrival of a message can be as catastrophic as loss of a message.

Communication faults depend upon the kind of underlying communication system used. For instance, a simple point-to-point link may inject random noise into the data it carries leading to lost or corrupted messages, but messages if delivered correctly, are delivered only once and in the correct order sent [Fischer 90].

The problems of message corruption should also be considered. A message may become corrupted at either end of/or during transmission, and may lead to the receiver taking

inappropriate actions. This is normally overcome through the use of integrity checks, where checksums on the message are included as part of the transmitted message.

2.2 Distributed Agreement

In distributed systems replication of resources, in processing or storage, is often used to increase the overall robustness. In distributed database systems properties of transactions ensure the consistency of replicated data [Sharp 94], while in distributed control systems the commit protocol can be used to ensure the consistency of operations [Hill & Holding 90][Hill 90].

2.2.1 Transaction Processing

In distributed database systems the *transaction* is used to perform the basic unit of work, such as a *'read'* or *'write'* to a record. Transactions exhibit the ACID properties of atomicity, consistency, isolation and durability. The transaction mechanism provides both 'failure atomicity' and 'concurrency atomicity'. Failure atomicity is the property that a database transaction must either complete or have no effect on a record, even in the presence of hardware and software faults. While concurrency atomicity is the property that the partial results of a transaction are not available to other transactions. The commit protocols developed in this research are based on the two-phase commit (2PC) protocol, which are adaptations of database transactions CCR requirement (Concurrency control, Commitment and Recovery), which are the mechanisms that provide failure atomicity. Commit protocols achieve this by ensuring a consistent distributed decision is reached.

Transactions use locks on data to achieve mutual exclusion, locking schemes can be adapted to increase concurrency by the use of hierarchical locking. Data objects are locked at various levels of abstraction. Failures that lead to deadlock are a serious problem in distributed systems, especially where locking schemas are used. This situation occurs when two or more transactions are blocked waiting for each other to release locks on resources (data). Deadlock can be illustrated with the following example from [Sharp 94]. Transactions *T1* and *T2* both operate on objects *A* and *B*, deadlock occurs when the following order of events occur:

i) T1 locks A,
ii) T2 locks B,
iii) T1 waits for locks on B to be released,
iv) T2 waits for locks on A to be released.

To avoid such situations deadlock detection schemes are used to abort transactions once a deadlock occurs [Gray 78].

In a distributed database system an atomic commit protocol (ACP) is a distributed commit protocol designed to be atomic. All processes either commit or abort the transaction, thus ensure the consistency of replicated data. ACPs must satisfy the following rules:
AC1) All processes that reach a decision reach the same one.

AC2) A process cannot reverse its decision once it has reached one.

AC3) The commit decision can only be reached if *all* processes voted *commit*.

AC4) If there are no failures and all processes voted *commit*, then the decision will be to commit.

AC5) Consider any execution containing only failures that the algorithm is designed to tolerate. At any point in this execution, if all existing failures are repaired and no new failures occur for sufficiently long, then all processes will eventually reach a decision [Bernstein *et al* 87].

2.2.2 Voting Schemes

In a distributed database where copies of the same data object are replicated at several sites, standard transactions synchronise access to copies of an item by requiring locks to be obtained on all copies in order to maintain consistency. An adaptation of this scheme is *majority voting* which requires a transaction to gather locks on a majority of copies of a replicated object before performing read or write operations.

A similar but more flexible approach is termed *weighted voting*. In this scheme every copy is assigned some number of votes, which may differ, and transactions need to collect a read quorum (r) of votes or a write quorum (w) of votes in order to access an object. In this manner access to an object to write can be restricted to one transaction at a time, while access to an object to read can be made less restrictive [Levi & Agrawala 94].

2.2.3 Clock Synchronisation

The successful co-operation of sites of a distributed system depends upon the systems ability to understand the functioning of its partners [Motus 92]. In a distributed real-time system recording the time of occurrence of an event which can be observed from different sites requires a common time reference amongst the sites. Clock synchronisation is the process of reaching distributed agreement on such a common time base. This agreement should be to a

required accuracy known as the synchronisation *tightness*, which is the amount that local clocks are allowed to differ from each other while still being considered in agreement.

A distributed *real-time* system requires two levels of synchronisation, internal and external, [Kopetz & Ochsenreiter 87]. *Internal Synchronisation* is the agreement of an approximate global time base amongst the sites of a distributed system, while *external synchronisation* is the synchronisation of this approximate global time base with the external physical time, i.e. the time in the physical environment with which the system interacts. In [Christian 89] it is noted that maintaining external synchronisation guarantees internal synchronisation, however the converse is not true as internal synchronisation may be maintained while drifting from the external physical time.

In general each site of a distributed system has a *physical clock* (H/W quartz clock) with a specified drift range in a non-faulty behaviour. It is due to this drift that some form of periodic re-synchronisation is required. The *granularity* of a physical clock is the period of this oscillation and limits the resolution of time measurement [Kopetz & Ochsenreiter 87]. Each site also maintains some form of *logical clock* , which is based on the value of the physical clock plus some offset. It is the values of the logical clocks which are adjusted in order to provide synchronisation [Lamport & Melliar-Smith 85].

2.2.3.1. Fault tolerant clock synchronisation.

[Kopetz & Ochsenreiter 87][Lamport & Melliar-Smith 85][Pfluegl & Blough 91a, 91b & 92][Pfluegl 92] and [Hoogeboom & Halang 92] all address the problem of providing fault tolerant clock synchronisation. The approaches differ in the algorithms used and the suggested use of extra hardware [Kopetz & Ochsenreiter 87][Christian 89] and [Hoogeboom & Halang 92]. The following are all factors that can affect clock synchronisation:

i) Communications delay. One of the main difficulties in synchronising distributed system's clocks is the unpredictability of communication delays. To overcome this a probabilistic approach to clock synchronisation is proposed by [Christian 89].

The probabilistic method of [Christian 89] is based on an asynchronous message passing communications network which allows sites to measure the round trip delay of a 'clock value request'. The assumption is made that the smaller the round trip delay, the smaller the clock reading error. Replies with a round trip delay greater than a set maximum are discarded to increase the precision of reading a set of remote clocks.

This is a probabilistic approach in that it does not guarantee reaching rapport with other sites, unlike the deterministic algorithms of [Lamport & Melliar-Smith 85] and [Kopetz &

Ochsenreiter 87]. However, it offers a greater synchronisation accuracy when agreement with other sites is reached.

ii) Communication failure resulting in missing clock value replies. In [Pfluegl & Blough 91b] if the number of clock estimates received in response to a sites request, is less than a certain threshold, the convergence function returns the local clock value in order to increase the fault tolerance of the algorithm. In the case where no clock values are received for a particular site then the estimate is replaced by a unique value *nil* in the convergence function.

iii) Byzantine failures resulting in two-faced clocks. Lamport & Melliar-Smith [85] define *two-faced* or *Byzantine* clocks as clocks that present different values to different sites. The clock synchronisation algorithms presented in [Lamport & Melliar-Smith 85] are adapted from algorithms that solve the *Byzantine Generals* problem [Lamport *et al* 82].

iv) Physical clock drift. In normal behaviour a physical clock has a specified drift rate, if the drift rate exceeds this the physical clock/site can be considered to exhibit faulty behaviour.

The sliding window algorithm (SWA) presented in [Pfluegl & Blough 91a], can tolerate bursts of intermittent or transient faults that affect many or all clocks. This is in contrast to [Lamport & Melliar-Smith 85], whose algorithm cannot bring clocks back into synchronicity once they have passed beyond the boundary of a *good* clock. Another difference to the algorithms of [Lamport & Melliar-Smith 85] and [Kopetz & Ochsenreiter 87], is that the SWA can bring clocks into synchronisation without the assumption of initial synchrony.

2.2.3.2. Clock synchronisation and system performance.

The techniques of [Kopetz & Ochsenreiter 87][Lamport & Melliar-Smith 85] [Pfluegl & Blough 92][Pfluegl 92] and [Christian 89] all require an overhead in processing (*CPU Load*) and message passing (*communications*), and as such there is normally inherent in these techniques some trade-off between synchronisation accuracy, fault tolerance and system performance. Measures to guarantee fault tolerant clock synchronisation normally effect:
i) The achievable synchronisation accuracy, and

ii) The quality of the service provided by the system.

To decrease the processing overhead and the amount of message passing required, [Kopetz & Ochsenreiter 87] and [Christian 89] both proposed the use of a subset of sites to perform the synchronisation task. In [Kopetz & Ochsenreiter 87] the use of a master site in a distributed system that would maintain the necessary external synchronisation is also

proposed and the use of a H/W clock synchronisation unit (CSU) is presented, the CSU is an accurate physical clock designed to provide a high degree of external synchronisation, and reduce the CPU overhead and network traffic required to maintain synchronisation to less than 1% of the systems operation.

In the above section it can be seen how several factors need to be considered in order to achieve this form of distributed agreement (clock synchronisation). Failure assumptions and the fault model used affect the requirements of the algorithms and their assessment. The available hardware (such as the use of fault tolerant clock synchronisation units), the underlying communication system (are there bounds on message transmission times, can messages be lost) and the processing overhead required are all factors that must be considered when deciding upon the approach used.

2.3 Commit Protocols

The commit protocols developed in this research are based on the 2PC protocol. This is an adaptation of the database transactions CCR requirements, using only the mechanisms that provide failure atomicity. The centralised versions of this protocol are static schemes with pre-designated coordinator and participant processes. The decentralised commit protocol is used where any process may initiate a request for commitment. The standard centralised 2PC is based on a coordinator and a set of participants. The coordinator site, where the transaction originates, is responsible for orchestrating the transaction.

2.3.1 Blocking Commit protocols

A major distinction in commit protocols can be made between *blocking* and non-*blocking* protocols, which affect the two conflicting properties of *consistency* and *availability* for distributed systems. The relative importance of these two properties will depend upon the application.

A blocking protocol is one where, due to a site failure, a consistent decision cannot be made by the active sites until the failed site recovers. In a database this type of transaction will reduce data availability because any locks on data will be held until the failed site recovers and the transaction can terminate successfully. This approach will guarantee strong data consistency between sites at the expense of availability, as other transactions may not be able to proceed once one transaction has failed. In systems which require high availability, such as real-time systems, it may be more practical to have temporary inconsistency between sites in order to make data or resources available. This would allow the required response times to be satisfied[Skeen & Stonebraker 83].

The 2 phase commit illustrated below, is a blocking protocol. The centralised form is detailed, with one site acting as the coordinator, whose job it is to ensure that all sites reach a consistent decision, and the other site/s as the participant/s. The blocking 2PC Protocol, offering low availability & high consistency takes the following form:

i) *Coordinator*

Phase 1 a ready log file is written
ready messages sent to each participant
wait for participants to reply with their votes

Phase 2 when all participants have replied a decision is made
a log file of the decision is created
the decision is sent to all participants

ii) *Participant*

Phase 1 wait for *ready* message
if participant can commit send *yes* else *no*
write yes log file or no log file and abort

Phase 2 either *commit* or abort *message* is received
write decision log file
execute decision

Blocking protocols are unrecoverable using only local information. Communication with the surviving sites is required in order for the recovered site to terminate the failed protocol consistently. Blocking protocols are not considered applicable to real-time systems as sites do not have bounded recovery times.

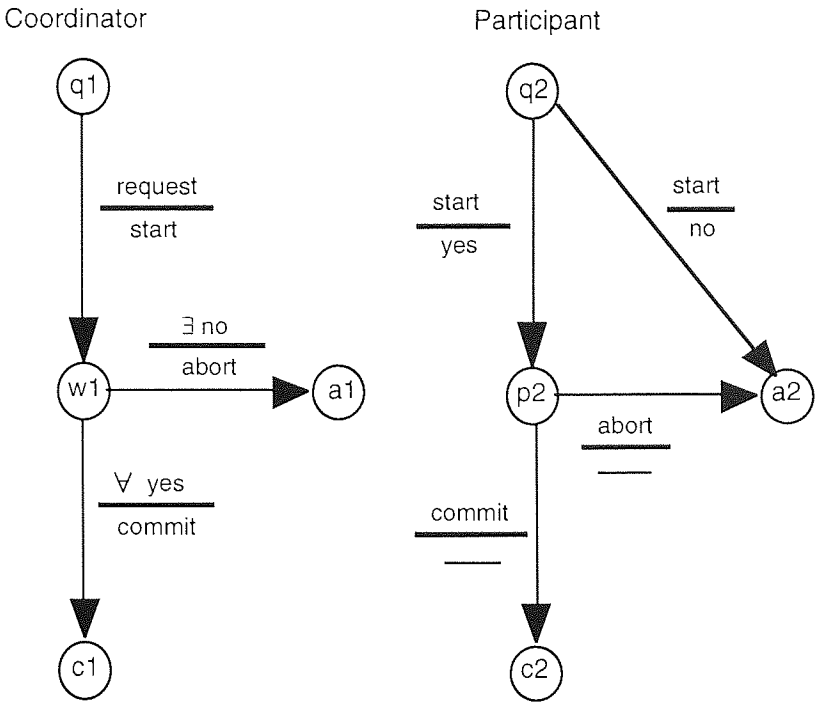


Fig.2.1. Two-phase commit protocol

Fig.2.1. is a finite state machine (FSM) model of the two phase commit protocol. An informal interpretation of the FSM is as follows; The circles denote the states and the edges represent the state transitions. The labels associated with the edges and above the bar denote the reason for the state change (such as a received message) and those below the bar denote the action that is the result of the state transition (such messages sent). A formal definition of state machines is given in section 3.2.

The 2PC protocol is the basic commit protocol, the two phases are the *voting* phase and the *decision* phase. During the voting phase one party to the protocol, known as the *coordinator* of the commit protocol, asks all the other *participants* to prepare to commit. If for any reason the participant is unable to commit then it sends a *No* vote to the coordinator, otherwise if a participant can guarantee to perform the outcome requested a *Yes* vote is sent. During the decision phase of the protocol the coordinator propagates the outcome of the protocol to all the participants. If all the participants voted *Yes* then the commit outcome is propagated. If any participant voted *No* then the abort outcome is propagated to each participant that voted *Yes*. Each participant of the protocol commits or aborts the effect of the transaction locally based on this outcome [Samaras *et al* 95].

To tolerate site failures each site keeps a local log of decisions in stable storage for recovery purposes. In the case of a site failing during a commit protocol, upon recovery the log is interrogated in order to complete the transition, the coordinator will append a commit/abort decision record in its log. This is the decision point of the protocol, before this point the transaction is termed *undecided*. Participants record their own decisions and the coordinators

decision in their local logs. The undecided state is where a recovering site has a prepared log but no decision log [Triantafillou 96].

2.3.2 Non-Blocking Commit protocol

A commit protocol is termed non-blocking if, in the event of a failure, the operational sites can be terminated without waiting for the recovery of the failed sites. The 3-phase commit (3PC) and the extended 2-phase commit (E2PC) are both adaptations of the basic 2PC, which produce non-blocking behaviour.

These protocols provide high data availability, because data is never locked indefinitely waiting for a failed site to recover. However, there may be temporary inconsistencies between the failed site and operational sites, while all operational sites remain consistent. It should be noted that non-blocking protocols generally require about 50% more communications overhead than corresponding blocking protocols [Dwork & Skeen 83]. However, non-blocking protocols are applicable to real-time applications as all operational sites can be guaranteed to terminate within a known deadline.

The non-blocking 3 Phase-Commit Protocol offers high availability at the cost of strict consistency, and takes the following form:

i) *Coordinator*

Phase 1 — *start* message sent to all participants, and the coordinator enters wait state.

Phase 2 — votes are collected, any *no* then an *abort* command is sent, else a *prepare to commit* message sent. The coordinator enters prepared state.

Phase 3 — The coordinator collects participants acknowledgements *ack*, sends *commit* messages and commits the transaction.

ii) *Participant*

Phase 1 — This is as for 2PC, either a *yes* or *no* vote is returned to the coordinator after the *start* message is received. If no then the participant unilaterally aborts, otherwise it enters a wait state.

Phase 2 — Wait for command *abort* or *prepare to commit*, and enter abort or prepared state is entered. Acknowledgement of transition to prepared state sent to the coordinator.

Phase 3 — The final *commit* command received to commit the local participant.

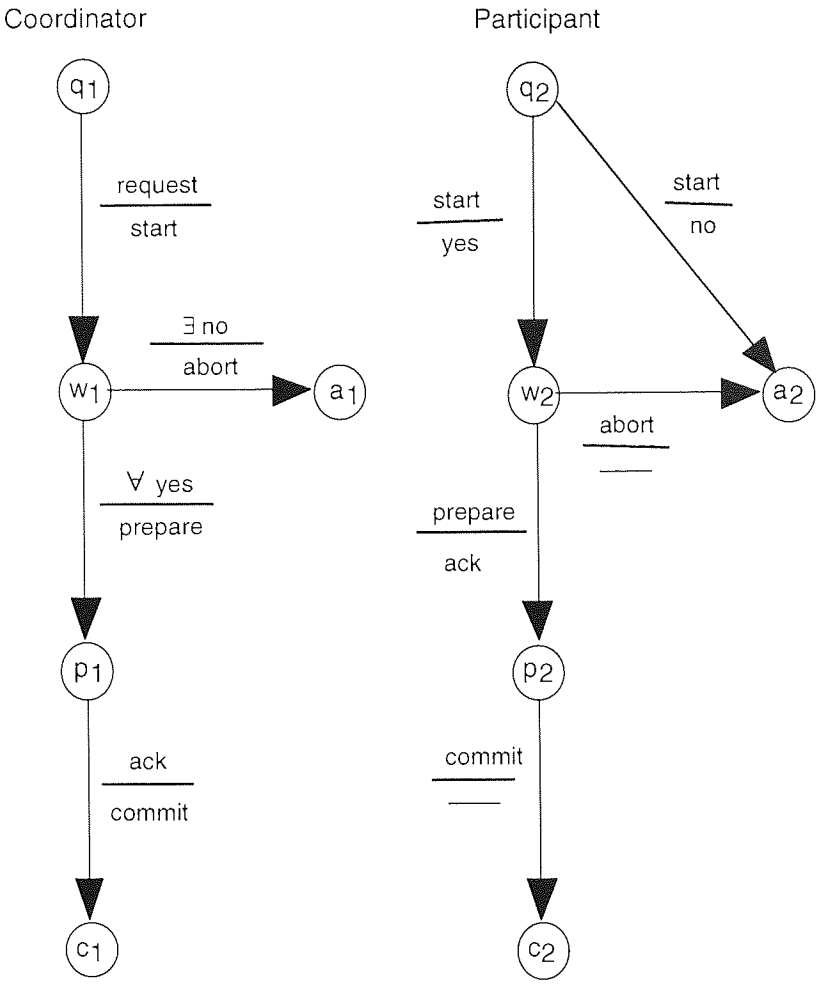


Fig.2.2. Three-phase commit protocol

The 3PC commit is based upon the 2PC with the addition of buffered states in both the coordinator and the participant. This protocol is non-blocking, but not independently recoverable (section 2.3.3.). This protocol has the same states for the centralised implementation and the fully decentralised form, although the decentralised form involves extra rounds of messages. Decentralised versions of the 3PC that elect leaders are possible, but inherently more complicated. These require sub-protocols to be invoked, such as election protocols [Garcia-Molina 82], in order to designate coordinator and determine the last process to fail.

2.3.3 Independent Recovery in Commit protocols

An important property of commit protocols is *independent recovery*, which is a property of a site recovering from a failed state. Independent recovery [Bernstein *et al* 87] is the ability of a recovering process to reach a decision consistent with the surviving processes without communication. The extended 2 phase-commit protocol (E2PC) is a non-blocking version of the 2PC that can be made independently recoverable, using a prepared state in the coordinator only. The 2 and 3-phase commit protocols cannot be made independently recoverable.

The E2PC protocol offers high availability at the cost of consistency between operational and failed sites, prior to recovery.

i) *Coordinator*

Phase 1 *start* message sent to all participants, coordinator enters wait state.

Phase 2 votes are collected, any *no* then an abort command is sent, else *commit* message sent, coordinator enters prepared state.

Phase 3 Coordinator collects acknowledgements and commits the transaction.

ii) *Participant*

Phase 1 as for 2PC, either a *yes/no* is returned to Coordinator after the ready received. If *no* then participant unilaterally aborts, otherwise enters ready state.

Phase 2 wait for command *abort* or *commit*, abort or commit state entered. acknowledgement' sent to coordinator, if 'commit' received.

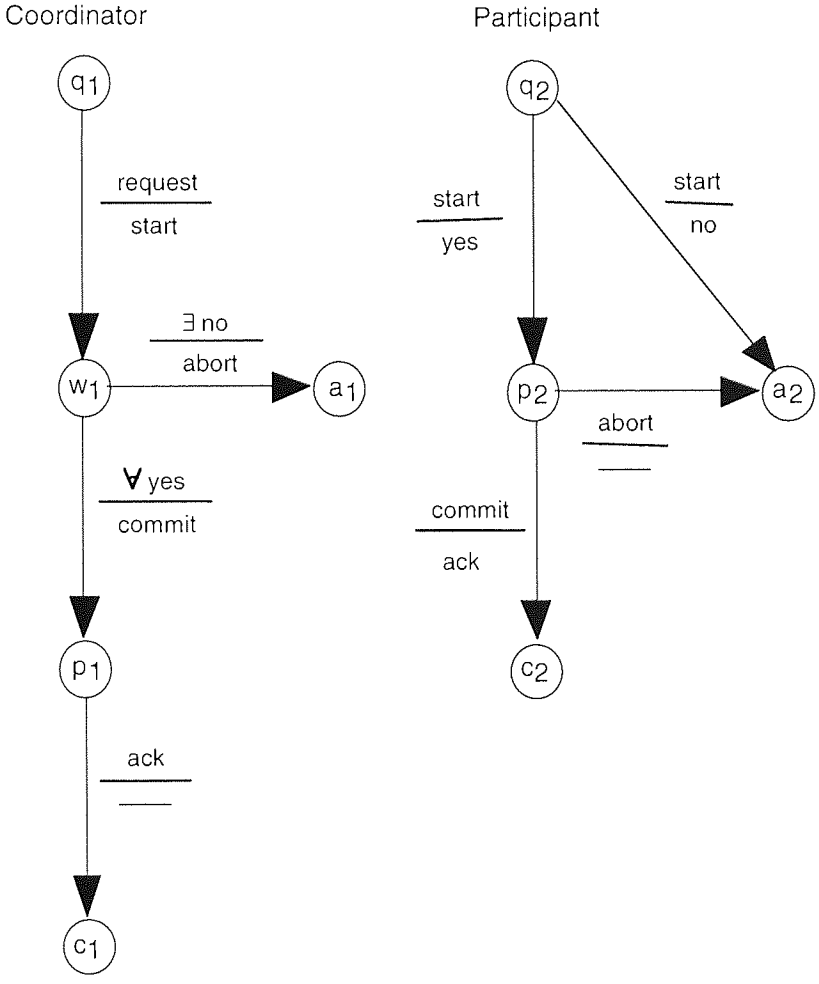


Fig.2.3. Extended two-phase commit protocol

A protocol that uses independent recovery would be applicable to real-time systems, because deadlines could be applied as the recovery time of operational sites from a failed transaction would be deterministic. The effect being that each partition would operate correctly but independently and achieve consistent states for each partition. This is important for real-time control as link failures are more likely than processor failure, due to the hazardous environment for links.

In [Skeen & Stonebraker 83] two theorem concerning independent recovery are given:

Theorem 1) "There exists no protocol resilient to a network partitioning when messages are lost", as this is a possible situation using asynchronous communications, the use of synchronous communications would be recommended.

Theorem 2) "There exists no protocol resilient to multiple partitions".

These theorems were used in [Hill 90] in order to make a version of the non-blocking E2PC protocol that was independently recoverable in the presence of a single partition, through the use of timeout guarded communications.

To determine if a protocol can be made independently recoverable it is necessary to examine the states other parties may occupy when a site fails. If it is impossible to determine the actions taken by operational sites without extra communication the recovering site may not terminate consistently and an independent recovery scheme is not possible [Levi & Agrawala 94].

The scheme for independent recovery proposed in [Skeen & Stonebraker 83] is applicable to situations where there is a single site failure in a two site system. This restriction on Skeen's rules to systems with a limited number of sites is due to the system model used, which assumes the atomic broadcast of messages to all sites. However, the coordinator site could fail after sending a message to only a partial number of participant sites. This would result in some participant sites receiving the message from the failed site, while others timeout the communication. Thus participant sites receiving the messages will have a different view of the failed coordinator site from those participants who took a timeout action.

In [Yuan & Jalote 89] a method is presented to overcome the problem of correct protocol operation with multiple site failures. This scheme uses independent recovery where possible and communication by recovered sites if independent recovery is not possible. This approach is based on the use of an exceptional states termed *recovery* states. A failed site moves to a

recovery state upon recovery, and if possible performs independent recovery to a final state. Communication with the operational sites is used when independent recovery is not possible. A scheme to determine which recovery states are independently recoverable is presented [Yuan & Jalote 89]. This scheme avoids the election of backup coordinators to replace failed coordinator sites, and the possibility of site failures during termination protocols leading to the domino effect.

A termination protocol is a recovery scheme that uses voting by the operational sites in order to elect a new coordinator in the centralised protocol forms, or to reach a consensus decision in the decentralised form. Protocols that can handle multiple site failures, but are not independently recoverable, are possible. These protocols use termination protocols in the event of failures [Yuan & Jalote 89]. These schemes can require many rounds of message passing between operational sites if further failures are encountered during the voting procedure.

A method for independent recovery applicable to systems with more than two sites is presented in [Yuan & Jalote 89]. This approach uses two responses to a timeout on a communication occurring, either timeout to abort or to commit. This decision is propagated to all other operational sites in order to maintain consistency. This message propagation by sites that detect a timeout ensures that all operational sites have the same response to a failure. In the case of a coordinator failure, the number of *timeout* messages propagated by participants will be dependent upon the number of messages sent prior to the coordinators failure. However, in the non-failure case the number of messages sent during the protocol is unchanged. The variable number of messages sent in the failure case would make this approach impractical for real-time systems. It should be noted that this approach is only applicable to systems with a single site failure and no link failure.

In [Carpenter 92] a distributed serial commit protocol is presented, this is a form of the E2PC where the coordinator communicates with the participants in a predetermined order. This allows a daisy-chain structure to be used for the propagation and collection of messages. Samaras *et al* [95] describe a similar approach with dynamic allocation of coordinator/participant, called the cascaded coordinator approach, requests are passed from coordinator to participant to further participants (the cascaded coordinator propagates messages downstream, and collects responses and sends them upstream). A participant in this approach does not know if its coordinator is the root of the commit tree or a cascaded coordinator.

2.3.4 Extensions to Commit protocols

The performance of the commit protocol has a substantial effect upon the transaction volume that a database system can support. For transaction processing applications the commit processing takes up a substantial part of the transaction time. It has been shown that the commit part of a transaction typically takes a third of the transaction time[Samaras *et al* 95], in distributed systems the relative time is much higher. A faster commit protocol could improve transaction throughput in two ways, firstly by reducing the duration of each transaction, and secondly by releasing locks on resources earlier, thus reducing the wait time of other transactions [Samaras *et al* 95].

Samaras *et al* [95] present several two-phase commit optimisations that are currently used in commercial database systems. Optimisations that are intended to improve performance in the normal case are shown on the left hand side of Fig.2.4. These optimisations are based on two assumptions:

i) that networks and systems are becoming more reliable,

ii) that the need to support high-volume traffic requires a streamlined protocol in the normal case.

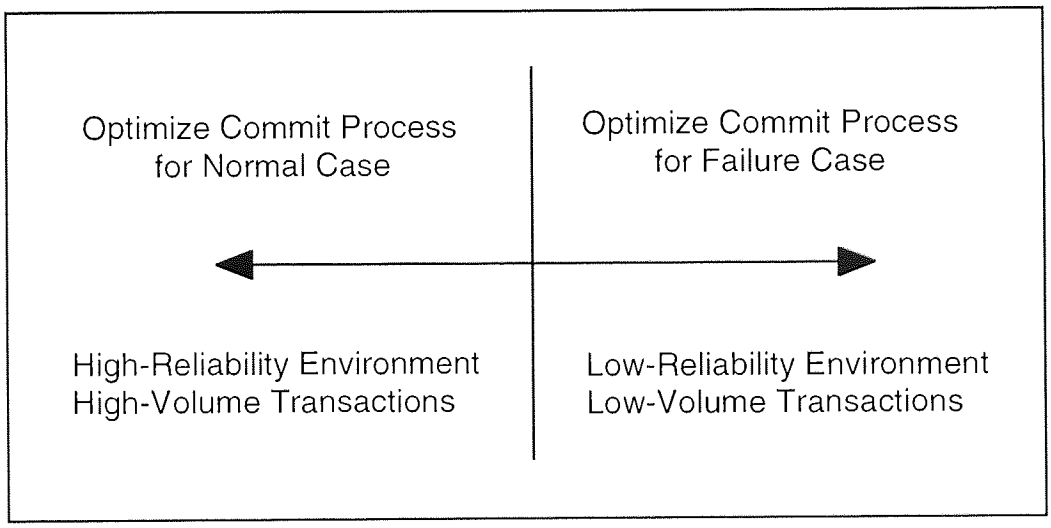


Fig.2.4 Commit performance optimisation for different environments

The 2PC protocol can be optimised according to two different approaches illustrated in Fig.2.4. The first approach concentrates on reducing the recovery time for failure cases. A lot of research effort has gone into providing the non-blocking property to commit protocols under certain failure modes. Extra messages have been added to the basic 2PC protocol in order to reduce the blocking delay required to resolve the transaction outcome following a failure.

This approach leads to a slowing down of the normal (non-failure) case in order to prevent unacceptable delays following failures. The trade-off (between reducing recovery time in

failure cases at the expense of increasing duration of normal commit operations) may not be acceptable in a highly reliable environment characterised by high volume transactions. The other approach is based on reducing the number of messages and/or local processing required for the non-failure case, sometimes at the cost of greater recovery processing and subsequent delay for the failure case.

The *Optimistic* two phase commit (O2PC) protocol [Levy *et al* 91] is an ACP that is based on the optimistic assumption that failures are rare. In the O2PC protocol participants after sending their votes go ahead and commit, instead of waiting for a decision in reply from the coordinator. This would violate the atomic commitment rules (AC1..4 section 2.2.1) if any other member of the transaction aborts. However, this is overcome by the use of a *compensation* transaction which undoes the local effects of a transaction without resorting to cascaded aborts.

This form of commit protocol is based on the assumption that the probability of O2PC protocol terminating unsuccessfully is very low. This results in an effective increased throughput through the early release of locks on data. Compensating transactions are used to undo the local effects of a sub-transaction, as well as handling situations where there is a requirement to undo the effects of a committed sub-transaction whose results have been used by other transactions. O2PC along with compensating transactions ensure the eventual atomicity of transactions [Levy *et al* 91].

In [Desai & Boutros 96] an adaptation of the standard 2PC protocol termed the *Prudent* two phase commit protocol (P2PC) is presented. This protocol attempts to avoid aborting transitions needlessly in the presence of failures, by giving the protocol a second chance to commit before deciding to abort. The second chance could save the transaction from an abort result when a transient communication error occurs, this would improve system performance and reliability. However, the P2PC is still a blocking protocol and is intended for environments where total failures are rare, but transient errors may be common.

The requirements for the handling of failures determines distributed databases commit protocol choice. It is not generally possible to achieve global atomicity in the presence of failures whilst also preserving complete local autonomy [Yoo & Kim 95]. Three distinct approaches to maintaining global transaction atomicity in the presence of failures exist:
i) Redo - Where the actions of the failed local part of a transaction (*sub-transaction*) are completed by executing a *redo transaction* that repeats the write operations of the sub-transaction only [Bernstein *et al* 87][Soparkar *et al* 91].

ii) Retry - The entire read and write actions of the aborted sub-transaction are repeated.

iii) Compensate - At each site where a sub-transaction of a global transaction committed, a compensating transaction is run to semantically undo the effects of the committed sub-transaction. [Levy *et al* 91]

The standard 2PC treats transient communication failure as a catastrophic communication or site failure, this leads to abort decisions which affect the system throughput and fault tolerance. Neither the O2PC or the P2PC distinguish between site and communication failure. Communication failure may cause a message to be lost, the second chance before aborting offered by the O2PC should compensate for most lost messages. The O2PC is more resilient to transient failures than standard 2PC, however this is achieved at the cost of additional rounds of messages and the associated increase in the number of timeouts required. In [Desai & Boutros 96] it is noted that in the absence of failures the P2PC is equivalent to the standard 2PC and is suitable for environments with rare communication failures but high site failures. The design of the P2PC is suitable for environments where the communication failure rate does not exceed the site failure rate.

The P2PC is suitable for situations where site or communications failures do not occur or the probability of occurrence is very low. To implement P2PC it is also required that compensation transactions can be used without cascading aborts throughout the entire system in a domino effect [Desai & Boutros 96].

The Reliable Two-Phase Commit (R2PC) of Yoo & Kim(95) guarantees fault-tolerant global atomicity in multi-database systems that do not provide the facility for an explicit prepared state. The R2PC is used for guaranteeing global data consistency and is not applicable to systems where communications and total site failure must be tolerated. Late messages and site failures in the R2PC are handled using timeouts that result in local locks on data, the re-sending of enquiry messages to determine the protocol outcome and the compensation of aborted transactions. The uses of compensation protocol, to repeal committed transactions that need rolling back, is done in a way that avoids cascaded aborts. This is achieved using a delayed-lock-release on data. The R2PC protocol includes termination and recovery procedures and is non-blocking due to site failures during a recovery procedure. However, the R2PC is blocking if a timeouts occurs in the commit state of a participant.

The O2PC [Levy *et al* 91] uses a compensating transaction to achieve atomicity, and has no prepared state. This solves the blocking and the delay problem, but the persistence of the compensation cannot be implemented according to Yoo & Kim [95].

The 2PC protocol is optimised in VAX cluster transactions, both coordinator and participants have access to a consistent log during a site failure [Samaras *et al* 95]. Coordinator migration is also used to improve reliability by transferring the commit decision to reliable partners [Gray & Reuter 93].

Triantafillou [96] attempts to resolve the commit uncertainty of recovering sites for large scale distributed systems. The non-blocking property is achieved by identifying uncertain data and marking them with an *uncertain* flag. This data is then dealt with in either a pessimistic (uncertain data treated as aborted) or an optimistic (uncertain data are treated as committed) manner.

Pre-transaction images are kept to allow recovery from incorrect assumptions. The notion of quorums are also used, where a transaction can be committed if performed on a specified quorum size of sites, where some sites fail to complete. The probability of Blocking is reduced by replicating each sites recovery log to a predetermined group of replicated sites. The extra cost of appending the log is high as an extra two phases are now required, in effect making this somewhat similar to the four-phase commit protocol. However, this append log operation may itself cause blocking [Triantafillou 96].

2.3.5 Responsive Commit Protocols

Communication protocols which perform recovery from any abnormal state to a normal state are called *self-stabilising* protocols [Gouda & Multari 91], these protocols offer a recovery function without the requirement for exception handling routines when loss of coordination occurs.

Real-time recovery functions are required for continued processing of distributed real-time systems when faults occur in the underlying communication system. Systems such as these that posses both real-time and fault-tolerant performance are termed *responsive* systems [Malek 90]. Kakuda *et al* [92, 94] define responsive protocols as ones where responsive systems technologies are applied to communication protocols.

In [Kakuda *et al* 94] responsive protocols are presented where recovery to a normal state from an abnormal one in real-time is included. The abnormal state can be caused either by a delay in communications, transfer error of messages or transient failure in the communicating processes. The maximum time taken for recovery from abnormal to normal state using these protocols is computed using an extended finite state machine model (section 3.2.) of the protocol, and real-time recovery is verified by comparing the maximum recovery time with a deadline given by the protocol design requirements.

Deadlock can occur when a participant to a protocol is waiting indefinitely because of some site or communication failure. This can be overcome where synchronous communication are used by the setting of timeouts [Hill & Holding 90][Hill 90]. However, the number of timeouts required to implement this for 2PC and E2PC protocols are quite large, as every input and output must be bounded with a timeout mechanism.

Fault-tolerant commit protocols offer two basic forms of fault tolerance, forward and backward. The occurrence of a failure during the execution of a transaction, but before the final commit phase, should result in the transaction being aborted. This should return the system to the state it was in before initiating the transaction and is a form of backward *error recovery*. When a failure occurs before the end of a transaction, but after the commit phase has been reached, the transaction must ensure that all actions are completed. This latter case is a form of forward *error recovery*.

2.3.6 Commit Protocols in Real-time Control

The addition of timeout guarded communications and recovery procedures can yield commit protocols that are applicable to real-time applications. Several timed commit protocols have been presented [Kopetz & Grunstedl 93, 94][Davidson *et al* 89] and [Gheith & Schwan 89] in the literature, although all of these rely to some degree on the support of the real-time implementations of the underlying communications system.

A version of the standard 2PC adopted for real-time environments is presented in [Davidson *et al* 89a], with particular emphasis on real-time distributed control applications in [Davidson *et al* 89b]. The problem of coordinating all or nothing behaviour under timing constraints is called *timed atomic commitment* for hard real-time control applications. These are where components execute concurrently on distributed sites in order to perform a control task. This approach extends the outcome of standard 2PC protocol (*commit*, *abort*) with an *exception* state which can provide a safe alternative to an inconsistent decision.

The standard 2PC only requires parties to *eventually* commit or abort, there is no explicit deadline by which decision and action must be completed. Timed correctness criteria require intermediate deadlines on the phases of the protocol , Dv, Dd, Dp shown in Fig.2.5. The use of these intermediate deadlines allows early detection of faults.

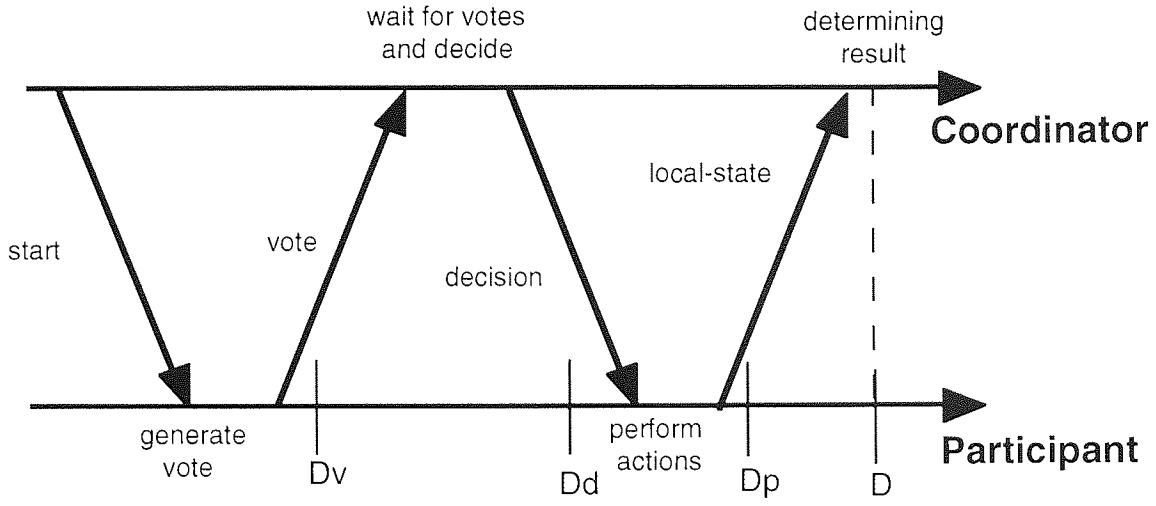


Fig.2.5. Messages in a CT2PC Protocol

This notion of *timed atomic commitment* (TAC) has an element of a *look-ahead* decision. An atomic action does not proceed unless there is knowledge, that in the absence of failure, it will be able to complete the required action by the deadline. An understanding of the time taken to complete the task is used in the commit decision. This is as opposed to the simplistic approach of enclosing an atomic commitment statement in a deadline. Davidson *et al* [89a] state that it is impossible to place a deadline on a traditional atomic commitment if processor failure or message loss can occur.

This approach allows the outcome of timed atomic commitment to be either:
i) all actions complete within the deadline,

ii) no actions are performed, or

iii) the system is in an exceptional state indicating that a fault has caused timing constraints to be violated.

The notion of a global clock is used to measure the deadline for commitment (D) (the skew of this clock (the difference in local times) is used in the calculation of timeout values). The Timed Correctness Criteria (TCC) are:
TCC1) all parties reach the same decision.
TCC2) decision is commit only if all parties vote *yes*.
TCC3) at D, the parties local state either reflect the completed action or is EXCEPTION.
TCC4) in fault free case,
- a) all parties each decision,
- b) if parties vote YES then decision is to commit
- c) all parties complete the decided upon action by D,
- d) a parties local state reflects the parties completed action.

To achieve TAC, *temporal scopes* [Lee & Gehlot 85] are used to express time dependent behaviour. A temporal scope consists of a start time and a deadline, optional statements that are to be performed in this interval, and an optional exception handler. The TAC approach relies on an operating environment that provides bounds on message delays, clock synchronisation, and guarantees resources.

In CHAOS^{art} [Gheith & Schwan 89] real-time transactions are used at the heart of a real-time operating system. The invocation of real-time atomic transactions form the basis for programming embedded real-time applications. The transactions used are objects with guaranteed timing, consistency and recovery attributes.

2.4 Conclusion

In this Chapter methods for coordinating distributed systems were considered in section 2.1., and the 2PC based protocols were identified as applicable to this research. The basic forms of this protocol were then described in section 2.2. The main features of the commit protocols considered are summarised in Table 2.1.

Protocol	2PC	3PC	E2PC	O2PC	P2PC	R2PC	CT2PC
Blocking	√	×	×	×	√	timeout	×
Non-Blocking	×	√	√	√	×	during recovery	√
Independent Recovery	×	×	√	×	×	×	×
Timeout	×	×	√	√	√	√	√
Transaction Volume	-	-	Low	High	-	-	Low
Consistency	High	Low	High	Low	Low	High	√
Availability	Low	High	High	High	High	Low	High
Use of Sub-Protocols	×	decentralised	×	√	√	√	√
Optimised for success	-	×	×	√	×	×	×
Optimised for failure	-	√	√	×	√	√	√

Table 2.1 Summary of commit protocols

In section 2.3 the 2PC optimisations for database systems were evaluated in terms of the reduction in network traffic, reduction in the number of logs written and decreased resource lock times. However, these factors can all effect reliability. The responsive commit protocol optimisations were of particulate interest, as these provide the fault tolerance and real-time response required.

The 2PC protocol was emphasised as it is becoming the standard for distributed transaction processing [X/Open 91] [ISO 92]. Therefore extensions to 2PC protocol were investigated as this atomic protocol forms the basis for those used in used in industry and academia.

The type of communication used affects protocol design and resiliency to failures, as noted in [Skeen & Stonebraker 83]. Commit protocols cannot be made resilient to multiple partitions or message loss, this is equivalent to the Byzantine generals problem [Lamport *et al* 82]. Therefore protocols that can tolerate a single site or communications link failure are considered, as link failures are expected to predominate in distributed real-time control.

The commit protocol is required for the coordination of distributed control systems with hard real-time constraints. The failure of coordination in such systems can lead to hazardous states resulting in damage to the systems environment. Site and communication failure can lead to these coordination failures. These types of failures must therefore be anticipated in the design of mechanisms for the control of distributed systems, and mechanisms used to provide fault tolerance. Techniques that provide fault avoidance must also be employed in the design

The available techniques for modelling and analysing functional and timing properties of commit protocols and distributed control systems are investigated in Chapter 3. The approach chosen should be unified, allowing the commit protocol to be modelled and analysed as an integral part of its target system.

The E2PC protocol is investigated and a responsive version is developed in Chapter 4, using the modelling and analysis techniques considered in Chapter 3. This commit protocol is considered suitable as it can be made non-blocking , as well as subject to real-time deadlines when enhanced with timeout mechanisms.

Chapter 3 Techniques for Modelling Distributed Systems and Protocols

3.1 Introduction

The following Chapter presents a survey of techniques suitable for the modelling, design & analysis of distributed systems and protocols, and evaluates their suitability for application to real-time systems. Particular emphasis is placed on techniques that can offer a unified approach to both the controllers, for the high speed machinery of interest, and the underlying interprocess communication protocols that define their interaction.

This Chapter investigates techniques currently used for modelling real-time systems in an attempt to choose the technique most suited to this research work. These include State Machine based techniques [Ostroff 89, Shaw 92], Petri Net based techniques [Peterson 81, Reisig 85, Murata 89], Temporal Logic based techniques [Manna & Pnueli 92, Jahanian & Mok 86, Ostroff 89], process algebras [Hoare 85][Milner 89]. This chapter also investigates techniques that have been used particularly for the treatment of protocols [Suzuki *et al* 90, Turner 93, Ardis *et al* 96]. These include SDL [CCITT 92], LOTOS [ISO 89b] and ESTELLE [ISO 89a], within protocol engineering these techniques are generally referred to as Formal Description Techniques (FDTs) [King 91]. Each approach has it own intended scope of application and is applied accordingly.

In particular this Chapter concentrates on the fields of Petri nets, Temporal Logic and the combination of both termed Temporal Petri nets [Suzuki & Lu 89][He & Lee 90]. These techniques were chosen as they are applicable to distributed control systems that are complex, concurrent, quality-critical and safety-critical. Quality-critical systems are ones that require high dependability and reliability, but whose failure is not normally dangerous. This type of system includes telecommunication systems and financial applications [Turner 93]. Safety-critical systems are computer control systems employed in the telecommunications, medical applications, defence, nuclear power industry and aerospace applications whose failure can lead to damage to its environment or injury to human beings.

The following two properties are described at this point as they are important behavioural properties of the systems under consideration and are referred to often in the survey. Safety Properties and Liveness Properties [Lamport 77], are described as follows:

i) Safety Properties: These dictate those actions which a system must not perform.

ii) Liveness Properties: These dictate those actions which a system must perform.

Informally, safety properties state that *bad things* will not happen, while liveness properties state that *good things* will happen. A more formal treatment is presented in [Alpern & Schneider 85]. The term *liveness* is also used in this Thesis to refer to a structural property of a Petri net model [Peterson 81] in section 3.3.3.3, but is always distinguished when used in this sense.

3.2 State Machines

Finite state machines (FSMs) are formally defined as an ordered quintuple [Hopcroft & Ullman 79]

$$M = (S, I, \delta, q_0, F) \tag{3.1}$$

(a) S, a finite non-empty set of states;
(b) I, a finite non-empty set of inputs;
(c) δ: $S \times I \rightarrow S$ is the state transition function;
(d) $q_0 \in S$, is the initial state; and
(e) $F \subseteq S$ is the set of final states.

Thus FSMs are machines without outputs. These are considered when the state-transition behaviour of a system is of importance.

3.2.1 Moore Machines

A state machine augmented with an output associated with the state is termed a Moore Automata, a *Moore type sequential machine* is an ordered six tuple:

$$M = (S, I, O, \delta, q_0, F) \tag{3.2}$$

(a) S, a finite non-empty set of states;
(b) I, a finite non-empty set of inputs;
(c) O, is the output function; δ: $S \rightarrow O$
(d) δ: $S \times I \rightarrow S$ is the state transition function;
(e) $q_0 \in S$, is the initial state; and
(f) $F \subseteq S$ is the set of final states.

The output, given by δ, for Moore machines occurs *after* a transition and depends only on the new state of the machine.

3.2.2 Mealy Machines

Mealy Automata associate the output with the transition, a *Mealy type sequential machine* is also an ordered six tuple

$M = (S, I, O,, \delta, q_0, F)$ (3.3)

(a) S, a finite non-empty set of states;

(b) I, a finite non-empty set of inputs;

(c) O, is the output function; δ: $S \times I \rightarrow O$

(d) δ: $S \times I \rightarrow S$ is the state transition function;

(e) $q_0 \in$ S, is the initial state; and

(f) $F \subseteq S$ is the set of final states.

For Mealy machines, an output occurs during each transition and depends on the input and the current state. Thus, for a given initial state and a sequence of inputs there is a defined set of outputs. An example of the standard graphical representation of this type of state machine, known as a state transition diagram, is given in section 2.3.

FSMs can be used to represent complete systems at an abstract level, however their analysis suffers from what is termed as the state space explosion [Scholefield 90]. Generally as a model grows in size and complexity there is an exponential increase in the number of states. Also, the models are non-compositional and timing constraints do not fit well into the model. Another drawback to FSMs are their sequential nature, concurrency is not expressed in an intuitive way [Ostroff 89].

3.2.3 Communicating Real-Time State Machines

Communicating real-time state machines (CRSM) [Shaw 92] are an extension to FSMs that model concurrent real-time systems and their environment. Communication through synchronous, unidirectional channels is modelled in a manner similar to CSP [Hoare 85].

State transitions are constrained by a guarded command, the guard is a Boolean expression over local variables, and the command is an input/output (I/O) or an internal action. A transition can only be executed if the guard is evaluated to true.

The notion of time is introduced by the interval bounded by a minimum and maximum time $[t_{min}(c), t_{max}(c)]$. In the case of I/O, this interval represents the earliest and latest times that a state I/O can occur. While for an internal command, they represent a bound on execution time of $[t_{min}(c), t_{max}(c)]$, such that the duration d of the action c is in the interval $0 \leq t_{min}(c) \leq d \leq t_{max}(c)$.

3.2.4 Statecharts

Statecharts [Harel 87] are an abstraction mechanism based on FSMs and aim to resolve some of the problems associated with state diagrams. Statecharts use a hierarchical approach to modelling by the layering of different levels of abstraction. Composition of large systems is possible, aided by the provision of communication mechanisms and a global discrete clock model.

An automated software package STATEMATE [Harel *et al* 90] has been written to provide a development environment for statechart models. STATEMATE provides simulation and analysis tools based on exploring a systems state space for properties such as non-determinism, freedom from deadlock, reachable space.

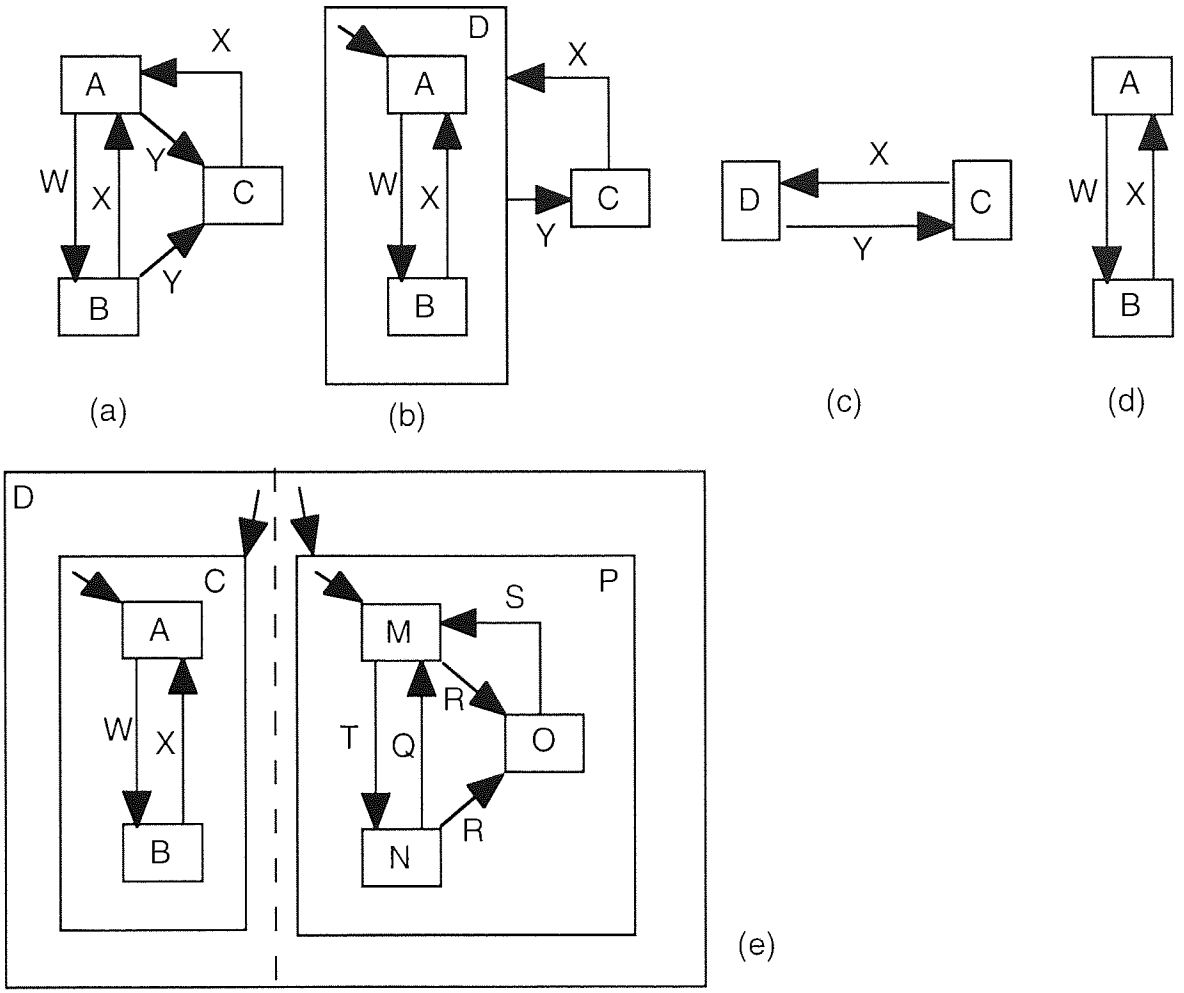


Fig.3.1. Example of a Statechart

Statecharts are illustrated in Fig.3.1., where states and encapsulation are represented by rectangles. Transitions between states, which model events have a guard associated with them, are represented by arrows.

Fig.3.1.(a) represents a system with three states (A, B & C) and three events which transfer the system from state to state (W, X & Y). Fig.3.1(b) shows how states are layered *'clustered'* event Y takes state A or B to state C, thus states A and B can be clustered as a new higher level state D. At this abstraction, event Y causes the same transition to state C and is represented by a single transition. The default entry point to state A within the new state D is shown by the single arrow at the top left-hand corner of A. Fig.3.1.(c) shows zooming-out to a level showing only the higher level states D and C. While Fig.3.1.(d) shows zooming-in to show only the lower level states and events within the state D. This zooming-in and -out allows a view of the system at different levels of abstraction, which is a significant aid as the complexity and size of models increases.

Fig.3.1.(e) shows the modelling of concurrency, the sub-states of states P and C are concurrent with each other.

3.3 Petri Nets

Petri nets are a popular technique and have been the subject of extensive research, they were developed by C.A.Petri in the early 1960's [Petri 66][Peterson 81]. Petri nets are a mathematical theory with an associated graphical notation for the modelling and analysis of the causal properties and the control flow in systems that exhibit concurrency and non-determinism. Petri nets are an established technique for the modelling and analysis of concurrent systems, that provides a simple graphical structure with an established background of analysis techniques [Murata 1989][Peterson 1981], as well as offering suitable synthesis strategies [Jeng & DiCesare 1993]. Introductory texts and review can be found in [Murata 89][Peterson 81], while Zurawski & Zhou [94] offers a tutorial to their use in industrial applications.

The graphical notation is a bipartite graph that consists of two types of node (known as places and transitions) and arcs that interconnect them [Reisig 85], as illustrated in Fig 3.2.

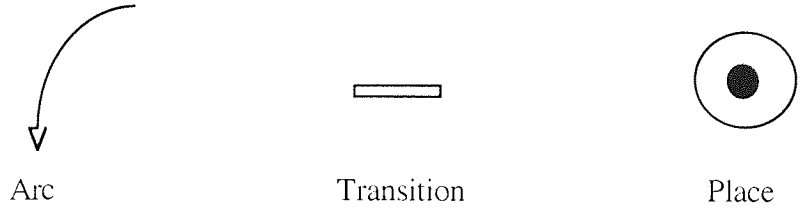


Fig 3.2. Graphical notation of Petri nets.

- a circle represents a place, a bar represents a transition,

- an arrow represents an arc which connects a place to a transition, or vice versa,
- a black dot which resides in a circle represents a token.

Petri nets are formally defined as an ordered quintuple, the definitions in [Murata 89] and [Peterson 81] differ, but are equivalent. This thesis follows the form of [Peterson 81],

$$PN = (P, T, I, O, M_0) \tag{3.4}$$

(a) $P = \{p_1; p_2; ..., p_n\}$ is a finite set of places;

(b) $T = \{t_1; t_2; ..., t_n\}$ is a finite set of transitions, $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$;

(c) $I : (P \times T) \rightarrow \mathbf{N}$, is an input function that defines directed arcs from places to transitions, where $\mathbf{N}$ is the set of non-negative integers;

(d) $O: (P \times T) \rightarrow \mathbf{N}$, is an output function that defines directed arcs from transitions to places;

(e) $M_0: P \rightarrow \{0, 1, 2, \ldots\}$ is an initial marking (or state);

In Petri nets a state is defined by the distribution of tokens on places and is called the Petri net marking M, defined as $M: P \rightarrow \{0, 1, 2, ..., m\}$. The presence or absence of a token in a place can indicate the truth value of a condition represented by the place. While the set of transitions can represent events. The dynamic behaviour of a Petri net is modelled by the changing in the markings caused by enabling and firing transitions.

The rules that determine the enabling and firing of transitions are as follows;

i) A transition $t \in T$ is enabled iff each input place $p \in P$ of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t as defined by the input function $I(p, t) = w$. Similarly, for the output function $O(p, t) = w$, where w is the weight connecting transition t to place p.

ii) a transition once enabled may or may not fire, dependent upon the additional interpretation of whether or not an event actually takes place.

iii) the firing of an enabled transition t removes from each input place p the number of tokens equal to the weight w of the directed arc connecting p to t. It also deposits in each output place p the number of tokens equal to the weight of the directed arc from t to p.

A great deal of research has been done on Petri net theory for a variety of applications [Peterson 81, Murata 89, Zurawski & Zhou 94], as a consequence many types of extensions to Petri nets exist [Reisig 85], [David 91]. These can be broadly divided into the categories of low-level and high-level Petri nets, and Petri nets that have some notion of time.

3.3.1 Low Level Petri net

Low-level Petri nets have a simple graphical notation for specifying the causal relationships of concurrent systems, an example from [Peterson 81] which models the well-known dining philosophers problem is used to illustrate the graphical form of Petri nets, Fig.3.3.

The dining philosophers problem consisting of five philosophers, each of whom can either be thinking or eating; each philosopher requires two chopsticks for eating. Problems arise, however, when philosophers attempt to eat because five chopsticks must be shared amongst five philosophers, so each chopstick is shared by two philosophers. This problem, one of deadlock [Hoare 85], is represented by Fig.3.3 in the following manner.

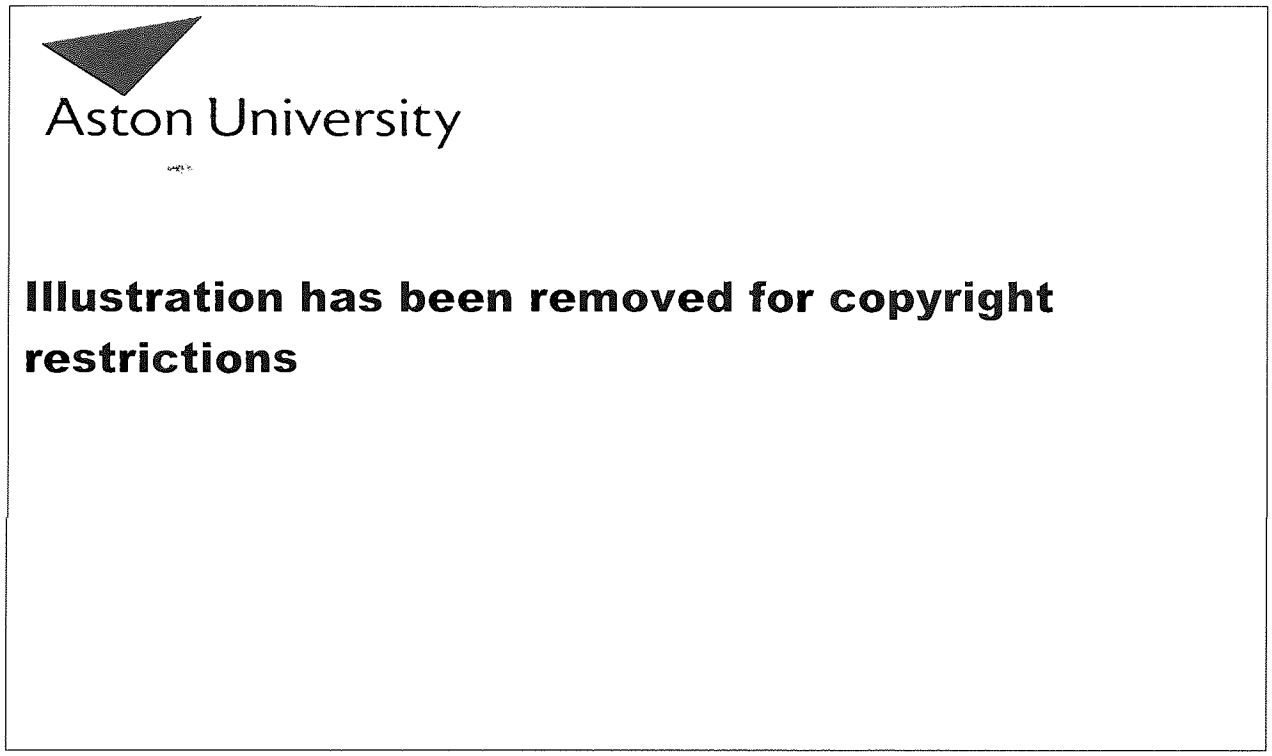


Fig.3.3 A Petri net model representing the dining philosophers problem

In Fig.3.3. places represent the states of the philosophers and the availability of resources (chopstick), and transitions represent events. The semantics assigned to the following places are:

i) p_1 to p_5 marked represent the condition that philosophers are thinking,

ii) p_6 to p_{10} marked represent available chopsticks,

iii) p_{11} to p_{15} marked represent philosophers eating,

and the semantics assigned to the following transitions are:

iv) t_1 to t_5 firing stand for the actions of picking up chopsticks and
v) t_6 to t_{10} firing stand for the actions of putting down chopsticks.

The initial marking *M0* shown in Fig.3.3. represents the state when all philosophers are thinking. The enabled condition of transitions t1 to t5 represents the philosophers having the potential to start eating. However, only the combination of philosophers represented by the following places can actually eat:
i) p_{11} and p_{13}, if t_1 and t_3 are fired,
ii) p_{11} and p_{14}, if t_1 and t_4 are fired,
iii) p_{12} and p_{14}, if t_2 and t_4 are fired,
iv) p_{12} and p_{15}, if t_2 and t_5 are fired, or
v) p_{13} and p_{15}, if t_3 and t_5 are fired.

Since the philosophers must share the available resources (chopsticks), a deadlock situation can easily occur in which certain philosophers cannot eat due to the unavailability of (chopsticks). Wu and Murata [Wu & Murata 83] present a starvation-free solution to this problem.

The behavioural properties of low level Petri nets such as reachability, boundedness, liveness, freedom from deadlock and reversibility depend upon the initial marking [Peterson 81, Murata 89, Zurawski & Zhou 94]. Analysis of these properties is based on the enumeration of the nets state space. All possible markings are generated by the successive firing of enabled transitions in each marking. A sequence of transition firings produces a sequence of markings. The following properties are based on reachability analysis, which requires the enumeration of all markings reachable from the initial marking.

3.3.3.1 Reachability

Reachability is an important property in the analysis of distributed systems. It can be used show whether a system can reach a specific state (e.g. a hazardous system state where sites are inconsistent) or exhibit a particular functional behaviour [Zurawski & Zhou 94] from the initial conditions. A marking *M* is said to be reachable from *M0*: iff there exists a sequence of transitions that transforms *M0*: to *M* . *S* denotes a firing sequence of transitions, $S = M0: t_1\ M1\ t_2 \ldots t_n\ Mn$, or simplified as $S = t_1\ t_2 \ldots t_n$. Where *Mn* is reachable from *M0* then we can write $M0\ [S>Mn$. Therefore *Mn* is reachable from *M0* iff $\exists S: M0\ [S>Mn$.

The set of reachable markings from *M0* is denoted by *R(M0*), and the problem of finding if $Mn \in R(M0)$ has been shown to be decidable, but subject to state space explosion [Scholefield 90].

A reachability tree is a graphical representation of all the possible markings reachable from an initial marking for a bounded net [Murata 89], and as such is the equivalent of a state transition diagram. Reachability trees are a fundamental analysis method used with Petri nets and is generated by executing each enabled transition of the Petri net and recording the new markings produced as a node on a tree. Where more than one transition is enabled then extra branches are added to the tree, each representing the firing of a transition.

The coverability tree reduces this by merging nodes that contain the same markings, this overcomes the problem of reachability trees of unbounded nets which grow indefinitely. The complete reachability tree shows all the possible states that a system may exhibit during its execution.

3.3.3.2 Boundedness

A place is k-bounded if the number of tokens in that place never exceeds k. A Petri net is k-bounded if each place can accept a maximum of k tokens (where $k \geq 1$) for any marking reachable from $M0$ [Peterson 81, Murata 89]. A 1-bounded Petri net is termed *safe* . For *safe* Petri nets the reachability tree can show 'Deadlock' and 'State Consistency' properties

Checks for boundedness ensure that places do not exceed their token capacities. Places are often used to represent communications and buffers. Where a place in a k-bounded net is used to represent such a buffer, the property of k-boundedness guarantees that there will be no overflow of the buffer (buffer size $\leq$ k) for all firing sequences [Murata 89].

A condition-event (CE) net is a low level net which is one bounded (the capacity for each place p is restricted to {0, 1}). The places represent conditions in a system, and transitions represent events (as illustrated in Fig.3.3.). A generalisation of the CE net is the place-transition (PT) net which are k-bounded rather than 1-bounded. As such these can produce models that are structurally less complex and hence more compact, when compared to CE nets.

3.3.3.3 Liveness

A Petri net is said to be *live* if for any marking M_n , where $M_n \in R(M0)$, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence [Peterson 81, Murata 89]. If a Petri net is *live* then this guarantees that it is deadlock free. Deadlock refers to a situation in a Petri net model in which a marking is reached where no further transitions are fireable.

The major disadvantage of state space based Petri net analysis (approaches involving the complete enumeration of the reachable states) is that the reachability graph can easily

become complex, large and unmanageable, even for relatively small Petri net models [Murata 89].

3.3.3.4 Reversibility

For an initial marking *M0* , a marking *Mh* is termed a *home state*, if there exists a sequence of transitions that can transform the marking to *Mh* from any reachable marking. A Petri net is reversible if the home state is the initial state[David & Alla 92] allowing a system to return to its initial state from any reachable state. Reversibility is an important property for the study of error recovery [Zurawski & Zhou 94] as systems may be required to make a transition from a *failed* state to a preceding *correct* state.

3.3.3.5 Incidence matrix and State Equations

This is an approach to Petri net analysis that does not rely upon generating the complete state space of the model. Instead this approach uses matrix equations to represent the dynamic behaviour of Petri net models. Matrix equations and linear algebra can be used to perform T-Invariant (token) and P-Invariant (place) analysis of the models. The incidence matrix **N** is fundamental to this approach and defines the interconnections between places and transitions. The Petri net markings are represented by state equations of the form (3.5). The incidence matrix **N** captures the Petri nets causal properties by relating the input and output places of each transition. The firing of transition (t_j) is represented by the elementary vector u_k = [0 0 .. 1 .. 0], where the 1 is the i^{th} element. The state equation can then be used to generate the sequence of markings $M_0, M_1, \ldots M_n$

$$\mathbf{M_k} = \mathbf{M_{k-1}} + \mathbf{N}^T \mathbf{U_k} \qquad (3.5)$$

where

- $\mathbf{M_{k-1}}$ is a column vector defining the current net marking,
- **N** the incidence matrix whose elements represent the arcs connecting a transition to places or places to transitions. $\mathbf{N}^T$ is the transpose of **N**,
- $\mathbf{U_k}$ is a column vector which defines the transition that will fire to give the new marking $\mathbf{M_k}$,
- $\mathbf{M_k}$ is the immediate successor marking to $\mathbf{M_{k-1}}$.

Analysis of place-invariants (P-invariants) shows the set of places in which the total number of tokens is constant. P-invariants can be calculated by finding the solutions (**y**) to the set of homogeneous equations given by:

$$\mathbf{Ny} = 0 \qquad (3.6)$$

T-invariants show the number of times a particular marking is reachable from a transition firing sequence of a Petri net, without specifying the order of appearance. This invariant can be calculated by finding the solutions (**x**) to the equation:

$$\mathbf{N}^{\mathbf{T}}\mathbf{x} = 0 \qquad (3.7)$$

P-invariants and T-invariants have been used for the analysis of concurrent systems [Lautenbach & Schmid 74] [Reisig 85], and communication protocols [Berthelot & Terrat 82] and are considered a powerful proof technique associated with Petri nets [Murata 89, Manna & Pnueli 92]. Knowledge of S-invariants and T-invariants can be used to deduce properties of the net, such as reachability, boundedness and deadlock.

3.3.2 High Level Petri net

High level Petri nets, such as Coloured Petri nets (CP) [Jensen 90] and Predicate-transition nets (PrT)[Genrich & Lautenbach 81] can be considered as structurally folded forms of low level Petri nets [Murata 89]. These were developed as an attempt to overcome some of the limitations of low level Petri nets by producing smaller and more manageable graphical representation of complex systems. The criticisms of low level Petri nets concerns their following aspects:

i) The lack of composition,

ii) Complexity management; for modelling of large or complex systems has the disadvantage of producing large models [He & Lee 90], which are not graphically intuitive,

iii) Intractability of analysis, large models can become unmanageable and difficult to analyse [Genrich & Lautenbach 81],

iv) The lack of automation of proofs.

A CE net can be viewed as a set of propositional variables, with changing truth values modelled by the dynamic behaviour of the net. Whereas, for PrT nets [Genrich & Lautenbach 81] a *predicate* is associated with each place, tokens represent individual variables and transitions are associated with logical formulas that define which tokens are involved in the enabling and firing of the transition.

Sets of places in a low level Petri net are represented by a predicate in a PrT net, while transitions represents sets of transitions in low level nets. A relational expression is associated with the arcs of a PrT net that specify which combination of tokens can engage in the firing process and the extent of change caused by the firing of the transition. Fig.3.4. gives an example of a PrT that models the five dining philosophers problem (section 3.3.1.), taken from [He and Lee 91].

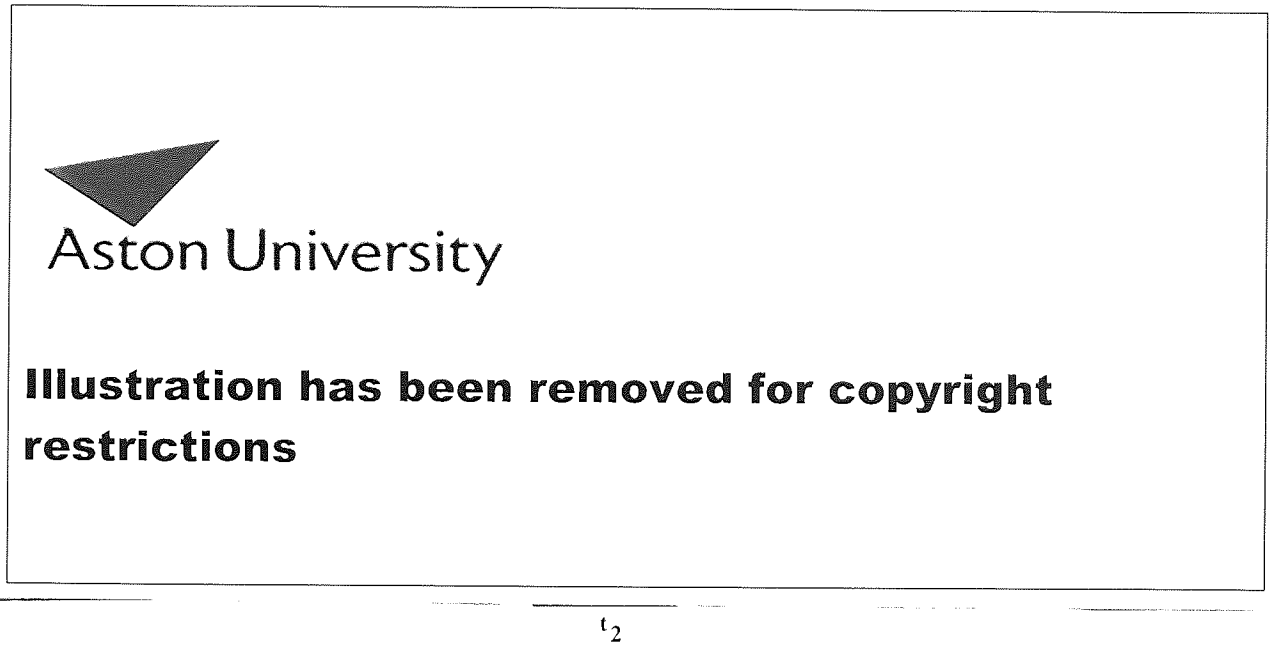


Fig.3.4 A PrT net model representing the dining philosophers problem

where the semantics are assigned as follows:

i) p_1 philosophers are thinking is the predicate associated with this place; philosopher are referred to as $p_1(n)$ (where $0 \leq n \leq 4$),

ii) p_2 chopsticks available is the predicate associated with this place; chopsticks are referred to as $p_2(n)$ (where $0 \leq n \leq 4$),

iii) p_3 philosophers are eating is the predicate associated with this place, philosophers are referred to as $p_3(n)$ (where $0 \leq n \leq 4$),

iv) t_1 and t_2 these transitions are associated with the actions of picking up and putting down chopsticks, respectively.

The type and number of tokens removed from input places and placed in output places by the firing of transitions are shown by the variables associated with arcs. For the firing of t_1, {x, y};p_2 loses two tokens, $p_2(x)$ and $p_2(y)$, as $y = x \oplus 1$ is the relational expression associated with this transition.

For instance when philosopher 2 is thinking (expressed as $p_1(2)$) the firing of transition t_1 means that $p_1(2)$ can only use chopsticks 2 and 3 (expressed as $p_2(2)$ and $p_2(3)$), as the enabling conditions of t_1 ($p_1(x)$, $p_2(x)$, $p_2(y)$ and $y = x \oplus 1$).

The invariant analysis as used in low level nets needs much simplification to be practicable for PrT nets [Genrich 87], due to the elements of the incidence matrix containing logical expressions rather than the integers of low level Petri nets. However, invariant analysis is

applicable to Coloured Petri nets (CP), and a coloured Petri net solution to the dining philosophers can be found in [Valmari 91].

(CP) nets [Jensen 81] are developed from PrT nets, but have the advantage of being easier to analyse due to a more refined invariant calculus. CP nets associate colours with tokens, transitions and places. The colour set of a place specify which coloured tokens may reside in it. Transition enabling and firing is dependent upon the numbers and colours of tokens in its input places. Functions associated with the input and output arcs of a transition specify which coloured tokens should be removed from its input places and which should be deposited in the output places. A transition firing can change the colour of tokens, and this can be used to represent complex information , a detailed formal definition of CP nets may be found in [Jensen 90].

3.3.3 Petri nets and Time

The low level and high level Petri nets considered have no explicit notion of time, and thus need extending for the analysis of real-time systems. These are necessary to describe and analyse concurrent systems whose behaviour is dependent upon the notion of time, and to formally verify time dependent systems [Berthomieu & Diaz 91]. Two early extensions to Petri net models for handling time were the Time Petri nets (TPN) of [Merlin & Farber 76] and the Timed Petri nets of Ramchandani [74].

Timed Petri nets [Ramchandani 74] were originally introduced to study the performance of pipeline processors. A finite firing time is associated with each transition and the transition firing rule is altered, each transition takes a fixed amount of time to fire and must fire as soon as it is enabled. Each transition has two values associated with it, an enabling time and a firing time. Once a transition has been continually enabled for a period equal to its enabling time the transition immediately fires. Once firing begins the token(s) are absorbed for the period of the firing time, and then deposited at the output places of the transition.

In [Zuberek 80] a model similar to that of Ramchandani [74] is presented where a stochastic model of time is used. This approach combines the transition delay with a probability of the transition firing. The transition must start firing as soon as it is enabled and tokens are absorbed from the input places and are placed in the output places after the delay has expired. The analysis of these nets uses a timed reachability graph, where time vectors are associated with states. These timed Petri nets have been used to model the performance of communication protocols.

In [Sifakis 78] Petri nets are extended by having time delays applied to places in order to represent that a given condition is true for a certain amount of time. This approach retains the original Petri net definition of instantaneous transition firing time.

The TPNs of Merlin & Farber [76] associate a minimum (EFT) and maximum (LFT) firing time with each transition in order to define a range of delays. These times (earliest and latest firing times) represent upper and lower bounds on events. This is a more general model than that used in timed Petri nets, which can be modelled using TPNs. In Merlin and Farber's [76] model, an enabled transition must fire instantaneously within the range specified by (EFT) to (LFT), unless it becomes disabled by the firing of a conflicting transition. In [Merlin & Farber 76] TPNs are used to model recovery procedures in simple protocols, this allowed the modelling of timeout ranges for correct protocol operation.

Razouk & Phelps [85] present a version of timed Petri nets where each transition has a enabling time (E_t) and a firing time (F_t) associated with it. A transition is ready to fire if it has been continuously enabled for (E_t), during firing tokens are absorbed (unavailable) for a period of (F_t). During the firing phase the transition cannot become disabled and must complete firing. This extension was proposed in order to model communication protocols effectively, and to provide certain performance estimates.

The TPN approach retains instant transition firing rules for the ease of standard Petri net analyses. In [Menasche & Berthomieu 83]TPNs were used with the firing intervals expressed in positive rational numbers, this ensured boundedness characteristics not guaranteed using real numbers [Berthomieu & Diaz 91]. This approach was necessary in order to describe and analyse concurrent systems whose behaviour is dependent upon explicit values of time, and also to formally verify time dependent systems [Berthomieu & Diaz 1991]. However, this analysis still required an exhaustive state space enumeration of the model.

TPNs have been successfully used in the analysis of safety and fault tolerance properties of hard real-time systems. In [Leveson & Stolzy 85, 87] an algorithm is presented that can detect the occurrence of defined hazardous states and state sequences. This algorithm is based on identifying a hazardous state and then using the reachability tree to backtrack through firing sequence to find the cause of the hazardous state. Once identified the cause of these hazardous sequences can be addressed.

A similar approach to [Leveson & Stolzy 85, 87] is used in [Kakuda *et al* 94], where an extended state machine is used to find worst case timings for state sequences. In [Kakuda *et al* 94] an automated approach for the verification of real-time self-stabilising protocols is

presented using a form of extended finite state machine. This model augmented with a restricted notion of time and related definitions of normal and abnormal states. In this approach, protocol states are represented by predicates and the property of self stabilisation is proven by verifying that sequences of states lead from an arbitrary abnormal state to converge in a normal state. The timeliness property is proven by defining timing values for the sequences that converge from abnormal to normal states, under the assumption that each state transition takes a maximum time value.

Suzuki *et al* [90] present a Petri net technique based on a combination of numerical Petri nets and the Timed Petri nets of Razouk & Phelps [85] for the modelling of communication protocols. The approach of Suzuki et al [90] is used for performance analysis of protocols, and extends the Petri net with the ability to model time dependency and data manipulation.

Time Petri nets have been used to guide the required range of timeout settings for correct protocol operation, directly from the model in [Hill & Holding 90][Hill 90]. However, knowledge of the communication system used and the transmission times is required for this procedure.

3.4. Logic Based Modelling and Analysis

Another approach to the specification and analysis of systems behaviour is to use mathematical logic. Most formal methods are based to some extent on three established fields of mathematical logic, Propositional Logic, Predicate Logic and Modal Logic.

3.4.1 Propositional & Predicate Logic

A proposition is a statement of a formal language which is either true or false and which remains so regardless of any environmental considerations. Like any formal language, propositional logic has a precisely defined syntax and a semantic interpretation. A statement in propositional logic is interpreted by its truth table, which maps the statement to a member of the set (TRUE, FALSE). The truth value of a statement is established by examining the truth values of its constituents.

Predicate logic is concerned with the relative truth of a statement, unlike the absolute truth of a statement in propositional logic. A predicate is an expression which defines a mapping from the subjects to a member of the set (TRUE, FALSE). Predicates are combined with connectives in the same manner as used in propositional logic, but there are two additional operators associated with predicates, the universal quantifier and the existential quantifier. For predicate R and set G, the universal quantifier is shown:

$$\forall g \in G \bullet R(g) \tag{3.7}$$

Which reads 'for all elements g which are members of G predicate R holds'. While, for a predicate R and set G, the existential quantifier is shown:

$$\exists g \in G \bullet R(g) \tag{3.8}$$

Which reads as ' There exists at least one element g which is a member of G for which R holds '.

Propositional and predicate logic are appropriate for describing static situations. Modal logic is an extension of predicate logic [Chellas 80] that allows reasoning about systems in which changes occur. Modal logic extends the notions of predicate logic by relating the dynamic changes between situations.

3.4.2. Modal & Temporal Logic

There exist a number of modal logic's, one of the most common and useful is temporal logic [Pnueli 86], which defines changes in properties through the passage of time. Temporal logic is of particular interest due to its application in the modelling and analysis of real-time systems. As well as using the standard connectives of predicate calculus it possesses temporal operators such as *always*, *next* and *eventually*[Pnueli 77]. It is a suitable formalism for specifying and analysing the behaviour of concurrent systems, such as distributed control systems, as it allows reasoning about statements whose truth value may change for different states.

The underlying computational model of modal logic [Chellas 80] consists of a universe of states (W) and an accessibility relation (R), which specifies the possibility of getting from one state to another. Temporal logic is a form of modal logic [Manna & Pnueli 92] where the accessibility relation is defined as the passage of time. Thus, a state S_n is accessible from another state S_{n-1} if through a process in time S_{n-1} can transform into S_n.

Temporal logic can be based on three models of the underlying nature of time[Rescher & Urquhart 71][Manna & Pnueli 92]. The first is that time is linear: at each moment there is only one possible future. Logics of this type are known as linear time temporal logic (LTL) [Pnueli 77]. The second model is that time has a branching, tree-like nature: at each moment, time may split into alternative courses representing different possible futures. Logics of this type are known as branching time temporal logic (BTL) [Emerson 86]. The third model is that time is represented as the partial ordering of events, these types of logic are known as partial order temporal logic[Reisig 88].

In all forms of temporal logic, the temporal operators are used to define how the truth value of properties of a system vary over time. This allows expression of properties such as:

i) Invariance properties, describing properties that are always true for a system over time,

ii) Eventualities, properties that become true at some instant of time,

iii) Precedence, assertions that can be made about the relative order of events over time.

LTL and BTL are both based on an interleaving semantics representation of concurrency [Manna & Pnueli 92] where the concurrent activity of two programs is represented by the interleaving of their atomic actions. The linear model is unable to distinguish between a non-deterministic choice and one due to the *interleaved* model of concurrency [Manna & Pnueli 88]. The notion of *fairness* is introduced to the interleaved model in order to overcome this limitation.

In partial order semantics a structure of states, which are possibly infinite, are defined by two basic relations. One relation is the partial order which represents the *precedence* ordering of events. While the *conflict* relation considers two events to be in conflict if they cannot participate in the same execution. Partial order semantics identify concurrency as a unique phenomena which is not translatable to an interleaved representation [Manna & Pnueli 88]. Therefore, partial order semantics can distinguish between inherent concurrency and internal non-deterministic choice.

Temporal logic has been proved useful in the specification and analysis of concurrent and real-time systems [Pnueli 86 , Emerson 86, Ostroff 89, Manna & Pnueli 92], and is an example of a technique ideally suited to the specification of requirements [Holzmann 92].

3.4.2.1. Linear Time temporal Logic

In LTL [Pnueli 86] the notion of qualitative and quantitative time is used. In qualitative time, the notion of eventuality and fairness are used to guarantee that an event can occurs at some point, but without providing a bound on the amount of time this may take. Using quantitative time, time is expressed as a value which defines when an event may occur. Quantitative time temporal logics are suited to the specification and verification of hard real-time systems, and as such are referred to as real-time temporal logics (RTTL) by Pnueli & Harel [88].

LTL is suitable for reasoning about concurrent programs in which a state may have several possible futures. The choice of which is taken is made in a non-deterministic manner, but can be resolved by imposing a fairness requirement.

The computational model of qualitative time LTL consists of a possibly infinite sequence of states (σ) and an accessibility relation (R), represented as:

$$\sigma = s_0, s_1, \ldots \tag{3.9}$$

$$R(s_n, s_{n+1}) \tag{3.10}$$

where s_n (where $n \geq 0$) represents a state,
and $R(s_n, s_{n+1})$ is the accessibility relation.

Within LTL temporal operators are used in conjunction with logical formulas, to specify the progression of time over the sequence of states σ, the following are the LTL temporal operators syntax:

always - $\Box$ eventually - $\Diamond$ next - O until - $\mathcal{U}$

The following defines the semantics of LTL temporal operators . For a state sequence (σ), let the sequence $\sigma^{(k)}$ be the k-shifted sequence given by:

$$\sigma^{(k)} = s_k, s_{k+1}, \ldots \tag{3.11}$$

then,

(i) if w is a classical formula (constructed from propositions or predicates and logical operators such as NOT ($\neg$), AND ($\wedge$), OR ($\vee$) and implication ($\Rightarrow$)) containing no temporal operators then,

$$\sigma \models w \quad \text{iff} \quad s_0 \models w \tag{3.12}$$

in this case w can be interpreted over a single state in σ, which is the initial state s_0,

(ii) the temporal operator always ($\Box$) is defined as:

$$\sigma \models \Box w \quad \text{iff} \quad \forall k \geq 0, \quad \sigma^{(k)} \models w \tag{3.13}$$

$\Box w$ holds on σ iff all states in σ satisfy w,

(iii) the temporal operator eventually ($\Diamond$)

$$\sigma \models \Diamond w \quad \text{iff} \quad \exists k \geq 0, \quad \sigma^{(k)} \models w \tag{3.14}$$

$\Diamond w$ holds on σ iff at least one state in σ satisfy w. Alternatively, eventually ($\Diamond$) can be defined in terms of the temporal operator always ($\Box$) as: $\Diamond w \equiv \neg\Box \neg w$

(iv) the temporal operator next (O)

$$\sigma \models O w \quad \text{iff} \quad \sigma^{(1)} \models w \tag{3.15}$$

Ow holds on σ iff $\sigma^{(1)}$ satisfies w,

(v) the temporal operator until ($\mathcal{U}$)

$$\sigma \models x \,\mathcal{U}\, y \quad \text{iff} \quad \exists k \geq 0 \text{ such that,}$$
$$\sigma^{(k)} \models y, \text{ and } \forall i,\ 0 \leq i \leq k,\quad \sigma^{(i)} \models x \tag{3.16}$$

$x \,\mathcal{U}\, y$ holds on σ iff at some time y holds and until then x holds continuously.

3.4.3 Z Notation

The Z specification languages has received widespread acceptance in industry [Mahony & Hayes 92, Woodcock & Davies 96]. This is a specification language based upon typed set theory and first order predicate logic, and is centred on a state-transition based computational model [Spivey 88, Woodcock & Davies 96].

The common style of specification using Z is based on the use of schemas which allow portions of mathematics to be grouped and named and a schema calculus, which is used to join these schemas in order to form a complete specification. These schemas can be used to represent state information and state changes.

In [Mahony & Hayes 92] Z is extended with a notation for specifying discrete and analogue properties in order to model real-time systems, while Z and duration calculus have also been combined in the ProCosII project [Hoare *et al* 94] in order to be applicable to real-time systems.

3.4.4 Process Algebras

Process algebras give an explicit model of concurrency and represent the behaviour of processes by means of constraining the allowable observable communication between them. Examples include CSP (Communicating Sequential Processes)[Hoare 85] and CCS (Calculus of Communicating Systems) [Milner 80, 89].

In CCS, systems are modelled as algebraic expressions termed *agents* and the events that a system can perform are modelled as actions of the agents. The notation describes concurrent functions and arguments competing and choosing interactions. This technique can deal with concurrency and supports the hierarchical decomposition of systems.

CSP is a mathematical approach for specifying and verifying concurrent systems and their associated communications. CSP models concurrent systems by using the notion of processes that interact with their environment. A process may represent a single sequential

system or several concurrent systems. Processes interact by transmitting information to each other through communication lines termed *channels* . Two processes can rendezvous if one process is ready to send information over a channel and simultaneously the other is prepared to receive information from the channel. This rendezvous achieves communication and provides synchronisation during message transmission.

A CSP representation provides a convenient abstraction for modelling large complex systems. Processes in CSP are defined using a process algebra which is used to construct rules that describe the behaviour of the process.

The three basic models that constitute a specification are:
i) a trace model for a process is a sequence of events, which are the instantaneous actions performed by that process. Thus the relative ordering of a sequence of events can be recorded to provide a trace of a process's behaviour. The trace model can be used to prove safety properties.

ii) a failure model, which determines between deterministic and non-deterministic behaviour of a process, the failure model can be used to prove liveness properties of a process.

iii) and a stability model, which models the internal actions of a model.

Although CSP is a useful formalism for modelling concurrent systems, the standard form does not incorporate the notion of time and is thus not suitable for real-time systems. The notion of time has been introduced into an extension of CSP, termed Timed CSP (TCSP), in order to describing real-time systems. Davies and Schneider[89] used TCSP to model time-outs for the alternating bit communication protocol.

However, the proof system of CSP & TCSP is difficult to use and its verification procedures need to be simplified before it will find widespread application. Another disadvantage of these techniques are that they are event based and states are not specified explicitly, unlike Petri nets or Temporal logic.

3.5 Temporal Petri nets

Petri nets and temporal logic are both applicable to real-time concurrent systems, Petri nets can capture the causal aspects of a system, which essentially represent the safety properties of a system, while Temporal logic can be used to reason about the temporal behaviour of the system. Temporal Petri nets combine Petri nets to validate safety properties and temporal

logic to prove liveness properties [Suzuki & Lu 89] or safety and liveness properties [He & Lee 90] of the model.

The attraction of combining Petri nets and Temporal logic was to relate a graphical model with a formal model. However, some approaches allow separate specifications to be written in each formalism, which has the disadvantage that there appears to be no means of checking the consistency between these specifications [Anttila 83]. While approaches such as [Suzuki 89] [He & Lee 90] produce a unified model. Petri net models cannot explicitly describe certain properties of concurrent real-time systems, which are ideally suited to expression through temporal logic:

i) Fairness, if a transition becomes enabled infinitely often, then it must fire infinitely often,

ii) Eventuality, certain places must eventually become tokenised, and certain transitions must eventually become enabled, and

iii) model specific constraints, the tokenising of two places must be mutually exclusive.

The Temporal Petri nets of [Suzuki & Lu 89] are a combination of LTL and Petri nets. The propositions of these Temporal Petri nets, which formalise the transition firing sequences of Petri nets, form the basis for proving liveness properties in an axiomatic manner. Following the style of [Suzuki & Lu 89] a temporal Petri net is defined as the pair:

$$TN = (PN, f) \tag{3.17}$$

where PN is the Petri net structure defined in Equ 3.4 and f represents a set of temporal formulas that describe the temporal behaviour of PN. A language based on LTL [Manna 83b] is used to describe the temporal formulas f, which is referred to as L_N [Suzuki & Lu 89]. The syntax of the temporal formulas of L_N over the PN structure consists of:

i) atomic propositions, where $p \in P$ and $t \in T$:
(p has a token), (t is fireable), (t fires) and (p has no token),

ii) logical operators (3.4.1): $\neg$ (NOT), $\wedge$ (AND), $\vee$ (OR) and $\Rightarrow$ (implication),

iii) temporal operators (3.4.2): O (next), $\Box$ (henceforth or always), $\Diamond$ (eventually) and $\mathcal{U}$ (until).

The formation rules for L_N are:
if $f_1, f_2 \in L_N$, then $(f_1 \wedge f_2), (f_1 \vee f_2), \neg f_1, (f_1 \Rightarrow f_2), Of_1, \Box f_1, \Diamond f_1$ and $f_1 \mathcal{U} f_2 \in L_N$.

The following semantics assigns an interpretation to each temporal formula of L_N over the PN structure.

The atomic propositions i) are intended to mean:

(p has a token) - p has at least one token at the current marking,
(t is fireable) - t is fireable at the current marking,
(t fires) - t fires at the current marking,
(p has no token) - p has no tokens at the current marking.

The meaning of the logical operators ii) follow the standard form.

If formula f uses temporal operators then its intended meanings are:

Of_1 - f_1 is satisfied at the next marking,
$\Box f_1$ - f_1 is satisfied at every marking reachable from the current marking,
$\Diamond f_1$ - f_1 is satisfied at some marking reachable from the current marking,
$f_1 \mathcal{U} f_2$ - f_1 remains satisfied at least until f_2 becomes satisfied at a marking reachable from the current marking.

The formal semantics of L_N are given as follows. Let α be a possibly infinite firing sequence from marking M. For each i, $0 \le i \le |\alpha|$ let β_i and γ_i be sequences such that $|\beta_i| = i$ and $\alpha = \beta_i{}^\wedge\gamma_i$. That is, β_i is the prefix of α with length i, and γ_i is the postfix of α excluding β_i. - ($|\alpha|$ denotes the length of α)

The notation $M \Rightarrow_t M'$ is used to denote that, if $t \in T$ is fireable at M, then it *may* fire and yield another marking M'. Let M_i be the marking such that $M[\beta_i\rangle M_i$, denoting that from marking M it is possible to reach marking M_i by a firing sequence β_i. (α^j is the sequence obtained by concatenating j occurrences of α, therefore α^0 is defined to be λ, where λ represents the empty sequence).

For a formula f, $\langle M,\alpha\rangle \models f$ is satisfied under the pair $\langle M,\alpha\rangle$:

$\langle M,\alpha\rangle \models$ (p has a token) iff p has at least one token at M (3.18)
$\langle M,\alpha\rangle \models$ (t is fireable) iff t is fireable at M (3.19)
$\langle M,\alpha\rangle \models$ (t fires) iff $\alpha \neq \lambda$ and $t = \beta_1$ (3.20)
$\langle M,\alpha\rangle \models f_1 \wedge f_2$ iff $\langle M,\alpha\rangle \models f_1$ and $\langle M,\alpha\rangle \models f_2$ (3.21)
$\langle M,\alpha\rangle \models f_1 \vee f_2$ iff $\langle M,\alpha\rangle \models f_1$ or $\langle M,\alpha\rangle \models f_2$ (3.22)
(note $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$)
$\langle M,\alpha\rangle \models \neg f_1$ iff not $\langle M,\alpha\rangle \models f_1$ (3.23)
$\langle M,\alpha\rangle \models f_1 \Rightarrow f_2$ iff $\langle M,\alpha\rangle \models f_1$ implies $\langle M,\alpha\rangle \models f_2$ (3.24)
(note $f_1 \Rightarrow f_2 \equiv \neg f_1 \vee f_2$)
$\langle M,\alpha\rangle \models Of$ iff $\alpha \neq \lambda$ and $\langle M_1,\gamma_1\rangle \models f$ (3.25)
$\langle M,\alpha\rangle \models \Box f$ iff $\langle M_i,\gamma_i\rangle \models f$ for every $0 \le i \le |\alpha|$ (3.26)

$<M,\alpha> \models \Diamond f$ iff $<M_i,\gamma_i> \models f$ for some $0 \le i \le |\alpha|$ (3.27)

$<M,\alpha> \models f_1 \mathcal{U} f_2$ iff $(<M_i,\gamma_i> \models f_1$ for every $0 \le i \le |\alpha|)$ or
$(\text{for some } 0 \le i \le |\alpha|, <M_i,\gamma_i> \models f_2 \text{ and}$
$<M_j,\gamma_j> \models f_1 \text{ for every } 0 \le j < i).$ (3.28)

The temporal formulas of L_N are constructed from the places and transitions of the Petri net structure; these are abbreviated in the following manner:

$p \equiv$ (p has a token), $\neg p \equiv$(p has no token)
t(ok) $\equiv$ (t is fireable), t $\equiv$ (t fires), t($\neg$ok) $\equiv$ (t is not fireable).

The following temporal formulas are expressed in this manner:

i) $<M,\alpha> \models Ot,$ (3.29)
transition t must fire at the next marking reachable from M when α occurs,

ii) $<M,\alpha> \models \Box p,$ (3.30)
place p must not become token-free at any marking reachable from M when α occurs,

iii) $<M,\alpha> \models \Diamond p,$ (3.31)
place p must eventually have at least one token at any marking reachable from M when α occurs,

iv) $<M,\alpha> \models \Box(t_1 \Rightarrow \Diamond t_2),$ (3.32)
whenever transition t_1 fires, transition t_2 must eventually fire in α,

v) $<M,\alpha> \models \Box\Diamond t,$ (3.33)
transition t fires infinitely often in α,

vi) $<M,\alpha> \models p \Rightarrow (p \mathcal{U} t)$: (3.34)
if place p has at least one token at M, then p must not become token-free at least until t fires.

The temporal Petri net of Suzuki and Lu [Suzuki 89] defines a temporal Petri net (given by the pair: TN = (PN, f)) where f is interpreted as a restriction on the firing sequences generated from PN, thus only those firing sequences that satisfy f are allowed to occur. This approach to Temporal Petri nets allows the proof of Liveness properties, while reachability based Petri net analysis is used for the proof of Safety properties.

PrT nets have the capability of producing concise and compact specifications for relatively large systems. The integration of first order LTL temporal logic [Manna & Pnueli 83] and PrT nets was presented in [He & Lee 90]. This produced a unified approach to the modelling and analysis of concurrent real-time systems that is consistent in each formalism. The approach is unified as both formalisms use the same semantic model, the Petri nets use

an interleaved sequence of markings and the temporal logic also describes a systems behaviour using an interleaved sequence of states.

A PrT net structure is translated in temporal logic, by considering set of places (predicates) and individual tokens. The proof system comprises system independent axioms and inference rules, which are obtained from the temporal logic proof system [Manna & Pnueli 83b][Pnueli 86][Pnueli & Harel 88], and system dependent axioms and inference rules that are unique to each Petri net structure under consideration.

The PrT has been translated into the temporal logic axioms and inference rules in an algorithmic manner,
i) the initial marking of the PrT net is expressed as a system dependent axiom, and

ii) the pre-conditions and post-conditions of transitions of the PrT net are converted into an system dependent inference rules,

i) and ii) form the basis of the proof system used to prove safety and liveness properties using a refutation proof procedure [He & Lee 90]. The property that needs to be proved is negated and the proof system derived above is used to form a contradiction of this negated property.

The system dependent axioms and inference rules are obviously unique to each PrT, consider the PrT structure Fig.3.4., the system dependent axiom for this fragment are given by the following:

A1. $p_1(0) \wedge p_1(1) \wedge p_1(2) \wedge p_1(3) \wedge p_1(4) \wedge p_2(0) \wedge p_2(1) \wedge p_2(2) \wedge p_2(3) \wedge p_2(4)$
This axiom (A1) describes the initial marking of Fig.3.4., all philosophers are thinking and all chopsticks are available. While, the system dependent inference rules are given by:

I1. $p_1(x) \wedge p_2(x) \wedge p_2(y) \wedge \neg p_3(x) \wedge (y = x \oplus 1)$
$\Rightarrow O(\neg p_1(x) \wedge \neg p_2(x) \wedge \neg p_2(y) \wedge p_3(x) \wedge (y = x \oplus 1))$
The logical form of the pre-conditions of transition t_1

I2. $\neg p_1(x) \wedge \neg p_2(x) \wedge \neg p_2(y) \wedge p_3(x) \wedge (y = x \oplus 1)$
$\Rightarrow O(p_1(x) \wedge p_2(x) \wedge p_2(y) \wedge \neg p_3(x) \wedge (y = x \oplus 1))$
The logical form of the post-conditions of transition t_2.

In [Sagoo & Holding 90, 91][Sagoo 92] the technique is extended from PrT nets to condition/event (CE) nets to form *extended temporal Petri nets*, using the above algorithmic approach. System dependent axioms are derived from the initial marking of the CE net

structure, $M_0 = [p_1, p_2, \ldots p_n]$, and translated to a logical from $p_1 \wedge p_2 \wedge \ldots\ldots p_n$. While the system dependent inference rules are constructed from a conjunction of the input places that form a pre-condition to a transition, and a conjunction of the output places that from the post-condition.

For the simple detail of a CE net structure, shown in Fig.3.5., the system dependent inference rule for transition t would be expressed as $p_1 \wedge p_2 \wedge \neg p_3 \Rightarrow O\neg p_1 \wedge \neg p_2 \wedge p_3$. The system dependent axiom is formed from the initial marking M_0 and is expressed as a propositional logic formula $p_1 \wedge p_2$. This approach has been successfully applied to real-time control systems [Sagoo & Holding 90, 91].

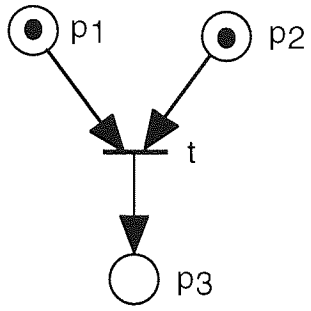


Fig.3.5. Detail of a CE net

3.6 Formal Description Techniques

As both real-time systems and interprocess communication including protocols, are addressed in this Thesis it is worth considering techniques used specifically in protocol engineering. Formal description techniques (FDTs) are generally used to refer to specification methods used in protocol engineering [King 91][Holzmann 92] within the telecommunications field [Turner 93]. However, the term can be applied to other formalisms such as Petri nets, the relational notation of Shankar & Lam [87] or the Z notation [Spivey 88][Woodcock & Davies 96]. There is a large variety of design methods and FDTs employed to contend with the inherent complexity of communications systems and no single FDT seems completely satisfactory for all types of systems [King 91].

The three most prominent FDTs in the telecommunications field are considered: SDL (Specification and Description Language)[Sharp 94], LOTOS (Language of Temporal Ordering Specification)[ISO 89b] and ESTELLE (Extended State Transition Model) [ISO 89a].

3.6.1 SDL

SDL is an FDT developed in the field of telecommunications in the mid 1970's. It has been continuously refined since then from a small informal diagrammatic notation used for design,

into a fully fledged FDT [Turner 93] retaining its graphical flavour. SDL is widely used in the telecommunications industry and is supported by a wide variety of tools.

SDL comprises a graphical and a textual representation. The system model is based upon extended finite state machines, which communicate with each other and their environment by exchanging messages. The extension to the state machine model is that each machine may have associated variables that hold information about their history. The data typing used in SDL is based on that used in ACT ONE [Ehrig & Mahr 85], a similar approach is also used in LOTOS.

In SDL, a system model consists of constructs for representing behaviour, interfaces and communication links, plus constructs for abstraction, module encapsulation and refinement. These constructs are specifically designed to aid the modelling of services and protocols in telecommunications systems [Turner 93].

3.6.2 LOTOS

LOTOS, defined in [ISO 89b], is so called as it is used to model the order of events which occur within a system. The model consists of two parts, a behavioural model derived from process algebras and a data typing model.

The behavioural model is based principally on CCS (Calculus of Communicating Systems) [Milner 89], but also to some extent on CSP (Communicating Sequential Processes) [Hoare 85]. An abstract data typing (ADT) language ACT ONE [Ehrig & Mahr 85] is used for the data typing model. This provides a well defined mathematical foundation that particularly aids in the analysis of systems modelled using this technique, and also in the development of tools.

The modelling of sequential behaviour, choice, concurrency and non-determinism in an unambiguous manner is possible using LOTOS. The explicit modelling of both asynchronous and synchronous forms of communication is also possible, while the specification of data typing as ADTs allows for an implementation independent model.

A LOTOS specification is based on the observable behaviour of a system as the sequence of all possible interactions of the system, as represented by events. These events can be abstract models of real-world occurrences [Turner 93]. For example, the transmission of data through an interface could be modelled as a single event, the beginning and end of the transmission could be modelled as separate events, or the transmission of the individual bits that constitute the data could be modelled as an event.

3.6.3 ESTELLE

ESTELLE [ISO 89a], is a formally defined specification language for distributed systems, especially for systems implementing the OSI communication services and protocols[Sharp 94].

The system model is based upon communicating, non-deterministic state machines. These state machines are hierarchically structured, like Statecharts [Harel 87], and communicate through bi-directional channels between defined communication ports with attached messages queued at either end. Synchronous or asynchronous parallelism between state machines can be represented, while actions are specified in a form based on standard PASCAL. System models are described as being *natural* by [Turner 93] due to their being based on finite automata

Although widely used for the modelling and analysis of communication protocols, these FDTs do not offer a unified approach to the modelling and analysis of both real-time distributed control systems and the communication protocols used.

3.7 Petri net Design and Analysis

Techniques that offer a unified approach to distributed real-time systems, for the control of high speed machinery, and the underlying communication protocols used, are required in this research. The three main considerations taken into account were:
i) The technique should be able to express a wide range of system properties, especially safety and liveness properties, along with the means to reason about such properties with confidence.

ii) The technique should be able to cope with real-time control systems that are inherently distributed. The complexities inherent in distributed systems also imply that a technique that permits modelling at several levels of abstraction would aid design.

iii) The critical nature of temporal constraints, in the real-time distributed control systems under consideration, requires a technique well developed to specify and prove temporal properties.

Petri nets were considered as they offer:
i) the ability to represent concurrency, distribution and explicit communications,
ii) explicit representation of causal dependencies and independence's,
iii) some level of automation in analysis is supported,
iv) a well developed field with over 25 years of research and mathematical basis,

v) a graphical representations of models that can be concise and intuitive.

The modelling of real-time distributed control systems is intuitive due to the correspondence between concepts in Petri nets and these systems such as events, states, state changes and activities. Petri nets are also particularly suited for real-time control systems as their analysis is able to identify undesirable properties such as deadlock, boundedness (related to system resources) and the occurrence of hazardous concurrent states.

The design and analysis of communication protocols requires some type of formal or semi-formal technique [Suzuki *et al* 90], due to their inherent complexity and concurrency and the need to consider explicit communications. Petri nets have been applied extensively to the modelling and verification of communication protocols [Merlin & Farber 76][Berthelot & Terrat 82][Suzuki *et al* 90][Berthomieu & Diaz 91], as the liveness and safeness properties that their analysis provides can be used for proving important *correctness* criteria of protocols [Murata 89].

Along with the modelling and analysis techniques offered by Petri nets and temporal logic, methods to guide design and analysis are also worth considering. There are several established approaches to aid in the design and analysis of Petri nets such as top-down [Valette 79], bottom-up [Suzuki & Murata 83] and hybrid design [Zhou and DiCesare 89, 93], along with net reductions to aid analysis [David & Alla 92]. These are considered below.

In top-down Petri net design methods a general model of the system is expanded in a stepwise refinement procedure. Valette [79] presents such a *stepwise refinement* method for the synthesis of large Petri nets. Transitions and places in a net are replaced by subnets, called *well formed blocks*. This is performed in a structured, step-by-step manner that preserves properties such as liveness and boundedness in the refined Petri net. These *well formed blocks* of [Valette 79] are considered in detail in section 5.3.1.

Suzuki and Murata [83] extend this technique to the refinement of systems by replacing transitions and places with subnets termed *well-behaved nets* in the top-down design of Petri net models. In the refinement procedure the specification considered becomes more detailed at each successive step, while the part of the net which is considered in detail becomes smaller. Interaction amongst subnets is difficult to specify using this approach as refinements are local and required to have no side effects [Jeng & DiCesare 1993].

In the bottom-up approach to Petri net design, sub-systems are modelled separately and then grouped or clustered to form larger systems. This approach allows for the modular

specification and manageable analysis of subnet and refined net [Suzuki & Murata 83]. The separate analysis of the constituent parts allows the construction of systems from reusable, fully understood subnets or templates. At each stage of the procedure the interaction between sub-systems must be considered, and common places and transitions merged to form larger sub-systems. The problem with this approach is in the design of the correct interaction between existing subnets [Zhou and DiCesare 89, 93]. Hybrid Petri net design combines a top-down approach followed by a bottom-up approach in an attempt to overcome this problem.

In the hybrid approach of Zhou and DiCesare [89, 93], a top down procedure is taken from a highly abstract Petri net, through the gradual replacement of places or transitions by refined subnets. Each step taken contains increased detail, and is made in a manner that preserves properties of the system such as freedom from deadlock. The bottom-up approach is used to design correct interaction among the sub-nets employed.

Petri net reduction methods [Murata 89][David & Alla 92] are procedures for reducing the number of places and transitions in a net in order to reduce the state space of the model. These rules allow a reduced net to replace a larger Petri net, where the reduced net maintains the desired properties of the original net, while having a reduced reachability graph [Murata 89]. Dwyer *et al* [95, 96] applies two Petri net reduction techniques termed *parallel transitions* and *forced communication pairs*. These are applied to aid the detection of deadlock properties, which are preserved in these reductions. *Forced communication pair* net reduction is based on abstracting sequences of communications between two tasks, where no other tasks attempt to communicate with the pair during the sequence.

The complexity of state based analysis can be reduced significantly if a structuring mechanism that also supports analysis is available. Common structuring mechanisms [Milner 89][Valmari 93] allow the description of a system by composing less complex process specifications in a well defined way. The central paradigm of this type of approach is to define local equivalence relations. *Processes* are replaced by less complex, but equivalently behaving substitutes [Valmari 93], which can give a significant reduction in the size of the state space generated. However, these hierarchical structuring methods are generally restricted to net specification and only the most detailed level is executable and allows for analysis [Bucholz 94]. A similar approach is taken with Petri nets in Chapters 5 and 6, however the problems of restricted analysis are addressed.

3.8 Conclusion

This chapter has addressed different methods of modelling and analysing distributed systems and their underlying protocols. The simple graphical structure of Petri nets can provide an approachable, intuitive system model and aid in the designer's understanding of a system. Petri nets also possess well defined analysis techniques, based on enumeration of the reachable state space, that allow the proof of safety properties. However, low level Petri net theory is only effectively able to communicate the flow of control information between concurrent processes, as opposed to data. This is an sufficient basis upon which to describe the flow of synchronisation messages and control communications. In terms of verification, the combination of the two techniques possesses the required flexibility.

Time dependencies are necessary for accurate protocol models, including modelling timeouts [Suzuki 90]. The Temporal Petri nets of Suzuki & Lu [89] and the TPNs of Merlin & Farber [76] are used for modelling and analysis of this type in Chapters 5 & 6. The TPNs are used to model timing constraints on commit protocols, and the Temporal Petri nets to provide the formal basis for reasoning about liveness properties for the protocol and controller designs developed.

The important features of the main techniques considered are summarised in Table 3.1.

Features	Formal Base	Computational Model	Concurrency	Real-time	Analysis Method
FSM	Set Theory	Interleaved State-transition	×	×	Reachability Graph
Petri nets , low & high level	Set Theory	Interleaved State-transition	√	×	Reachability Graph or Matrix Theory
Petri nets, Time(d)	Set Theory	Interleaved State-transition	√	√	Reachability Graph or Matrix Theory
LTL	Modal Logic	Interleaved State-transition	√	×	Deductive proof system
RTTL	Modal Logic	Interleaved State-transition	√	√	Deductive proof system
CSP	Set Theory & Predicate Logic	Event-based	√	×	Deductive proof system
TCSP	Set Theory & Predicate Logic	Event-based	√	√	Deductive proof system
Z	Set Theory & Predicate Logic	State-transition	×	√	Deductive proof system
Temporal Petri nets	Set Theory & Predicate Logic	Interleaved State-transition	√	√	Reachability Graph & Deductive proof system

Table 3.1 Summary of modelling techniques

Chapter 4 Design of Commit Protocols for Distributed Control

4.1 Introduction

The commit protocols examined in Chapter 2, are typically used as part of transaction processing component in distributed databases, but they can also be used to provide coordination for distributed control systems [Hill & Holding 90]. The 2-phase commit (2PC) protocol is an accepted standard commit protocol [ISO 92], while the extended 2-phase commit (E2PC) protocol has been identified as having certain properties that make it applicable in real-time systems.

The modelling of commit protocols using FSMs is established and many examples exist in the literature [Skeen & Stonebraker 83][Yuan & Jalote 89][Sharp 94][Levi & Agrawala 94][Yoo & Kim 95]. FSMs were discussed in Chapter 3, and are used for the modelling and analysis of commit protocols in section 4.2 of this chapter. The limitations of FSM models for commit protocol modelling and analysis are discussed, and problems with the modelling of communications and failures are highlighted. The Petri net methods selected in Chapter 3 are then applied to commit protocols in section 4.3 to overcome these limitations.

In this chapter the 2PC and E2PC protocols are considered in terms of their ability to coordinate distributed processes and their resilience to faults. In section 4.3 Petri net models of commit protocol are extended with failure modes, and the resulting behaviour analysed in order to demonstrate the protocols blocking or non-blocking properties. The blocking property can be used to guarantee strict data consistency between sites when failures occur. While, the non-blocking property can be used to avoiding deadlock of a distributed system due to system failure. The non-blocking property is therefore essential if a commit protocol is required to provide a real-time response even in the presence of failures.

It is intended to design commit protocols that can offer real-time response, and are resilient to clearly defined failure modes. This is required for the coordination of real-time manufacturing systems, which are examined in Chapter 6. In section 4.2 the models of standard commit protocols are presented and analysed, and a limited form of failure modelling is examined. In section 4.3 Petri nets are used for the modelling and analysis of commit protocols using explicit communications, and a multi-participant E2PC protocol is developed. In section 4.4 detailed failure modelling is included in the Petri net models and a responsive commit protocol is developed. In section 4.5 this version of the E2PC protocol is

optimised using timeout mechanisms and analysed. The behaviour of the developed commit protocol, with inherent timeout mechanisms, is analysed using Time Petri nets in section 4.6.

4.2 Modelling Commit Protocols

As explained in Chapter 2, commit protocols can be classified as either blocking or non-blocking depending upon their termination criteria. There are two types of termination conditions for commit protocols:

i) the *weak termination condition* - states that if there are no failures during the execution of the protocol then all processes will eventually reach a consistent commit or abort decision,

ii) the *strong termination condition* - states that if failures occur during the execution of the protocol, then all non-faulty processes eventually reach a consistent commit or abort decision [Lynch 96].

Protocols that meet the first condition are termed blocking protocols and the second condition are termed non-blocking protocols.

4.2.1 Blocking & non-blocking protocols

A protocol can be said to be non-blocking if the operational sites do not deadlock when a site or link failure occurs during the operation of a protocol [Skeen & Stonebraker 83]. Whereas blocking protocols are those that allow the operational sites to block until the failed site (or the communication link) returns to service. The 2PC is a blocking protocol and E2PC a non-blocking protocol, and they are examined in the following sub-sections.

4.2.1.1. Two Party 2-Phase Commit Protocol

The FSM shown in Fig.4.1. illustrates the operation of the basic 2PC protocol, which is a blocking protocol for both site and link failure occurrences.

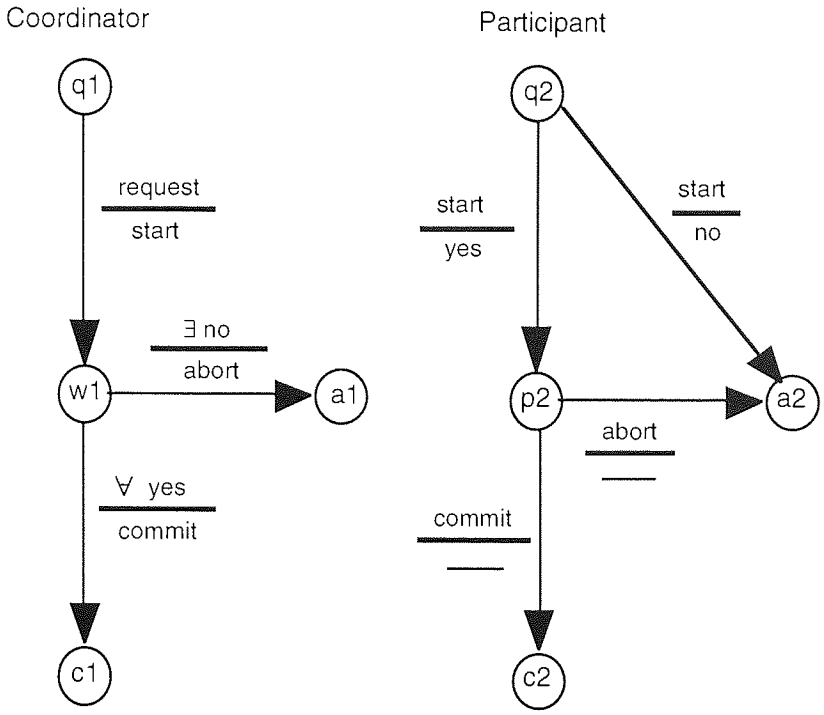


Fig.4.1. FSM of centralised 2PC protocol

Fig.4.1. shows a centralised form of the protocol (which was described in section 2.3.1.), with a pre-determined coordinator and participant arrangement. The protocol would be initiated by a request from the user to commit a transaction, this would initiate state q_1 of the coordinator.

Fig.4.2. shows a decentralised form of the 2PC protocol where all processes in the protocol are symmetrical participants, there is no coordinator. This form of the protocol relies upon rounds of message passing between each site. For a system of n sites (i in Fig.4.2. denotes process $(1..n)$), there are n start messages sent to initiate the protocol, and each site replies to all other sites $(1..n)$. This decentralised form of the 2PC protocol relies on $(2n^2 - n)$ messages to achieve a commit of n participants [Levi & Agrawala 94], while $3(n - 1)$ messages required by the simple centralised 2PC protocol for the same number of sites.

This decentralised approach is more flexible, allowing the dynamic assignment of which site initiates a commit action. However, this entails a marked increase in the number of messages required to reach a decision.

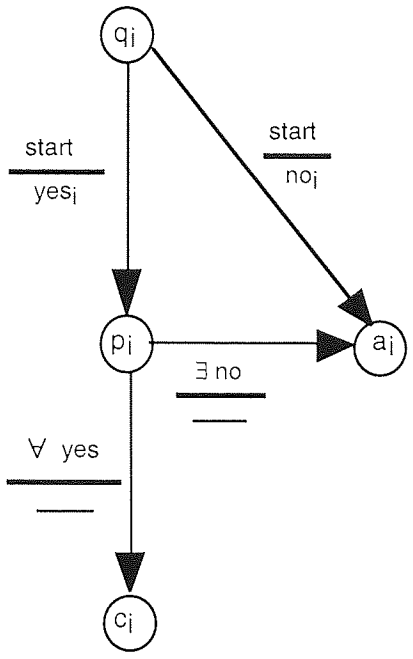


Fig.4.2. FSM of de-centralised (symmetric) 2PC protocol

The following are the local states that each protocol process can occupy, in the FSMs of Fig.4.1. and Fig.4.2.:

i) q - the initial state,

ii) w - the ready state of the coordinator, awaiting the participants votes,

iii) p - the prepared state of the participant, awaiting the commit or abort decision,

iv) a - the abort state, the protocols terminating state for an *abort* decision,

v) c - the commit state, the protocols terminating state for a *commit* decision.

A local state can be termed *commitable* if a process's occupancy of that state implies that all other processes have voted *yes* on committing [Yuan & Agrawala 88][Levi & Agrawala 94][Desai & Boutros 96]. As such, it can be said that the occupancy of *non-commitable* local states cannot be used to infer the states of other processes.

The behaviour of the basic 2PC protocol shown in Fig.4.1. is as follows; The coordinator starts in state q_1 initiated by a *request* command from the user, and issues a *start* message to all participants (process-2 in Fig.4.1.), before entering a wait state w_1. The participant's initial state is q_2, upon receipt of the *start* message it takes one of two possible actions. If it is capable of committing it replies *yes* and enters the *prepared* state p_2, or if it is not then it replies *no* and enters the final state a_2 *abort*. If all the participants vote *yes* and the coordinator is also capable of committing then the *commit* message is sent to the participants, and the coordinator enters the final state c_1 *commit*. If any of the participants voted *no* or the coordinator is unable to commit then the *abort* message is sent to the participants that voted *yes* and the coordinator enters the final state a_1 *abort*. A participant in the prepared state

enters the final state c_2 *commit* on receipt of the commit message, or a_2 on receipt of the *abort* message from the coordinator.

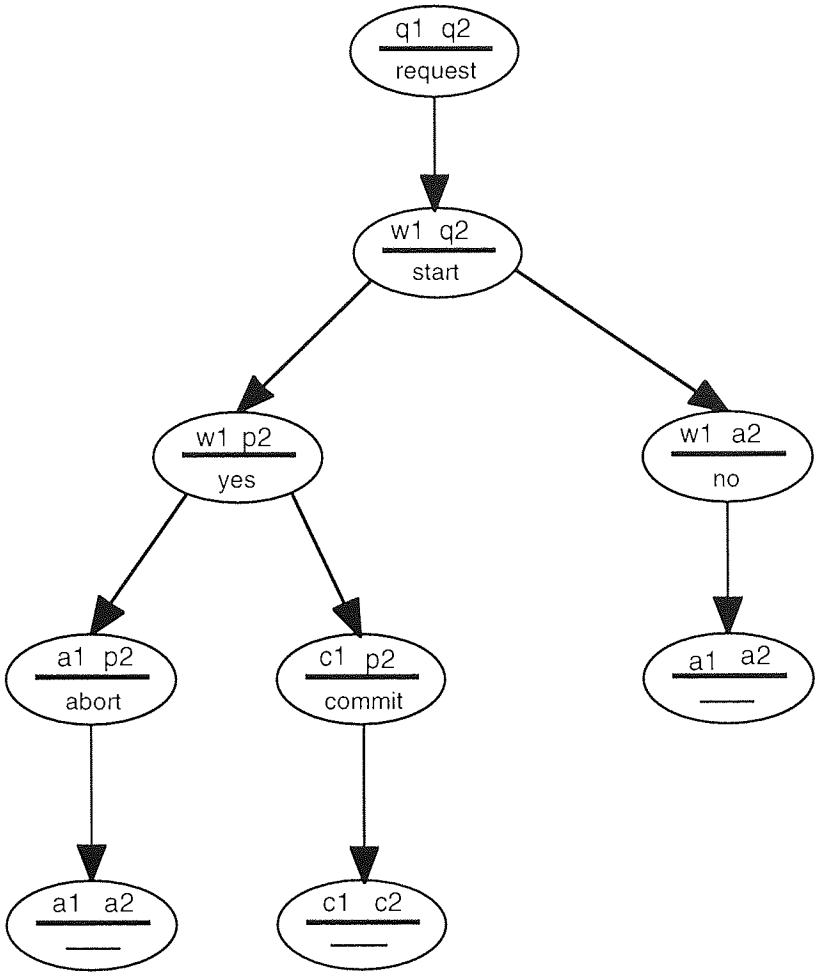


Fig.4.3. Reachability tree for the centralised 2PC protocol.

In Fig.4.1. the local states of the coordinator and participant processes are shown, and in Fig.4.3. the reachability tree is used to show the global states of the protocol. This is a two party implementation of the protocol, where the coordinator site is denoted (1) and the single participant (2). The protocol with a single participant is described in order to aid understanding of the basic protocol, and avoid the description of concurrent states of the participants. There are drawbacks to the use of FSM to model commit protocols for multi-participant systems, this is addressed in section 4.2.5.

Two sets of reachable states of the FSM model are introduced in order to aid analysis of state based protocol models. The *concurrency set* $C(s_i)$ and the *sender set* $S(s_i)$, for local state s_i [Skeen & Stonebraker 83]. The concurrency set of a local state s_i is the set of all local states that other concurrent processes can potentially occupy. From Fig.4.3. it can be seen that the concurrency set for the coordinators wait state is:

$$C(w_1) = \{ q_2, p_2, a_2 \} \qquad (4.1)$$

From $C(w_1)$, it can be seen that the participant process can be in one of potentially three states; initial (q_2), prepared (p_2) and abort (a_2). The sender set of a local state s_i is the set of all local states of other concurrent processes that send messages to this site.

$$S(w_1) = \{ q_2 \} \qquad (4.2)$$

Set $S(w_1)$ shows that in the wait state (w_1) the coordinator receives messages sent from the participant in its initial state (q_2). The complete sets for Fig.4.1. are given in Table 4.1 for the 2PC protocol, where $\varnothing$ represents the empty set. The sender sets are used in this research for determining timeout placements (in section 4.2.4.), in order to make protocols resilient to site and communication failures.

Local State s_i	Concurrency Set $C(s_i)$	Sender Set $S(s_i)$
q_1	q_2	$\varnothing$
w_1	q_2, p_2, a_2	q_2
a_1	p_2, a_2	$\varnothing$
c_1	p_2, c_2	$\varnothing$
q_2	q_1, w_1	q_1
p_2	w_1, a_1, c_1	w_1
a_2	w_1, a_1	$\varnothing$
c_2	c_1	$\varnothing$

Table 4.1 Concurrency & sender sets for two party 2PC protocol

The concurrency set is derived from the reachability tree, and shows all the potentially concurrent states for each state within the model. Concurrency sets can also be used to make deductions about the recoverability properties of a protocol following a site failure. [Skeen & Stonebraker 83][Hill 90].

An independent recovery scheme is one where the recovering site makes a direct transition to a final state, without communicating with the operational sites [Bernstein *et al* 87]. If a local state contains both an *abort* and a *commit* state in its concurrency set then it cannot be made independently recoverable. This is because upon recovery, the failed site would need to communicate with the operational sites to determine which decision had been reached. It would not be possible for a recovered site to determine this decision using only local information.

It can be deduced from the 2PC protocol concurrency set, Table 4.1, that if the coordinator site fails while the participant is in the local state p_2, then the participant cannot progress to a final state. As p_2 is an uncommitable state the protocol is blocked, $C(p_2) = \{ w_1, a_1, c_1 \}$ contains both a commit and an abort state.

4.2.1.2. Two Party Extended 2-Phase Commit Protocol

The E2PC protocol is an extension of the 2PC protocol, with the addition of a prepared state in the coordinator. This removes the coordinator's commit (c_1) and abort (a_1) states from the concurrency set of the participants prepared state $C(p_2)$.

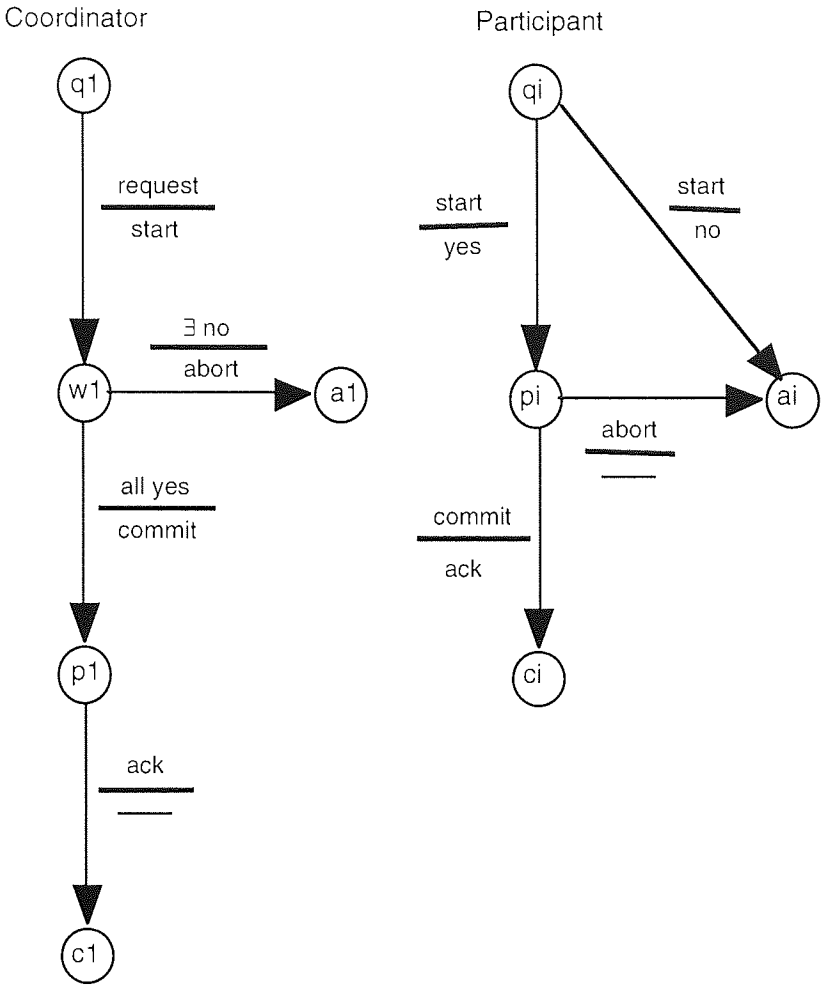


Fig.4.4. FSM of E2PC protocol

The FSM of the E2PC protocol Fig.4.4. and its associated reachability tree Fig.4.5., show the essential difference of this protocol to the 2PC. The coordinator enters a prepared state after sending the *commit* message, and the participant process has an additional *acknowledgement* message to confirm its commit state. The 3-phase commit (3PC) is also a non-blocking protocol, with the addition of a prepared state to both the coordinator and participant processes. The 3PC protocol is a decentralised scheme which allows the use of a symmetric form of the protocol [Levi & Agrawala 94].

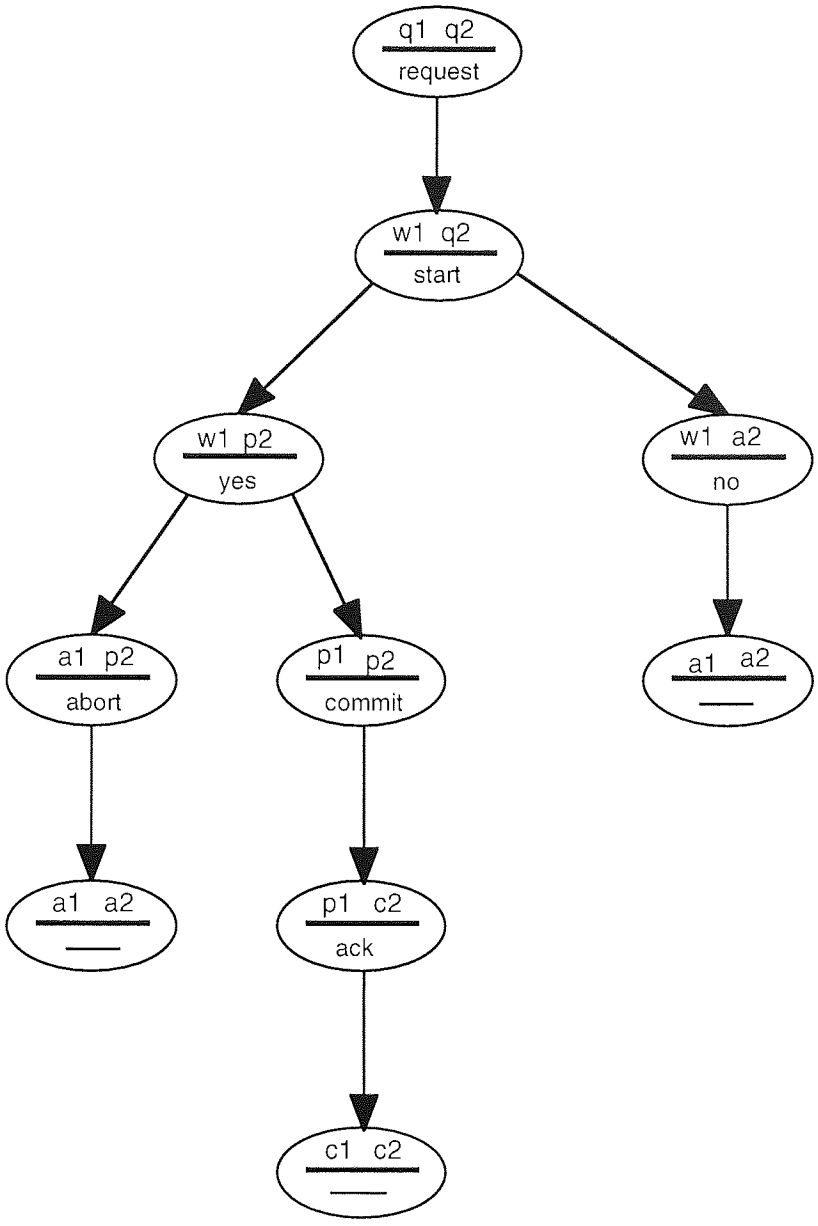


Fig.4.5. Reachability tree for extended 2-phase commit

The reachability tree, Fig.4.5., is shown for a coordinator and a single participant process (in Fig.4.4. 1 denotes the coordinator and i is 2 for the single participant). Examination of the two reachability trees, Fig.4.3. and Fig.4.5. shows they satisfy the weak termination condition, i.e. without failures for both processes the protocol terminates with a consistent final state, either both commit or both abort. This condition can be seen clearly in the concurrency set of the final states of each process (c_1, c_2, a_1, a_2) shown in Table 4.2., and in the leaf nodes of the reachability tree.

Local State s_i	Concurrency Set $C(s_i)$	Sender Set $S(s_i)$
q_1	q_2	$\varnothing$
w_1	q_2, p_2, a_2	q_2
p_1	p_2, c_2	p_2
a_1	p_2, a_2	$\varnothing$
c_1	c_2	$\varnothing$
q_2	q_1, w_1	q_1
p_2	w_1, p_1, a_1	w_1
a_2	w_1, a_1	$\varnothing$
c_2	p_1, c_1	$\varnothing$

Table 4.2. Concurrency & sender sets for two party E2PC

4.2.2. Failure and Timeout Transitions in FSMs

In order to make commit protocols resilient to failures [Skeen and Stonebraker 83] extends the protocol model with timeout and failure transitions, which represent the actions taken by a site when it detects a failure. Failure transitions are the action taken by a failed site when it recovers. This action would be based upon the failed sites recovery procedure examining the log records, which are written by the protocol after completing each phase, prior to the failure. This action should bring a recovering *failed* site into a consistent state with the operational sites.

The timeout transition represents the action taken by an operational site when it detects a failure while awaiting to send or receive a communication. This is detected by the expiration of local timers, and is mostly due to link failures or site failures [Levi & Agrawala 94]. These types of transitions are illustrated Fig.4.6. for the E2PC protocol.

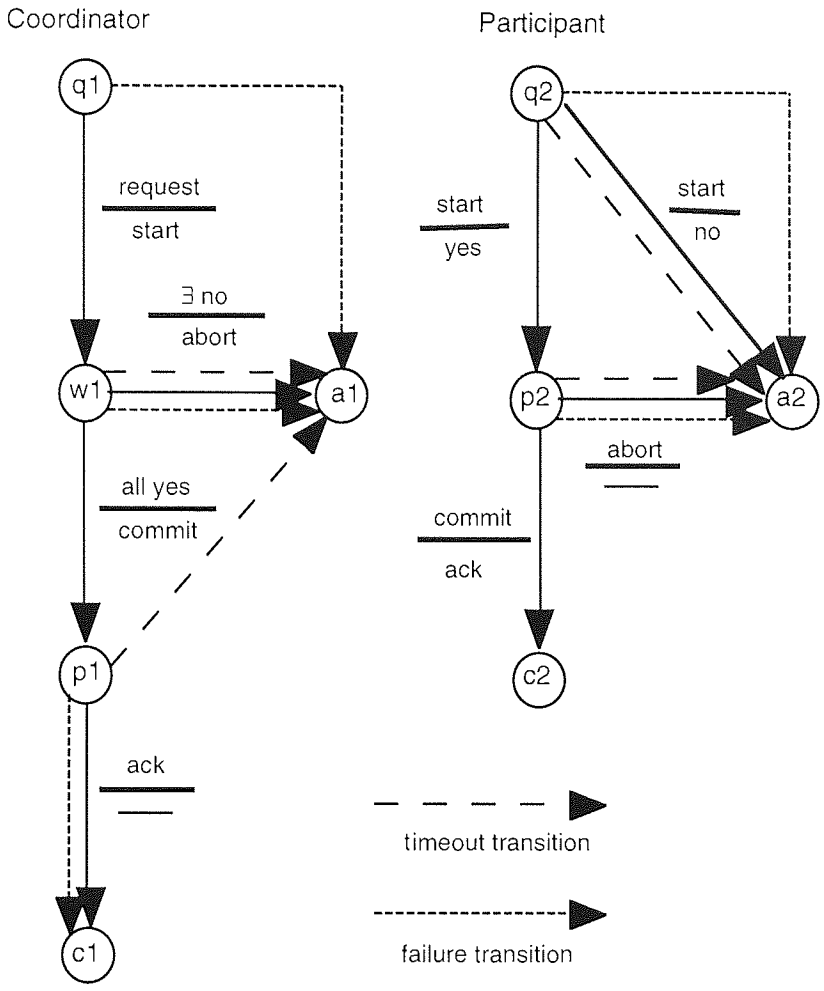


Fig.4.6. FSM of two-party E2PC with failure and timeout transitions

Two rules for the assignment of these types of transitions are presented in [Skeen and Stonebraker 83] as:

i) for every intermediate (non-final) protocol state s_i, if the concurrency set $C(s_i)$ contains a commit state then assign a failure transition from state s_i to a commit state. Otherwise, assign a failure transition to an abort state; (4.1)

ii) for every intermediate protocol state s_i, if there exists a state τ_i in the sender set of the state s_i, which has a failure transition to a commit state (or an abort state), assign a timeout transition from s_i to the commit state (or an abort state). (4.2)

Hence, to extend the E2PC protocol of Fig.4.4. with site failures by using (4.1) and (4.2) produces the protocol shown in Fig.4.6. According to rule (4.1) failure transitions are assigned as follows:

i) from q_1 to a_1, as $c_2 \notin C(q_1)$

ii) from w_1 to a_1, as $a_2 \in C(w_1)$

iii) from p_1 to c_1, as $c_2 \in C(p_1)$

iv) from q_2 to a_2, as $c_1 \notin C(q_2)$

iv) from p_2 to a_2, as $c_1 \notin C(p_2)$

The set of failure transitions F_t for the E2PC protocol FSM are therefore:

$$F_t = \{ (q_1, a_1) (w_1, a_1) (p_1, c_1) (q_2, a_2) (p_2, a_2) \}$$

According to rule (4.2) timeout transitions are assigned as follows:

i) from w_1 to a_1, as $q_2 \in S(w_1) \wedge (q_2, a_2) \in F_t$

ii) from p_1 to a_1, as $p_2 \in S(p_1) \wedge (p_2, a_2) \in F_t$

iii) from q_2 to a_2, as $q_1 \in S(q_2) \wedge (q_1, a_1) \in F_t$

iv) from p_2 to a_2, as $w_1 \in S(p_2) \wedge (w_1, a_1) \in F_t$

The set of timeout transitions T_t for the E2PC protocol FSM are therefore:

$$T_t = \{ (w_1, a_1) (p_1, a_1) (q_2, a_2) (p_2, a_2) \}$$

These rules are sufficient to make the protocol resilient to single site failures. The analysis of the reachability tree for the protocol shown in Fig.4.6. can show that a correct, consistent decision is reached by the processes even in the presence of failures.

4.2.3. Independent Recovery

As detailed in section 2.3.3. the standard 2PC is a blocking protocol, while the E2PC and the 3PC are non-blocking protocols. Although operational sites can terminate using non-blocking protocol schemes (E2PC and 3PC), a failed site may require extra non-local information upon recovery in order to reach a final state which is consistent with those reached by the operational sites. A protocol that can recover to a consistent state using only local information is termed *independently recoverable* [Bernstein *et al* 87][Yuan & Jalote 89]. In [Skeen & Stonebraker 83] it is shown that there are no independent recovery schemes possible in the presence of multiple site failures, or multiple partitions due to link failures.

The work of [Yuan & Jalote 89] proposes a method, that uses an approach based on FSMs, which allow the independent recovery of a system composed of more than two sites. This approach is based on the use of two special types of timeout transitions: timeout-to-commit and timeout-to-abort. These require a site that times-out a communication, to propagate its timeout action to all other operational sites. These extra messages are sent in order to retain a consistent global view of a failed sites local state. However, this approach considers only site failures and not link failures.

From the concurrency sets of Table 4.2 it can be seen that for the E2PC protocol, no concurrent state contains both a commit and an abort action. However, in the FSM models (such as Fig.4.4 and 4.6.), it is possible for a site to fail in the middle of a state transition; for

instance the coordinator site could fail during the sending of messages to participant sites, and only some of the participants would receive the message. If there are k (where $k > 2$) participants and the coordinator fails during the state transition (w_1 to p_1), it can be seen from Fig.4.6. of the failure model, that those participants that receive the message will have a final local state of *commit* (c_k). While those participants that did not receive the message and acted upon a timeout mechanism will have a final state of *abort*, as there is a timeout transition from the participant's prepared state to the abort state $(p_k, a_k) \in T_t$. Therefore the final global state in this case would contain both commit and abort states and as such would be inconsistent.

In the FSM protocol models, the local state transitions can consist of both the receiving and/or sending of messages in an atomic step. The sending of the commit message from the coordinator to the participants in Fig.4.4. is modelled as an atomic action, which is valid for a single participant where at most one message is sent per state transition. But, as noted in [Yuan & Agrawala 88][Yuan & Jalote 89] for more than a single participant this action could fail semi-completed, and as such cannot be considered to be atomic. For a multiple participant system, each local state transition of the FSM model can represent multiple message send or receive operations. Thus, for this type of representation (where a failure of one of the communicating sites or the communication links could have occurred in the middle of a state transition), the state transition would be rendered a non-atomic action.

The use of Petri nets to model the E2PC protocol allows the above disadvantage of representing multiple message send and receives as atomic actions to be overcome. The modelling of the communications system and each message transfer can be made explicit, this allows the behaviour of the protocol under failure conditions to be fully analysed and understood.

4.3. Petri Net Protocol Models

4.3.1. Communication Model

As stated above, a method of overcoming the problem of state transitions becoming non-atomic is to replace the FSM models of the protocols with Petri net models extended with explicit communications. Rules 4.1 & 4.2 (shown in section 4.2.4) for failure and timeout actions in FSM models are still applicable to Petri nets [Hill 90].

The type of Petri nets considered in the rest of this thesis are known a safe nets; the places in these nets have a token capacity of one (i.e. all places are 1-bounded). A software workbench for the analysis of behavioural properties of Petri net models developed in the IT research

group of Aston University [Azzopardi 96] was used in this research. This tool automates the process of analysing a Petri net model, by allowing:

i) the enumeration of the net's reachability tree, and its concurrency and sender sets,

ii) the checking of properties such as liveness, boundedness, and freedom from deadlock,

iii) the calculation of P-Invariants and T-Invariants.

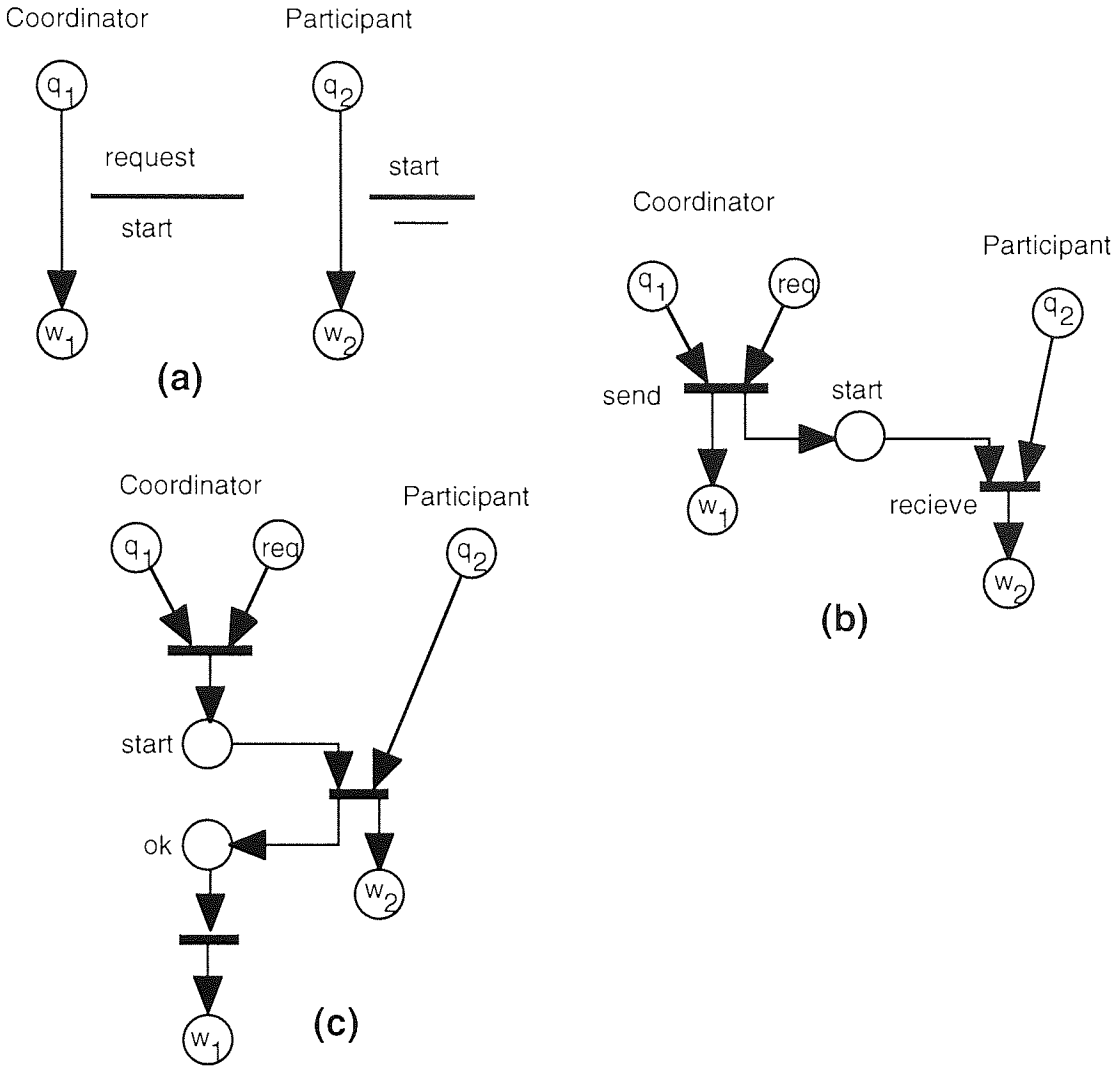


Fig.4.7. FSM and Petri net models of interprocess communication; (a) Finite state machine, (fragment from 2PC section 4.2.1.), (b) equivalent Petri net with explicit asynchronous communications, (c) equivalent Petri net with explicit synchronous communications.

In Fig.4.7.(a) the FSM model of interprocess communication is shown, this represents a single message transmission, the *start* message being sent from the coordinator to the participant. But as shown in Fig.4.4., the same state transition is also used where there are multiple participants, and represents the sending of the *start* message to each participant. In this representation multiple communications are considered as atomic actions and explicit communication detail is omitted.

The Petri net equivalents of this communication structure are shown in Fig.4.7.(b) & (c), and represents the level of atomicity of communications. The representation of asynchronous communications are shown in Fig.4.7.(b), in this model the sender process changes state when the *start* message is sent, and the receiver process changes state when message is received. There is a shared place (marked *start*) to represent the state of message in transmission, and separate send and receive transitions for each process, as shown in Fig.4.7.(b).

Fig.4.7.(c) represents synchronous communications, the sender (i.e. the coordinator) is unable to proceed with its operation until it receives an acknowledgement from the receiver (i.e. the participant). The states of the sender and the receiver processes change concurrently, upon firing of the shared communications transition. When state { ok, w_2 } becomes reachable, it shows that a message has been successfully sent and received.

4.3.2. E2PC Protocol

The FSM model of the E2PC protocol, shown in Fig.4.4., was used as a basis for the Petri net model shown in Fig.4.8. This model shows a coordinator and single participant process, using synchronous communications. The Petri net model was created by modelling each state in the FSM (Fig.4.4.) with a place, and replacing each FSM communication (as shown in Fig.4.7.(a)) with an equivalent Petri net structure for synchronous communications (shown in Fig.4.7.(c)).

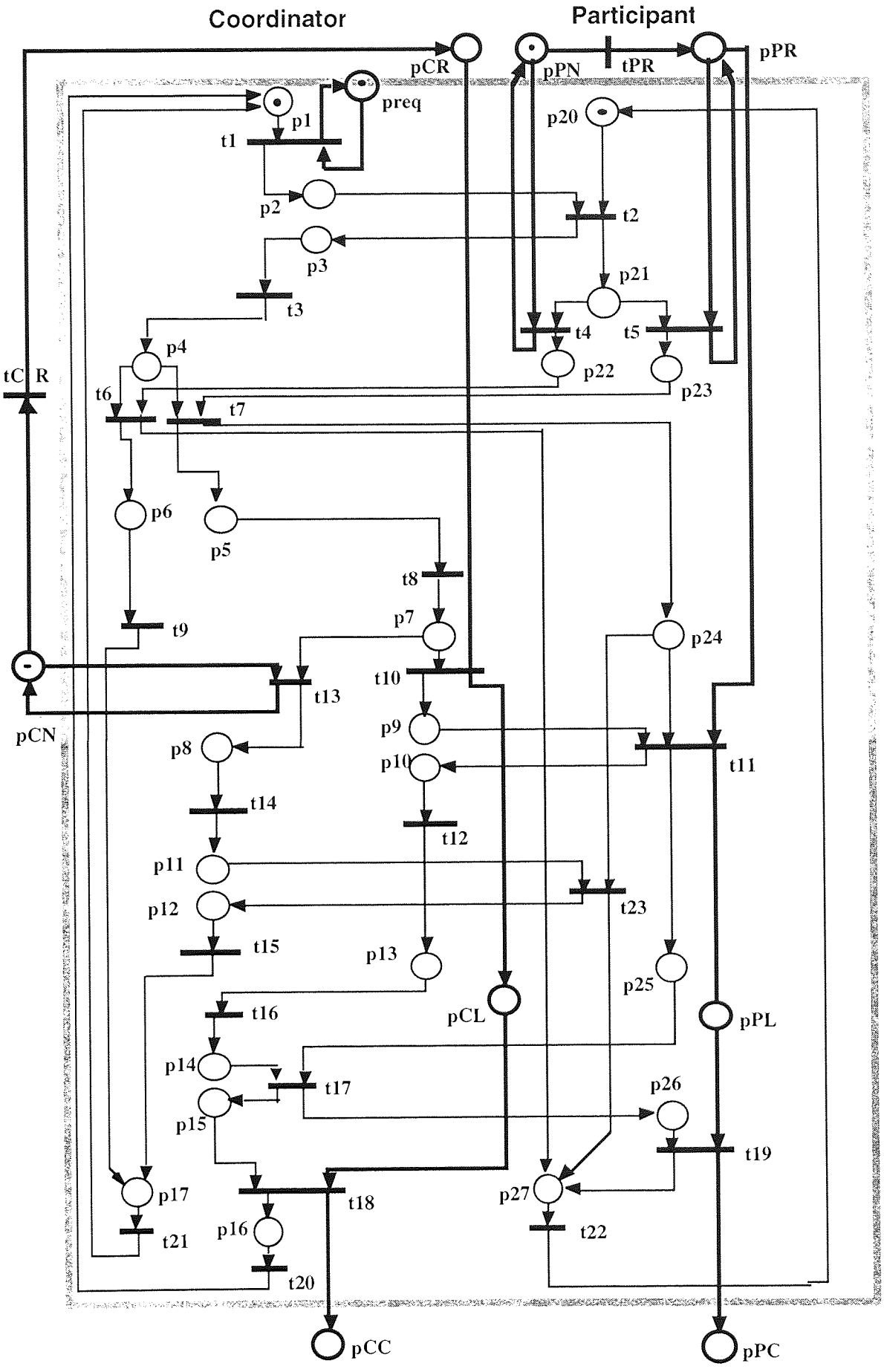


Fig.4.8. Petri net of 2-party E2PC protocol (shown on previous page)

The interaction of the coordinator and the participant can be seen clearly, through the transitions { t_2, t_6, t_7, t_{11}, t_{23}, t_{17} } which represent synchronous communications. In the FSM model of this protocol, Fig.4.4., this interaction is illustrated only through the notation used. The following are the equivalent communications in the Petri net:

i) t_2, the *start* message from coordinator to participant

ii) t_{11}, the *commit* message from coordinator to participant

iii) t_{23}, the *abort* message from coordinator to participant

iv) t_6, the *no* vote message from participant to coordinator

v) t_7, the *yes* vote message from participant to coordinator

vi) t_{17}, the *ack* message from participant to coordinator

The states in the FSM of Fig.4.4. that are directly equivalent (in terms of their semantics) to the Petri net of Fig.4.8. are shown as follows:

i) coordinator process:{ $q_1 \equiv p_1$ }, { $w_1 \equiv p_4$ }, { $a_1 \equiv p_{17}$ }, { $p_1 \equiv p_{14}$ }, { $c_1 \equiv p_{16}$ },

ii) participant process: { $q_i \equiv p_{20}$ }, { $p_i \equiv p_{24}$ }, { $a_i \equiv p_{26}$ }, { $c_i \equiv p_{27}$ }.

The user *request* that initiates the functions of the protocol in Fig.4.4. and is modelled by the tokenising of the place p_{req} in Fig.4.8. The places { p_{CN}, p_{CR}, p_{CC}, p_{CL}, p_{PN}, p_{PR}, p_{PL}, p_{PC} } are termed environment places. These are the minimal representation of the environment of the protocol necessary to generate all firing sequences of the model.

In the commit protocol each process must decide whether it is prepared to commit, this would depend upon the local state of the protocol. In the Petri net model this is represented by the state of the environment places. For instance in Fig.4.8., when the participant has received the *start* message (t_2) it is in state p_{21}, the decision to send the *yes* or *no* vote depends upon the state of the participant's environment places (p_{PN} and p_{PR}). The marking of these places determine the firing of either t_4 or t_5 (representing the *yes* or *no* vote decision). As can be seen from the Petri net, the firing of these transitions are mutually exclusive.

The initial marking of environment places p_{CN} and p_{PN} represents a particular behaviour model of the environment. This marking is sufficient to exercise all possible decision combinations for the protocol parties, coordinator and participant being able and unable to commit.

The *yes* or *no* vote decision for the coordinator process is represented by the firing of either t_{10} or t_{13}, determined by the marking of the coordinator's environment places p_{CN} and p_{CR}. In this Thesis, where the firing of a transition in a protocol is dependent upon an environment

place, the action of firing this transition is termed place sampling in the same manner as [Sagoo & Holding 92].

The decision by a process to vote *yes* to commit often requires the locking of local resources, this is represented by the places p_{CL} and p_{PL} for the coordinator and participant. The release of this local resource is represented by places p_{CC} and p_{PC}, this only occurs once the *ack* message has been sent and both protocol processes have made a transition to their committed state. For a control application the final environment places p_{CC} and p_{PC}, could represent the global state that both processes have committed to take a certain action which must be consistent, such as the lifting of a component by two independently controlled robot arms.

As for the FSM model, the abort message is only sent (t_{14}) when the participant has voted *yes* to the commit and the coordinator subsequently votes *no*, and the *commit* is sent if the coordinator votes *yes*.

The Petri net of Fig.4.8. is equivalent to the FSM model of Fig.4.4., but the former is more refined as the communications are shown explicitly and the environment conditions which determine the *yes* and *no* vote decisions are also shown. This Petri net model has been analysed and shown to be live, safe and deadlock free.

4.3.3. Multi-party E2PC Protocol

The commit protocol is modelled using Petri nets so that the intermediate states associated with communication can be shown explicitly, in order to consider failures at these points in the protocol. As described in section 4.3.1., the FSM model is only valid for representing failures in a two party protocol, an equivalent Petri net model was developed in section 4.3.2. This is extended to a multi-party protocol in order to consider the consequences of failures at the intermediate communication stages.

Fig.4.9. shows a Petri net model of the E2PC protocol with the addition of a second participant process. The sending and receiving of multiple messages between coordinator and participants are represented by local atomic actions, rather than the global atomic actions of the FSM commit protocol models.

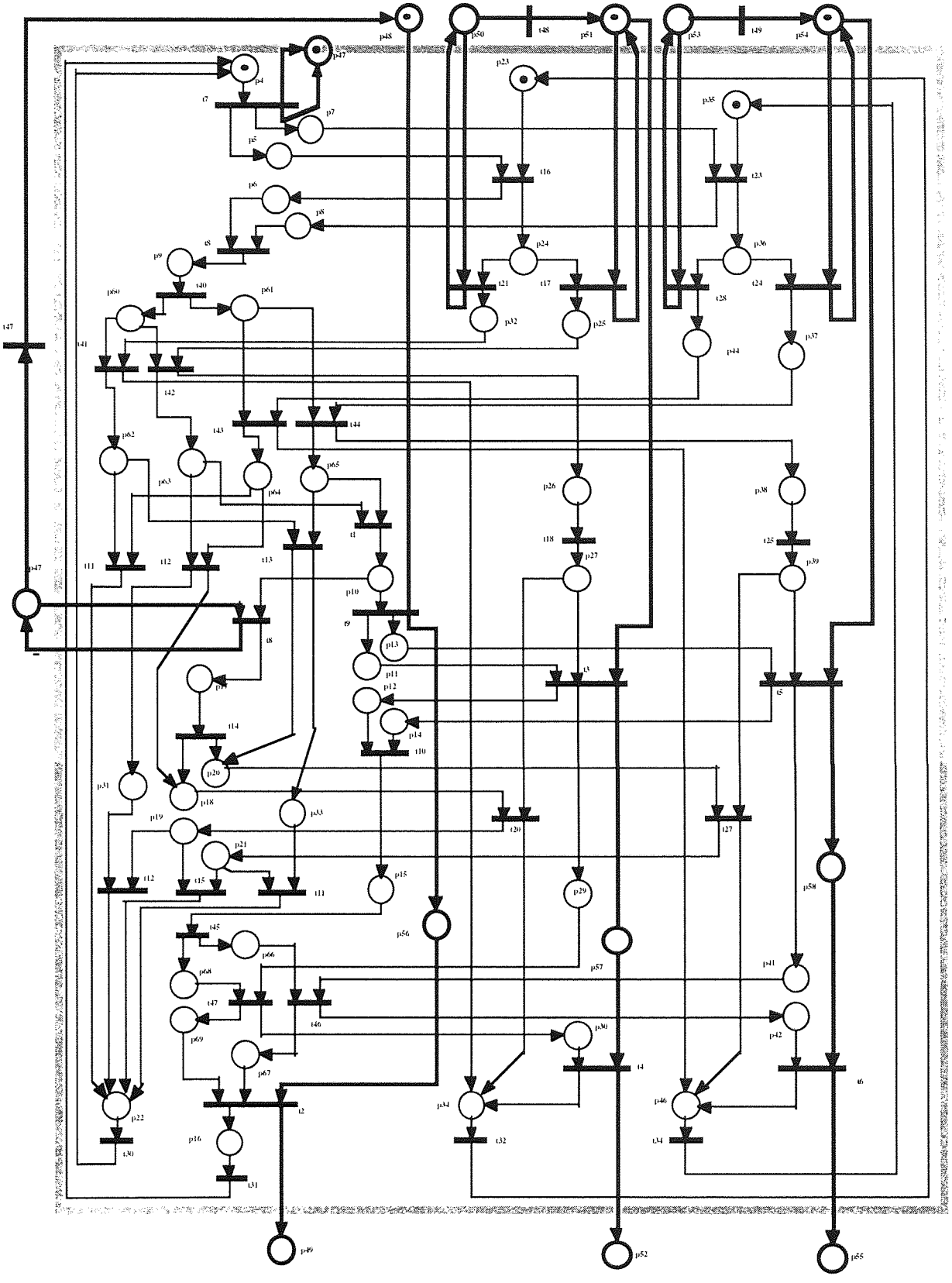


Fig.4.9. Petri net of 3-party E2PC

The voting mechanism of the coordinator process is represented by the firing of transitions { t8, t9 }. The enabling conditions of which are based on the votes collected from the participants, along with the place sampling of the coordinators environment places { p47, p48 }. The state space for the model was enumerated and analysed, and the protocol model

shown to be live, safe and deadlock free. The functionality was also assessed and found to be correct, the protocols behaviour is an atomic action, either all processes commit or all abort.

The places { p62, p63, p64, p65 } represent the intermediate states of the coordinator process, where the participants votes are collected. The intermediate communication states are also shown for the commit messages { p11, p12, p13, p14 }, the abort messages { p18, p19, p20, p21 } and the acknowledgement messages { p66, p67, p68, p69 }. The above Petri net can be extended with site and failure models in order to analyse the behaviour of the protocol when these occur.

4.4. Modelling Failures in Petri Nets

All the possible types of failure that can occur in the system must be included in the model, to analyse the effect of failures on the behaviour of the system [Leveson & Stolzy 87]. Site and communication failures can be included in the Petri net commit protocols. Failure transitions can be added to model the actions taken by sites when failures are detected, these were used in [Merlin and Farber 76][Skeen & Stonebraker 83] for analysing protocols, and in [Leveson & Stolzy 87] for the analysis of safety critical systems.

The site failures modelled include those types of failure that are termed:
i) *fail-silent* [Powell *et al* 91] or *fail-stop* [Bernstein *et al* 87], where a site stops processing completely,
ii) *fail-uncontrolled* [Powell *et al* 91], these being where arbitrary or malicious actions are performed by the failed site, and are very difficult to rectify, the problem is similar to solving the Byzantine generals problem [Lamport *et al* 82]. Those type of failures that involve erroneous execution, termed Byzantine failures [Levi & Agrawala 94], are not considered.

A Petri net model that includes site failures is shown in Fig.4.10. In this model, a site failure is represented by a transition that is shown in outline. The firing of a failure transition represents a total site failure, and the subsequent places { f_1, f_2 } represents the action taken by the failed site upon recovery.

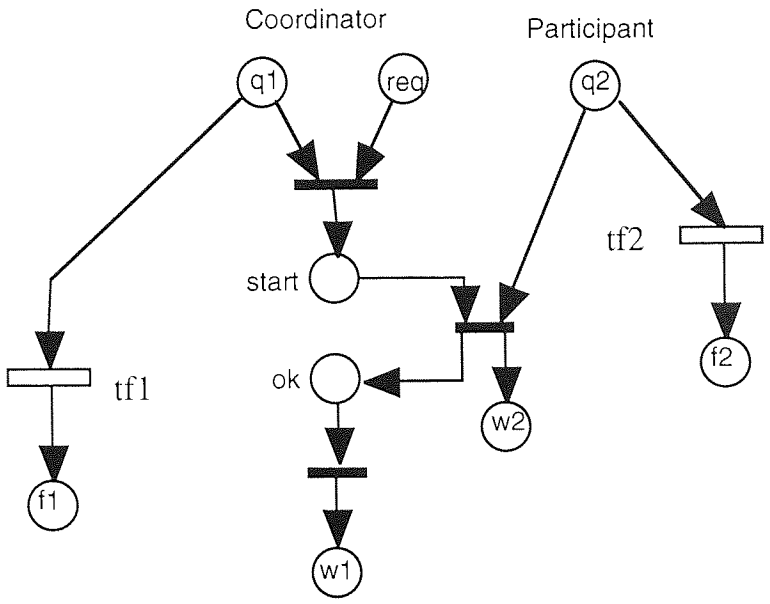


Fig.4.10. Modelling site failure using failure transitions

In Fig.4.10. the places f_1 and f_2 represent the failure states of coordinator and participant processes (respectively) at the first stage of a commit protocol. The recovery action taken by a site would depend on the last recovery log written by the site before the failure occurred.

The types of communication failure considered in this Thesis depend upon the communication subsystem. For synchronous communications only total communication failure, or link failure is considered. The use of point-to-point synchronous communications prevent message loss and message ordering failure. While, for asynchronous communications, failure can mean link failure and message loss.

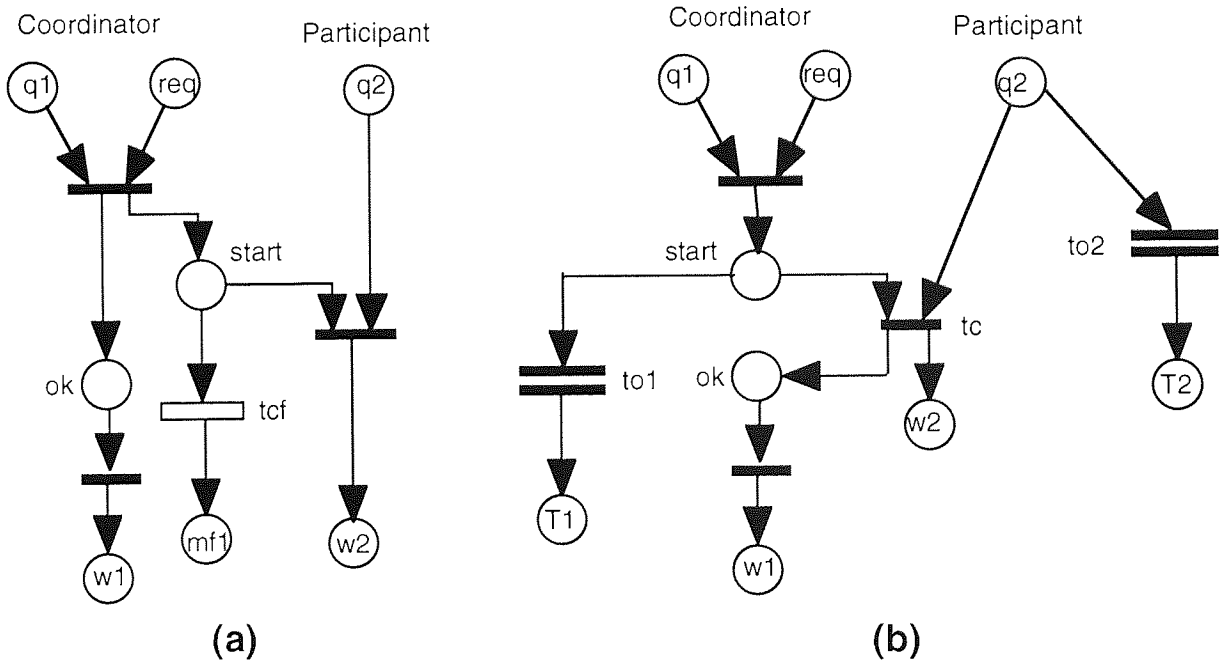


Fig.4.11. Modelling communication failure

Fig.4.11.(a) shows the Petri net representation of communication failure for asynchronous communications, where message loss is possible. This is modelled by a token being removed from a message place (*start*), due to the firing of a communication failure transition (t_{cf}) shown in outline. The removed token is placed in (m_{f1}) place in the Petri net model.

To simulate a link failure for synchronous communications, a communication transition that never fires is used, (i.e. t_c in Fig.4.11.(b)). Link failures are assumed to be a physical disconnection of a communication link between processors, and messages are of short duration so that disconnection cannot occur half way through a message. Fig.4.11.(b) shows timeout transitions (t_{o1}, t_{o2}), shown as a 'double bar', these transitions model the timeout mechanisms used to guard against deadlock, where a process waits indefinitely on a failed communication. The places marked T_1 and T_2 represent the action taken by a process when a timeout occurs and they indicate that a communication failure has occurred

The resilience of commit protocols to failures was studied in [Skeen & Stonebraker 83] and it was found that it depended upon the type and number of failures. The following was shown:
i) there is no commit protocol resilient to communication failure where messages are lost,
ii) there is no protocol resilient to multiple network partitions (where communications failure.

In [Hill 90] the use of point-to-point synchronous communications was used to overcome the possibility of message loss, and this is the approach taken in the rest of this Thesis. The commit protocol developed in the next sections will be designed to tolerate a single link failure leading to a single network partition.

The assumption is taken that the interprocess communication system delivers all messages within a pre-defined time limit and a late message is considered to be a failure (an inability to either send or receive a message). Sites are assumed to fail in a fail-silent manner [Levi & Agrawala 94], and that if a site sending a message fails, the receiver site will use a timeout mechanism to detect this occurrence. This has the implication that for a site waiting to send or receive a message, the failure of the site (with which it is attempting to communicate) is indistinguishable from a communication link failure. The site which detects the failure is required to take the appropriate action for both cases.

For real-time control applications, appropriate responses to failures must be built into the system. In [Leveson & Stolzy 87]. Three basic forms of acceptable response are defined:

i) fault-tolerant - a system continues to provide full functional performance in the presence of faults,
ii) fail-soft - a system provides continued service with degraded performance or reduced functionality,
iii) fail-safe - a system takes action to limit potential damage, and no attempt at continued service is attempted.

The recovery times of failed sites are large relative to the commit protocols normal duration and so can be considered unbounded. The recovery actions of failed sites modelled in the manner of Fig.4.10. are not considered, as in the intended real-time control applications (such as the applications considered in Chapter 6), site failures are expected to be infrequent and the recovery response is assumed to be a safe restart of all sites. The use of recovery logs is therefore unnecessary.

The response of the operational sites if communication failures are detected (due to a site or link failure) are of prime concern. Particular emphasis is placed on the communication, or link failures as these failure types are expected to predominate in distributed control applications. The E2PC protocol was chosen for development as this is known to be non-blocking [Skeen & Stonebraker 83] and can provide independent recovery with the addition of timeouts [Hill 90]. These properties of the protocol allow processing to continue at operational sites independently, in order to provide any required fail-safe response.

Within the commit protocols it is necessary to provide a timeout on either side of all synchronous communications in order to prevent deadlock. Deadlock can occur when a process is waiting indefinitely for a response, due to some site or communication failure. Deadlock states can be identified explicitly in the Petri net reachability tree. A timeout occurring initiates the actions assumed to be taken by other sites, when they detect that a site or communication link has failed. In Fig 4.11.(b)., the places T_1 and T_2 represent the actions associated with the timeouts { t_{o1}, t_{o2} }, these actions are used to reach a consistent final state for both coordinator and participants. However, the number of timeouts required (for the commit protocol such as Fig.4.9.) are quite large, as every input and output must be bounded with a timeout mechanism.

The commit protocol models (Fig.4.8. and Fig.4.9.) can be extended with the addition of link failures, in order to show where mechanisms are required in order to produce a resilient protocol. The protocols developed in this Chapter are enhanced with timeout mechanisms so as to tolerate the failures modelled in the Petri nets. The rules of Skeen & Stonebraker [83] for the placement of timeout transitions, and for determining the actions associated with timeouts occurring, do not apply to protocols using synchronous communications [Hill 90].

This is because the firing of the shared communication transitions, alters the state of the sender and receiver process concurrently.

4.4.2. Timeout actions

Each input place to a communication transition needs to be guarded with a timeout. As the concurrency set cannot be used to determine the action associated with such a timeout occurring (as used in [Skeen & Stonebaker 83]), inspection of the reachability tree will be used. Where link failures lead to deadlock, the action associated with a timeout expiring will depend upon whether the deadlocked state is commitable. From Fig.4.11.(b) it can be seen that a communication transition failing will result in a pair of timeout transitions firing{t_{o1}, t_{o2}}. For the commit protocol shown in Fig.4.9. each communication will have a pair of timeouts assigned to its input places (these timeout transitions will be denoted by T_1 to T_{18}). Each communication (in Fig.4.9.) will be considered in turn, to determine the required action for a timeout occurring:

i) p_5 and p_{23} are the input places to the communication transition t_{16}. Timeout transitions are assigned - (T_1 from p_5, and T_3 from p_{23}), that fire if communication fails and t_{16} cannot fire. This case represents the situation in which the coordinator and participant-1 are deadlocked, unable to send/receive the *start* message. These communication input places are not commitable states and so the action associated with the timeouts (T_1,T_3) occurring is to enable the abort state. The same reasoning follows for participant-2's communication, and the assigned timeout transitions T_2 and T_4. The timeout transitions connect the following places T_1 (p_5 to p_{22}), T_2 (p_7 to p_{22}), T_3 (p_{23} to p_{34}), T_4 (p_{35} to p_{46}}. (4.3)

ii) place p_{60}, and places p_{32} and p_{25} are the input places to the communication transitions t_{41} and t_{42}, (the same timeout is associated with sending the *yes* and *no* votes for the participant). Timeout transitions are assigned - (T_5 from p_{60}, and T_7 from p_{32} and p_{25}). The communication input place for participant-1 is not a commitable state, and so the participant times-out to the abort state. In this situation the coordinator will timeout the communication with participant-1, this is equivalent to receiving a *no* vote from participant-1. In this case, if participant-2 sent a *yes* vote then the coordinator will need to send it an *abort* message. The timeout action for the coordinator is thus the same as for receiving a *no* vote. The timeout transitions connect the following places T_5 (p_{60} to p_{62}), T_6 (p_{61} to p_{64}), T_7 (p_{32} & p_{25} to p_{34}), T_8 (p_{44} & p_{37} to p_{46}). (4.4)

iii) Timeout transitions are assigned - (T_{11} from p_{11}, and T_{13} from p_{27}). This situation represents a deadlock of the *commit* message from the coordinator. The action associated with a timeout occurring for a participant is to timeout to abort. The timeout action for the

coordinator requires an alteration to the protocol design. If a single link failure occurs then it can be assumed that one participant receives the *commit* message, and the other participant times-out the communication. In this situation the coordinator will need to send an *abort* message to the participant that received the *commit* message successfully. Thus an extra *abort* message and communication transition is required. For participant-1 the abort message would be sent to it in place p_{29}, and for participant-2 in place p_{58}. The timeout transitions connect the following places T_{11} (p_{11} to p_{18} & p_{31}), T_{12} (p_{13} to p_{20} & p_{33}), T_{13} (p_{27} to p_{34}), T_{14} (p_{39} to p_{46}} (4.5)

iv) Timeout transitions are assigned - (T_9 from p_{18}, and T_{10} from p_{20}). This situation represent the coordinator deadlocked, attempting to send the *abort* message. The use of the *abort* message will be altered through optimisation of the protocol using timeouts in the next section. The timeout transitions T_9 and T_{10} (coordinator to participant-1 and participant-2 communications) will not be required, as the *abort* message will only be required, as detailed in case iii) above. (4.6)

v) Timeout transitions are assigned - (T_{15} from p_{68}, and T_{17} from p_{29}), this situation represents where an *acknowledgement* message from a participant is deadlocked. The input places to this communication are commitable states, and so the timeout action taken by coordinator and participants is the same is if the *ack* was sent/received. All processes time-out to the commit the state. The timeout transitions connect the following places T_{15} (p_{68} to p_{69}), T_{16} (p_{66} to p_{67}), T_{17} (p_{29} to p_{30}), T_{18} (p_{58} to p_{42}} (4.7)

In section 4.5. the actions associated with timeouts occurring are used as a basis to optimise the E2PC protocol, in order to design a responsive commit protocol.

4.5. Protocol Optimisation

Fault-tolerant distributed real-time systems need a way to detect failures and initiate a recovery response consistently. This is achieved in this research by integrating timeout mechanisms into the commit protocols used for coordinating such systems. The timeouts are used to detect communication and site failures, and the actions associated with such timeouts occurring prevent coordination failure.

Responsive protocols are those that have real-time properties and are self-stabilising [Kakuda *et al* 92, 94]. A protocol is known as *self-stabilising* if it performs recovery from any abnormal state to a normal state [Gouda & Multari 91]. This form of protocol offers provides recovery without the use of exception handling routines when failures occur. The

E2PC protocol can be made responsive by using timeouts to detect failures (abnormal states), and actions associated with timeouts to perform recovery to normal states. In section 4.4. the actions taken by the E2PC protocol when timeout occurred were defined, these actions provide a form of error recovery. The timeout to the *abort* state can be considered as backward error recovery, and the timeout to the *commit* state as a form of forward error recovery.

As mentioned in section 4.4., the use of timeouts requires some alteration of the E2PC protocol design, such as the ability to send an *abort* message to a participant that has received the *commit* message. The actions associated with timeouts occurring can also be used to optimise the protocol:
i) by reducing the number of messages required,
ii) by allowing messages to be synchronisation signals that have no explicit information, the information is implied by the state of the protocol for the receiving process.

To determine the possible optimisations of the E2PC protocol, each of the timeout actions defined in section 4.4.2., will be considered in turn:
i) the actions associated with the timeout for the *start* message (4.3) are unchanged in the optimised E2PC protocol. Consider a single link failure at this communication stage. The coordinator and the participant that timeout this communication will proceed to the abort state. The participant that received the *start* message will then deadlock at the subsequent vote communication, and will timeout to the abort state. Therefore, all processes reach a consistent state. This optimisation removes the need for the coordinator to send an explicit *abort* message to the participant that received the *start* message;

ii) the action associated with a timeout of the vote communications (4.4) can be used to optimise the protocol. For a single link failure, the action taken by the coordinator when a timeout occurs is the same as for receiving a *no* vote. Therefore, the sending of an explicit *no* vote by the participant can therefore be removed. A participant that makes a *no* vote decision will proceed directly to the abort state. Using this optimisation the coordinator only receives *yes* vote messages, and so the message can be made a synchronising signal which is implicitly a *yes* vote. This type of optimisation which uses messages only for synchronisation and not data, avoids the problem of message corruption [Birman & Joseph 88]. This provides an efficiency in resources rather than time. There being a saving of one message transmission, at the cost of the protocol will always requiring the duration of the timeout to complete when a *no* vote decision is made;

iii) the action associated with a timeout of a *commit* communication (4.5) is the sending of the *abort* message. Where a timeout of a *commit* communication occurs, it can be assumed

that the other *commit* communication was successful due to a single link failure occurring. The action of the coordinator for this timeout occurring, is the sending of an *abort* message to the participant that received the *commit* message;

iv) the action associated with a timeout of the *abort* communication (4.6) allows an optimisation of the E2PC protocol. The design is optimised in that there are no *abort* messages sent to participants (that voted *yes*) when other processes (participant or coordinator) vote *no*. A timeout occurring for a *commit* communication has the same effect for a participant as receiving an *abort* message. Therefore, this timeout is used to replace the *abort* message. The only *abort* message, in the optimised protocol, is the one detailed in point iii), where a link failure has occurred. Under the assumption of a single link failure occurring no timeout is required on this *abort* message;

v) The timeout actions associated with the acknowledgement messages (4.7), are unchanged as the inputs to this communication are still commitable states.

Using the above approach the number of messages can reduced, the *no* vote is removed and the abort message is only required when a link failure has occurred. The *start* message could be removed, if the processes could be guaranteed to be initially synchronised. As the *start* messages main function is to synchronise the sites party to the commit protocol. For a practical system this initial synchronisation signal is necessary.

The E2PC protocol could be optimised by removing the *ack* (acknowledgement) message. The actions associated with the timeouts on this communication would bring all sites to a consistent (commit) state. However, the *ack* message is retained in the design as its use allows the commit protocol to complete before the commit deadline. The use of timeouts to replace the *ack* message would mean the time to reach the commit/abort decision would be determined by the timeout setting. The sending of the *ack* messages could occur before this deadline, and thus allow the commit decision to be reached earlier. This is efficiency in time rather than resources, one extra message transmission is required, but the protocol can complete earlier than the timeout deadline in the commit case.

The responsive commit protocol design incorporating these optimisations is shown in Fig.4.12.

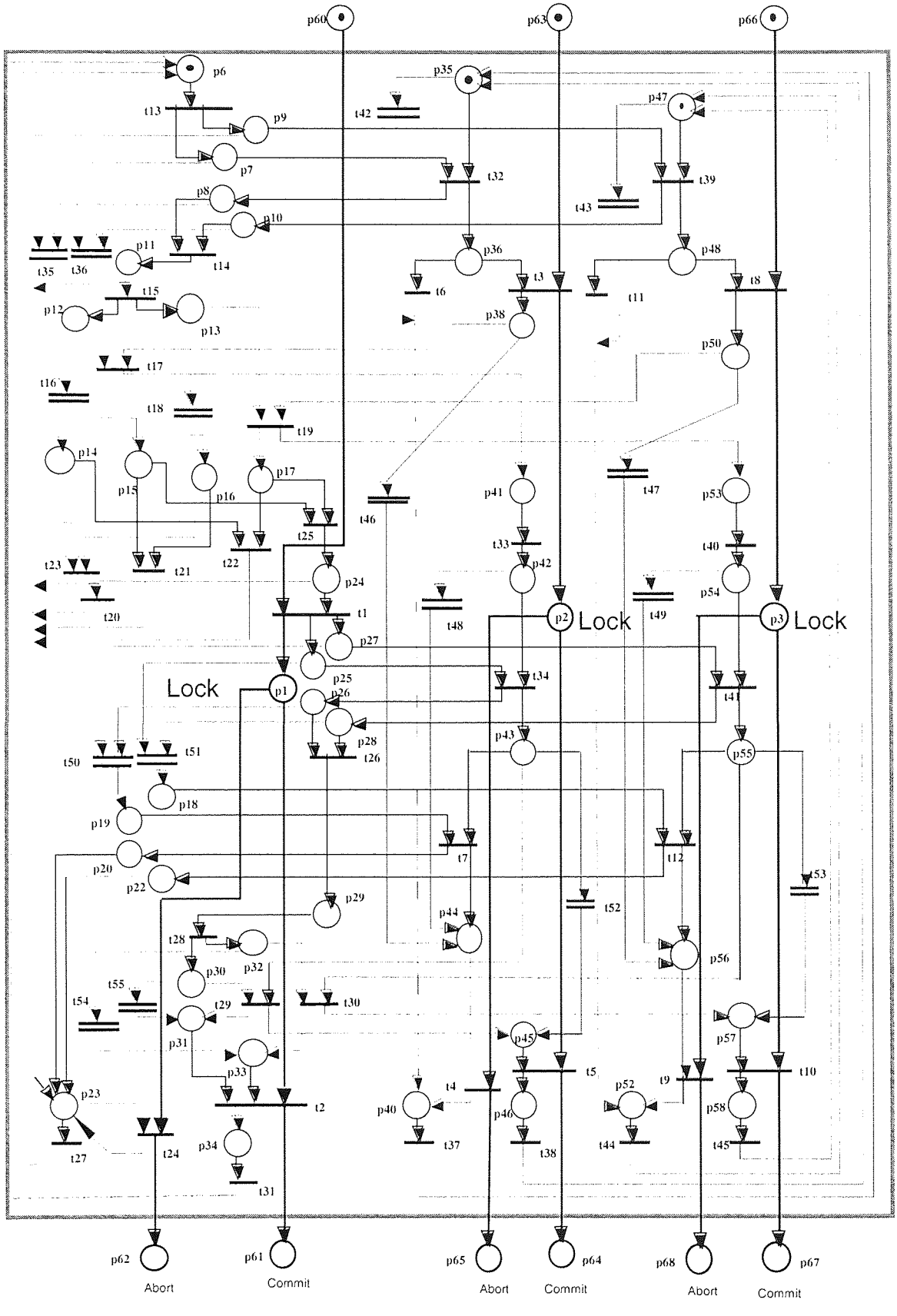


Fig.4.12. Optimised responsive commit protocol

The timeout transitions are shown clearly as double bars in Fig.4.12. The locking of local resources is shown by places { p_1, p_2, p_3 }, and the release of tokens in these places is to either the *abort* or *commit* final outcome states for the protocol.

There are two final states for each process reflecting the outcome of the commit decision p_c = { p_{61}, p_{64}, p_{67} }, and abort decision p_a = { p_{62}, p_{65}, p_{68} }. Examination of the reachability tree for this optimised commit protocol shows that it is non-blocking as expected. While, the application specific grading of these timeouts will allow the protocol to meet real-time deadlines. It can also be shown that inconsistent local states are not reached in normal or failure operation, for the clearly defined failure modes. The final global states of commit (p_c) or abort (p_a) are always reached. The reachability tree generated had 812 nodes and analysis showed the Petri net model to be live, safe and deadlock free for an initial marking M_0 { p_6, p_{35}, p_{47}, p_{60}, p_{63}, p_{47} }. The protocol always produced a consistent abort of commit decision for all processes.

The design of the optimised E2PC protocol provides both a timed atomic commit (TAC) and integral recoverability through the use of timeout actions. The use of timeouts in the protocols decision process allowed a reduction in the number of messages used and a simplification of the design, although this places a greater emphasis on the calculation of accurate timeout values.

4.6. Time Petri Nets & Timeouts

The protocol models need to include time as they are intended for use in real-time control applications, where deadline constraints for decisions are required and timeout values need to be calculated and analysed. Timing properties can be added to Petri net models in several ways, as described in chapter 2. The Time Petri nets of Merlin & Farber [76] are considered the applicable to the commit protocol model Fig.4.12.

Time Petri nets [Merlin & Farber 76] can be used to restrict the behaviour of the net and prevent false timeout conditions from occurring. This allows the desired outcome, to commit rather than *abort*, to occur whenever possible. The addition of timing constraints, using Time Petri nets, can restrict the set of possible markings of the un-timed reachability tree[Leveson & Stolzy 87]. In the work of [Srinivasan & Jafari 93] Time Petri nets are used in conjunction with the backward firing of transitions, to compute the maximum time before a watchdog (timeout) can safely be disabled. Backward firing of transitions involves tracing firing sequences back from a particular state, to a previous state of interest, i.e. where a decision was made or an action initiated.

The timing analysis possible through the use of Time Petri nets [Merlin & Farber 76][Leveson & Stolzy 87][Berthomieu & Diaz 91] and [Bucci & Vicario 95] allow the derivation of constraints on timeout settings to be estimated directly from the net. However, knowledge of the underlying communication system used and its performance is required to perform this estimation.

Using Time Petri nets minimum and maximum firing times are associated with transitions, these are the earliest firing time (EFT) and latest firing time (LFT) for the transition once enabled. Consider the timeout guarded communication shown in Fig.4.11.(b), each transition has an earliest and latest firing time associated with it, which determine when it fires after the transition has become enabled. The following constraints on the timeout values are required in order to prevent false timeouts,

i) EFT (t_{O1}) > LFT (t_C),

ii) EFT (t_{O2}) > LFT (t_C),

Where t_C is the communication transition, t_{O1} and t_{O2} are the timeout transitions.

The timeout settings for the responsive E2PC protocol, Fig.4.12., can be given in terms of the behaviour of the Petri net. Timeout transitions differ from the other transitions in that an extra delay, the timeout value (D_t), is associated with them, modelling the value of the timeout setting. The grading of timeout settings must guarantee deadlines by preventing deadlock states, while also preventing false (early) timeouts. Preventing deadlock states can be essential to a real-time control system, as in such applications providing a late response can be as hazardous as performing an incorrect action. The actions associated with a timeout occurring guarantee a response within a certain time.

The following will show how Time Petri nets can be used for deriving constraints for the setting of timeouts.

Consider the timeout transitions associated with the sending and receiving of the *yes* vote from participant-1 to the coordinator process, t_{16} and t_{46} in Fig.4.12. The last synchronisation point for the two processes was the firing of communication transition t_{32} (the *start* message). A synchronisation point is the firing of a transition common to both processes, representing a synchronous communication event. This is represented by point (0) in Fig.4.13., and the transition firing times will be related to this point in the firing schedule.

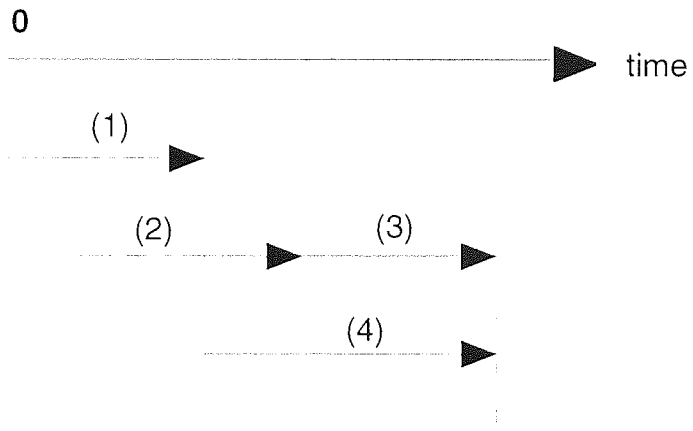


Fig.4.13. Firing schedule for timeout

The setting of timeout t_{46} is considered first, this is the timeout that prevents deadlock of the *yes* vote communication (sent from participant-1 to the coordinator), represented by transition t_{17} in Fig.4.12.

From the firing of t_{32} there are two sequences that need to be considered that enable transition t_{17}: the firing of t_{32} before t_{39}, or the firing of t_{39} before t_{32}. Assuming the worst case for successful communication, the timeout transition is enabled at the earliest point EFT(t_{32}) and the communication transition is enabled at the latest point LFT(t_{39}). The times from the firing schedule Fig.4.13. are then given:

(1) the earliest time that the timeout t_{46} is enabled is EFT(t_3).

(2) the latest that communication transition t_{17} is enabled is

$$\text{LFT}(t_{15}) + \text{LFT}(t_{14}) + [\text{LFT}(t_{39}) - \text{EFT}(t_{32})].$$

(3) the latest time that communication can occur is LFT(t_{17}).

(4) the earliest time the timeout can fire is D_{t46} + EFT(t_{46}).

From Fig.4.13. it can be seen that:

(4) = (3) + (2) - (1), replacing these with the times above the minimum timeout setting of D_{t46} for the participant to avoid false timeout is given by:

$$D_{t46} \geq \text{LFT}(t_{17}) + \text{LFT}(t_{15}) + \text{LFT}(t_{14}) + [\text{LFT}(t_{39}) - \text{EFT}(t_{32})] - \text{EFT}(t_3) - \text{EFT}(t_{46}) \qquad (4.8)$$

Consider a firing sequence that enables the timeout transition before the communication transition (where the firing times given by EFT(t_{32}) and EFT(t_{39})), then the setting of the timeout transition t_{16} for the coordinator is given by:

(1) the earliest time that the timeout t_{16} is enabled is EFT(t_{15}) + EFT(t_{14}).

(2) the latest that communication transition t_{17} is enabled is LFT(t_3).

(3) the latest time that communication can occur is LFT(t_{17}).

(4) the earliest time the timeout can fire is D_{t16} + EFT(t_{16}).

The setting of t_{16} to avoid false timeout is given by

$$D_{t16} \geq LFT(t_{17}) + LFT(t_3) - EFT(t_{15}) - EFT(t_{14}) - EFT(t_{16}) \qquad (4.9)$$

The settings for the rest of the timeouts associated with the coordinator and participant-1, and the timeouts associated with participant-2 can be shown in the same manner.

The deadline for the protocol to produce a *commit* or *abort* decision can be given as D_d. The timeout value and the minimum and maximum firing times for the timeout pairs are assumed to be the same; i.e. for the coordinator the pair of timeout transitions assigned to the vote communication (t_{16} and t_{18}), $D_{t16} = D_{t18}$, $EFT(t_{16}) = EFT(t_{18})$ and $LFT(t_{16}) = LFT(t_{18})$. This follows for participant timeout pairs, such as t_{46} and t_{47}. If absolute times are assumed then the constraints of the timeout settings for the responsive E2PC protocol, Fig.4.12., can be given:

i) $D_{t55} + LFT(t_{55}) + LFT(t_2) < D_d$, timeout to commit outcome for coordinator (p_{61}).

ii) $D_{t52} + LFT(t_{52}) + LFT(t_5) < D_d$, timeout to commit outcome for participant-1(p_{64}).

iii) $D_{t53} + LFT(t_{53}) + LFT(t_{10}) < D_d$, timeout to commit outcome for participant-2(p_{67}).

iv) $D_{t50} + LFT(t_{50}) + LFT(t_7) + LFT(t_{24}) < D_d$, the timeout action of the coordinator is to initiate the sending of an abort signal to participant-1, and reach an abort state (p_{62}).

v) $D_{t16} + LFT(t_{16}) + LFT(t_{23}) + LFT(t_{24}) < D_d$,timeout of *yes* vote communication from participant-1, followed by the abort state (p_{62}).

vi) $D_{t46} + LFT(t_{46}) + LFT(t_4) < D_d$,timeout of *yes* vote communication to coordinator, to reach an abort state (p_{65}).

The constraints of case v) and vi) give the maximum values for the setting of timeouts D_{t46} and D_{t16}. These can be combined with the minimum settings to avoid false timeouts given above, in order to set the upper and lower limits on the timeouts. The other timeout constraints can be given in the same manner.

4.7. Conclusion

In this Chapter the use of FSM for modelling commit protocols [Skeen & Stonebraker 83][Yuan & Jalote 89][Levi & Agrawala 94][Yoo & Kim 95] was considered, and the limitations of this approach were discussed. Petri nets were then chosen for the development of a responsive commit protocol as used in [Hill & Holding 90][Hill 90].

The limitations of Skeen & Stonebaker's [83] rules for failure and timeout transitions were considered, and the modification of these rules made in [Yuan & Jalote 89] and [Hill 90] were used in the commit protocol design. The representation of multiple communications as

atomic actions in the FSM protocol models, highlighted the benefit of using a Petri net model with explicit communications shown, as an approach to commit protocol design and analysis.

The communications used in the commit protocol are concerned with the flow of control information rather than the transmission of data, because of this such communications can be expressed easily in terms of synchronous events [Carpenter 92] such as used in the Petri net protocol models.

The commit protocol developed was enhanced with timeout mechanisms and associated recovery actions, so as to tolerate the clearly defined set of hardware faults modelled. Once the Petri net model of the E2PC protocol, shown in Fig.4.8. was extended with communication failures, examination of the resulting reachability tree showed where these recovery and timeouts mechanisms were required. The addition of these mechanisms resulted in the Petri net shown in Fig.4.12.

An alternative method for the placement of timeout transitions by examination of the reachability tree was given by Hill [90], and this approach was applied to a multi-party commit protocol in this chapter. The rules of Skeen & Stonebaker [83] are only applicable to systems using asynchronous communication and for single coordinator/participant systems as shown in [Yuan & Jalote 89]. This is because in FSMs the state transitions representing message transfer alter the states of two process concurrently.

The multi-participant protocol developed in section 4.3 and optimised in section 4.4 offers a timed atomic commitment, along with inherent recovery and fault tolerant properties. The protocol is complete in that the timeout mechanisms used to detect communication failure are part of the protocol, and no other sub-protocols are required. Only *failure atomicity* is of concern, the preserving of consistency in the presence of failures, and not the concurrency control of transaction processing that is used to achieve serialisability.

The Petri net model was then transformed to a Time Petri net model in section 4.6, by associating minimum and maximum firing times with transitions. These firing time bounds are used to illustrate a method for calculating relative timeout settings. This method is dependent upon the accuracy with which communication times can be estimated for the intended applications.

Standard and Time Petri net analysis of the developed commit protocol, with timeouts and using synchronous communication, has shown that it provides a timed atomic commitment that it is resilient to link and site failures. This was shown by the inclusion of failures within the Petri net model and analysing the subsequent behaviour of the protocol.

The commit protocols developed are intended for use in the applications addressed in chapter 6., which concern the coordination and synchronisation of distributed controllers for high speed, independently driven machinery.

An independently recoverable commit protocol augmented with timeouts is well suited for use in a real-time environment since it has the capability of surviving link failures, and satisfying timing constraints. Synchronous communication should be used where the application is safety critical and link failures are expected, in order to prevent message loss.

However, it should be noted that the application of the commit protocols developed in this chapter, for the coordination of distributed controllers, would depend upon an implementation of the controller using point-to-point synchronous communication. The protocol was optimised, with timeouts being used to provide implicit information for the *no* votes and *abort* messages, this allowed the use of synchronising signals rather than data messages, and removed the need to detect and tolerate data corruption. The robust, real-time protocol design and protocol optimisation rely upon the use of synchronous communication, and accurate information on the timing of communications is needed in order to calculate timeout values. This requirement for synchronous point-to-point communication is a necessary but limiting factor for the use of the robust real-time commit protocol developed.

The standard Petri net design and analysis of the multi-party, responsive commit protocol developed in this chapter is used as a comparison for the modular approach to design and analysis presented in Chapter 5. The functional and temporal behaviour of the protocol presented in section 4.6. will form a template for the modular commit protocol designs in Chapter 5.

As can be seen from Fig.4.12., the final Petri net protocol design with the inclusion of explicit communications and all timeout mechanisms, is large and complex, which results in a correspondingly large reachability graph. The automation of the reachability graph and concurrency set generation aided the analysis, but the state space was still prohibitively large (1563 nodes). The combination of the commit protocol and distributed control system model could prove difficult to analyse, due to the exponential growth in reachable states. The modular approach to the design and analysis of commit protocols presented in Chapter 5 is intended to overcome these drawbacks to the use of Petri nets models and to reachability based analysis.

Chapter 5 Modular Design & Analysis of Commit Protocols

5.1 Introduction

In chapter 4 the development of responsive atomic commit protocols using Petri net techniques was presented. It was shown how Petri nets offered an improvement on FSM based methods for the modelling and analysis of protocols. The use of Time Petri nets [Merlin & Farber 76] was also shown to be an aid to the grading of the timeout mechanisms required to guarantee real-time deadlines.

The intended application for the commit protocols developed in chapter 4 is in the coordination of distributed controllers. These controllers will be independent and use message passing communications in order to synchronise and coordinate there operation. This coordination will require the use of fault-tolerant real-time commit protocols. It is intended to use Petri nets for the design of the distributed controllers, and then to embed the Petri net commit protocol models within these controller designs. This composition of the controller and the protocol models should allow the analysis of the complete controller system in a unified manner.

State enumeration is a main method of analysis for Petri nets, involving the elaboration of the complete reachable state space of the model. This is termed reachability analysis as each reachable process state is considered to determine whether a given property, such as *boundedness* and *liveness*, is satisfied [Dwyer *et al* 95].

It is anticipated that the Petri net designs for the distributed controllers will be both large and complex. This will result in a correspondingly large state space, due to the problem of state space explosion [Scholefield 90], discussed in chapter 3. This problem can render state enumeration based analysis costly, if not intractable. It is intended to use a modular approach to both the development and analysis of Petri net models in order to cope with the problem of a large state space. This modular approach will conceal local events of sub-nets, whilst preserving properties of interest, such as boundedness and liveness. The systematic concealment of local events allows the management of state space, and permits a modular approach to the analysis of complex systems [Bucci & Vicario 95].

A set of reusable templates are developed in order to reduce the complexity of the Petri net protocol models, and to aid the readability of designs. These templates are termed

communication blocks and are based upon the sets of communication actions within the commit protocol. The structure and behaviour of these communication blocks are modelled and analysed using Petri net and Temporal Petri net methods. In this approach a 'divide and conquer' philosophy is used to handle complexity in both the design and the analysis of commit protocols.

In section 5.2. it is shown how communication blocks can be used in the modular design of a protocol model that is equivalent to the protocol model developed in section 4.5. The communication blocks are reusable templates of interprocess communications, and the analysis of their structures are used to aid the analysis and formal verification of properties of the protocol design.

In section 5.3. the notion of *well formed multi-blocks* (WFMB) is developed from Valette's [79] *well formed blocks* (WFB). The WFMBs are used to formalise the reusable Petri net templates termed *blocks*. The application of these blocks to the atomic commit protocols developed in order to create what are termed *commit blocks* is shown. The application of these commit blocks models within the design and analysis of distributed controllers is then discussed.

5.2 Communication Block

In this section a method for reducing the complexity of the Petri net protocol models by abstracting sets of communication actions using communication blocks is presented. It is then shown how a modular approach to the design of commit protocols can be taken using these templates. This approach is illustrated through the modular design of a multi-party E2PC protocol, which is equivalent to the model presented in section 4.5. The templates are analysed using Petri net theory to gain knowledge of their behaviour. The repeated instance of these templates allows the repeated use of their analysis. This is applied to the analysis of the protocol designed using the communication block approach.

5.2.1 Communication structures

As can be seen from the protocol developed in section 4.5. (see Fig.4.12.), each phase of the protocol consists of a series of timeout-guarded synchronous communications, that involve either two or three independent processes.

Fig.5.1. shows a model of the timeout guarded two-party synchronous communication, (as used in Fig.4.12.) for the *abort* message sent from the coordinator process to one of the participant processes. The outcome of this structure is either a successful communication of the two processes or the firing of both processes timeout guards, aborting the communication

attempt. The communication failure could be due to a late communication or a link failure. This simple communication structure is represented by nine places and six transitions, which includes two timeout transitions { t_2, t_6 }.

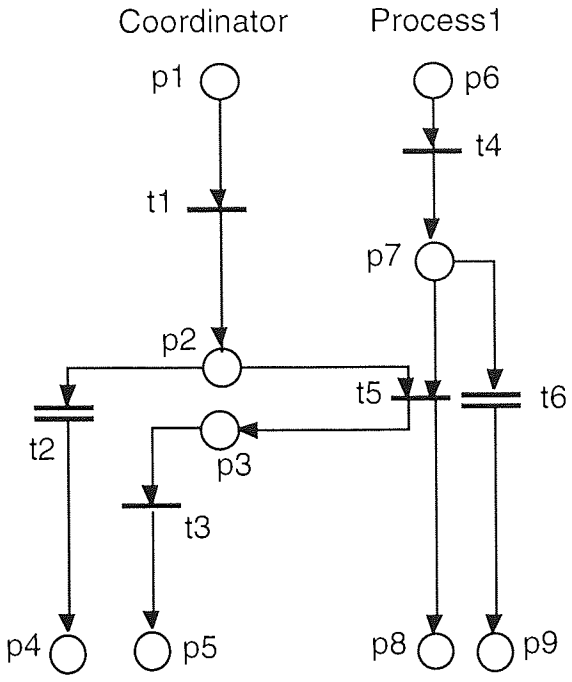


Fig.5.1. detail of 2-party synchronous communication with timeouts

The three-party version of this structure is illustrated in Fig.5.2. This communication structure differs from the above, in that the process termed *coordinator* , also requires a decision mechanism to differentiate between the outcome of the communication with each processes. The outcome of this decision mechanism indicates the success of each communication attempt.

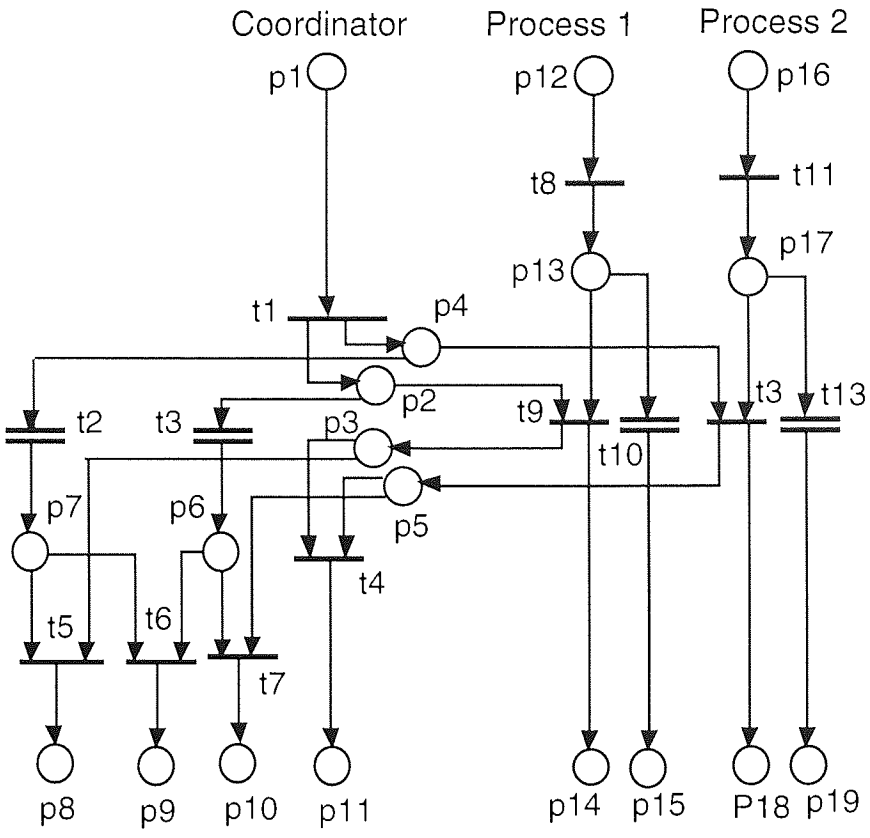


Fig.5.2. detail of 3-party communication with timeouts

As can clearly be seen from Fig.4.12., a protocol design with fully elaborated communication structures can become highly complex. The use of abstractions of the above two communication structures (Fig.5.1. and Fig.5.2.) would offer a technique to simplify the protocol design. A reduced model of the synchronous communication structures, termed blocks, could serve as a reusable template. The use of these blocks as templates would offer a method of modular design for protocols. The blocks could be used in protocol designs to avoid explicitly modelling communication using multiple transitions and places. It is the intention of this research to reduce the complexity of the Petri net protocol models (such as Fig.4.12.) using this method.

As noted in [Christensen & Hansen 94], the modelling of systems using hierarchical or modular Petri nets would be aided by the use of a construct that modelled the interaction between modules. Where these constructs represented message passing communication, their use would aid the design of distributed systems based on this form of interprocess communication. The work of Christensen & Hansen [94], used this approach with coloured Petri nets [Jensen 90], but the concept applies equally to the standard Petri nets used here.

The use of templates that model communication structures in modular Petri net design, would allow greater emphasis to be placed on the system to be modelled, rather than on the communication sub-system and synchronisation primitives inherent in the design. A similar

approach has been used in state machines [Shaw 92] and Petri nets [Christensen & Hansen 94] for the composition of modular sub-nets, based on the use of communication channels or ports [Bucci & Vicario 95] . The concept of these communication channels is influenced by CCS [Milner 80, 89] and CSP [Hoare 85]. Without these communication constructs, it is necessary to explicitly model each communication using transitions and places, resulting in a complex net structure. However, the approach of [Christensen & Hansen 94] is not practical for the analysis of timing or for safety properties.

The use of *checkpoint* transitions in [Shieh *et al* 90] is a similar approach, but is not applied to modular Petri nets. The taking of a checkpoint in a distributed system requires the agreement of all processes to record their current states for error recovery purposes. This would typically involve the use of an agreement protocol, requiring interprocess communications in order to synchronise and reach agreement. In [Shieh *et al* 90] Petri net designs of distributed systems that use a single transition to represent this checkpoint action are presented. This *checkpoint* transition is a high level abstraction of the underlying communication structures involved.

5.2.2 Communication blocks

The communication *blocks* structures illustrated in Fig.5.3.(a) and (b), represent abstractions of the two communication mechanisms illustrated in Fig.5.1. and Fig.5.2. These show only the initial and final places for each process communicating (coordinator or participants). This reduces the Petri net structure to the external behaviour of the communication. Each process has a single input place and a pair of output places, the output places represent the two outcomes of the communication.

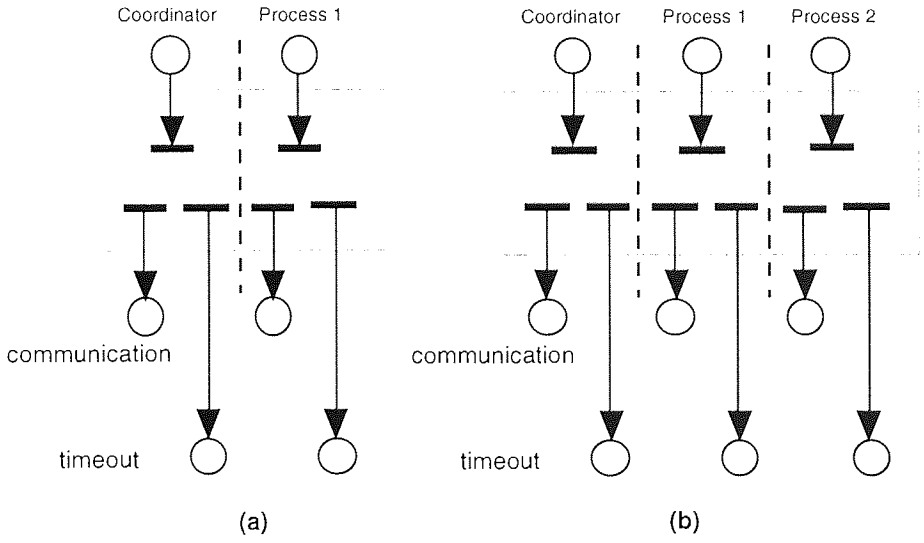


Fig.5.3. communication block with timeout, (a) two-party, (b) three-party.

The use of communication blocks as reusable templates is illustrated through the modelling and analysis of the two-party timeout guarded communication. Fig.5.4. shows a simple

synchronous point-to-point communication between two processes. This is extended with timeouts in Fig.5.5. in order to make the communication resilient to link failure, in the same manner as used in the commit protocols of chapter 4. It can be seen that each process in Fig.5.4. has a single path between its input and output places and is termed a single-input single-output (SISO) block. However, for Fig.5.5. each process has two output paths, one path represents the timeout and the other a successful communication. This type of structure is termed a single-input multiple-output (SIMO) block.

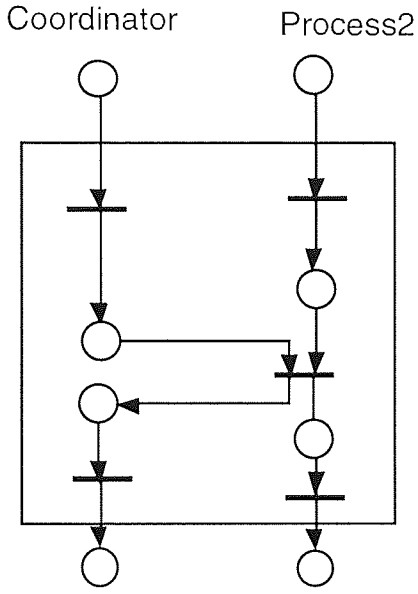


Fig.5.4. 2-party communication block

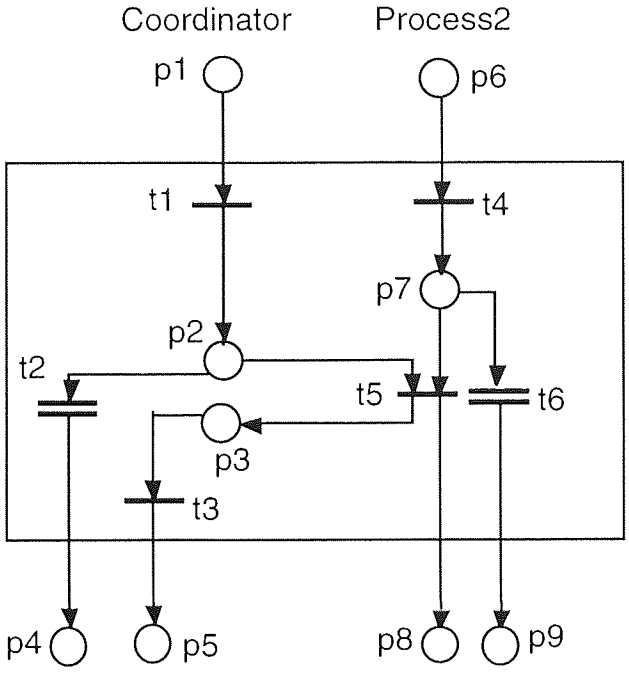


Fig.5.5. 2-party communication block with timeout

The reachability tree is shown for the Petri net structure of Fig.5.5., and the marking for each state is given in Table 5.1. The reachability tree is based on an initial marking of the Petri net $M_0 = \{ p_1, p_6 \}$.

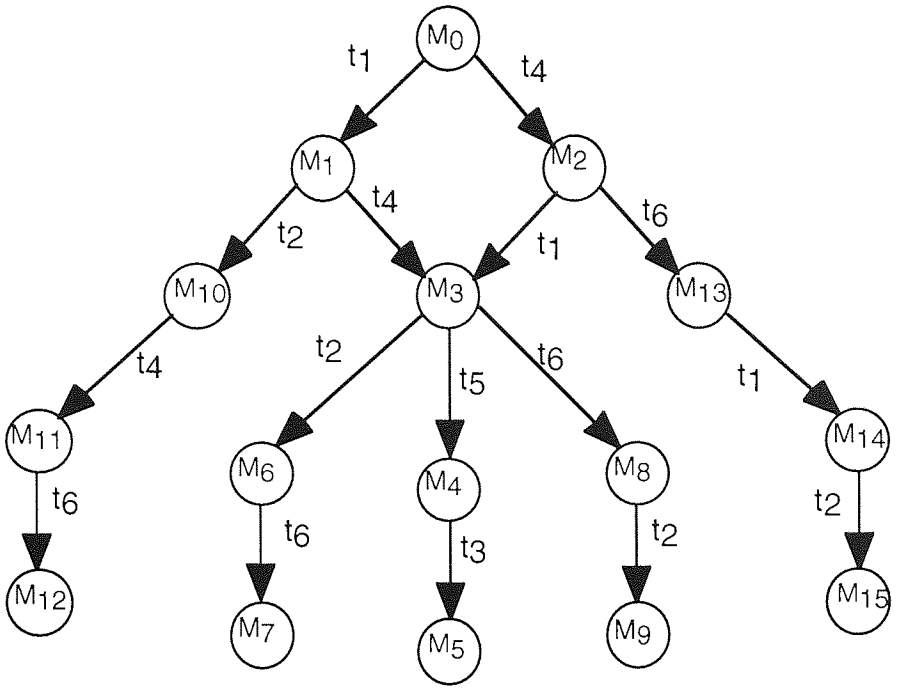


Fig.5.6. 2-party communication block reachability tree

Marking	Places	Marking	Places
M_0	p1, p6	M_8	p2, p9
M_1	p2, p6	M_9	p4, p9
M_2	p1, p7	M_{10}	p4, p6
M_3	p2, p7	M_{11}	p4, p7
M_4	p3, p8	M_{12}	p4, p9
M_5	p5, p8	M_{13}	p1, p9
M_6	p4, p7	M_{14}	p2, p9
M_7	p4, p9	M_{15}	p4, p9

Table.5.1 Markings from the Petri net Fig.5.6.

As can be seen from Figs.5.5. - 5.6., that even for a relatively small Petri net the reachability tree can become relatively large. The concurrency set (not shown) offers a more compact representation of certain net properties. For instance, the concurrency set for the coordinator place that represents a timeout occurring { p4 } state is given by C(p4) = { p6, p7, p9 }. This represents all the concurrent states that it is possible for the participant process to occupy while the coordinator is in this state. The concurrency set of the final places (denoted C_f) of each process offers a concise view of the outcome of the communication, and thus of the outcome of the communication block. Thus $C_f(p5)$ = { p8 } from marking { M_5 }, and $C_f(p4)$ = { p9 } from markings { M_7, M_9, M_{12}, M_{15} }. The final markings { M_5, M_7, M_9, M_{12}, M_{15} } along with the initial marking M_0 define the external behaviour of the communication block.

For the multi-party communication block, such as Fig.5.3.(b), a typical communication structure is shown explicitly in Fig.5.7. For this structure, with all input places marked, the possible outcomes of the communications are either:

i) Communications successful { C_c, $P1_c$, $P2_c$ },

ii) Coordinator - Process-1 timeout { C_a, $P1_a$, $P2_c$ },

iii) Coordinator - Process-2 timeout { C_a, $P1_c$, $P2_a$ }, or

iv) Timeout of both communications { C_a, $P1_a$, $P2_a$ }.

These sets of final places (i) - (iv), along with the initial marking, represent the external behaviour of a multi-party communication block that is an abstraction of this communication structure.

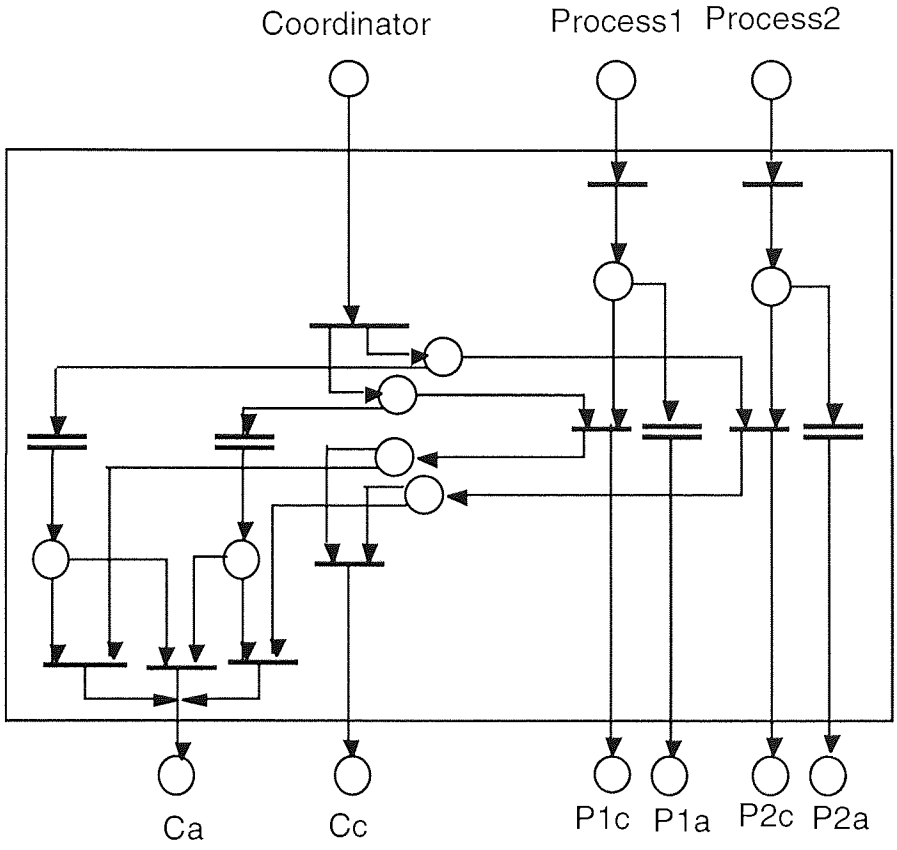


Fig.5.7. 3-party communication block with timeout

The Petri net reachability graph and concurrency sets for this structure (Fig.5.7.) are given in Appendix.A.1. Using these it has been shown that the net is live, 1-bound and deadlock free.

For each of the three processes in this Petri net, there should be at most one final place marked per firing sequence. For the coordinator process there are two final places, these represent communication complete (C_c) and abort (C_a). The marking of these two places should be mutually exclusive. Examination of the markings of the final places in the reachability graph, not shown, confirms this mutual exclusion property. This property holds for each process, as can be seen from the net.

There are a limited number of combinations of final place markings for this block structure. For instance the coordinator final place, communication complete (C_C), should only be tokenised when the participants final places $P1_C$ and $P2_C$ are tokenised. It can be seen from the net that case (i), Communications successful { C_C, $P1_C$, $P2_C$ }, only occurs through a firing sequence that does not include a timeout transition.

The set of clearly defined final place markings for each block, will be used in the composition of the blocks. The analysis of the blocks underlying template can be reused whenever the block is used.

5.2.3 Hybrid protocol design

In this section a commit protocol design is developed, using the communication blocks as templates for the communications required at each stage of the E2PC protocol (*prepare*, *vote* etc.). The use of communication blocks allows a hybrid approach to the commit protocol design. Hybrid approaches to Petri net design [Zhou and DiCesare 89, 93] were discussed in section 3.7.

The protocol synthesis is termed a hybrid approach as it combines both a top-down refinement of the basic protocol structure with a bottom-up composition of the communication blocks. The composition of the blocks preserves properties such as liveness and boundedness, in order for the analysis of the blocks subnet to be reusable.

In Fig.5.8. below, a refinement of the commit protocol structure that is the template for the E2PC protocol is shown. This is the top-down stage of the design. Each of the basic steps of the E2PC protocol (*prepare, vote, commit, abort & acknowledge*) are refined using the communication blocks of Fig.5.3. This was achieved by examining the protocol structure, as modelled in Fig.4.12., and identifying each set of communications that could be modelled by a template such as Figs.5.1. or 5.2. Each instance of this communication structure was then represented by the appropriate communication block, Fig.5.3.(a) or (b). These communication blocks are templates of either 2 or 3-party communications with timeouts.

The bottom-up stage of the design process involves the composition of communication blocks. This is achieved by merging the input and output places of the blocks with a minimal set of places necessary to model the protocol. This minimal set of places would include the decision places { p24, p36, p48 }, and the abort places { p23, p40, p52 } for each process in Fig.4.12.

In order to compose the various communication blocks of the protocol, it is necessary to:
i) examine the Petri nets for each communication block,
ii) formally define the external behaviour of each communication block (the input and output place markings),
iii) analyse the Petri net for the defined external behaviour,
iv) determine the appropriate action for each outcome of the communication block,
v) merge the communication blocks output places with the places representing these actions.

Step (i) determines which templates are used for each communication block in the top-down design (Fig.5.8.). Step (ii) defines the input and output constraints for each communication block. In step (iii) the external behaviour of the communication block is checked against the behaviour of the Petri net template . This involves determining the possible sets of output place markings of the net, using the input place markings as an initial marking (M_0). The input and output constraints on the communication blocks, defined in step (ii), are then used to determine the correct interconnection of the block structures, in steps (iii) and (iv).

To illustrate the merging of places (steps iv and v) consider the coordinator process for the *prepare* communication block in Fig.5.8. When the coordinators input place to the communication block becomes marked, this will result in one of the coordinators output places becoming marked. These two output places represent communication complete (C), or communication abort due to time-out (A) in Fig.5.8. The action associated with the timeout outcome is to enter the coordinators abort state, (represented by place (p_{23}) in Fig.4.12). So the coordinator output place (A) for the *prepare* communication block is merged with the abort place (p_{23}). The action associated with the success outcome of the *prepare* communication block, is for the coordinator to receive the participants votes. To do this would require the coordinators input place to the *vote* communication block to be come marked. So the coordinators output place (C) for the *prepare* communication block is merged with the coordinators input place for the *vote* communication block.

Steps (iv) and (v) are considered for each process and communication block in turn, until all blocks are composed in a bottom-up manner. This composition by place merging is considered in detail in section 5.2.5.

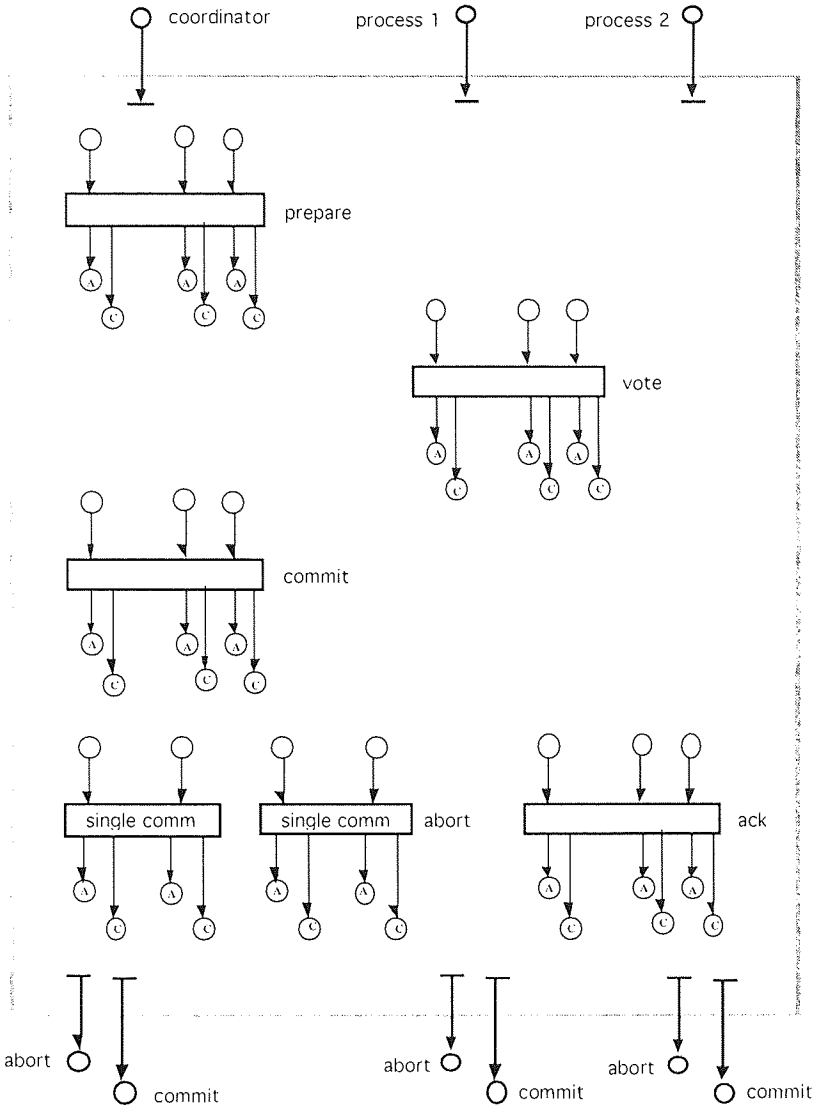


Fig.5.8. block structure for hybrid protocol design

In the approach presented below the Petri net template for each communication block is analysed in terms of its external behaviour. The procedure involves taking each communication block defined in the top-down design (Fig.5.8.) and using Petri net analysis of the subnets to confirm the *input* and *output* place sets, prior to the communication blocks interconnection.

5.2.4. Block structures for E2PC

The Petri net template of Fig.5.7. is the basis for the following communication blocks, apart from the abort block. Analysis of the Petri net of Fig.5.7. has shown it to be live, 1-bound and deadlock free, and to have a single firing path per process modelling communication success or failure.

5.2.4.1 Prepare block

The prepare block is based on the Petri net structure shown in Fig.5.9. This represents a 3-party communication with timeouts. Tokens in the input places are absorbed by the firing of the transitions { t_1, t_{10}, t_{12} }. The input places form the nets initial marking M_0 = { p_1, p_{10}, p_{14} }. These represent the coordinator, participant-1 and participant-2 input places. Each of these processes has an outcome of either communicate (c) or communicate abort (a) due to timeout.

The marking immediately reachable from M_0 are the internal places { p_2, p_3, p_{11}, p_{15} }. The single partition and no message loss criteria, discussed in section 4.2.3., are assumptions that hold for all communication blocks in the protocol. The assumption of a single link failure constrains the net behaviour to a single timeout transition pair firing, either { t_5 & t_{11} } or { t_4 & t_{13} } in Fig.5.9..

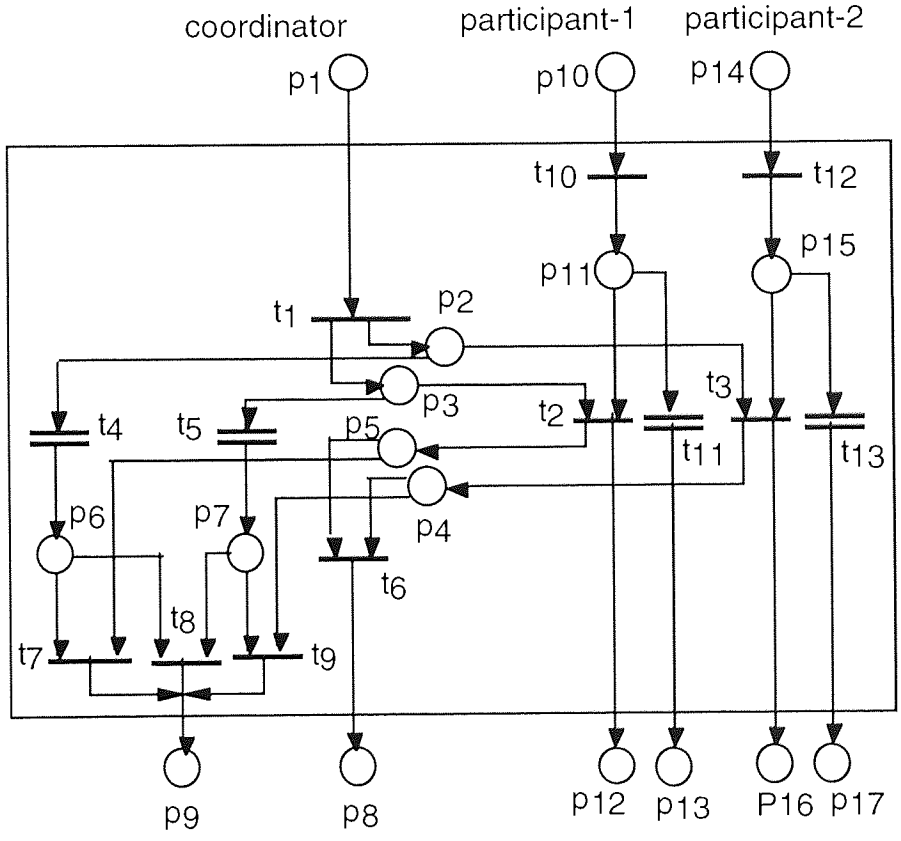


Fig.5.9. 3-party communication block with timeout

For the prepare communication block under the assumption of a the single link failure, transition t_8 is not required, as this represents the coordinator process timing out both participants communications.

Only a single pair of timeout transitions associated with a communication link can fire, and this only occurs after the communication transition associated with the other link fires

successfully. This behaviour is based on the assumption of a single communication link failure and the setting of the timeouts allowing the maximum time required for a successful communication. These failure modes form a restriction on the behaviour of the Petri net template, and thus partially define the input/output constraints of the communication block.

From Fig.5.9. consider a communication link failure between the coordinator and participant-2, the following timeout firing sequences occurs:
1) communication transition t_2 fires (coordinator - participant-1 link), and is followed by the independent firing of the timeout transition pair t_4 and t_{13}. The firing sequence is (t_2, t_4, t_{13}) or (t_2, t_{13}, t_4).

Where the communication link between the coordinator and participant-1 fails, the following timeout firing sequences occurs:
2) communication transition t_3 fires (coordinator - participant-2 link), and is followed by the independent firing of the timeout transition pair t_5 and t_{11}. The transition firing sequence is (t_3, t_5, t_{11}) or (t_3, t_{11}, t_5).

The definition of the communication blocks external behaviour, must forms an input/output constraint necessary for the correct interconnection of communication blocks. The constraint on the timeout firing sequence can be explicitly represented in the Petri net model of the communication block by using:
i) inhibitor arcs [Murata 89] - as shown in Fig.5.10.(a), or
ii) places that form an interlock - Fig.5.10.(b) shows the additional Petri net structure that needs to be used to implement an interlock. Where no timeouts occur, tokens in places p_X and p_Y would need to be consumed by the firing of transitions not shown in the diagram, this would preserve boundedness of the Petri net.

The above approaches can lead to a complex Petri net structure and a corresponding increase in the net's state space. An alternative approach is to represent the relevant constraints in an implicit manner by not modifying the net but only considering those sequences in the reachability graph that exhibit the desired behaviour. This is a similar approach to that used in [Ostroff 89] where only certain portions of a complete reachability tree are considered. This would involve only considering those portions of the reachability tree for this form of 3-party communication block (shown in Appendix.A.1.), that contain the firing sequences 1) or 2) above.

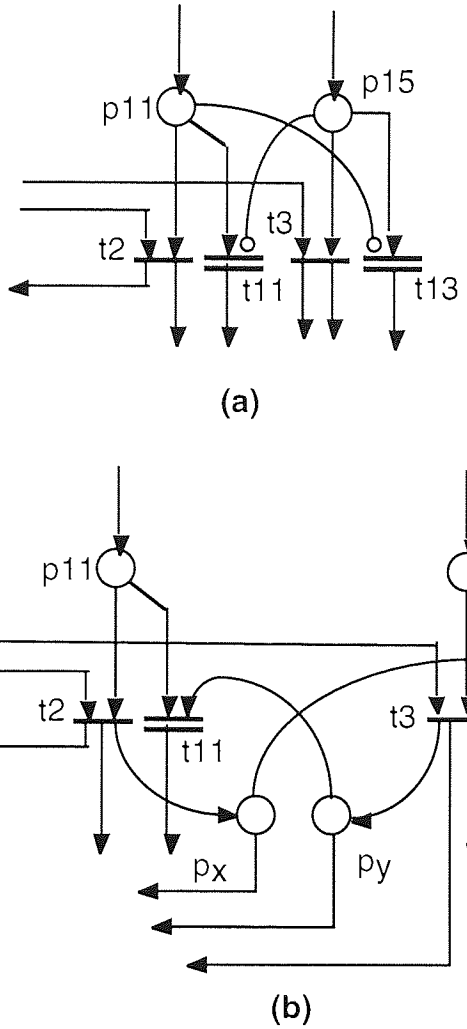


Fig.5.10. fragment from (Fig.5.9.), showing (a) inhibitor arc and (b) interlock structure.

In this thesis the second approach of using interlock places (Fig.5.10.(b)) to constrain the Petri net behaviour is taken. By constraining the behaviour of the net in the above manner the model incorporates an implicit notion of the failure mode of the system. A single communication link failure, the firing of t_{11} or t_{13}, can only occur after a successful communication has occurred, the firing of t_2 or t_3. Where both communications are successful, t_2 and t_3 both fire, the tokens in places p_x and p_y are removed by the firing of a subsequent transition. Appendix.A.2 presents the reachability graph and concurrency sets for the constrained behaviour of the Petri net structure Fig.5.9. The use of the interlock places, as illustrated in Fig.5.10.(b), are also detailed.

	Input set	Output set	Timeout
1	p1, p10, p14	p8, p12, p16	None
2	p1, p10, p14	p9, p12, p17	Coord - Part2
3	p1, p10, p14	p9, p13, p16	Coord - Part1

Table 5.2 external behaviour of the prepare communication block

This approach produces the external behaviour for the communication block defined in Table 5.2., (Coord is coordinator, Part1 is participant process 1 and Part2 is participant process 2). This forms the input/output constraint necessary for composition, achieved by merging the prepare blocks input and output places.

5.2.4.2 Vote block

The vote block structure is that shown in Fig.5.9., although the implicit constraints differ from those for the prepare block. The use of different constraints make the generic Petri net applicable to different communication blocks. For the vote block the reachability tree of Appendix.A.1. can be used for analysis. This communication block consists of a 3-party communication with timeouts, and with input place tokens being absorbed. Each process has an outcome of either communicate(c) or communicate abort (a) due to timeout. The single non-multiple partition criteria discussed in section 4.1. applies to this net.

This is the same structure as the prepare block, with the same failure assumptions of a single link failure. However, due to the structure of the responsive E2PC protocol the coordinator may need to timeout communications with both participants. However, the constraint that timeouts occur after any successful communication still applies. Transition t_8 in Fig.5.9. is required for this communication block, as this represents the coordinator process timing out both participants communications.

As mentioned in section 4.4.1 the E2PC is optimised in that after receiving the prepare message, a participant either decides *yes* and attempts to send the *yes* votes or decides *no* and unilaterally aborts the protocol. The participants *yes* vote is explicit while the no vote is implicit. In the latter case the coordinator will timeout communications with the participant, as if a link failure had occurred. The action of the coordinator is required to be the same for an implicit *no* vote as for a link failure.

Considering the input sets in Table 5.3. will illustrate the difference in the required external behaviour of the vote and prepare blocks.

i) cases 1. to 3. - this occurs where both participants decide to send the *yes* vote. The behaviour is the same as for the prepare block, the firing of a only a single timeout pair is allowed.

ii) cases 4. & 5. - this occurs where participant-1 decides to send the *yes* vote, and participant-2 decides to unilaterally abort. In these cases the coordinators timeouts associated with communications with participant-2 always fire. If a link failure occurs for participant-1, then both the coordinators timeouts will fire, this is case 5.

iii) cases 6. & 7. - are for participant-1 deciding to unilaterally abort.

iv) case 8. - this is a consequence of a timeout outcome of the prepare block. This occurs where participant-1 receives the prepare message and attempts to send a *yes* vote to the coordinator. If the coordinator to participant-2 link failed for the prepare block, the outcome would be that both these processes would unilaterally abort the protocol. As a consequence only the timeout outcome for participant-1 is possible.

v) case 9. - as case 8. , for a participant-1 link failure.

vi) case 10. - this occurs where both participants decide to unilaterally abort the protocol.

	Input set	Output set	Timeout
1	p1, p10, p14	p8, p12, p16	None
2		p9, p12, p17	Coord - Part2
3		p9, p13, p16	Coord - Part1
4	p1, p10	p9, p12	None
5		p9, p13	Coord - Part1
6	p1, p14	p9, p16	None
7		p9, p17	Coord - Part2
8	p10	p13	Part1
9	p14	p17	Part2
10	p1	p9	Coord

Table.5.3 external behaviour of the vote block, Petri net Fig.5.9.

Cases 4. to 10. shown in Table 5.3 illustrate the major difference between the external behaviour of the prepare and vote blocks. In these cases only one or two of the processes are part of the behaviour of the communication block, however, execution of the net with these initial markings has shown that the Petri net template is still live, 1-bound and most importantly deadlock free. In case 4. only input places p1 & p10 are tokenised, and execution of the Petri net template produces the correct output marking of p9 & p12.

5.2.4.3 Commit block

The commit block is shown in Fig.5.11. This Petri net models a 3-party communication with timeout on the communications, and with input place tokens being absorbed. Under a single link failure assumption, the firing of only a single timeout transition pair is allowed.

Although based on the same Petri net template, this communication block differs from the prepare and vote blocks in its timeout behaviour. There are two timeout outcomes for the coordinator, these explicitly identify the communication link that has failed. The firing of t7 to tokenise p9 indicates the coordinator timed-out communication with participant-2, while the firing of t8 to tokenise p18 indicates the coordinator timed-out communication with participant-1.

This identification is required in order to initiate the sending of an *abort* message to a participant that has received the *commit* message. This requirement was discussed in section 4.5. The firing of transition t3 indicates participant-2 receiving the *commit* message from the coordinator. If the communication link to participant-1 subsequently fails, the timeout transition pair (t5 & t10) will fire. In this case the coordinator will be required to send an *abort* message to participant-2. The tokenising of place p18 will initiate this action.

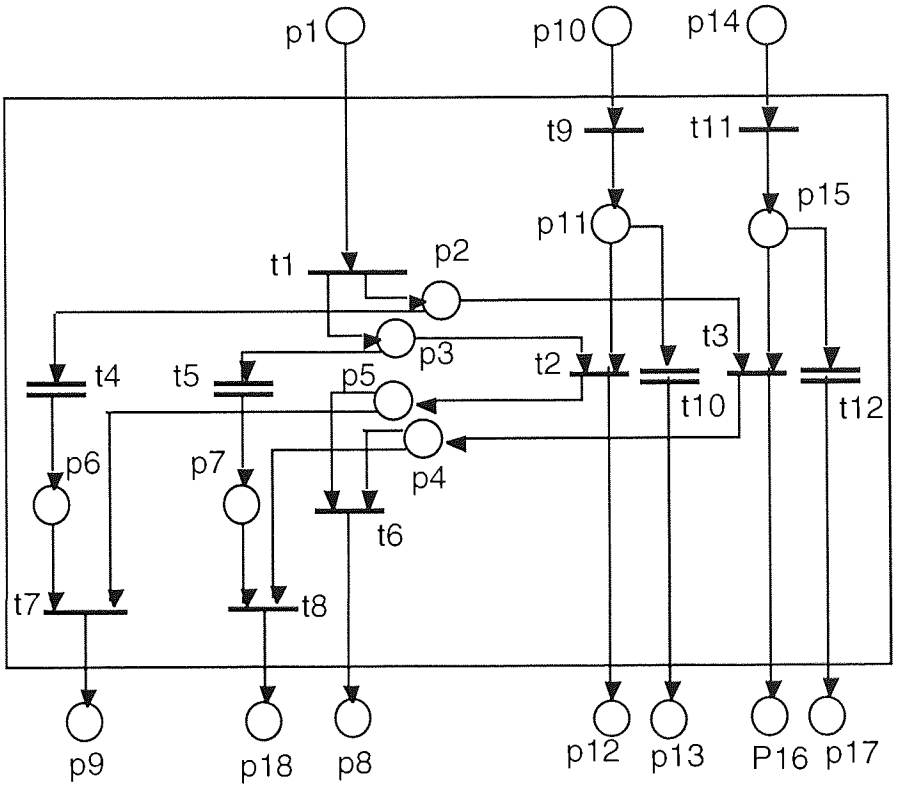


Fig.5.11. *commit* communication block

As can be seen from the net the output place p9 of the communication block indicates that (coordinator - process 1) link has failed, while place p18 indicates that the (coordinator - process 2) link has failed.

	Input set	Output set	Timeout
1	p1, p10, p14	p8, p12, p16	None
2		p9, p12, p17	Coord - Part2
3		p9, p13, p16	Coord - Part1
4	p10, p14	p13, p17	Part1 and Part2
5	p10	p13	Part1
6	p14	p17	Part2

Table 5.4 external behaviour of the *commit* communication block, Petri net Fig.5.11.

As discussed in section 5.2.4.2., after the prepare block, the participants reach a decision point. At this point the participants decide to proceed and send a *yes* vote or unilaterally abort the protocol. The decision point for the coordinator is after the vote block. In Table 5.4 case 4. is where the coordinator votes no and unilaterally aborts the protocol, only the participant input places are tokenised. As can be seen from the net Fig.5.11. the outcome for both participants in this case is timeout.

This is part of the optimisation of the E2PC protocol where participants are left to timeout to the abort decision, instead of the coordinator sending an explicit *abort* message, the action to be taken by the participants in either case being the same.

It is at the point before the commit communication that the coordinator takes the decision to continue or unilaterally abort the protocol. The latter case is shown in the bottom three input and output sets in Table 5.4. Cases 5 and 6 in Table 5.4 show the situation in which a single participant has a successful outcome to the vote block, and as a result is the only process that initiates the commit block.

5.2.4.4 Abort block

The two abort blocks represent a 2-party timeout guarded communication. The initiation of this abort message block was discussed in section 5.2.4.2. Referring to the structure of the E2PC protocol, Fig.4.12. and the hybrid protocol template of Fig.5.8., these are only used when there has been a timeout outcome in the commit block.

A coordinator timeout outcome of the commit block indicates a link failure has occurred, and also explicitly identifies the link. Under the assumption of a single link failure the other communication link can be assumed to be operational. If the communication link (coordinator - process 1) fails, then the communication link (coordinator - process 2) is available and operating correctly, and *vice versa*. The resulting constraint on the timeout behaviour is that they never fire, as the assumption can be made that if necessary, this communication is possible within a known deadline. The Petri net template is shown in Fig.5.12.

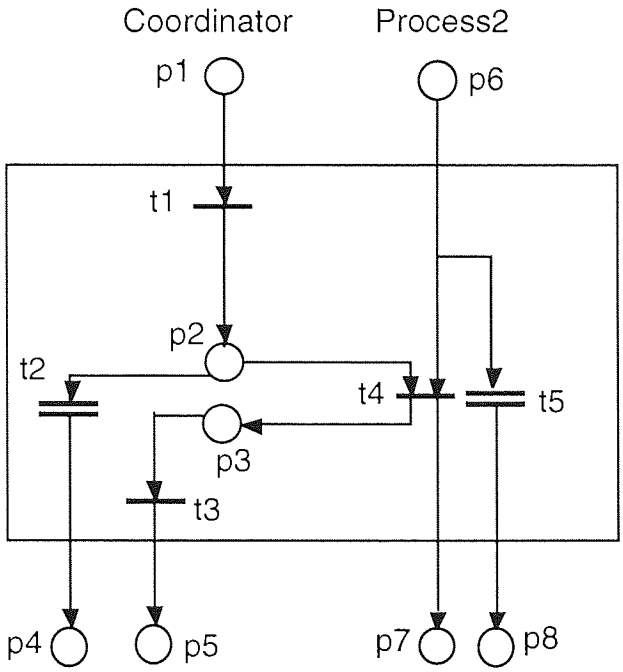


Fig.5.12. abort communication block

From the structure of the E2PC protocol, Fig.4.12., the following structure can be observed. The participants input place to the abort block is in conflict with the input places to the acknowledgement (ack) block. Therefore, tokens in these input places are not absorbed into the block, through the use of what is termed *atomic feedthrough* of participant input place tokens. In atomic feedthrough the communication block is an atomic action, where the firing of a transition internal to the block removes a token from an input place and deposits it directly in an output place.

This block is required for the aborting of some participant that has received the commit message from the coordinator, and the coordinator subsequently being unable to send the commit message to the other participant due to a link failure.

	Input set	Output set	Timeout
1	p1, p6	p5, p7	None
2			Coord - Part1/2

Table 5.5. external behaviour of the *abort* communication block, Petri net Fig.5.12.

Due to the assumed failure modes the outcome of this block is restricted to both parties communicating. This is possible as the timeout action would be constrained, due to the single link failure criteria. This assumption is also important for the timeout action of participants in the acknowledgement block.

5.2.4.5 Acknowledgement block

The ack block is shown in Fig.5.13. This Petri net structure models a 3-party communication with timeout on the communications. The coordinator's input place tokens are absorbed, while atomic feedthrough of the participants input place tokens is used. This requirement is due to the participants input places being in conflict with the abort blocks input places as mentioned in section 5.2.4.4. The timeout period and action for these shared input places is defined by the timeouts in the ack block. As shown in chapter 4, these input places are *commitable* protocol states, and so a communication failure results in a timeout to a final commit state.

As mentioned in chapter 4, a further optimisation of the E2PC protocol would be to have no explicit *acknowledgement* message. In this case participants would timeout to a protocol commit state unless an *abort* message were received. As such, the use of the *acknowledgement* message does not increase the robustness of the protocol.

The use of the *ack* block allows a possible earlier time to commit than a timeout action alone. The timeout action will always require a set maximum time to complete, while the sending of the *acknowledgement* message could complete earlier than the protocol deadline.

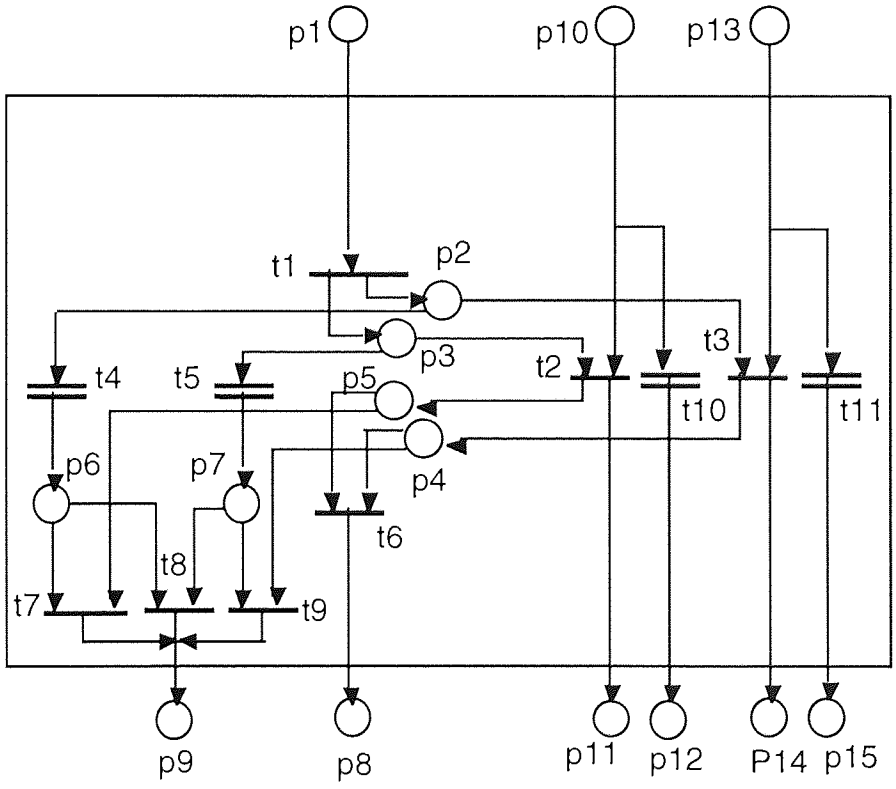


Fig.5.13 acknowledgement communication block

	Input set	Output set	Timeout
1	p1, p10, p13	p8, p11, p14	None
2		p9, p11, p15	Coord - Part2
3		p9, p12, p14	Coord - Part1

Table 5.6. external behaviour of the *acknowledgement* communication block

The outcome of the ack block for each process is communicate (c) or communicate abort (a). Under the single partition criteria the assumption of a single link failure can be made, and the firing of a single timeout transition pair is allowed.

The input places to the ack block are *commitable* states, and so the actions of the processes are the same for the communicate or timeout outcome of this block. Thus, for each process, the two output places of this block are merged when the blocks are interconnected. The following places would be merged (p8 - p9), (p11 - p12) and (p14 - p15).

It should be noted that the validity of all communication block's output sets were confirmed by executing the Petri net templates with each input set as an initial marking (M_0). This also verified that the templates were live, safe and deadlock free. Examples of the reachability trees generated and analysis results can be found in Appendix.A.1. and A.2.

5.2.5. Block structured responsive E2PC

In order to use a hybrid design approach for the commit protocol, a number of steps were proposed. The top-down phase of the design was completed in section 5.2.3., giving the block structured Petri net of Fig.5.8. The bottom-up phase of the hybrid design involves the merging of the communication blocks.

In section 5.2.3. a five step procedure for this composition was presented. The first three steps involved (i) determining the Petri net template for each communication block, (ii) defining the required external behaviour of the block and (iii) analysing the Petri net template using the initial markings defined in step (ii). These three steps were completed for each of the communication blocks in section 5.2.4.

The final two steps of the composition procedure involve the merging of the communication blocks input and output places. In step (iv) the appropriate action for each outcome of the block is determined, and in step (v) the places representing these actions are merged. In section 5.2.3. this was illustrated by examining the output places of the prepare block, for the coordinator process.

In this section the commit protocol is constructed through the composition of the communication blocks defined in section 5.2.4. The interconnection of the blocks should preserve properties of the Petri net templates, such as boundedness, liveness and freedom from deadlock. These properties are preserved if the external behaviour of the blocks is maintained.

The interconnection of the communication blocks involves the merging of the blocks output places with combinations of places that are either:
i) input places to subsequent communication blocks,
ii) abort places, these represent final internal states for each process, these initiate the protocols abort action, { p23, p40, p52 } in Fig.4.12.,
iii) decision places, this is the point where each process takes the decision to proceed or to unilaterally abort the protocol, { p24, p36, p48 } in Fig.4.12.,
iv) commit places, these represent final internal states for each process, these initiate the protocols commit action, { p34, p46, p58 } in Fig.4.12.

The block structured form of the E2PC protocol is shown below in Fig.5.14. The merging of places to compose the communication blocks is illustrated by the input and output places of the prepare and vote block. The prepare blocks output places for the coordinator process were considered in section 5.2.3., so Participant-1's output places are considered here. The numbering of places and transitions in Fig.5.14. is taken from the version of the protocol given in Appendix.B.2.

The output places of the prepare block represent communication complete (C) and communication abort (A). For participant-1 the appropriate action for communication abort (A) is the final abort state, and so this output place is merged with the abort place (p32). The appropriate action for communication complete (C) is the decision state, and so Participant-1 output place (C) is merged with the decision place (p31).

At the decision state Participant-1 proceeds or unilaterally aborts the protocol, as modelled by the firing of the transitions marked *Yes* (t6) and *No* (t7). The appropriate action for the No decision is the abort state, and so the output place of this transition (t7) is merged with the abort place (p32). The appropriate action for the Yes decision is to send a *yes* vote to the coordinator. So the output place of the *Yes* transition (t6) is merged with the Participant-1 input place of the vote block (p33).

This procedure is repeated for the coordinator and participant-2, until all output places of the prepare block, and all input places of the vote block are merged. The merging of the input and output places for all the communication blocks is realised in the same way. This

completes the bottom-up composition to produce the hybrid commit protocol design, shown in Fig.5.14.

The merging of places can be checked by considering the input and output sets for each communication block. The possible markings for the output places of the prepare block are defined in Table 5.2. Consider case 1. of this output set, this gives the prepare block output place marking { p_{21}, p_{31}, p_{41} }. By firing the *Yes* transitions (t_6, t_{11}) the following marking is produced { p_{21}, p_{33}, p_{43} }. This marking gives cases 1, 2, & 3 of the vote blocks input set (Table 5.3). This sequence represents the coordinator successfully sending the *prepare* message to both participants, and the participants deciding to proceed with the protocol and return a *yes* vote.

By considering each case of the prepare blocks output set, along with the firing of the *Yes* (t_6) and *No* (t_7) transitions, a set of markings for the vote blocks input places is produced. If this set of markings is equivalent to the vote blocks input set, then the composition of the prepare and vote block is valid. This process is repeated for each communication block to check the composition of the commit protocol.

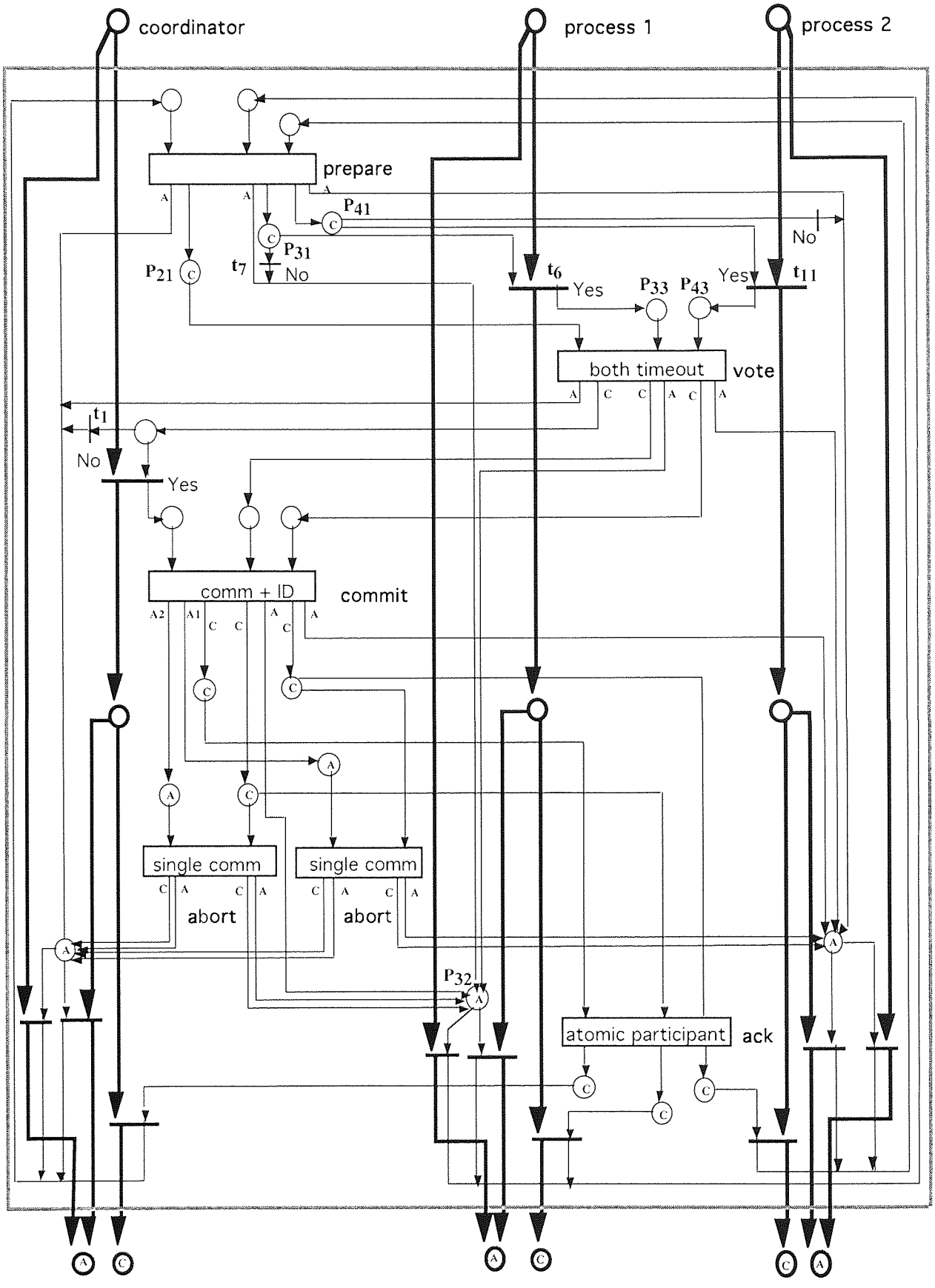


Fig.5.14. Hybrid design commit protocol, using communication blocks.

This commit protocol includes a reduced model of the environment, in the same manner as the commit protocol designs in chapter 4. The places, arcs and transitions of the environment

model are shown in bold in Fig.5.14. This minimal environment model is used in both the design and analysis of the commit protocol.

For the protocol shown in Fig.5.14. each process has an external input place and two external output places that represent the minimal environment model. The output places represent the final outcome of the protocol. In this version of the E2PC protocol, there is locking of input place tokens of the environment model. These could represent a process locking a local resource until a commit protocol completes. The output places for the environment model would represent a process committing to or aborting an action.

The block structured form of the E2PC protocol shown in Fig.5.14., is termed a commit block. It is intended to use the commit block as a Petri net template of the commit protocol, this is addressed in section 5.3.

The form of the commit protocol in Fig.5.14. is termed commit block-1 (CB1), and has locking of all input place tokens (for the environment model). This locking action is modelled by the firing of the Yes transitions (t_1, t_6, t_{11}). This occurs after each process has made the decision to proceed within the protocol. Atomic feedthrough of (the environment models) input place tokens is used if an abort action occurs before the decision point. This is modelled by the firing of transitions (t_1, t_6, t_{11}).

A version of the protocol, commit block-2 (CB2), with atomic feedthrough of (the coordinator process environment model) input place tokens is shown in Appendix.B.3. A version of the protocol, commit block-3 (CB3), with atomic feedthrough of all processes input place tokens is shown in Appendix.B.4. The application of the different versions is discussed in chapter 6.

5.2.6. Verification of communication blocks behaviour

The composition of the communication blocks, presented in the last section, relied upon the validity of the input and output sets for each block. In this section it will be verified that the input and output sets are a correct representation of the external behaviour, for the underlying Petri net template, of each communication block.

The interconnection of the communication blocks in order to form a block structured commit protocol is shown in Fig.5.14. The verification of properties for the individual communication blocks will show that the resulting block structured Petri net exhibits certain desired behaviours.

The analysis based on temporal Petri nets (described in section 3.5.) will be used to formally verify that from the initial marking M_0 a particular set of outputs from the block will be generated. The proofs of each set of outputs will form the properties of the underlying Petri net model.

The Temporal Petri net analysis of [Suzuki & Lu 89] is used to prove liveness properties for the communication blocks. This analysis will be based on using the notation of Petri nets and Temporal logic to make assertions about firing sequences for particular net markings. The input sets of the communication blocks will form the initial marking M_0 for the underlying Petri nets.

This verification procedure is based on using Propositions that make assertions about the firing of transitions in a Petri net. Specifically, for the analysis shown in this chapter, Propositions 1 and 2 will be used and are shown below. These propositions are applied after first checking the validity of their premises against the reachability graphs of the communication blocks Petri nets. Examples of these reachability graphs are given in Appendix A.1 and A.2.

Proposition 1 and Proposition 2 are used to make assertions about the firing of transitions in the Petri net. Proposition 1 is applicable to the firing of transitions where the input places are not in conflict, and Proposition 2 to the firing of transitions where input places are in conflict (in which the firing of a transition disables itself and another enabled transition). The validity of these transitions are checked against the reachability graph of the net, and then used to prove certain liveness properties for an initial marking M_0. Propositions 1 and 2 can then be used along with propositions unique to the Petri net under analysis, and are defined as follows:

Proposition 1:
For a temporal Petri net $TN_1 = (C,f)$ whose initial marking is M_0.
(i) For a marking M reachable from M_0, if transition t is enabled at M, then t remains enabled until t fires:

$$<M_0,\alpha> \models \Box[\, t(ok) \Rightarrow t(ok)\ \mathcal{U} t\,] \qquad (5.1)$$

(ii) f implies

$$<M_0,\alpha> \models \Box[\, t(ok) \Rightarrow \Diamond t(\neg ok)\,] \qquad (5.2)$$

where f represents the condition that for marking M_0 and firing sequence α, that whenever t becomes enabled then it will eventually become disabled, and C represents the structure of the Petri net. If (i) and (ii) hold, then for any firing sequence α from marking M_0, using (i) and (ii) it can be deduced that:

(iii) $$<M_0,\alpha> \models \Box[\, t(ok) \Rightarrow \Diamond t\,] \qquad (5.3)$$

Proposition 2

For a temporal Petri net TN1 = (C,f) whose initial marking is M_0.

(i) for any marking M reachable from M_0, if t_1 (input function $I(t_1)$) and t_2 (input function $I(t_2)$) are fireable at M and are in conflict, which is defined as:

$$I(t_1) \cap I(t_2) \neq \varnothing$$

where $\varnothing$ is the empty set. Then either t_1 remains enabled until either t_1 itself fires or t_2 fires, or t_2 remains enabled until either t_2 itself fires or t_1 fires. This statement is formalised as:

$$<M_0,\alpha> \models \Box[\, t_1(ok) \Rightarrow t_1(ok)\ \mathcal{U}(t_1 \vee t_2)\,] \vee \Box[\, t_2(ok) \Rightarrow t_2(ok)\ \mathcal{U}(t_1 \vee t_2)\,] \quad (5.4)$$

(Only one transition is allowed to fire at any instant using the interleaving model of concurrency, the choice of which fires is made non-deterministically or according to a fairness requirement if specified),

(ii) f implies

$$<M_0,\alpha> \models \Box[\, t_1(ok) \Rightarrow \Diamond t_1(\neg ok)\,] \wedge \Box[\, t_2(ok) \Rightarrow \Diamond t_2(\neg ok)\,] \quad (5.5)$$

If (i) and (ii) hold, then for any firing sequence α from marking Mo, using (i) and (ii) it can be proved that:

(iii) $$<M_0,\alpha> \models \Box[\, t_1(ok) \wedge t_2(ok) \Rightarrow \Diamond(t_1 \vee t_2)\,] \quad (5.6)$$

Property 5.7, for the *Prepare* communication block Fig.5.9. whose input and output sets are shown in Table 5.2., the following property can be proved:

$$<M_0,\alpha> \models \Box\,[\, M_0 \Rightarrow \Diamond(M_1 \vee M_2 \vee M_3)\,] \quad (5.7)$$

where $M_0 = \{ p_1, p_{10}, p_{14} \}$, $M_1 = \{ p_8, p_{12}, p_{16} \}$, $M_2 = \{ p_9, p_{12}, p_{17} \}$ and $M_3 = \{ p_9, p_{13}, p_{16} \}$.

The proof for this property (5.7) is presented in Appendix.C.1.

The proof for property (5.7) verified that for the prepare communication block and Petri net, for all possible firing sequences the set of input places eventually produce the set of output place markings .

The input set of the *vote* block differs from the *prepare* block, although the same underlying Petri net Template is used. The input sets for the vote block contains markings for only one or two input places (these are cases 4 to 10 in Table 5.3). This represents cases such as only one participant attempting to send a *yes* vote to the coordinator.

Due to the structure of the protocol the coordinator may timeout communications with both participants at this stage of the protocol, although a single link failure is still assumed. All successful communications still occur before any timeout action.

The reachability graph for the Petri net with this implicit behaviour is given in Appendix.A.1. The *vote* block is verified by considering each of the input sets in turn.

Property 5.8 The first three cases in Table 5.3 are the same as for the *prepare* block and the properties are proved in the same manner. This property can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_1 \vee M_2 \vee M_3)] \qquad (5.8)$$

The proof for this follows from that presented in Appendix.C.1.

The proof for Property 5.9 is presented so as to illustrate the Petri net behaviour where only two processes are present in the input set. This is the fourth and fifth case in Table 5.3.

Property 5.9 The following will illustrate a proof of a property for the vote communication block using Temporal Petri net analysis. The Petri net for this communication block is shown in Fig.5.9. and its set of input and output places are shown in Table 5.3

This property can be interpreted as the following behaviour for the protocol: Prior to the vote block the coordinator has successfully sent the *prepare* message to both participants. At this point each participant takes the decision to proceed or unilaterally abort the protocol. Participant-2 unilaterally aborts, and Participant-1 proceeds. In this case only Participant-1 attempts to send a *yes* vote to the waiting coordinator (input sets 4. and 5. in Table 5.3). The outcome is that either the *yes* vote is successfully sent, or there is a link failure and the coordinator and Participant-1 both timeout the communication (output sets 4. and 5. in Table 5.3).

Let TN_1 be a temporal Petri net for the Petri net PN shown in Fig.5.9., and its initial marking be $M_0 = \{ p_1, p_{10} \}$. Whenever M_0 becomes reachable in TN_1, then eventually either one of the following markings is reachable:

(a) $M_1 = \{ p_9, p_{12} \}$ or

(b) $M_2 = \{ p_9, p_{13} \}$.

Hence property 5.9 can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_1 \vee M_2)] \qquad (5.9)$$

Proof

1. At M_0, transitions t_1 and t_{10} are fireable, this can be formalised as:

$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(t_1(ok) \wedge t_{10}(ok))]$

From an observation of the Petri net of Fig.5.9. and its associated reachability graph (shown in Appendix.A.1), it can be seen that t1 and t_{10} can fire independently.

2. Using Proposition 1 for the firing of t_1 and t_{10} yields:

$<M_0,\alpha> \models \Box [t_1(ok) \Rightarrow \Diamond t_1]$

3. $<M_0,\alpha> \models \Box [t_{10}(ok) \Rightarrow \Diamond t_{10}]$

The firing t_1 and t_{10}, will yield a new sub-marking, these are shown by:

4. $<M_0,\alpha> \models \Box [t_1 \Rightarrow (p_2 \wedge p_3)]$

5. $<M_0,\alpha> \models \Box [t_{10} \Rightarrow p_{11}]$

6. Combining (2., 4.) and (3., 5.) with 1. produces:

$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_2 \wedge p_3 \wedge p_{11})]$

Let $M_3 = \{ p_2, p_3 , p_{11} \}$, hence the above can be re-written as:

$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond M_3]$

7. At the marking M_3 both t_2 and t_4 are fireable, however only one of these transitions can fire. This can be formalised as:

$<M_0,\alpha> \models \Box [M_3 \Rightarrow \Diamond(t_2 \wedge t_4)]$

8. The following will consider the consequences of firing each transition t_2 and t_4. Specifically:

(a) the firing of t_2 will be succeeded by the transition sequence:

$< t_4, t_8 >$

(b) the firing of t_4 will be succeeded by the transition sequence:

$< t_5, t_{11}, t_8 >$

9. Considering case 8(a):

(i) the firing of t_2 produces the submarking:

$<M_0,\alpha> \models \Box [t_2 \Rightarrow (p_5 \wedge p_{12} \wedge p_2)]$

(ii) From the structure of the net TN_1 it can be seen that

$<M_0,\alpha> \models \Box [p_2 \Rightarrow t_4(ok)]$

(iii) Using proposition 1 on the firing of t_4 produces:

$<M_0,\alpha> \models \Box [t_4(ok) \Rightarrow \Diamond t_4]$

(iv) From the structure of the net TN_1, it can be seen that the firing of t_4 tokenises p_6, i.e.

$<M_0,\alpha> \models \Box [(t_4) \Rightarrow p_6]$

(v) Combining 9(i) - (iv) produces:

$<M_0,\alpha> \models \Box [t_2 \Rightarrow \Diamond(p_5 \wedge p_6 \wedge p_{12})]$

(vi) From the structure of the net TN_1, it can be seen that the sub-marking of $\{p5, p6\}$ enables t7, i.e.

$<M_0,\alpha> \models \Box [(p5 \wedge p6) \Rightarrow t7(ok)]$

(vii) Using proposition 1 on the firing of t7 produces:

$<M_0,\alpha> \models \Box [t7(ok) \Rightarrow \Diamond t7]$

(viii) From the structure of the net TN_1, it can be seen that the firing of t7 tokenises p9, i.e.

$<M_0,\alpha> \models \Box [t7 \Rightarrow p9]$

(ix) Combining 9(vi) -(viii) produces:

$<M_0,\alpha> \models \Box [(p5 \wedge p6) \Rightarrow \Diamond p9]$

(x) Combining 9(ix) -(v) produces:

$<M_0,\alpha> \models \Box [t2 \Rightarrow \Diamond(p9 \wedge p12)]$

Since $M_1 = \{ p9, p12 \}$ then the above can be re-written as:

$<M_0,\alpha> \models \Box [t2 \Rightarrow \Diamond M_1]$

---

10. Considering case 8(b)

(i) the firing of t4 produces the sub-marking:

$<M_0,\alpha> \models \Box [t4 \Rightarrow p6]$

(ii) the firing of t5 produces the sub-marking:

$<M_0,\alpha> \models \Box [t5 \Rightarrow p7]$

(iii) the firing of t_{11} produces the sub-marking:

$<M_0,\alpha> \models \Box [t_{11} \Rightarrow p13]$

(iv) Since t4, t5 and t_{11} can fire, combining 10.(i) - (iii) produces:

$<M_0,\alpha> \models \Box [(t4 \wedge t5 \wedge t_{11}) \Rightarrow (p6 \wedge p7 \wedge p13)]$

(v) From the structure of the net TN_1, it can be seen that the sub-marking of $\{p6, p7\}$ enables t8, i.e.

$<M_0,\alpha> \models \Box [(p6 \wedge p7) \Rightarrow t8(ok)]$

(vi) Using proposition 1 on the firing of t8 produces:

$<M_0,\alpha> \models \Box [t8(ok) \Rightarrow \Diamond t8]$

(vii) From the structure of the net TN_1, it can be seen that the firing of t8 tokenises p9, i.e.

$<M_0,\alpha> \models \Box [t8 \Rightarrow \Diamond p9]$

(viii) Combining 10(iv) -(vii) produces:

$<M_0,\alpha> \models \Box [(t4 \wedge t5 \wedge t_{11}) \Rightarrow \Diamond(p9 \wedge p13)]$

(ix) Simplifying 10(viii) produces:

$<M_0,\alpha> \models \Box [t4 \Rightarrow \Diamond(p9 \wedge p13)]$

Since $M_2 = \{ p9, p13 \}$, the above case can be re-written as:

$<M_0,\alpha> \models \Box [t4 \Rightarrow \Diamond M_2]$

11. Combining 10(ix) and 9(x) with 7, produces:

$<M_0,\alpha> \models \Box [M_3 \Rightarrow \Diamond(M_2 \vee M_1)]$

12. Combining 11 and 6, produces:
$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_1 \vee M_2)]$ ■

Property 5.10, for the *vote* communication block shown in Fig.5.9, whose input and output sets are cases 6 and 7 in Table 5.3.

Let TN_1 be a temporal Petri net for the *vote* communication block Fig.5.9, with an initial marking $M_0 = \{ p_1, p_{14} \}$. The property of TN_1 that will be proved is that:
Whenever M_0 of TN_1 becomes tokenised, then eventually either one of the following markings is reachable:

(a) $M_4 = \{ p_9, p_{16} \}$
(b) $M_5 = \{ p_9, p_{17} \}$

This property can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_4 \vee M_5)] \quad (5.10)$$

The proof of property (5.10) follows from the proof of property (5.9)

Property 5.11, for the *vote* communication block shown in Fig.5.9, whose input and output sets are cases 6 and 7 in Table 5.3.

Let TN_1 be a temporal Petri net for the *vote* communication block Fig.5.9, with an initial marking $M_0 = \{ p_{10} \}$. The property of TN_1 that will be proved is that:
Whenever M_0 of TN_1 becomes tokenised, then eventually the following marking $M_6 = \{ p_{13} \}$ is reachable. This property can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_6)] \quad (5.11)$$

Proof

1. At M_0, transition t_{10} is fireable, this can be formalised as:
$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond t_{10}(ok)]$

2. The firing of t_{10} using Proposition 1 produces:
$<M_0,\alpha> \models \Box [t_{10}(ok) \Rightarrow \Diamond t_{10}]$

3. From an observation of the Petri net of Fig.5.9 and its associated reachability graph (as shown in Appendix.A.1), it can be seen that the firing of t_{10} tokenises p_{11}, i.e.

$$<M_0,\alpha> \models \Box [t_{10} \Rightarrow p_{11}]$$

4. Combining 1. - 3. yields:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_{11})]$$

5. The tokenisation of p_{11} enables t_{11}, i.e.

$$<M_0,\alpha> \models \Box [p_{11} \Rightarrow t_{11}(ok)]$$

6. The firing of t_{11} using proposition 1 gives:

$$<M_0,\alpha> \models \Box [t_{11}(ok) \Rightarrow t_{11}]$$

7. The firing of t_{11} tokenises p_{13}, i.e.

$$<M_0,\alpha> \models \Box [t_{11} \Rightarrow p_{13}]$$

8. Combining 7 -1 produces:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_{13})]$$

Since $M_6 = \{ p_{13} \}$, the above can be re-written as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond M_6]$$ ■

Property 5.12 and 5.13 are for the input set cases 9 and 10 in Table 5.3.
These properties can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond M_7] \qquad (5.12)$$

From an initial marking $M_0 = \{ p_{14} \}$, where $M_7 = \{ p_{17} \}$.

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond M_8] \qquad (5.13)$$

From an initial marking $M_0 = \{ p_1 \}$, where $M_7 = \{ p_9 \}$. The proof for property (5.12) and (5.13) follows from that presented for (5.11).

The verification of the input and output sets for the *commit*, *abort* and *ack* communication blocks have been performed in the same manner to the above. This verification of the communication blocks, using Temporal Petri net analysis, gives confidence in the hybrid design of the E2PC protocol synthesised using these communication blocks.

5.3 Commit Protocol Block

The intended application of the commit protocols developed in the section 5.2. are in the design of distributed controllers, for the coordination of real-time control operations. The

Petri net models of the protocols will form reusable templates, that can be applied in Petri net models of the distributed controllers. The reusable templates of the commit protocols are termed *commit* blocks.

The modelling of large and complex systems using Petri nets can result in problems in the readability of the design, and in the analysis of the model. In general as the Petri net model increases in size and complexity, the associated state space increases exponentially [Dwyer *et al* 95]. Methods of analysis based on generating the reachability graph of a model can become impractical to use due to the processing time and effort required, but more importantly because it may be impossible to generate the complete state space.

It is anticipated that the distributed controller models, incorporating the atomic commit protocols, will result in large and complex Petri nets. This will result in problems when using reachability based analysis. There are various ways of approaching this problem, such as the use of stubborn sets [Valmari 91], the projection of the behaviours of composed sub-nets onto *interesting* and *environment* components [Valmari 93, 94], these components are similar to net reductions [Murata 89], and the composition of sub-net reachability graphs using sub-net interface specifications [Bucci & Vicario 95].

In section 5.2.6. the state space of each communication block's Petri net template was separately enumerated and assessed under the intended environment for the module. This external behaviour (environment) was specified by the set of input place markings for each block. This approach is similar to that used by Bucci & Vicario [95], where sub-nets are composed using reading and writing *ports,* resembling communication channels [Shaw 92]. The input and output behaviour for these *ports* represents the environment for the sub-nets.

In the following chapter, it is intended to use a modular approach to the design and analysis of the Petri net models. The modular approach is intended to overcome the problem of state space explosion [Scholefield 90], while still being based on the use of a reachability graph.

The modular approach to the design of the commit protocols, used in section 5.2, will be employed in the distributed controller design in chapter 6. This approach allows the use of specified and verified re-usable templates. In the development of the commit protocols, the templates formed the underlying Petri nets of the communication blocks. The Petri nets of the commit protocols are to be used in the same manner as the templates for the commit blocks.

A hybrid approach will be taken to the design of the distributed controllers, as described in section 3.7. This will involve the top-down refinement of models for separate local

controllers. Followed by the bottom-up composition of these modules using commit blocks. This will be achieved by merging the input and output places of the commit blocks with places in the local controllers, to form a distributed controller model. It is intended to reuse the analysis of the commit block's Petri net template in the analysis of the distributed controller.

In section 5.2, the hybrid design of the commit protocols was presented, using the composition of the communication blocks. This composition was based on the definition of the blocks external behaviour, and required the merging of the input and output places for the communication blocks.

The constraints on the external behaviour of the communication blocks were defined in the input and output sets. In section 5.2.4. each communication block was analysed separately from the total system, under the assumption of these input and output sets. This analysis proved salient properties of the Petri net template, such as liveness, boundedness and freedom from deadlock.

In section 5.2.6 the formal verification of the external behaviour of each communication block was presented. This confirmed the validity of the composition of the blocks. It is intended for the modular design of the communication blocks presented in section 5.2, to be extended to aid the modular analysis of the commit protocol. This would allow the reuse of the analysis, and the formal verification, of the communication blocks for the analysis of the total system.

In order to allow this modular analysis, a formal notion of the block structures is developed in sections 5.3.1 and 5.3.2., that is based upon the well formed blocks of Valette [79]. A projection of the external behaviour of the communication blocks is used to generate the reachability graph for the commit protocols composed from these blocks, such as Fig.5.14. This allows the use of reachability based Petri net analysis for the commit protocols, which is presented in section 5.3.3.

The verification of the communication blocks has been shown for certain sets of input place markings. The reachability graph for the composed commit protocol (section 5.3.3.), can be used to confirm these sets of input place markings. Where these input sets for the communication blocks hold, the verified properties for each block also hold. The combination of the reachability graph for the commit protocol with the verified properties for each block, can be used to deduce properties for the composed design. This modular verification is presented in section 5.3.4.

5.3.1 Well formed blocks

The Petri net block structures used in section 5.2. are based on Valette's [79] well formed blocks (WFB), the definition of these will be considered in order to define the notion of well formed multi-blocks (WFMB). These WFMB form the basis for the application of the communication and commit blocks.

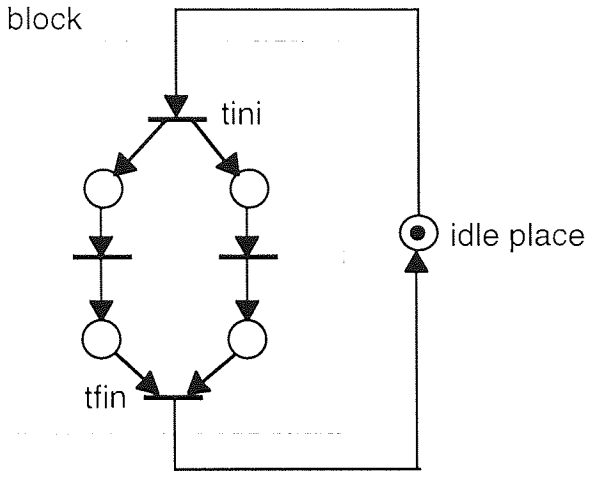


Fig.5.15. well formed block with idle place

Valette [79] proposed a top-down approach that allowed a step-wise refinement to replace transitions in these nets with block structures, whilst preserving the properties of safeness, boundedness and liveness. The Petri net, Fig.5.15., within the (block) outline has one initial transition (t_{ini}) and one final transition (t_{fin}) and is termed a block.

A refined Petri net PN' = (P', T', I', O', M_0') , termed the associated Petri net, is obtained from the block Petri net PN = (P, T, I, O, M_0) , by the addition of the (idle) place p_0 , such that:

i) the only output transition of p_0 is t_{ini},

ii) the only input transition of p_0 is t_{fin},

iii) $M_0' = M_0 + \{p_0\}$.

The following definitions can then be given:

i) the block PN is bounded iff the associated Petri net PN' is bounded,

ii) the block PN is safe iff the associated Petri net PN' is safe,

iii) the block PN is live iff the associated Petri net PN' is live.

A block Petri net PN is a well formed block [Valette 79] iff the associated Petri net PN' is such that:

i) PN' is live,

ii) $M_{0'}$ is the only marking in the set of markings reachable from $M_{0'}$ such that the idle place is not empty,
iii) the only transition enabled by $M_{0'}$ is the initial transition t_{ini}.

A WFB is such that once activated (the initial transition is fired) there exists a firing sequence that contains the final transition, and that after the firing of the final transition the marking of a WFB is the initial marking[Valette 79].

5.3.2 Well formed multi-blocks

The WFB presented in [Valette 79] are not directly applicable to the definition of the blocks used in section 5.2. These blocks are intended to be used for the substitution of several independent transitions, that represent the interaction of concurrent processes. Thus simple transition substitution of WFB, is not possible in these cases. The WFMB used differ from Valette's [79] WFB, in that:
i) the underlying Petri net for the block represents the interaction of several processes,
ii) there is an initial transition (t_{ini}) for *each process*,
iii) there is a final transition (t_{fin}) for *each process*,
However, the properties such as liveness, boundedness and freedom from deadlock, are to be preserved during the refinement procedure in the same manner as for WFB.

In order to develop the notion of the WFMB, the idle place is replaced by an input and output place (p_{in}, p_{out}) for the block , along with what is termed an environment transition (t_{env}), for the WFB shown in Fig.5.16.(a). This transition and places can be reduced to a single place using Petri net reductions [Murata 89], and can thus be considered equivalent to the WFB idle place.

The environment transition along with the input and output places, are intended to be a minimal representation of the environment for the WFMB. There is still an initial transition (t_{ini}) and final transition (t_{fin}) in this type of block for each process.

To form the WFMB another process is added to the single process of a WFB block to form a multi-process block, where each process has an initial and final transition and an input and output place. The input and output places for each process are connected by a single environment transition. This block is shown in Fig.5.16.(b). This form of multi-block is termed *simple*, as there exists a single-input and a single-output (SISO) place for each process. The *abort* communication block in section 5.2. is an example of a simple WFMB.

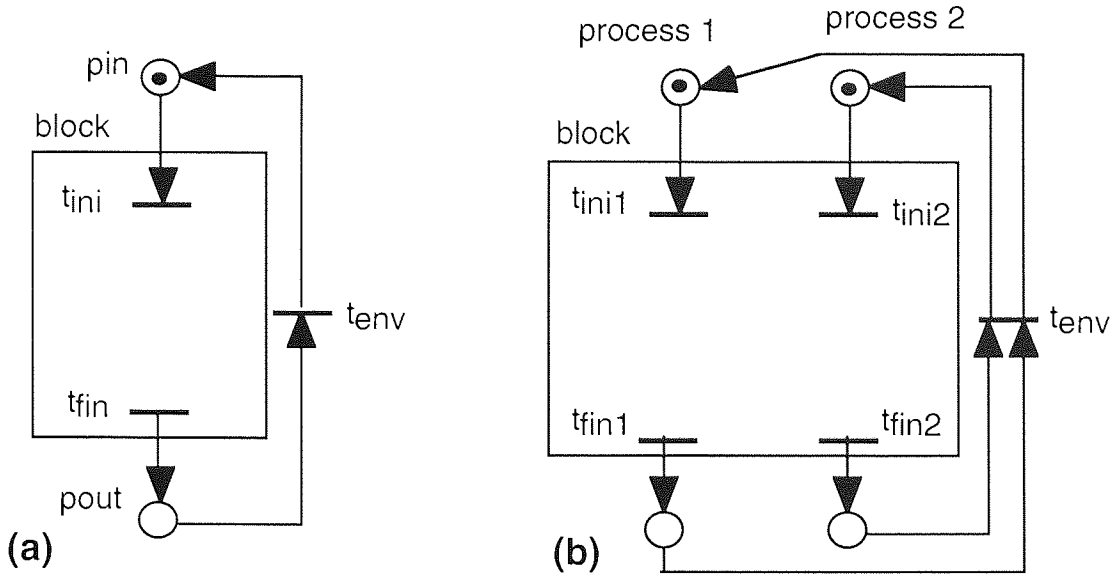


Fig.5.16. (a) well formed block with environment transition, (b) well formed multi-block with environment transition, (SISO).

The associated Petri net PN' = (P', T', I', O', M_0') , is obtained from the block Petri net = (P, T, I , O, M_0) , by the addition of the input and output places { p_{in}, p_{out} }, and the environment transition t_{env}. Such that for each WFMB process:

i) the only output transition of p_{in} is t_{ini} for *each process*,

ii) the only input transition of p_{in} is t_{env} for *each process*,

iii) the only input transition of p_{out} is t_{fin} for *each process*,

iv) the only output transition of p_{out} is t_{env} for *each process*,

v) $M_0' = M_0 + M_{in}$.

Where M_{in} is the marking of the input places for each process.

M_{out} represents the markings of the output places for each process. For the simple WFMB there is one place per process in each of the sets, M_{in}+ M_{out}. The firing of the environment transition transforms the marking M_{out} to the marking M_{in}.

v) $M_0' = M_0 + M_{in}$.

Where M_{in} is the marking of the input places for each process.

M_{out} represents the markings of the output places for each process. For the simple WFMB there is one place per process in each of the sets, M_{in} and M_{out}. The firing of the environment transition transforms the marking M_{out} to the marking M_{in}.

The following definitions can then be given:

vi) the block PN is bounded iff the associated Petri net PN' is bounded,

vii) the block PN is safe iff the associated Petri net PN' is safe,

viii) the block PN is live iff the associated Petri net PN' is live.

A block Petri net PN is a well formed multi-block (WFMB) iff the associated Petri net PN' is such that:

ix) the PN' is live,

x) $M_{0'}$ is the only marking in the set of markings reachable from $M_{0'}$ such that marking M_{in} holds, (each processes input place is not empty),

xi) the only transitions enabled by $M_{0'}$ are the set of initial transitions t_{ini} for each process,

xii) the only transition enabled by the firing of the final transitions t_{fin} for each process, is t_{env},

xiii) the firing of the environment transition t_{env}, produces the marking M_{in}.

The block structures such as the *prepare* and *vote* communication blocks differ from this form of WFMB. They have single-input multiple-output (SIMO) places for each process and are termed complex WFMB.

Analysis of the underlying Petri net templates for these blocks in section 5.2.5. has shown them to be live, 1-bounded and deadlock free. The verification of properties for these blocks in section 5.2.6., showed each marking in the set of output markings M_{out} contained only one output place per process. This analysis only holds for the set of initial markings defined by the input sets for each block.

The complex WFMB requires a set of environment transitions, one for each member of M_{out}, The firing of the associated environment transitions (t_{env_i}) transforms the output marking (M_{out_i}) to the single input marking (M_{in}). The markings M_{in} and M_{out} are equivalent to the input and output sets for the communication blocks, this is shown in Fig.5.17.

The analysis of the multi-process blocks in section 5.2.5. showed that the underlying Petri nets were live, and deadlock free, even when only some processes input places are tokenised.

The verification of properties for the communication blocks in section 5.2.6. showed that only the processes with input places tokenised, have output places tokenised in the corresponding output set. The tokenising of each processes output places are mutually exclusive (for each process an input place token is absorbed and eventually produces a single output place token).

The dynamic behaviour of the underlying Petri net template involves only those processes present in the input set. Thus the WFMB behaviour can be considered an atomic action:

i) the block fully completes or aborts,

ii) for a composed Petri net (such as Fig.5.14.) for each process in the input set of the block, all transition steps either precede or succeed the steps of the block.
iii) only the external behaviour for each process is observable

The commit protocols developed in section 5.2. are restricted cases of the general (complex) WFMB structure, as shown below in Fig.5.17. M_{out} for this form of the complex WFMB contains only two markings, one for the commit outcome of the protocol, and one for the abort outcome of the protocol. The two associated environment transitions are thus marked t_{env_c} (commit outcome) and t_{env_a} (abort outcome) and represent a reduced model of the environment for the WFMB.

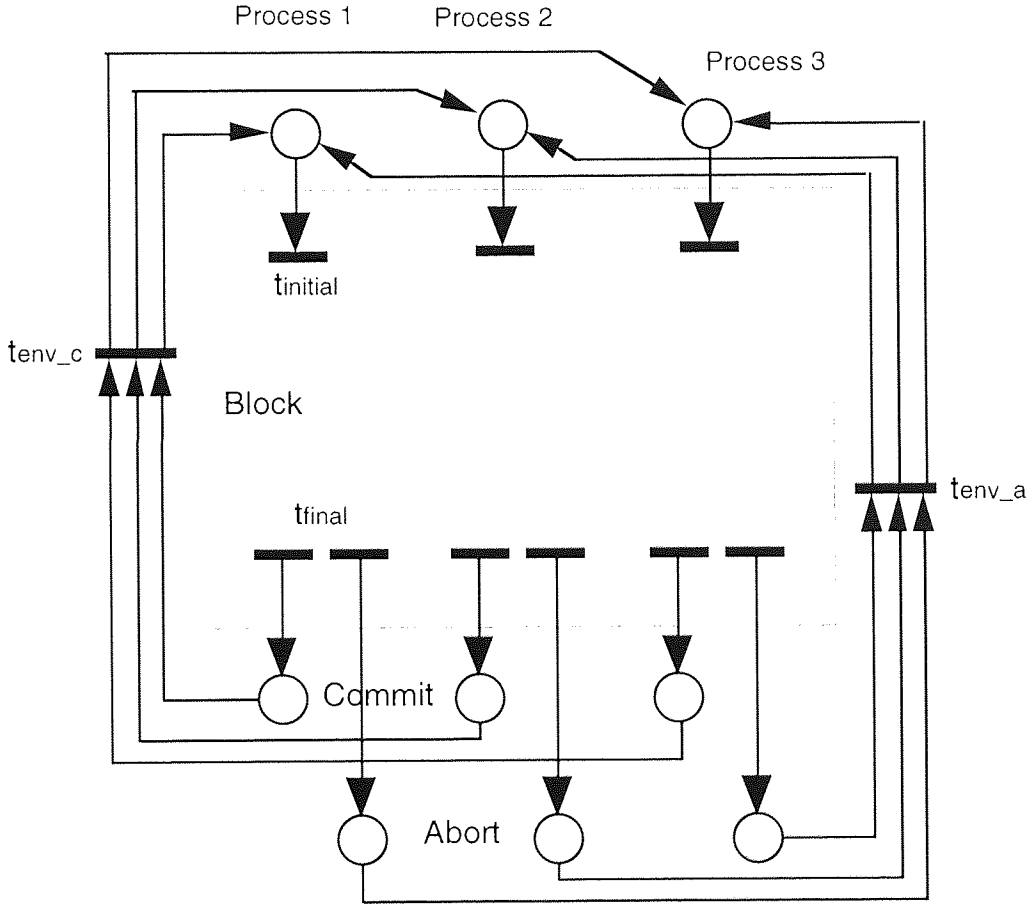


Fig.5.17. (complex) Well formed multi-block, with dual exit path

The sets of markings M_{in} and M_{out}, and environment transitions { t_{env} } form an abstract model of the environment for the WFMB. This is restricted to the behaviour of interest in the design and analysis of the module.

5.3.3 Reachability analysis of hybrid design E2PC

In this section the behaviour of the complex WFMB (defined in section 5.3.2.), are used for the modular analysis of the composed protocols (developed in section 5.2.5). The firing sequences of the WFMB's underlying Petri nets are replaced by direct mappings between the

external behaviours of the blocks. These mappings take the form of atomic transitions, which are fired to generate the WFMB dynamic behaviour. The use of these mappings conceal the local events of the blocks not required for analysis. Where a set of atomic transitions replace all the firing sequences for the composed WFMBs, their firing can be used to generate a reachability graph. In Fig.5.14., each communication block will be replaced by several atomic transitions, these can be fired to generate the reachability graph for the protocol, allowing reachability based analysis to be used.

In order to extend a modular design approach to enable a form of modular analysis, it is necessary to derive the behaviour of the composed system from the behaviour of the composed components [Kindler 97]. This should be possible without considering the component details at each stage of analysis. Bucci & Vicario [95] point out that although extensions to Petri nets that allow module construction exist, they are not accompanied by analysis techniques that exploit the modularisation. Generally, modularisation is restricted to the specification, and analysis must be performed on the (flat) detailed net. In [Valmari 93] the reachability set of a composed net is generated by composing the reachability sets of subnets. This is combined with behaviour preserving reductions between the composition steps. However, this can still result in a large state space for the composed system.

Reachability graphs of Petri net component modules (WFMB) can be *projected* onto reduced representations. These representations conceal events local to the module that are not essential for the purpose of analysis [Bucci & Vicario 95], while preserving an equivalent behaviour, in order to manage the state space problem.

$M_{in}[\alpha > M_{out}$ denotes marking M_{out} is reachable from M_{in} via the firing sequence α. For a WFMB, for each input set marking there are an associated set of output set markings. These are reachable by an associated set of firing sequences, $M_{in(i)}[\alpha_{(j)} > M_{out(j)}$. The input set and output set markings, M_{in} and M_{out}, are equivalent to the input and output sets used to define the external behaviour of the communication blocks in section 5.2.4.

In a composed Petri net design, such as Fig.5.14., each firing sequence of a WFMB is considered an atomic action. This atomic action transforms an input place marking to an output place marking. To generate the external behaviour of the WFMB each firing sequence is replaced by an atomic transition internal to the block. This is a projection which conceals events local to the block, where these events are not required for analysis.

To generate the reachability graph for a composed Petri net containing WFMB, each firing sequence $\alpha_{(j)}$ of the WFMB is replaced by an atomic transition $t_{(j)}$. The firing of this transition transforms the input marking of the WFMB $M_{in(i)}$ to the output marking $M_{out(j)}$.

This is possible as the firing sequence $\alpha_{(j)}$ involves only those processes present in the input set marking $M_{in(i)}$.

The atomic transition $t_{(j)}$ is a direct mapping from the input set marking to the output set marking. Where $M_{in(i)}[t_{(j)} > M_{out(j)}$ denotes $M_{out(j)}$ is directly reachable from $M_{in(i)}$ via the firing of $t_{(j)}$. Thus $M_{in(i)}$ and $M_{out(j)}$ are equivalent to the input and output functions *I* and *O* for a standard Petri net transition.

Atomic transitions of this form can be associated with each of the communication blocks input and output sets. The input and output sets of the communication blocks are defined for the commit protocols developed in section 5.2. (such as Fig 5.14.). Since confidence in these input and output sets has been generated by the verification procedure presented in section 5.2.6., there can be confidence in the mappings based on them. The mappings are of the form $M_{in(i)}[t_{(j)} > M_{out(j)}$, and can be used to generate the correct dynamic behaviour of the hybrid Petri net design.

$t_{(j)}$	Input set [$M_{in(i)}$]	Output set [$M_{out(j)}$]
1	p24, p34, p44	p25, p35, p45
2	p24, p34, p44	p26, p35, p42
3	p24, p34, p44	p27, p32, p45
4	p34, p44	p32, p42
5	p34	p32
6	p44	p42

Table.5.7. external behaviour of the *commit* communication block

Consider the external behaviour for the commit WFMB (The numbering of places in Table 5.7 is from the composed commit protocol given in Appendix.B.2.). To generate the dynamic behaviour of this WFMB would require six atomic transitions of the form $t_{(j)}$, where $M_{in(i)}[t_{(j)} > M_{out(j)}$. For example the firing of the atomic transition $t_{(4)}$, maps the input set { p34, p44 } to the output set { p32, p42 }.

Local events are concealed through the projection of the external behaviour of modules [Bucci & Vicario 95], this forms the basis of the clustering stage of the bottom-up approach. The dynamic behaviour of the WFMB, produced by the firing of the atomic transitions, is equivalent to this type of projection. These mappings are then used in the WFMB defined above for the execution of the dynamic behaviour of the hybrid Petri net design.

Applying this procedure to the hybrid protocol design (Fig.5.14. - CB1) results in a Petri net with 39 transitions and 41 places. This includes the nine environment places (three per

process) and two environment transitions for the abort and commit case. These are necessary to form a WFMB for the commit protocol block (termed the commit block). The reachability graph generated consists of a state space of 89 nodes. This state space is a considerable reduction in size from the fully enumerated state space for the equivalent protocol of chapter 4. (Fig.4.12) which had a state space of 1563 nodes. The reachability graph for the commit protocol (CB1) generated in this manner is given in Appendix.A.3., along with the concurrency set and results of the Petri net analysis. In Appendix.A.4. the equivalent information for the commit protocol (CB2) is given, this protocol is presented in Appendix.B.3.

Reachability based analysis of the associated Petri net for the commit block (presented in Appendix.B.2) using the state space generated has been performed. The Petri net has been shown to be live, safe (1-bound) and deadlock free and has a state space of 89 nodes. Examination of the markings of the reachability graphs has shown that all markings in M_{out} enable one of the two environment transitions (t_{env}). This represents a consistent abort or commit decision always being reached by the protocol. This property can be confirmed through examination of the concurrency sets for the output places of the commit block, { p_3, p_7, p_{11} } - *commit* and { p_4, p_8, p_{12} } - *abort* .

The reachability analysis has shown these hybrid designs to be consistent models of atomic commit protocols, and also that they are equivalent to the fully elaborated designs developed in chapter 4. Equivalent markings of the commit blocks input places { p_1, p_5, p_9 } produce equivalent external behaviour, markings of the commit blocks output places { p_3, p_7, p_{11} } - *commit* or { p_4, p_8, p_{12} } - *abort* .

The same number of synchronous communications and timeouts are used in both models (Fig.4.12. & Fig.5.14.), but their use is implicit in the hybrid design protocol (Fig.5.14.). The communication blocks are used to conceal the behaviour of these communications and timeouts, as this is not required for the analysis of the protocol at this level of abstraction.

The reachability analysis in this section is modular, as the individual analysis of each communication block (considered in section 5.2.4.) is reused in the analysis of the composed commit protocol.

The separate analysis of the communication blocks (section 5.2.4. & 5.2.6.) has verified the relationship between the input and output sets. The composition of these reduced reachability graphs, is achieved by generating the reachability graph for the composed protocol (using the defined mapping $M_{in(i)}[t_{(j)} > M_{out(j)}$ for each WFMB). Analysis of the composed protocol is possible using the reachability graph generated.

5.3.4. Verification of hybrid design E2PC

The composition of the communication blocks relied upon examination of the blocks input and output sets. These sets were defined in section 5.2.4., and the composition procedure in section 5.2.5. The verification of liveness properties for each communication block was performed in section 5.2.6., these properties related input place marking and output place markings.

The verification of properties for each communication block was performed using Temporal Petri nets. These properties can only be guaranteed for certain sets of input place markings. The reachability graph generated for the composed protocol model Fig.5.14., was developed in section 5.3.3. This reachability graph can be used to show the input place markings for each communication block. If the input set markings for the blocks can be shown to hold, then the properties of the blocks can be used to deduce properties for the composed design.

The reachability graph for the composed protocol (CB1) is given in Appendix.A.3 (protocol numbering is shown in Appendix.B.2), this shows that the input place markings for each of the communication blocks holds, therefore the properties verified for each of the communication blocks also holds.

From the guaranteed output set markings for each block, the possible firing sequences should lead to the input set marking for the subsequent communication block. This can be shown by examining each communication block in turn, for the composed protocol (Appendix.B.2). Consider the prepare block, the input and output set for this block are given in Table 5.8:

Input set	Output set
p20, p30, p40	p21, p31, p41
p20, p30, p40	p22, p31, p42
p20, p30, p40	p22, p32, p41

Table.5.8 external behaviour of the *prepare* block in Petri net Appendix.B.2.

The three output sets are guaranteed for the single input set by the proof of property (5.7). The firing sequences possible from these markings are given by the reachability graph. From observation of the net it can be seen that the transitions enabled are (t_6, t_7, t_{11}, t_{12}). Places p_{31} and p_{41} are the decision states for the participant processes, and the enabled transitions represent the *Yes* decision (proceed with protocol) and *No* decision (unilaterally abort protocol) for the participants. By considering the firing of these transitions in the reachability graph, for each of the output set markings in Table.5.8, it can be seen that the vote blocks

input set markings are produced. Therefore the properties for the vote block can be guaranteed. The proofs of properties (5.8) to (5.13) verified the output sets for the *vote* block.

By following each blocks output set marking in the reachability graph it can be seen that the next blocks input set markings hold, and therefore that the properties of the next block also hold. By considering these liveness properties of each communication block in a stepwise manner, liveness properties for the composed protocol can be deduced.

For the input marking { p_1, p_5, p_9 } of the composed protocol, the final output place markings produced are { p_3, p_7, p_{11} } or { p_4, p_8, p_{12} }. This property can be deduced using the properties of the communication blocks and the reachability graph for the composed protocol.

To verify this property using the Temporal Petri net approach presented in section 5.2.6., it would only be necessary to consider the firing sequences for the minimal number of transitions { t_1, t_2, t_6, t_7, t_{11}, t_{12}} that interconnect the previously verified communication blocks. This procedure requires the generation of the reachability graph for the hybrid protocols, presented in section 5.3.2.

The verification of the liveness property for the composed commit protocol, guarantees that for each process, the protocol produces a consistent decision. Eventually either:
i) all processes reach their commit states, or
ii) all processes reach their abort states.

Case i) is the output set for the *ack* communication block, and case ii) is a possible outcome for each of the communication blocks prior to this. The validity of this can be confirmed from the reachability analysis presented in section 5.3.3.

5.3.5 Dynamic behaviour of Commit block

The composed commit protocols can be used as the underlying Petri net template for the commit block. It is intended to use these commit blocks in modular Petri net designs, presented in chapter 6.

For the commit block protocol CB1 (Fig.5.18.), the underlying Petri net structure is given in Appendix.B.2. The input and output sets for this commit block are given in Table.5.8 The two output sets represent the *commit* and *abort* outcome of the protocol.

t_j	Input set (M_{in_i})	Output set (M_{out_j})
1	p1, p5, p9	p3, p7, p11
2	p1, p5, p9	p4, p8, p12

Table.5.8 external behaviour of the commit block CB1

The dynamic behaviour of the commit block can be generated by the following mappings between input and output sets:

i) $M_{in(1)}[t(1) > M_{out(1)}$, and

ii) $M_{in(1)}[t(2) > M_{out(2)}$.

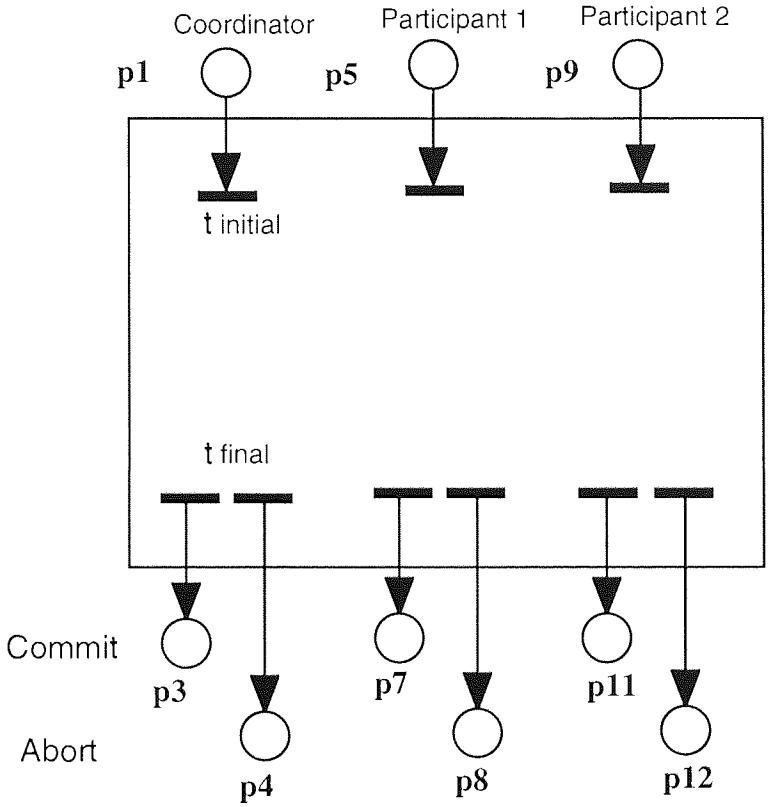


Fig.5.18. Commit block external behaviour

When this commit block is used in a composed Petri net design, the firing of the atomic transitions defined would produce the blocks dynamic behaviour. The commit block would be composed in a hybrid Petri net by merging the input and output places shown in Fig.5.18. This is illustrated in chapter 6. through the design of two distributed controller applications.

5.4. Conclusion.

This chapter has presented a hybrid approach to Petri net design for a restricted class of communicating processes. The approach is termed hybrid as a top-down refinement was used for the design of the commit protocols basic structure, followed by a bottom-up

composition of sub-nets (termed blocks) to create the final design. A modular approach to analysis was used, as the analysis and verification of the hybrid design commit protocol rely upon properties of the sub-nets (Petri net templates) which have been proven in section 5.2 using Petri net and Temporal Petri net analysis and verification.

These blocks structures were based on the notion of Well Formed Blocks (WFB), but differ from Valette's [79] in that each block can be used to model the interaction of several processes and the blocks can have more than one final transition. These types of blocks were called Well Formed Multi-Blocks (WFMB), and like WFB the behaviour of the underlying Petri net is live, bounded and deadlock free. The WFB presented in [Valette 79] are used in a top-down approach to Petri net design through stepwise refinement, while the WFMB presented in section 5.3 are used in a hybrid approach to Petri net design and as an aid to analysis. The communication and commit blocks offer a reduced representation of the underlying Petri nets behaviour that allows a composition procedure, based on the definition of WFMB presented in section 5.3.

The modular approach to design is based on the hybrid synthesis of Petri nets , where the top-down decomposition technique preserves the global view of the system, and the bottom-up approach helps to synthesise accurate models of low level operations. While the modular approach to analysis reduces the state explosion problem associated with fully elaborated nets.

According to Ramaswamy & Valavanis [96] the advantage of such a hybrid, block structured approach is that:
i) the top down decomposition technique preserves a complete global view of the details of the system modelled.
ii) the bottom up approach can preserve responsive communication and synchronisation structures for the different levels of abstraction.

The WFMB approach allowed the modular design and analysis of the communication and commit block, and then the hybrid protocol designs. The communication blocks underlying Petri nets were analysed using standard Petri net analysis, and subsequently the notation of Temporal Petri nets was used to verify certain liveness properties. Properties of the WFMB communication blocks were used to enable reachability analysis of the hybrid protocols. This allowed a unified approach, as the specification, design and analysis of the commit protocols developed are based on the same model.

This hybrid approach is suitable both for system modelling and design, and for analysis. In [David & Alla 92] [Ramaswamy & Valavanis 94, 96] [Zhou & DiCesare 93] [Suzuki *et al*

90] methods are described where Petri nets provide such a unified basis for the modelling and analysis of systems. However, the approach presented in this chapter is particularly suited for use with distributed systems as the underlying communication system model is used as the basis for the choice of sub-net (underlying each block) and for the composition procedure. To illustrate their application the commit blocks developed in section 5.3 will be applied to two real-time distributed control systems in Chapter 6.

In [Bucholz 94] a modular approach to Petri net design based on the communication between processes is presented. In this approach sub-net composition is used, where each sub-net is executable, the dynamic behaviour being described in a local reachability net, and the composition of each sub-nets generated state space is required for the analysis of the composed system. The behaviour of the environment is partially considered during subnet reachability generation in [Bucholz 94], a similar approach was taken in sections 5.2 and 5.3 for the analysis of the communication and commit blocks. However, the environment was modelled as an integral part of the commit protocol.

In [Shieh *et al* 90] the notion of synchronisation and checkpoint transitions for Petri net models of distributed systems are introduced, and are used in the modelling and analysis of fault tolerant schemes. The checkpoint transition represents the coordination of independent processes in order to take a consistent checkpoint of the system for rollback purposes. However, the coordination of the processes to achieve this checkpoint is not considered. In Chapter 6, it is shown that the commit blocks provide an equivalent function to these transitions, but are based on analysis of the actual mechanisms required to implement such a scheme.

For each communication stage of the composed commit protocol, presented in section 5.3, the input and output places of each communication block were shown to be sufficient for reachability analysis, based on the execution of the communication block as an atomic action. In section 5.3.5 it was discussed how the developed commit blocks can be used in the reachability analysis of a Petri net design incorporating them, in the same way as the reachability graph for the hybrid design commit protocol was generated using the commit blocks, as presented in section 5.3.4.

The hybrid design commit protocols developed in section 5.3 will be applied to two real-time distributed control applications in Chapter 6, in order to illustrate their use.

Chapter 6 Synthesis & Analysis of Distributed Controllers

6.1. Introduction

From the survey of modelling techniques presented in Chapter 3, Petri net based techniques were selected for use in this Thesis because they allow concurrency to be represented by a simple and intuitive graphical notation, and coupled with temporal logic they allow reasoning about time related behaviour. Commit protocols that provide real-time and fault-tolerant behaviour, termed responsive commit protocols, were developed in Chapter 4, and served as the basis for the design and analysis procedure developed in Chapter 5. In Chapter 5 commit protocols were modelled using a modular approach based on Petri nets, and reusable Petri net templates of responsive commit protocols were developed. These commit protocols were analysed using reachability based Petri net analysis and verified using the notation of Temporal Petri nets using a modular procedure based on *'block'* structures.

In this chapter it is intended to apply the Petri net templates (of commit protocols) developed in Chapter 5 in the design of distributed controllers for two real-time problems. The analysis and verification of the resulting controller designs will then be performed, based upon the modular analysis and verification of the commit protocols presented in Chapter 5.

The control of complex, independently driven, high-speed machinery was traditionally achieved through mechanical means in order to achieve synchronisation for their correct and safe operation, and forms the target application for real-time embedded applications in manufacturing. The flexible design and speed of such applications can be greatly increased through the replacement of the mechanical transmissions used, by independent software controlled drives [Sagoo & Holding 90].

The embedded real-time software controllers of these drives must achieve continuous or intermittent synchronisation for the correct operation of these systems. The synchronisation logic [Sagoo & Holding 90] to accomplish this must be fault-tolerant in order to operate safely under both normal and abnormal conditions; the latter are those that are likely to lead to failure.

These controllers can be classified as real-time safety-critical systems (RTSCSs) because the failure, of these systems, including the time constraints of their environment, can lead to accidents involving injury or damage to the environment [Burns & McDermid 94].

The distributed controllers for the two applications are developed and analysed in sections 6.2 and 6.3. These controllers are designed in a hybrid manner, first the centralised forms of the controllers are partitioned into the separate processes, and then these separate controllers are composed using the *commit blocks* developed in Chapter 5. These commit blocks are Petri net templates of the commit protocols required for real-time fault-tolerant coordination.

The separate local controllers developed for each process are examined and their synchronisation points determined. The synchronisation points represent where coordination of the controllers are required, it is intended to achieve this coordination using commit protocols. The controllers are then composed using the commit blocks developed in Chapter 5 at the identified synchronisation points. These interaction points are termed *commit transitions* in the Petri net models of the distributed controller. The composition of the distributed controller will follow the procedure used in Chapter 5 for the composition of the commit protocol models. This composition procedure is completed using protocol subnets which have been shown to exhibit certain structural behaviour (such as liveness and boundedness).

The analysis and verification of the resulting Petri net designs, for the distributed controllers, is then presented. This is based upon the properties proven for the (commit protocol) Petri net templates presented in Chapter 5.

The compositional approach to the design of Petri nets, and the modular approach to their analysis and verification is illustrated through its application to two industrial systems; the first is a prototype can sorting machine [Jiang 95][Holding *et al* 95] and the second a drum transfer mechanism [Sagoo & Holding 90].

6.2. Multi-Axes Can Packaging Machine

The use of commit blocks is illustrated by the application of these protocol templates (developed in Chapter 5) to the design of synchronisation logic for a controller of a high speed can packaging machine; a schematic of the can packaging machine is shown in Fig.6.1.

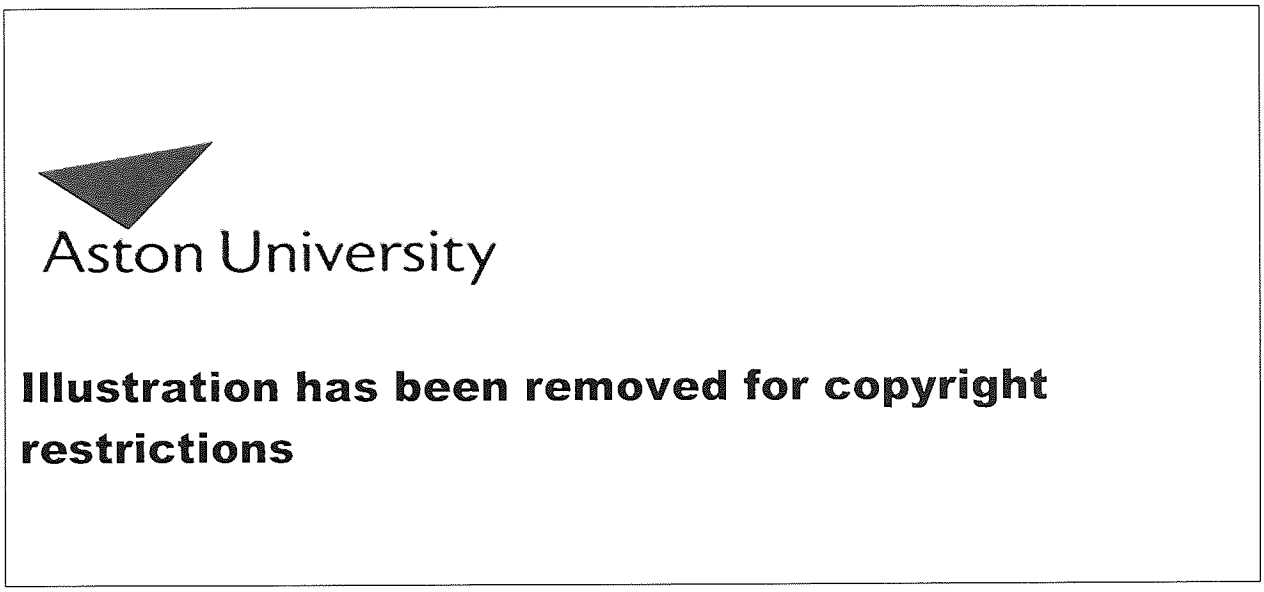


Fig.6.1. Can packager machine physical system.

The can packaging machine developed by Eurotherm Controls Ltd., comprises six independently driven axis, and was presented in [Jiang 95]. In that work a rule based approach was used to synthesise a controller for the mechanism, by the transformation of the synchronisation constraints on the system into a Petri net model. Petri net analysis techniques [Peterson 1981][Murata 1989] were then used, along with extended Temporal Petri nets [Sagoo 92] for analysis of the controller design. In [Azzopardi 96] an object-oriented Petri net technique was used for the design of a controller for the same system.

The above research developed designs for the controller based on the assumption that a centralised (single) controller was able to coordinate the motion of all the independent drives. A site failure for the centralised controller would be a single point of failure for the system, and would result in loss of control of all drives. A failure of the communication link between the centralised controller and an independent drive would result in the loss of control of the particular drive. In order to increase flexibility and robustness of the system design, this centralised controller is to be replaced by a set of physically distributed controllers co-placed with the independent drives. The co-ordination and synchronisation of these drives will thus require a distributed version of the synchronisation logic and the local control, incorporating real-time inter-process communications.

6.2.1 Application system

The system requires the intermittent synchronisation of six independent axis: drum1, drum2, drum transfer slider, feeder transfer slider, conveyor and conveyor transfer slider as illustrated in Fig.6.1.- 6.3. These six axis exhibit independent asynchronous motion until constrained into local synchronisation by the synchronisation logic.

The controller design is used to achieve the co-ordination and intermittent synchronisation of the independent drives of a high speed can packaging machine; a task achievable through the use of distinct local controllers whose actions are co-ordinated using responsive commit

Fig.6.2. Can packager physical system (side).

Fig.6.3. Can packager physical system (front).

6.2.2. Modelled system

Cans are transferred from the *feeder* to *drum1* by the *feeder transfer slider*, (as shown in Fig.6.2.) from *drum1* to *drum2* by the *drum transfer slider* (as shown in Fig.6.3.) and *drum2* to the *conveyor* by the *conveyor transfer slider* (as shown in Fig.6.2.). The conveyor/feeder, transfer sliders and drums are driven by independently driven motors, synchronisation is required in order to transfer the cans between the different component of the mechanism.

Petri net models of the separate controllers for each axis are created. These are illustrated in Fig.6.4., the (*commit*) transitions (where the processes are required to synchronise and reach a decision) are shown in outline. Table.6.1. gives the semantics for the places of these nets, with the places that represent the synchronisation logic for each controller are marked *, and Table.6.2. the semantics for the commit transitions.

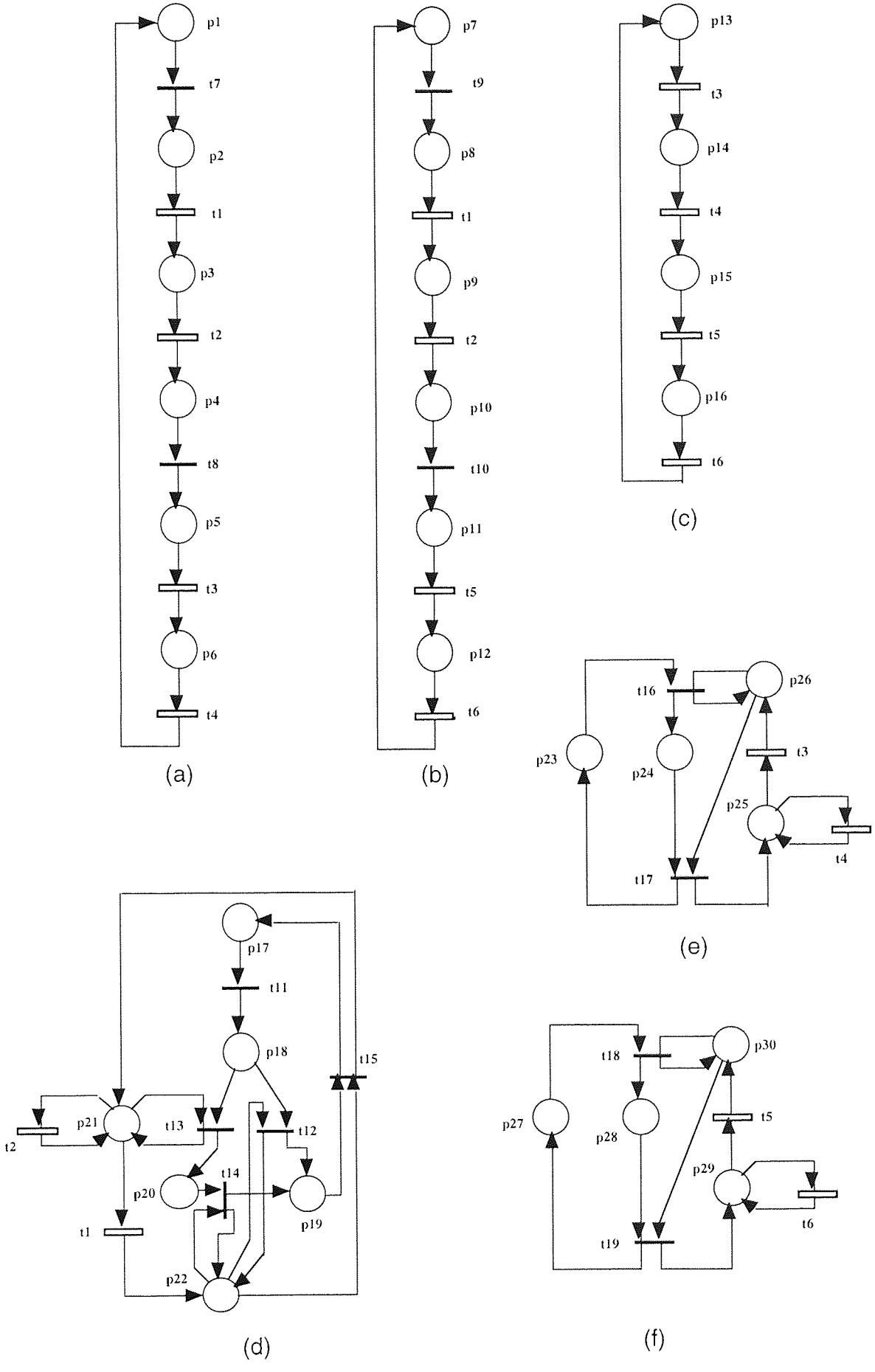


Fig.6.4. Can packager distributed controllers models.

In Fig.6.4. the following are the local controller models; (a) Drum1, (b) Drum2, (c) Conveyor/Feeder, (d) Drum transfer slider, (e) Feeder transfer slider & (f) Conveyor transfer slider.

A distributed controller is developed using a compositional design method. The Petri nets of the local controllers are composed through fusion of the commit transitions, to form the required synchronisation logic. While the commit transitions are refined using the commit blocks developed in Chapter 5. Fusion of Petri net transitions means the subnets share events. Shared transitions imply the use of synchronous communication (as used in the commit protocol designs developed in Chapters 4 and 5), while shared places are used (when shared resources are modelled), and implies the use of asynchronous communications [Khalifa & Nketsa 96].

Process	State	Semantic
Drum 1	p1	Drum1 rotate with can
	p2	Drum1 await transfer with can
	p3	Drum1 committed to drum transfer *
	p4	Drum1 rotate without can
	p5	Drum1 await feeder without can
	p6	Drum1 committed to feeder transfer *
Drum 2	p7	Drum2 rotate without can
	p8	Drum2 await transfer without can
	p9	Drum2 committed to drum transfer *
	p10	Drum2 rotate with can
	p11	Drum2 await conveyor with can
	p12	Drum2 committed to conveyor transfer *
Conveyor/Feeder	p13	Conveyor/Feeder can available
	p14	Feeder committed to transfer *
	p15	Conveyor/Feeder no can available
	p16	Conveyor committed to transfer *
Drum Transfer	p17	Slider return and approach motion
	p18	Slider decision point
	p19	Slider insert motion
	p20	Slider abort motion
	p21	Slider insert inhibit *
	p22	Slider insert active *
Feeder Transfer	p23	Feeder slider withdraw
	p24	Feeder slider insert

	p25	Feeder slider insert inhibit	*
	p26	Feeder slider insert active	*
Conveyor Transfer	p27	Conveyor slider withdraw	
	p28	Conveyor slider insert	
	p29	Conveyor slider inhibit	*
	p30	Conveyor slider active	*

Table 6.1. Semantic for 6-Axes Petri nets, Fig.6.4. & Fig.6.5.

Table 6.2. gives the semantics for the commit transitions, along with the processes which are composed through the fusion of these transitions. In Table 6.2 the assignment of the controller in the commit protocol shown; coordinator (Coord), participant-1 (Part1) and participant-2 (Part2).

Transition	Processes (Coord, Part1, Part2)	Commit decision
t1	Drum Transfer, Drum 1, Drum 2	Commit to drum transfer
t2	Drum Transfer, Drum 1, Drum 2	Drum transfer complete
t3	Feeder Transfer, Drum 1, Conveyor	Commit to feeder transfer
t4	Feeder Transfer, Drum 1, Conveyor	Feeder transfer complete
t5	Conveyor Transfer, Drum 2, Conveyor	Commit to conveyor transfer
t6	Conveyor Transfer, Drum 2, Conveyor	Conveyor transfer complete

Table 6.2. Semantic of Commit transitions for Petri net models, Fig.6.4. and Fig.6.5.

Fig.6.5. shows the composed controller, again with the commit transitions shown in outline. These commit transitions are abstractions of the commit blocks developed in chapter 5, which were proven to exhibit certain important properties (liveness, boundedness, freedom from deadlock and consistent atomic commit).

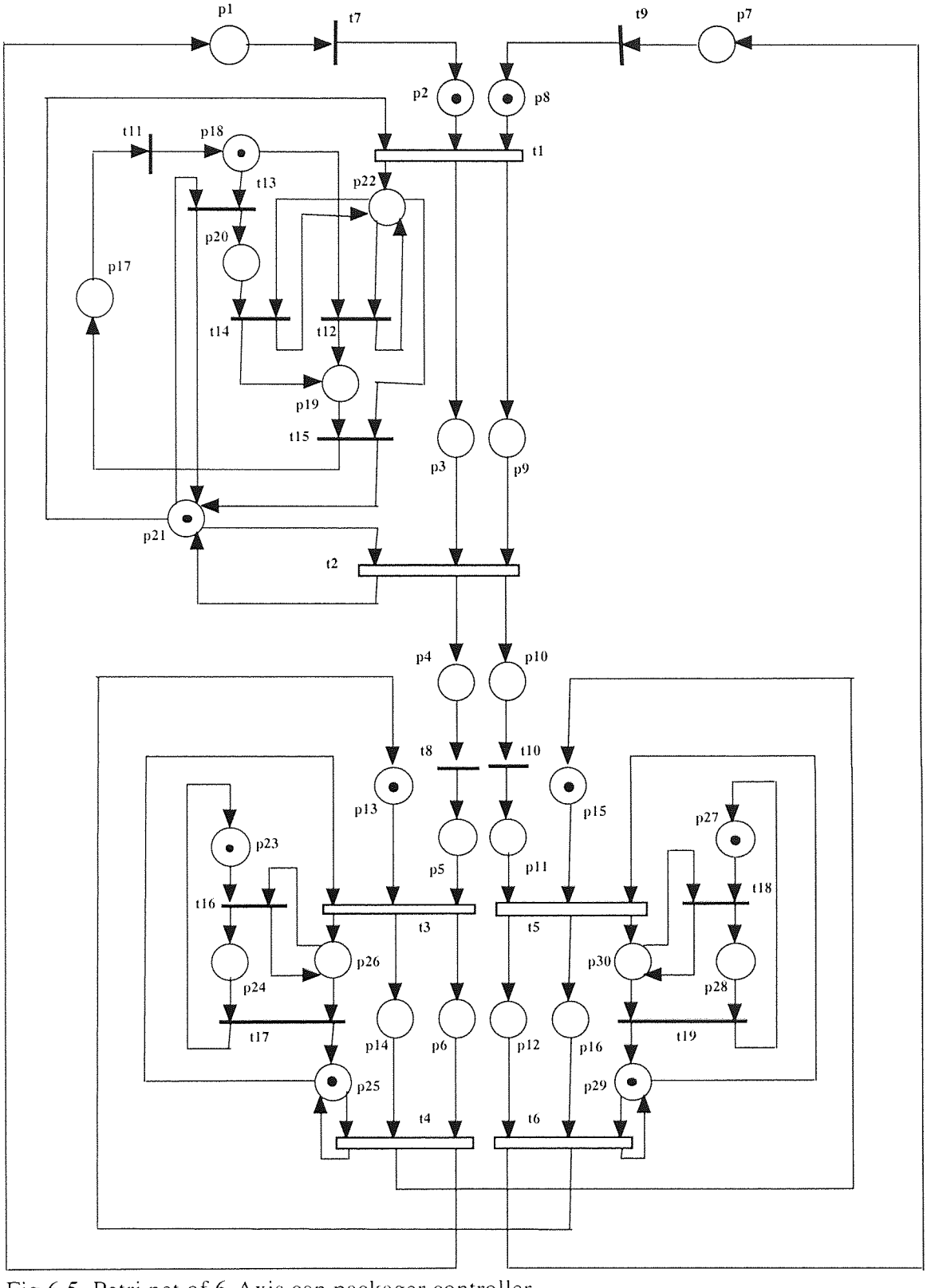


Fig.6.5. Petri net of 6-Axis can packager controller

The refinement of the commit transitions with commit blocks allows the distributed controller design to inherit the properties (such as application specific safety and liveness properties) of the commit blocks underlying Petri net template. The commit blocks are models of the commit protocols that guarantee all parties reach a timely and consistent decision. The application requires the use of responsive protocols [Kakuda *et al* 92, 94], as

both fault-tolerant and real-time performance constraints must be met. These properties of the commit blocks were verified in Chapter 5.

An inspection of commit transition t_1 of Fig.6.5. shows the procedure used to compose the Petri nets of Fig.6.4.(a) - (d) to yield the Petri net of Fig.6.5. Transition t_1 represents the coordination of the drum1, drum2 and drum transfer processes necessary to commit or abort the drum transfer action. The commit transitions in the local controllers are identified (such as t_1 in Figs.6.4.(a), (b) and (d)) and fused in such a way that the input and output places of the commit transitions are merged with the input and output place sets for the commit block, Fig.5.18. The commit transitions represent the point in the local controllers operation where coordination with other controllers is required. For transition t_1, the input places to the commit transitions (shown in Figs.6.4.(a), (b) and (d)) are merged with the input places to the commit block (Fig.5.18, and given in bold here):

i) **p1** - $p21$, drum transfer slider (commit protocol coordinator),
ii) **p5** - $p2$, drum1 (commit protocol participant-1),
iii) **p9** - $p8$, drum2 (commit protocol participant-2).

The commit protocol outcome of a *commit* decision can be seen as a successful firing of transition (t_1), where the output places of the commit transitions (shown in Figs.6.4.(a), (b) and (d)) are merged with the output places of the commit block (Fig.5.18, and given in bold here). The places merged are:

i) **p3** - $p22$, drum transfer slider (commit protocol coordinator),
ii) **p7** - $p3$, drum1 (commit protocol participant-1),
iii) **p11** - $p9$, drum2 (commit protocol participant-2).

This outcome of the commit block is the consistent decision required for the controllers to safely commit to the drum transfer action. This outcome enables the firing of transition t_{12} when p_{18} is tokenised at the decision point (shown in Fig.6.5.).

The commit protocol outcome of an *abort* decision represents the controllers reaching a consistent decision to abort the drum transfer action. This situation requires the re-tokenising of the commit blocks input places. The abort decision output places of the commit block (Fig.5.18, and given in bold here) are merged with the input places of the commit transitions (shown in Figs.6.4.(a), (b) and (d)) :

i) **p4** - $p21$, drum transfer slider and commit block coordinator,
ii) **p8** - $p2$, drum1 and commit protocol participant 1, and
iii) **p12** - $p8$, drum2 and commit protocol participant 2.

This outcome of the commit block represents a consistent decision by the drum1, drum2 and drum transfer proceed to abort the drum transfer action, and to re-attempt the commit decision. The abort outcome may be due to one of the controllers voting *no* at the vote stage of the protocol, or to the occurrence of a communication link failure. A processes vote depends upon its ability to perform the action requested, this could involve checks on the controlled systems current state or on the availability of resources. If the controlled system is not in the correct state or the allocation of resources necessary to reach a timely decision cannot be guaranteed, then the process votes *no* .

In the presence of a permanent failure, the protocols abort decision prevents processes committing to a hazardous action when the action of the other processes cannot be guaranteed. While, in the case of a transient communication failure the re-tokenising of the commit blocks input places should eventually lead to a commit decision being reached.

The outcome of the commit protocol is summarised in Table 6.3., where the output set of the commit block is always a consistent commit/abort decision of the controllers.

Input set	Output set
p2, p8, p21	p3, p9, p22
p2, p8, p21	p2, p8, p21

Table 6.3. external behaviour of the commit block (t_1), from Fig.6.5.

The resulting Petri net, which is an equivalent *distributed* controller shown in Fig.6.5., was composed by following the same procedure for each of the commit transitions of the local controllers, shown in Fig.6.4.(a) - (d)) and defined in Table 6.2. These commit transitons were replaced with the commit block of the type shown in Fig.5.18.

The reachability based Petri net analysis of the distributed controller shown in Fig.6.5. was performed, after substitution of the commit transitions (t_1, t_2, t_3, t_4, t_5, t_6) by commit blocks. The firing of atomic transitions, that offer equivalent behavior of the commit blocks underlying Petri net templates, are used to generated the state space of the distributed controller. The analysis showed that the Petri net of the distributed controller to be live, 1-bound and deadlock free. The reachability graph and concurrency sets used in this analysis are given in Appendix.A.5., the Petri net had a reachable space of 79 nodes. This information is also used in section 6.2.4. for the verification of properties of the controller (defined in section 6.2.3.).

6.2.3. Properties of the System

The system must be verified for correctness, to ensure that the design exhibits the required behaviour in an acceptable and safe manner. This type of verification is achieved by proving two types of properties that are known as safety and liveness. Informally *Liveness* properties are defined as 'what the system should do' and *Safety* properties as 'what the system should not do' [Lamport 77]. These properties are stated as follows:

Safety Property 1, 2 A situation will *never occur* in which drum1's controller has committed to a transfer and drum1 is rotating, this can be formalised as:

$$\Box\neg[(p3) \wedge (p1 \vee p4)] \quad (6.1)$$

$$\Box\neg[(p6) \wedge (p1 \vee p4)] \quad (6.2)$$

Safety Property 3, 4 A situation will *never occur* in which drum2's controller has committed to a transfer and drum2 is rotating, this can be formalised as:

$$\Box\neg[(p9) \wedge (p7 \vee p10)] \quad (6.3)$$

$$\Box\neg[(p12) \wedge (p7 \vee p10)] \quad (6.4)$$

Safety Property 5 A situation will *never occur* in which the drum_transfer's controller commits to insert when either drum1 or drum2 is rotating, this can be expressed in temporal as:

$$\Box\neg[(p22) \wedge (p1 \vee p4 \vee p7 \vee p10)] \quad (6.5)$$

Safety Property 6 A situation will *never occur* in which the conveyor_transfer's controller commits to insert when either drum2 is rotating or the conveyor controller has not committed to transfer, this can be formalised as:

$$\Box\neg[(p30) \wedge (p7 \vee p10 \vee \neg p16)] \quad (6.6)$$

Safety Property 7 A situation will *never occur* in which the feeder_transfer's controller commits to insert when either drum1 is rotating or the feeder controller has not committed to transfer, this can be expressed in temporal as:

$$\Box\neg[(p26) \wedge (p1 \vee p4 \vee \neg p14)] \quad (6.7)$$

Safety Property 8 A situation will *never occur* in which the conveyor_transfer's slider inserts when drum2 is rotating or the conveyor is not stationary, this can be formalised as:

$$\Box\neg[(\ p28\ ) \wedge (\ p7 \vee p10 \vee \neg p16\ )] \tag{6.8}$$

Safety Property 9 ... 12 The independent motion of the drums is permitted, the *rotating* states are not mutually exclusive, this can be expressed in temporal formula as:

$$\neg\Box\neg(p1 \wedge p7) \tag{6.9}$$
$$\neg\Box\neg(p1 \wedge p10) \tag{6.10}$$
$$\neg\Box\neg(p4 \wedge p7) \tag{6.11}$$
$$\neg\Box\neg(p4 \wedge p10) \tag{6.12}$$

Liveness Property 1 Whenever a can is transferred from the feeder to drum1 then eventually it must be exchanged from drum1 to drum2, this can be expressed in temporal formula as:

$$\Box[(p6 \wedge p14 \wedge p24) \Rightarrow \Diamond(p3 \wedge p9 \wedge p19)] \tag{6.13}$$

Liveness Property 2 Whenever a can is transferred from the drum1 to drum2 then eventually it must be exchanged from drum2 to the conveyor, this can be expressed as eventuality in temporal formula as:

$$\Box[(p3 \wedge p9 \wedge p19) \Rightarrow \Diamond(p12 \wedge p16 \wedge p28)] \tag{6.14}$$

Liveness Property 3 Whenever drum1, feeder/conveyor and feeder transfer processes reach a commit decision to initiate a feeder transfer, then eventually these processes will reach a commit decision that the transfer is complete. This can be expressed as eventuality in temporal formula as:

$$\Box[(p6 \wedge p14 \wedge p24) \Rightarrow \Diamond(p1 \wedge p10 \wedge p25)] \tag{6.15}$$

in terms of the commit blocks, the same property can be re-written as:

$$\Box(t3 \Rightarrow \Diamond t4)$$

6.2.4. Verification of System Properties

The hybrid controller design was analysed and shown to be live, deadlock free and 1-bound (safe) using reachability based Petri net analysis . The reachability graph and concurrency set for the Petri net model (shown in Fig.6.5.) are given in Appendix A.5.2. The following proofs of the above safety properties use the concurrency sets generated for the controller, as shown in Appendix A.5.2.

Safety property 1 & 2, (6.1) $\Box\neg[(p_3) \wedge (p_1 \vee p_4)]$ and
property (6.2) $\Box\neg[(p_6) \wedge (p_1 \vee p_4)]$

Proof

To verify these properties it is necessary to show that the *rotate* states of drum1 are mutually exclusive with states where drum1's controller has reached a commit decision to a transfer action with other processes. This property can be verified by examining the concurrency set (Appendix A.5.3), from where it can be shown that states *Drum1 rotate with can* (p_1) and *Drum1 rotate without can* (p_4) are not in the concurrency set of *Drum1 committed to drum transfer* (p_3) or the concurrency set of *Drum1 committed to feeder transfer* (p_6). This can be verified by inspection of the following concurrency sets:

i) $C(p_3) = \{ p_9, p_{15}, p_{17}, p_{18}, p_{19}, p_{20}, p_{21}, p_{22}, p_{23}, p_{25}, p_{27}, p_{29} \}$, and
ii) $C(p_6) = \{ p_7, p_8, p_{14}, p_{17}, p_{18}, p_{20}, p_{21}, p_{23}, p_{24}, p_{25}, p_{26}, p_{27}, p_{29} \}$

Safety property 7, (6.7) $\Box\neg[(p_{26}) \wedge (p_1 \vee p_4 \vee \neg p_{14})]$

Proof.

To verify this property it is necessary to show that the state where the feeder slider's controller has committed to transfer, is mutually exclusive with the *rotate* states of drum1, and states where the conveyor/feeder's controller has not committed to transfer. This property can be verified by examining the concurrency set, of Appendix A.5.3., from where it can be shown that states *Drum1 rotate with can* (p_1) and *Drum1 rotate without can* (p_4) are not in the concurrency set of feeder transfer *committed to slider insert* (p_{26}). And also that the state of the conveyor/feeder is not other than *feeder committed to transfer* (p_{14}). It can be seen from the concurrency set of p_{26} that states $\{ p_1, p_4 \}$ are not present:

i) $C(p_{26}) = \{ p_6, p_7, p_8, p_{14}, p_{17}, p_{18}, p_{20}, p_{21}, p_{23}, p_{24}, p_{27}, p_{29} \}$

It can also be seen from this concurrency set that state p_{14} is present, and so this state is potentially concurrent with p_{26} as required. From inspection of the Petri net for the conveyor/feeder, shown in Fig.6.4.(c), it can be seen that one of the following places is always tokenised $\{ p_{13}, p_{14}, p_{15}, p_{16} \}$. As the only state from this set in $C(p_{26})$ is p_{14}, then p_{14} is always tokenised whenever p_{26} is tokenised.

The same approach used for the proof of safety properties 1, 2 and 7 has been used to verify the other safety properties stated above.

The same type of Temporal Petri net analysis as used in chapter 5, can be used to prove a class of liveness properties of the system which are important and non-intuitive. These proofs are based on using Propositions that make assertions about the firing of transitions in a

Petri net. These propositions are applied after first checking the validity of their premises against the reachability graph Appendix A.5.3. These Propositions are re-stated as follows:

Proposition 1:
For a temporal Petri net TN_1 whose initial marking is M_0.

(i) $<M_0,\alpha> \models \Box[t(ok) \Rightarrow t(ok)\,\mathcal{U}\,t]$

(ii) $<M_0,\alpha> \models \Box[t(ok) \Rightarrow \Diamond t(\neg ok)]$

(iii) $<M_0,\alpha> \models \Box[t(ok) \Rightarrow \Diamond t]$

This proposition is applicable to the firing of a basic transition, which only disables itself by firing. For the firing of conflicting transitions, where the firing of a transition disables itself and another enabled transition, Proposition 2 is applicable:

Proposition 2:
For a temporal Petri net TN_1 whose initial marking is M_0.

(i) $<M_0,\alpha> \models \Box[\,t_1(ok) \Rightarrow\ t_1(ok)\ \mathcal{U}\,(t_1 \vee t_2)] \vee \Box[\,t_2(ok) \Rightarrow\ t_2(ok)\ \mathcal{U}\,(t_1 \vee t_2)]$

(ii) $<M_0,\alpha> \models \Box[t_1(ok) \Rightarrow \Diamond t_1(\neg ok)] \wedge \Box[t_2(ok) \Rightarrow \Diamond t_2(\neg ok)].$

(iii) $<M_0,\alpha> \models \Box[t_1(ok) \wedge t_2(ok) \Rightarrow \Diamond(t_1 \vee t_2)]$

By applying proposition 1(i) and 1(ii) in turn to each of the nineteen transitions in the net, it is possible to infer using proposition 1(iii) a further nineteen temporal formulae of the form:

$\Box[t_m(ok) \Rightarrow \Diamond t_m]$ where $m = \{\ 1, \ldots, 19\ \}$

The following liveness property was stated in section 6.2.3.

Liveness property 3, (6.15) $\Box[(p_6 \wedge p_{14} \wedge p_{24}) \Rightarrow \Diamond(p_1 \wedge p_{10} \wedge p_{25})]$

Which can also be stated as: (6.15) $\Box(t_3 \Rightarrow \Diamond t_4)$

Let TN_1 be a temporal Petri net translated from the Petri net PN shown in Fig.6.9., with an initial marking $M_0 = \{\ p_2, p_8, p_{13}, p_{18}, p_{21}, p_{23}, p_{25}, p_{27}, p_{29}\ \}$. This property can be formalised as:

$<M_0,\alpha> \models \Box[(\ t_3\ ) \Rightarrow \Diamond(\ t_4\ )]$

Proof

1. From M_0 and Fig.6.5., the firing of t_3 gives the following markings (only the markings of the conveyor/feeder, feeder transfer and drum1 controllers are shown),

$<M_0,\alpha> \models \Box[(\ t_3\ ) \Rightarrow O(\ p_6, p_{14}, p_{23}, p_{26}\ )]$

Let $M_1 = \{\ p_6, p_{14}, p_{23}, p_{26}\ \}$, 1. can be re-written as:

$<M_0,\alpha> \models \Box[(\ t_3\ ) \Rightarrow O(\ M_1\ )]$

2. Considering the sub-marking reached, transition t_{16} is enabled:

i) $\langle M_0,\alpha\rangle \models \Box[M_1 \Rightarrow t_{16}(ok)]$

ii) combining 2.i) with 1. produces:

$\langle M_0,\alpha\rangle \models \Box[(t_3) \Rightarrow Ot_{16}(ok)]$

3. Using proposition 1 the firing of t_{16} yields:

i) $\langle M_0,\alpha\rangle \models \Box [t_{16}(ok) \Rightarrow \Diamond t_{16}]$

ii) combining 3.i) with 2.ii) produces:

$\langle M_0,\alpha\rangle \models \Box[(t_3) \Rightarrow \Diamond(t_{16})]$

4. The firing of transition t_{16} will yield a new sub-marking, this is shown by:

i) $\langle M_0,\alpha\rangle \models \Box [t_{16} \Rightarrow (p_6, p_{14}, p_{24}, p_{26})]$

ii) Let $M_2 = \{ p_6, p_{14}, p_{24}, p_{26} \}$, 4.i) can then be re-written:

$\langle M_0,\alpha\rangle \models \Box [t_{16} \Rightarrow M_2]$

iii) combining 4.ii) with 3.ii) and 2.i) yields:

$\langle M_0,\alpha\rangle \models \Box[M_1 \Rightarrow \Diamond M_2]$

iv) by combining 4.iii) with 1. yields:

$\langle M_0,\alpha\rangle \models \Box [(t_3) \Rightarrow \Diamond M_2]$

5. By an observation of the Petri net of Fig.6.5. and its reachability graph (shown in Appendix A.5.3.), it can be seen that marking M_2 enables the firing of t_{17} this is shown by:

i) $\langle M_0,\alpha\rangle \models \Box[M_2 \Rightarrow t_{17}(ok)]$

ii) Using proposition 1 the firing of t_{17} yields:

$\langle M_0,\alpha\rangle \models \Box [t_{17}(ok) \Rightarrow \Diamond t_{17}]$

iii) the firing of t_{17} will yield a new sub-marking, this is shown by:

$\langle M_0,\alpha\rangle \models \Box [t_{17} \Rightarrow O(p_6, p_{14}, p_{23}, p_{25})]$

iv) This submarking enables t_4, hence:

$\langle M_0,\alpha\rangle \models \Box [(p_6, p_{14}, p_{23}, p_{25}) \Rightarrow t_4(ok)]$

v) using proposition 1 on the firing of t_4, produces:

$\langle M_0,\alpha\rangle \models \Box [t_4(ok) \Rightarrow \Diamond t_4]$

vi) by combining 5.i) - iii) produces:

$\langle M_0,\alpha\rangle \models \Box [t_{17} \Rightarrow \Diamond t_4]$

6. Combining 5.i) - 5.vi) yields:

$\langle M_0,\alpha\rangle \models \Box [M_2 \Rightarrow \Diamond t_4]$

7. Combining 6 with 4.iv) produces:

$\langle M_0,\alpha\rangle \models \Box [t_3 \Rightarrow \Diamond t_4]$ ■

6.2.5. Discussion

The verification of properties of the distributed controller's Petri net has shown that it does not have reachable states that are undesirable and that it conforms to a set of temporal constraints. This approach specified the system using standard Petri net and temporal logic in a unified manner, using the formal framework of Temporal Petri nets. The application of Temporal Petri net analysis can show that the control logic satisfies the properties of the system stated in section 6.2.3. and does not cause any hazardous behaviour of the mechanism. The safety properties of the system have been verified using the reachability graph and concurrency sets. While, temporal reasoning was used to verify liveness properties of the system.

The design of the distributed controller using commit blocks was achieved in a hybrid manner, and properties of the commit blocks proven in Chapter 5 were used to verify that the distributed controllers always reached a consistent decision. The underlying Petri nets of the commit blocks, used in the distributed controller, were shown to provide the type fault tolerant and responsive coordination necessary. Analysis of the distributed controller was based on generating its reachable state space, for this atomic transitions that offer equivalent behaviour of the commit protocols underlying the commit blocks were used.

The commit protocols developed in chapter 4 are resilient to single communication link failures, and also allow the remaining operational controllers to continue to function safely when a site failure of one of the controllers occurs. These properties of the commit protocols were shown in Chapters 4. and 5. using reachability based Petri net analysis. The commit blocks developed in Chapter 5 use the underlying Petri net templates of these commit protocols, and the composition procedure used in the design of the distributed controllers preserved the properties of these commit blocks. The resulting distributed controller designs are thus resilient to a single link failure and to site failure, allowing the remaining operational controllers to continue to function safely without loss of coordination.

6.3. Drums and Slider Application

The second application is for the control of a Drum and Slider mechanism that was used in the research work of [Sagoo & Holding 90][Jiang 95] and [Azzopardi 96]. This application is a time-critical real-time control system and will be used illustrates the timeout mechanism within the commit block's underlying protocol. The commit protocols used to coordinate the distributed controller for this system will be required to provide a real-time

response, as a late commit/abort decision (or an inconsistent decision) could result in damage to the mechanism.

The coordination of the drum and slider mechanism uses the commit protocols with timeouts and synchronous communication, developed in Chapter 4 & 5. These provide a timed atomic commitment, that is resilient to link failures. The use of the previously analysed commit blocks to prevent hazardous behaviour, shows how a link failure of the distributed controller can be tolerated. This property of the commit protocols was shown in Chapter 5 by analysing their behaviour with communication failures included in the model.

In Chapter 4 Time Petri net analysis [Merlin & Farber 76] was used to define constraints on the commit protocols, this was required to ensure their correct functionality and timeliness. The relative settings of the protocol's timeouts were defined in terms of a deadline for the commit decision. The Petri net of the distributed controller, developed in this section, is extended to a Time Petri net model in section 6.3.5. This is then used to define constraints on the commit blocks response, this defined deadline needs to be met for the controller to maintain coordination of the system.

6.3.1 Application system

This application involves controlling part of a high speed manufacturing plant used for can packaging, and is based on a mechanism made by Molins Engineering PLC of Coventry in the UK [Sagoo 90]. Specifically, it involves the distributed control of a slider and two drum mechanisms and their interaction. The slider is a moving arm which periodically inserts into an aperture located on the periphery of both of the high speed rotating drums, when they are aligned. The desired operation of the system is that the rotating drums should decelerate from rotation and stop, and that the slider is inserted while the drums are stationary. The slider is then retracted from the drums, which proceeds to accelerate and rotate again.

Originally the individual actions of the drums/slider mechanism were co-ordinated by gears and cams powered by a central driving mechanism. It was proposed to replace the central drive with a set of independent software controlled drives co-placed with each of the functional units, while the coordination previously supplied by mechanical means was to be replaced by software controllers and inter-process communications.

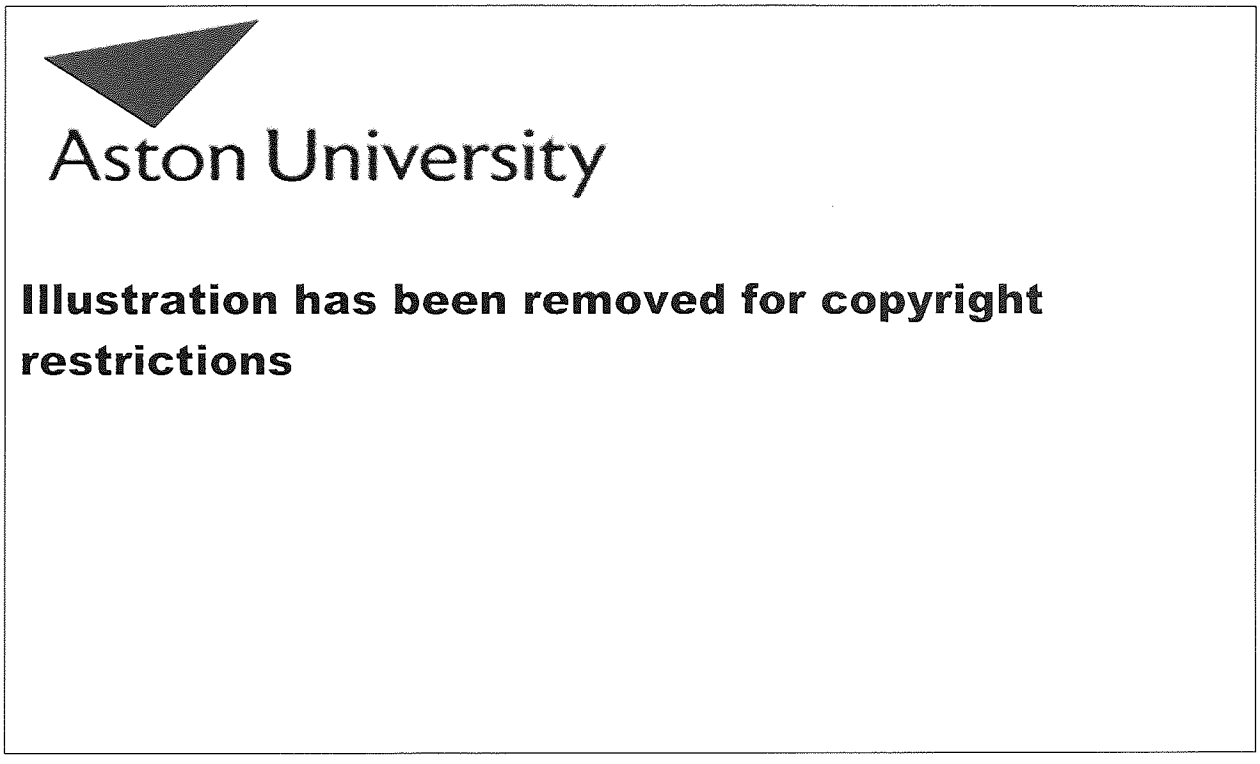


Fig.6.6. The Drums and Slider mechanism.

The plant involves the coordination of machinery which rotates at high speeds, and both the slider and the drums have significant inertia. The side view of the mechanism is shown in Fig.6.6. and the motion profile of the slider mechanism is shown in Fig.6.7.

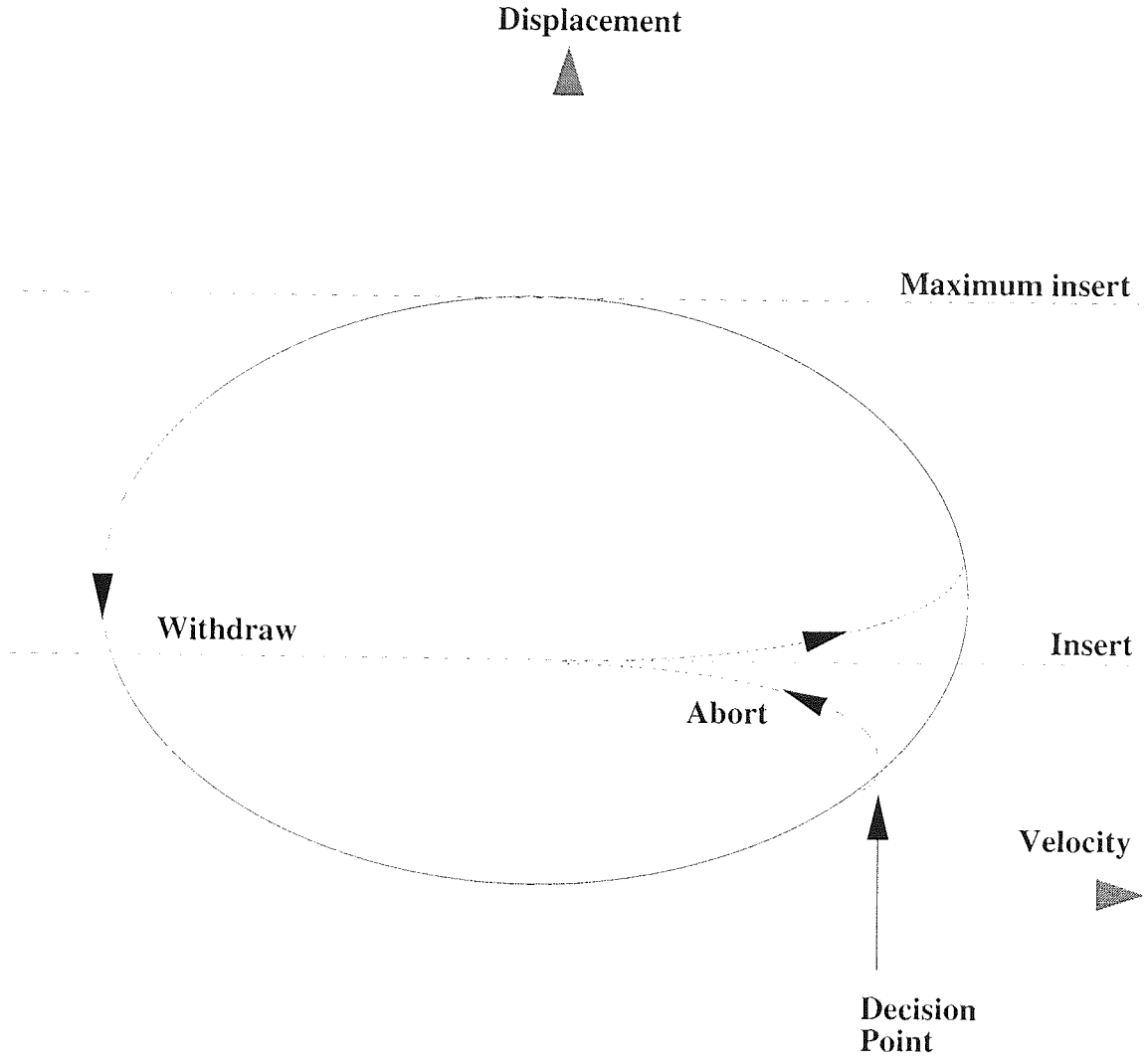


Fig.6.7. Motion profile and time-critical decision of the slider mechanism

The main hazardous behaviour of the system would be that of a collision between the slider and either of the drums. This could be caused by the drums accelerating from rest while the slider is still within their periphery, or the slider mechanism inserting while the drums are still rotating. Consequently, the controller design must ensure this behaviour does not arise.

A simple approach to avoiding a collision between the slider and drums would be to prohibit their asynchronous motion. A controller that enforced such behaviour would ensure that the slider does not insert until both drums are stationary and aligned, and that the drums do not start their rotation until the slider mechanism is withdrawn from the apertures, this would avoid a collision of drum or slider. However, there are periods during the motions of both the slider mechanism and the drums at which their concurrent motion is possible without such a collision occurring. To increase the operation speed of the system this concurrent motion must be employed whenever possible, but without compromising the hazard free operation of the mechanism.

The design of the controller should allow the drums and slider mechanism to move as freely as possible within the specified safety constraints. The slider's motion is initiated irrespective of whether the drums are rotating or stationary. The slider is allowed to approach the drums for insertion asynchronously and only at an advanced point in its motion (termed the *decision point* and shown in Fig.6.7.) must a (time-critical) decision be made for the insertion to proceed (*commit*) or for the slider to decelerate and stop (*abort*). A mechanism that provides the required consistent and timely decisions, to implement the synchronisation logic, at this point is provided by the commit block.

The decision point in the slider mechanisms motion profile (Fig.6.7.) corresponds to the point marked (2) in the side view of the mechanism shown in Fig.6.6. The overriding factor in the design of the distributed controller must be to ensure a collision free behaviour, while providing a high speed of operation of the system. In such a design the commit motion of the slider is the preferred behaviour, and the abort decision should be infrequent.

The distributed controller's synchronisation logic is required to allow the free motion of slider and drums around virtually all of their respective motion profiles. The only restrictions apply during the critical stage in their coordination when they have the potential to collide. This is indicated in Fig.6.6, the point marked (1) indicates the position where the slider mechanism begins its insertion motion, point (2) indicates the point at which the time-critical decision to commit or abort the sliders insert action must be complete, and point (3) the maximum insertion during the motion profile of the slider mechanism, Fig.6.7. The commit protocol employed in the commit block must allow the controller of the slider mechanism to make a delay-free decision to either commit or abort.

6.3.2. Modelled System

The hybrid design approach used in section 6.2 is used in the design of the distributed controller for the drum and slider mechanism. The *local* control for the drives co-placed with each unit of the mechanism (slider or drums) are designed from the simple functional requirements of each unit, and are shown in Fig.6.8.(a) - (c). The points at which coordination of these local controllers is required, are indicated by the commit transitions { t_5, t_6 }, shown in outline in Fig.6.8. These points in the control cycle represent the stage in which the processes must reach a consistent decision to *commit the insert action* of the slider (drum transfer mechanism), and the *commit to rotate action* of the drums.

The distributed controller is then composed from these local controllers (Fig.6.8.(a) - (c)) by the fusion of these commit transitions, and the commit transitions subsequent refinement using commit blocks.

The coordinator for the commit protocol is co-placed with the drum transfer controller, as this is the process which is required to abort or proceed with the sliders insert action, depending upon the (*commit* or *abort*) outcome of the commit protocol. The drum transfer controller is shown in Fig.6.8.(c), where place p8 represents the controller state at which a time-critical decision is required, and transition t8 represents the *abort insert* action and t7 the *insert* action.

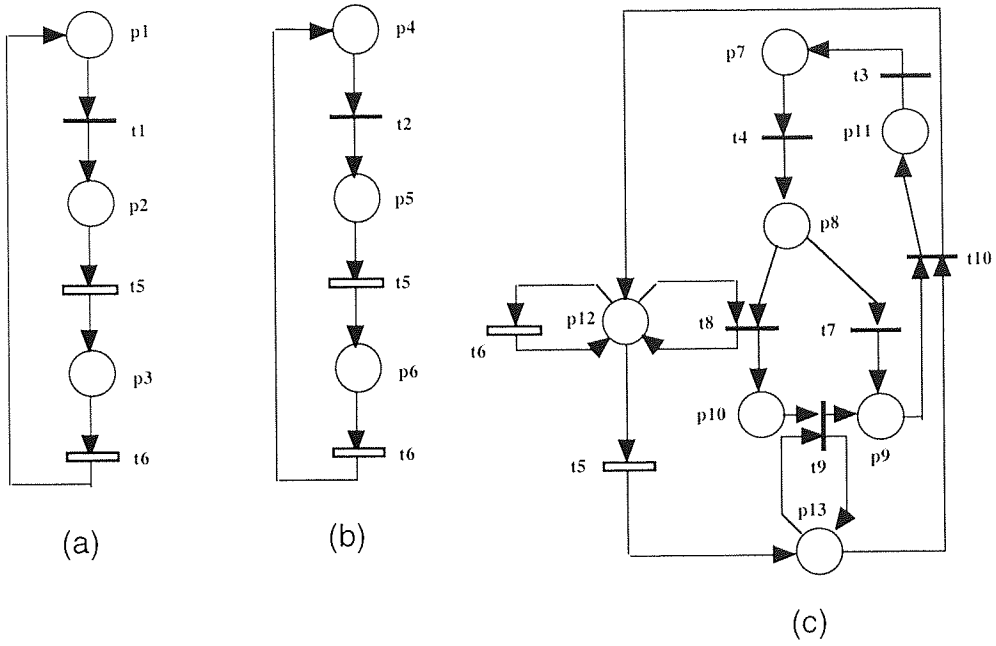


Fig.6.8. Drum and slider mechanism, individual controllers model; (a) Drum1, (b) Drum2, (c) Drum transfer.

The semantics for the places in the controllers designs are given in Table 6.4, with the required synchronisation logic indicated (*). The semantics for the commit decisions, and thus the commit blocks *commit* outcome are given in Table 6.5, along with the assignment of each process in the commit blocks underlying protocol ; coordinator (Coord), participant-1 (Part1) and participant-2 (Part2).

Process	State	Semantic
Drum 1	p1	Drum1 rotate
	p2	Drum1 await transfer
	p3	Drum1 committed to drum transfer *
Drum 2	p4	Drum2 rotate
	p5	Drum2 await transfer
	p6	Drum2 committed to drum transfer *
Drum Transfer	p7	Slider approach motion
	p8	Slider decision point
	p9	Slider insert motion
	p10	Slider abort motion
	p11	Slider withdraw motion
	p12	Slider insert inhibit *
	p13	Slider insert active *

Table 6.4. Semantic for 6-Axis Petri net, Fig.6.8.

Transition	Processes (Coord, Part1, Part2)	Commit decision
t5	Drum Transfer, Drum 1, Drum 2	Commit to slider insert
t6	Drum Transfer, Drum 1, Drum 2	Commit to drum rotate

Table 6.5. Semantic of Commit transitions for drum transfer mechanism, Fig.6.4.

The composed distributed controller design is shown in Fig.6.9., where the commit blocks are shown in outline. The distributed controller Petri net was composed form the three local controller Petri nets (shown in Fig.6.8.(a) - (c)) by the fusion of the commit transitions (t_5 and t_6). This was achieved by replacing the two sets of commit transitions (those marked t_5 and t_6 in Fig.6.8.(a) - (c)) by two commit blocks (marked t_5 and t_6 in Fig.6.9.). The input and output places of the commit transitions are merged with the commit blocks input and output places, on a per process basis; i.e. for commit block t_5, Drum1 is assigned the role participant-1 in the commit protocol, so place p_2 (in Fig.6.8.(a)) is merged with the participant-1 input place to the commit block (place p_5 in Fig.5.18). This procedure was described in section 6.2.

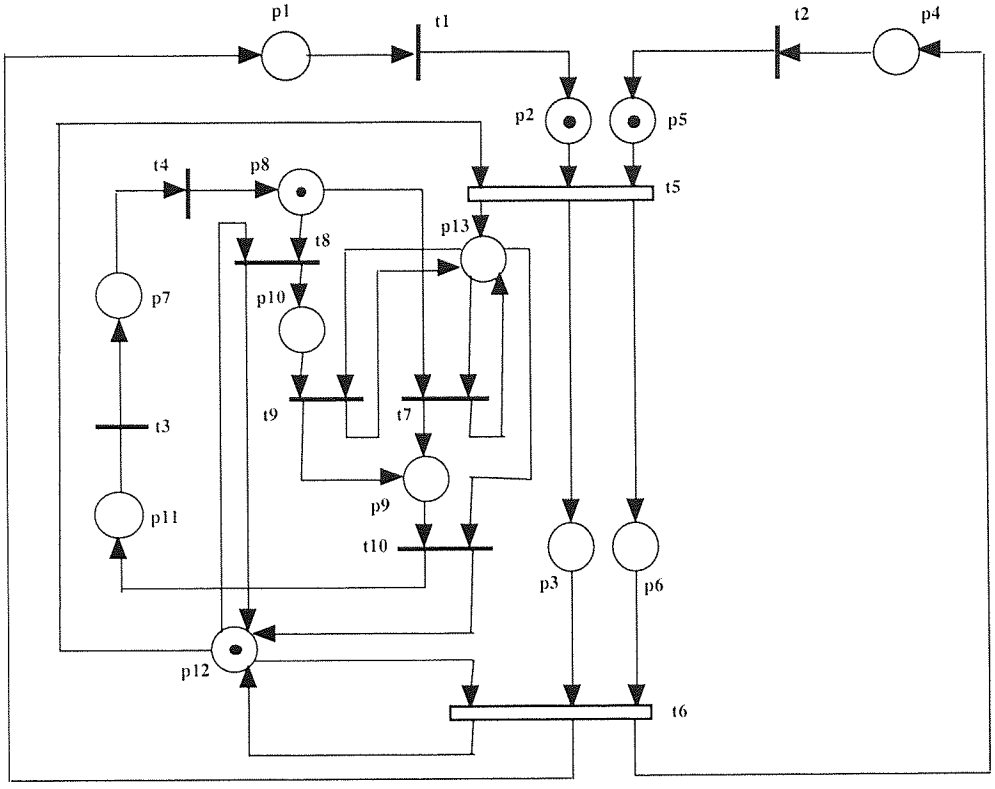


Fig.6.9. Distributed controller design.

The drum transfer process is assigned the role of coordinator in the (t5) commit blocks underlying commit protocol, to coordinate the insert *commit/abort* decision. This assignment is chosen as the initiation of the sliders withdrawal (represented by t10 in Fig.6.9.), by the drum transfer process, can also be used to initiate the commit protocol. The commit block CB2, whose behaviour was described in section 5.2.5, can be used for the (t5) commit block, as this uses atomic feedthrough of coordinator input place tokens. This type of commit block uses place-sampling of the coordinators input place token, this allows place p12 (in Fig.6.8.) to remain tokenised and thus the abort decision (represented by the firing of t8) is possible until the commit protocol completes. Due to the controller structure the locking of the drum processes input place tokens (by commit block t5) is possible without altering the behaviour of the net, while place sampling of the slider controllers input place token is required to maintain a non-hazardous sequence of states. In Section 6.3.5. the constraints necessary for the locking of all input place tokens using commit block CB1 is described.

In Chapter 5 it was shown that the commit protocols used have the timed atomic commit behaviour required for this application. The input and output places of the commit blocks are merged with the input and output places of the commit transitions in the local controllers (shown in Fig.6.8.(a) - (c)), these are shown along with the outcomes of the commit blocks in Table 6.6.

Commit block outcome	Input set	Output set
t_5 commit	p2, p5, p12	p3, p6, p13
t_5 abort	p2, p5, p12	p2, p5, p12
t_6 commit	p3, p6, p12	p1 p4, p12
t_6 abort	p3, p6, p12	p3, p6, p12

Table 6.6. external behaviour of the commit blocks { t_5, t_6 }, from Fig.6.9.

As can be seen in Table 6.6 the abort outcome of the commit protocol requires the re-tokenising of the commit block's input places. The functional and temporal properties for the commit blocks, as verified in chapter 5, will be used in stating and verifying the properties of the distributed controller in the following sub-sections. The commit blocks complete external behaviour was used in the generation of the reachable space for the resulting Petri net design.

The *abort* outcome of the commit block could be the result of a permanent or transient communication failure, or as a result of one of the protocol processes taking the decision to unilaterally abort. The protocols used in the commit blocks were analysed with the inclusion of link failures in Chapter 4 and 5, and shown to exhibit the external behaviour given in Table 6.6. This has guaranteed that the outcome of the commit blocks are always a consistent decision. The resiliency of the commit protocols to communication failures is important for the fault-tolerant coordination of the drum and slider mechanism. These type of failures are expected to predominate in real-time control applications, and so the fault-tolerant property of the commit protocol is necessary to prevent a collision, of the drum and slider mechanism, when a communication failure occurs.

In the case where the controlled process (the drum and slider mechanism) is not in the correct state or the controller cannot guarantee the resources necessary to complete the protocol in time, the decision is taken by the controller to abort the protocol. The timeouts used by the other controllers in the protocol to reach a decision consistent with this, also guarantee a fixed maximum deadline for the commit blocks outcome. The timeout to abort outcome allows for a safe decision to be reached by the controllers, while the re-tokenising of the commit blocks input places allows the commit protocol to be retried.

6.3.3. Properties of the System

The specification of the controller must be shown to be correct, by ensuring that certain liveness and safety properties (which are specified in the system requirements) are satisfied. The following safety and liveness properties for the synchronisation logic are examples of the properties required for the distributed controller. The first five properties of the controller

refer to behaviour which must not occur, and as such constitute safety properties, while the last three properties, termed liveness properties, and refer to the behaviour the controller should exhibit:

Safety Property 1 A situation will *never occur* in which the drum transfer controller *commits* to insert when either drum1 or drum2 is rotating, this can be expressed as invariance in temporal formula as:

$$\Box\neg[\,(p_{13} \wedge \neg\, p_3) \;\vee\; (p_{13} \wedge \neg\, p_6)\,] \tag{6.16}$$

Safety Property 2 A situation will *never occur* in which the drum transfer slider inserts when either drum1 or drum2 is rotating, this can be expressed as invariance in temporal formula as:

$$\Box\neg[\,(p_9 \wedge \neg\, p_3) \;\vee\; (p_9 \wedge \neg\, p_6)\,] \tag{6.17}$$

Safety Property 3 A situation will *never occur* in which the controller *commits* to drum rotate when the slider mechanism is inserting, this can be expressed as invariance in temporal formula as:

$$\Box\neg[(\,p_9\,) \wedge (\,p_{12}\,)] \tag{6.18}$$

Safety Property 4 A situation will *never occur* in which the slider is at the decision point and no decision can be made, this can be expressed as invariance in temporal formula as:

$$\Box\neg[\,p_8 \;\wedge (\neg\, p_{12} \wedge \neg\, p_{13}\,)] \tag{6.19}$$

Safety Property 5 The independent motion of the drums is permitted, and their *rotating* states are not mutually exclusive, this can be expressed in temporal formula as:

$$\neg\Box\neg[(p_1) \wedge (p_4)] \tag{6.20}$$

Liveness Property 1 Whenever drum1, drum2 and drum_transfer processes reach a commit decision (slider insert), then eventually these processes will reach the commit decision (drum rotate). This can be expressed as eventuality in temporal formula as:

$$\Box[(\,t_5\,) \;\Rightarrow\; \Diamond(\,t_6\,)] \tag{6.21}$$

Liveness Property 2 Whenever the slider reaches the decision point, it will eventually insert into the drum due to a commit to slider insert decision being reached, this can be expressed as eventuality in temporal formula as:

$$\Box\Diamond(p8) \Rightarrow \Box\Diamond(p9) \tag{6.22}$$

Liveness Property 3 Whenever drum1 and drum2 start to rotate then eventually their rotation cycle will end. This can be expressed as eventuality in temporal formula as:

$$\Box[(t6) \Rightarrow \Diamond(t2 \wedge t7)] \tag{6.23}$$

6.3.4. Verification of System Properties

The controller design (Fig.6.9.) has been shown to exhibit certain properties, such as liveness, 1-boundedness and freedom from deadlock using Petri net analysis (given in Appendix.A.6). The reachable state space of the controller model and the concurrency sets are shown below in Table 6.7 and Table 6.8, and will be used in the following proofs.

Marking	Places	Marking	Places
M_1	p2, p5, p8, p12	M_{14}	p1 p5 p7 p12
M_2	p2 p5 p10 p12	M_{15}	p2 p5 p7 p12
M_3	p3 p6 p10 p13	M_{16}	p3 p6 p7 p13
M_4	p3 p6 p9 p13	M_{17}	p3 p6 p8 p13
M_5	p3 p6 p11 p12	M_{18}	p2 p4 p7 p12
M_6	p1 p4 p11 p12	M_{19}	p1 p5 p11 p12
M_7	p1 p4 p7 p12	M_{20}	p2 p5 p11 p12
M_8	p1 p4 p8 p12	M_{21}	p3 p6 p11 p13
M_9	p1 p4 p10 p12	M_{22}	p2 p4 p11 p12
M_{10}	p1 p5 p10 p12	M_{23}	p3 p6 p7 p12
M_{11}	p2 p4 p10 p12	M_{24}	p3 p6 p8 p12
M_{12}	p1 p5 p8 p12	M_{25}	p3 p6 p10 p12
M_{13}	p2 p4 p8 p12		

Table.6.7 reachable markings from the Petri net Fig.6.9.

State	Concurrency Set
p1	Places { 4, 5, 7, 8, 10, 11, 12 }
p2	Places { 4, 5, 7, 8, 10, 11, 12 }
p3	Places { 6, 7, 8, 9, 10, 11, 12, 13 }
p4	Places { 1, 2, 7, 8, 10, 11, 12 }
p5	Places { 1, 2, 7, 8, 10, 11, 12 }
p6	Places { 3, 7, 8, 9, 10, 11, 12, 13 }
p7	Places { 1, 2, 3, 4, 5, 6, 12, 13 }
p8	Places { 1, 2, 3, 4, 5, 6, 12, 13 }
p9	Places { 3, 6, 13 }
p10	Places { 1, 2, 3, 4, 5, 6, 12, 13 }
p11	Places { 1, 2, 3, 4, 5, 6, 12, 13 }
p12	Places { 1, 2, 3, 4, 5, 6, 7, 8, 10, 11 }
p13	Places { 3, 6, 7, 8, 9, 10, 11 }

Table 6.8 concurrency sets for drum transfer mechanism, Fig.6.9.

In order to allow maximum freedom of motion for the mechanism, the controller allows the slider to begin insertion before the commit decision is reached. This decision to commit or abort the insertion must be made by a point in the slider motion termed the decision point. This decision point is situated just before the slider is irrevocably committed to inserting into the drum by its inertia. The decision point is marked as point (2) in Fig.6.6. and indicated on the slider's motion profile shown in Fig.6.7.

At this point in its motion the slider can still be brought to rest before it inserts into the drums. Safety property 4 stipulates that at its decision point the controller must always be able to decide upon the actions *slider commit* or *slider abort* at the next step.

Proof of safety property 1, (6.16) $\Box\neg[(p_{13} \wedge \neg p_3) \vee (p_{13} \wedge \neg p_6)]$

To verify this property it is necessary to show that whenever p_{13} is marked then both p_3 and p_6 are also marked. From inspection of the concurrency set for p_{13} (shown in Table 6.8) it can be seen that states p_3 and p_6 are concurrent states at some markings.

i) $C(p_{13}) = \{ p_3, p_6, p_7, p_8, p_9, p_{10}, p_{11} \}$

The loop $C = \{ p_1, t_1, p_2, t_5, p_3, t_6 \}$ shown in Fig.6.8.(a). has the property that the number of tokens in C remains invariant during all executions of the net (given by the minimal P-invariants in Appendix.A.5.3.) This means that mutual exclusion between places in the loop is guaranteed. As places p_1 and p_2 are not present in the concurrency set for p_{13} then

whenever p_{13} is tokenised p_3 must also be tokenised. The same can also be shown for state p_6 by examination of the concurrency set for p_{13}, where it can be shown that states p_4 and p_5 are not present in the concurrency set of p_{13}.

Proof of safety property 3, (6.18) $\Box\neg[(p_9) \wedge (p_{12})]$

To verify this properties it is necessary to show that the *insert* state of the slider mechanism is mutually exclusive with the state where the controller has reached a commit decision (committed with other processes to the drum rotate action). This property can be verified by examination of the concurrency set for Fig.6.9. (shown in Table 6.8), from where it must be shown that states *Slider insert motion*(p_9) and *Slider insert inhibit*(p_{12}) are not present in each others concurrency sets. This can be verified by inspection of the following concurrency sets from Table.6.8

i) $C(p_9) = \{ p_3, p_6, p_{13} \}$

ii) $C(p_{12}) = \{ p_1, p_2, p_3, p_4, p_5\ p_6, p_7, p_8, p_{10}, p_{11} \}$

The liveness properties will be verified using the Temporal Petri net analysis from section 6.2.4.

Proof of liveness property 1, (6.21) $\Box[(t_5) \Rightarrow \Diamond(t_6)]$

Let TN_1 be a temporal Petri net translated from the Petri net PN shown in Fig.6.9., with an initial marking $M_0 = \{ p_2, p_5, p_8, p_{12} \}$.

This property can be formalised as:

$$<M_0,\alpha> \models \Box[(t_5) \Rightarrow \Diamond(t_6)]$$

Proof

1. From M_0 and Fig.6.9., it can be observed that transitions t_5 and t_8 are fireable. Hence

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow t_5(ok) \wedge t_8(ok)]$$

2. By applying proposition 2, transitions t_5 or t_8 can fire, hence

$$<M_0,\alpha> \models \Box [t_5(ok) \wedge t_8(ok) \Rightarrow \Diamond(t_5 \vee t_8)]$$

3. Since t_5 and t_8 are in conflict, the firing of each transition will be considered.

(a) The firing of t_5 produces marking $M_{17} = \{ p_3, p_6, p_8, p_{13} \}$, (shown in Table 6.7). At this marking t_7 is fireable:

i) $<M_0,\alpha> \models \Box [t_5 \Rightarrow Ot_7(ok)]$

ii) By applying proposition 1 on 3(i), we have:

$<M_0,\alpha> \models \Box [t_7(ok) \Rightarrow \Diamond t_7]$

iii) The firing of t_7 enables t_{10}, hence:

$<M_0,\alpha> \models \Box [t_7 \Rightarrow O t_{10}(ok)]$

iv) By applying proposition 1 on 3(iii), we have:

$<M_0,\alpha> \models \Box [t_{10}(ok) \Rightarrow \Diamond t_{10}]$

v) The firing of t_{10} will enable transitions t_3 or t_6. Since these transitions can fire independently, hence the firing of t_6 will be considered:

$<M_0,\alpha> \models \Box [t_{10} \Rightarrow O t_6(ok)]$

vi) By applying proposition 1 on 3(v), we have:

$<M_0,\alpha> \models \Box [t_6(ok) \Rightarrow \Diamond t_6]$

vii) Combining 3(i) to 3(vi) we have that:

$<M_0,\alpha> \models \Box [t_5 \Rightarrow \Diamond t_6]$

---

viii) Considering the firing of t_3 from step 3(v) we have

$<M_0,\alpha> \models \Box [t_{10} \Rightarrow O t_3(ok)]$

ix) By applying proposition 1 on 3(v), we have

$<M_0,\alpha> \models \Box [t_3(ok) \Rightarrow \Diamond t_3]$

x) The firing of t_3 enables t_4, hence

$<M_0,\alpha> \models \Box [t_3 \Rightarrow O t_4(ok)]$

xi) By applying proposition 1 on 3(v), we have

$<M_0,\alpha> \models \Box [t_4(ok) \Rightarrow \Diamond t_4]$

xii) The firing of t_4 produces marking $M_{24} = \{ p_3, p_6, p_8, p_{12} \}$, (shown in Table 6.7) at this marking t_8 is fireable:

$<M_0,\alpha> \models \Box [t_4 \Rightarrow O t_8(ok)]$

xiii) By applying proposition 1 on 3(xii), we have

$<M_0,\alpha> \models \Box [t_8(ok) \Rightarrow \Diamond t_8]$

(b) The firing of t_8 produces marking $M_{25} = \{ p_3, p_6, p_{10}, p_{12} \}$, at this marking only t_6 is enabled:

xiv) $<M_0,\alpha> \models \Box [t_8 \Rightarrow O t_6(ok)]$

xv) By applying proposition 1 on 3(xiv), produces:

$<M_0,\alpha> \models \Box [t_6(ok) \Rightarrow \Diamond t_6]$

xvi) Combining 3(viii) to 3(xv) we have that:

$<M_0,\alpha> \models \Box [t_{10} \Rightarrow \Diamond t_6]$

xvii) Combining 3(xvi) to (3(i) - 3(iv)) we have that:

$<M_0,\alpha> \models \Box [t_5 \Rightarrow \Diamond t_6]$ ■

6.3.5. Time Analysis of System

Although the safety properties above guarantee that the controller avoids hazardous system states, such as a collision of the slider mechanism and the drums, it is still undesirable behaviour in failure free operation for an abort decision to be reached. The insert action performs the intended function of the mechanism, transferring an item from drum1 to drum2. While the abort action is a safety feature that prevents damage to the mechanism.

In the case of a communication failure the abort behaviour is unavoidable and increases the robustness of the controller. However, the deadline for the protocol, constrained by its timeout settings, should allow for a 'commit to insert' decision to be reached whenever possible.

The slider controller is always able to make a (commit or abort) decision due to either p_{12} or p_{13} being marked. This was verified by the proof for safety property 4 (6.19). The place p_{12} is the coordinator's input place for the commit block of t_5. Where commit block CB2 is used for commit transition t_5, the coordinators input place tokens are not absorbed. Therefore the (commit/abort) decision is not prevented by a late outcome for the commit block, as this commit block uses atomic feedthrough of the coordinators input place tokens. A late outcome for this commit block may result in an abort decision, but would not prevent a late decision being reached.

Where a commit block such as CB1 is used (where the coordinators input place (p_{12}) tokens are locked), the setting of the timeouts for the protocol are critical. These timeouts determine the maximum time for the protocol to complete. On completion a token is placed in either p_{12} or p_{13}, depending upon the protocols outcome. The proof for Safety property 4 (6.19) showed that a token in one of these two places is required for reaching a safe decision (section 6.3.4). Therefore the deadline for the completion of the commit protocol is critical to reaching a safe decision.

Timing constraints can be added to the Petri net in the form of minimum and maximum transition firing times [Merlin & Farber 76]. These are termed EFT (Earliest Firing Time) and LFT (Latest Firing Time). These represent the upper and lower bounds on event times, and for an enabled transition the times can be converted to absolute times by adding the time the transition is enabled to the firing times.

At marking $M_4 = \{ p_3, p_6, p_9, p_{13} \}$ of Fig.6.9, the slider must next withdraw from the drums. From this marking there is a firing sequence $<C>$ that leads to the slider inserting (t_7 firing) and a firing sequence $<A>$ that leads to the slider aborting (t_8 firing). For an insert to

be possible, the sum of EFTs for C must be less than the sum of LFTs for A, giving the timing constraint:

$$EFT(C) \leq LFT(A) \tag{6.24}$$

From marking $M_4 = \{ p_3, p_6, p_9, p_{13} \}$ the maximum time taken for the abort firing sequence is:

$$LFT(t_{10}) + LFT(t_3) + LFT(t_4) + LFT(t_8) = LFT(A)$$

This represents the maximum time taken to withdraw the slider mechanism from the point where it is clear of the drums to its starting point, to accelerate the slider towards the drums until it reaches the decision point, and to then abort its motion.

From marking $M_4 = \{ p_3, p_6, p_9, p_{13} \}$ the minimum time taken for the commit firing sequence is:

$$EFT(t_6) + MAX[\, EFT(t_1), EFT(t_2) \,] + EFT(t_5) + EFT(t_7) = EFT(C)$$

This represents the maximum time taken to withdraw the slider mechanism from the point where it is clear of the drums to its starting point, to accelerate the slider towards the drums until it reaches the decision point, and to then commit to insert.

Where $EFT(t_6)$ and $EFT(t_5)$ are the earliest times that the commit protocols used can complete, and $MAX[\, EFT(t_1), EFT(t_2) \,]$ is the maximum of the shortest times taken for the rotation of drum1 and drum2.

Timing constraint (6.24) must be guaranteed for a *slider commit* decision to be possible using a commit block of type CB2 at transition t_5. This gives the minimum duration for the commit block to reach a consistent decision. As there is no locking of the coordinator input place token (p_{12}) a *slider abort* decision is always possible. Thus the commit blocks duration does not require a hard real-time constraint for safe operation, as the firing of t_8 before t_5 does not result in a hazardous state.

Where a commit block of type CB1 is used then timing constraints must be guaranteed in order for a safe decision to be reached, even in the presence of a communication failure. A consistent and timely decision (either a commit block *abort* or *commit* outcome), is thus required to implement the synchronisation logic at this point. Since the slider is in motion when the decision is taken, a late decision could have as catastrophic a result as an incorrect decision. A timed atomic commit (TAC) behaviour for the commit blocks is thus required. The timing constraint can be expressed as:

$$LFT(C) \leq EFT(A) \tag{6.25}$$

Where LFT (C) is the total latest firing time for the commit firing sequence, and EFT (A) the earliest possible firing time for the abort firing sequence. From marking M_4 = { p_3, p_6, p_9, p_{13} } the minimum time taken for the abort firing sequence is:

$$EFT(t_{10}) + EFT(t_3) + EFT(t_4) + EFT(t_8) = EFT\ (A)$$

While the maximum time taken for the commit firing sequence is:

$$LFT(t_6) + MAX[\ LFT(t_1), LFT(t_2)\ ] + LFT(t_5) + LFT(t_7) = LFT\ (C)$$

The time $LFT(t_5)$ is the maximum time that the commit protocol can take to complete in order to guarantee a timely decision. The commit block outcome could be either a commit or abort decision for the slider. Therefore this constraint can be related directly to the maximum time for protocol completion D_d, used in section 4.6. for the calculation of E2PC protocols timeout settings. By setting the commit block timeouts relative to D_d, the protocol can be tailored to application specific constraints.

The value of the protocols timeouts must be set carefully so as to avoid false timeouts, which are due to not allowing enough time for communication to occur. However, timeout values must not be too large as undue delays are unacceptable and may upset the timing requirements of other parts of the system. It must be ensured that there are no redundant timeouts, and timeouts that depend on others. These factors were taken into account in the design of the commit protocols in chapter 4.

The environment of the distributed controllers are considered to be the motion drives upon which they sit. One factor to consider in such an arrangement is that every action initiated by the process is translated into a command to the motion controller and each of these commands takes time to be implemented. This must be taken into account when determining the upper and lower bounds on event times. For instance the EFT and LFT for t_3 represent the minimum and maximum time taken to withdraw the slider mechanism from the point where it is clear of the drums to its starting point, and then to initiate the sliders insert motion.

The operation of commit blocks such as CB2 using atomic feedthrough of input place tokens, can be considered an atomic action in terms of the Petri net of the controller. However, for the use of a commit block such as CB1 using locking of input place tokens, time constraints must be considered. Where the timeout settings of the commit block can guarantee timing constraint (6.25), then commit block t5 can be considered an atomic action in terms of the Petri net of the controller. The actions external to the commit protocol can be considered to occur before or after the protocol takes place.

6.4. Conclusion

This chapter has described the use of Petri net, Time Petri net and Temporal Petri net techniques in the specification and verification of distributed controllers for two real-time distributed systems. The applications chosen include both time-critical and safety-critical functions, and are representative of the type of systems commonly found in the control of manufacturing machinery. It has been shown that the above techniques can be used to reason about safety and liveness properties of the systems and to verify the defined hazard free operation of the controllers. The distributed controller designs are optimal control strategies as they permit the maximum asynchronous operation of the independent drives possible, whilst maintaining safe behaviour of the controlled system; i.e. the drums and slider mechanism move as freely as possible within the constraints of the specification.

The distributed controllers are intended to be implemented using local control processes for each independent software controlled drive, these being co-placed with each functional unit (the drums and slider mechanism). The control of the two application considered have three main properties which were used in the choice of the specification and analysis methods used:

i) the system is distributed, with concurrent actions whose operation is co-ordinated by message passing communications. Petri nets and Temporal Petri nets provides a framework around which such a system can be specified, modelled and analysed.

ii) the system must operate within time constraints, and therefore the distributed controller design requires a model which is able to capture and reason about time.

iii) the communications involved in the distributed controllers operation are concerned with the flow of control information rather than the transmission of data. Because of this such communications can be expressed easily in terms of synchronous events employed in the underlying Petri net templates of the commit blocks . These lead to a unified representation of the distributed controller system and the commit protocol used for coordination.

Analysis of the Petri net description of a controller design showed that it does not have any reachable hazardous states. The desired properties of the distributed controller were specified properties using the notation of Temporal Petri nets, and the verification of these properties using concurrency sets and Temporal Petri net analysis was presented. The application of a consistent temporal logic proof showed that the control logic satisfied its specification and did not cause hazardous operation. The use of Time Petri net allowed reasoning about the relative timing constraints for the commit protocol inherent within the controller design. The use of these modelling and analysis methods provided a measure of fault avoidance in the controller design.

The distributed control of the two applications considered has fault tolerant coordination in the presence of communication failure, through the use of the commit protocols developed in Chapters 4 & 5. However, the use of these protocols would depend upon an implementation of the controller using point-to-point synchronous communication, as the responsive protocols designs rely upon this form of communication, and accurate information on the timing of communications, in order to calculate timeout values. This requirement for synchronous point-to-point communication is a limiting factor for the use of these responsive protocol.

The use of the commit blocks allowed the properties proven for these structures, such as their consistent outcome and recoverability, to be inherited in the controller design. The hybrid design of the distributed controllers using commit blocks was shown to satisfy the requirements of the synchronisation logic.

Analysis of the Petri net model was only able to determine the untimed behaviour of the system, but as a result of extending this to a Timed Petri net model (in section 6.3.5) it was possible to reason clearly about the relative times that give a constraint for the protocols completion. Thus in the second application, presented in section 6.3, after reaching the decision point (place p_8 in Fig.6.9.) the system must perform either a commit or an abort action within a time interval (D_d) where a protocol that locks input place tokens is used. This gives a constraint for setting the deadline of the commit protocol, to guarantee a timely decision is always reached.

The recovery time of sites cannot be calculated, and so only link failures are considered for these real-time applications. The behaviour of the commit protocols was considered for single link failures, as there does not exist any protocol resilient to multiple partitions [Skeen & Stonebaker 83], or where messages are lost. When there is a site failure of one of the parties to a commit block, the action of the other parties is the same as for a communication failure, this always results in a consistent abort decision being reached by the operational sites. The assumption is taken that upon recovery the failed site would initiate a safe restart for the whole system. This is a valid assumption as controller site failure are expected to be extremely rare when compared to communication failure rates.

The type of failures termed Byzantine failures, discussed in Chapters 2 and 4 are not considered in the application of these commit protocols, these failure types being where incorrect actions are performed by the failed item, and are very difficult to rectify, being similar to solving the Byzantine generals problem [Lamport et al 82]. Only *fail silent* type failures are considered, where failed sites do not perform an expected action.

In the failure analysis link failures are assumed to be a physical disconnection of a communication link between processors, and that messages are short so that disconnection cannot occur half way through a message. In real-time control applications this should be an accurate assumption. Tolerating link failures in control applications such as these is important because the processors can still control the motion drives independently, and maintain safe system behaviour in the presence of such failures.

The synthesis of the synchronisation logic that performs intermittent synchronisation of the asynchronous processes controlling these high speed drives is presented. This synchronisation logic incorporates real-time fault-tolerant commit protocols. The analysis of the distributed controller designs relied upon the inheritance of certain properties of the commit blocks used, the verification of these commit blocks and their underlying Petri net protocol models was performed in Chapter 5. The coordination of the distributed controllers has been shown to be preserved even in the presence of certain defined failures, through the use of these fault tolerant commit protocols.

The analysis of the distributed controller designs relied upon properties of the commit blocks used in the design. These properties were proven separately by the analysis and verification of the commit blocks and their underlying Petri net templates , and presented in Chapters 4 and 5. Thus the analysis of the Petri net controllers in this chapter is a modular approach, that allows the management of the Petri net models state space. Some form of modular analysis is desired as the size of the reachable state-space of the composed system, with the protocols state space fully enumerated, could become unmanageable for manual analysis. The commit block approach allows the modular specification and manageable analysis for controller model and protocol model sub-net. This is required as even using automated tools the state explosion can render reachability based analysis unmanageable for real-world systems [Zhou & DiCesare 93].

In this chapter the verification of the distributed controller designs for the two applications (section 6.2 & 6.3), has shown that they provides equivalent safety and liveness properties to the centralised controller designs presented in [Sagoo 92][Jiang 95] and [Azzopardi 96]. The use of the commit blocks in the controller designs, and the inheritance of their previously proven properties, allowed a reduction in the size of the state space enumerated for analysis.

Chapter 7 Conclusions and Future Research

7.1 Conclusions

This thesis has been concerned with the development of responsive commit protocols for distributed real-time control systems, and the incorporation of these in the synthesis and design of distributed controllers. It has also been concerned with the modelling and analysis of such systems and the incorporation of a structuring approach, using Well Formed Multi-Blocks (WFMB), that aids the design and verification processes.

This approach was based on the use of WFMBs, Petri nets, Time Petri nets and Temporal Petri nets. In this thesis the Temporal Petri nets of Suzuki & Lu [89] were used, based on linear time temporal logic (LTL). The Temporal Petri nets offer a combination of Petri nets (which provide a graphical notation) and Temporal logic (which provides a formal mathematical basis for verification). The Petri nets are used for the proof of safety properties and the Temporal Petri nets for the proof of liveness properties, which specify the eventual occurrence of a state or event using qualitative time. Temporal Petri nets, unlike Petri nets and Time Petri nets, provide a notation that allows fairness and eventuality properties to be specified.

The WFMB approach developed in Chapters 4, 5 and 6 handles the following characteristics, which make it suitable for application in distributed real-time control systems.
i) Concurrency. Distributed real-time control systems have many components that behave in a parallel manner, as well as the parallel nature of the systems interaction with its environment. Petri nets model concurrency and allow the explicit modelling of interprocess communication. This is especially important for the modelling and analysis of distributed systems, as discussed in Chapter 3 and illustrated in Chapter 4. Reachability trees of Petri nets show the global state of a system, *comprised* of the local states for each concurrent process, and concurrency sets can be used to show all concurrent states possible.

ii) Safety. The Petri net (reachability based) analysis and the Temporal Petri net analysis allow the verification of safety and liveness properties. This was shown for both the commit protocols developed in Chapters 4 and 5, and the distributed controller designs developed for the two (real world) applications in Chapter 6. The use of Temporal Petri net verification techniques developed by [Suzuki & Lu 89] [Sagoo 92] can verify that the required safety, and liveness properties hold.

iii) Timing. Time Petri nets can be incorporated in the Petri net designs, and were used for deriving relative values for the settings of timeouts in the responsive commit protocols in Chapter 4 & 5.

iv) Environment. The modelling and analysis of the communication structures (communication blocks - section 5.2) and the commit protocols (commit blocks - section 5.3) both used a minimal representation of the environment. The target environment for the block structured commit protocols developed in Chapter 5 are the distributed controller designs developed in Chapter 6. In fact the use of environment constraints is the key to the structuring of WFMBs and the reuse of the WFMBs subnets verification.

v) Size and complexity. This is a hybrid approach to design, involving both top-down decomposition and bottom-up clustering to handle complexity. WFMBs were used in the design of commit protocols in Chapter 5 and in the design of distributed controllers in Chapter 6 to manage the size of models. They were also used to reduce the size of the reachable state space considered in analysis of these designs. Using WFMBs the flat net (fully elaborated subnet) is not required for the generation of the block structured Petri nets behaviour. This is to help prevent the problem of state space explosion [Scholefield 90] as discussed in Chapter 3.

The basis of the approach to design and analysis presented in Chapters 5 & 6, was the use of WFMBs which offer a reduced representation of Petri net structures. These blocks allowed the reuse of formally proven Petri net templates of two basic structures:
i) interprocess communications. Synchronous point-to-point message passing was used in these applications, with timeout guards to avoid deadlock. These are termed communication blocks and were presented in section 5.2;

ii) commit protocols. These protocol models were based on those developed in Chapter 4, and their development was based on the use of the above communication blocks. These are termed commit blocks and were presented in section 5.3.

The distributed controllers are intended to be implemented using local control processes for each independent software controlled drive, these being co-placed with each functional unit (such as each drum and slider mechanisms in the second application). The two control applications considered have three main properties were used in the choice of the specification and analysis methods used:

i) the systems are physically distributed and involve concurrent actions whose operation must be co-ordinated. Petri nets and Temporal Petri nets provide a framework around which such a system can be specified, modelled and analysed.

ii) the system must operate within time constraints, and therefore the distributed controller design requires a modelling approach which is able to capture and reason about time.

iii) the communications involved in the operation of the distributed controllers are concerned with the flow of control information rather than the transmission of data. Because of this such communications can be expressed easily in terms of synchronous events, employed in the underlying (low level) Petri net templates of each commit block. This led to a unified representation of the distributed controller system and the commit protocol used for coordination.

The use of these modelling and analysis methods provided a measure of fault avoidance in the controller design. The distributed control of the two applications considered has fault tolerant coordination in the presence of communication failure, through the use of the commit protocols developed in Chapters 4 & 5.

The refinement of the local controllers using commit blocks, allowed the distributed controller designs to inherit the verified properties of the commit protocol designs, such as fault tolerance and timed atomic commitment. Using the timed Petri net model (in section 6.3.5) it was possible to reason clearly about the relative times that give constraints for the protocols required behaviour.

The behaviour of the commit protocols was considered for single link failures, as there does not exist any protocol resilient to multiple partitions [Skeen & Stonebaker 83], or where messages are lost. When there is a site failure of one of the parties to a commit block, the action of the other parties is the same as for a communication failure, this always results in a consistent abort decision being reached by the operational sites. The assumption is taken that upon recovery the failed site would initiate a safe restart for the whole system. This is a valid assumption as controller site failure is expected to be extremely rare when compared to communication failure rates.

The type of failures termed Byzantine failures, discussed in Chapters 2 and 4 are not considered in the application of these commit protocols, these failure types being where incorrect actions are performed by the failed item, and are very difficult to rectify, being similar to solving the Byzantine generals problem [Lamport et al 82]. Only fail silent type failures are considered, where failed sites do not perform an expected action.

The use of the responsive commit protocols developed in this thesis allowed the distributed controller designs to tolerate communications failure. Tolerating link failures in control applications such as these is important because the processors can still control the motion drives independently, and maintain safe system behaviour.

The synthesis of the synchronisation logic, incorporating real-time fault-tolerant commit protocols, that perform intermittent synchronisation of the asynchronous processes controlling industrial control problems was presented in Chapter 6. The analysis of the distributed controller designs relied upon the inheritance of certain properties of the commit blocks used, the verification of these commit blocks and their underlying Petri net protocol models was performed in Chapter 5. The coordination of the distributed controllers was shown to be preserved even in the presence of certain defined failures, through the use of the fault tolerant commit protocols.

The analysis of the distributed controller designs relied upon properties of the (WFMB) commit blocks used in the design. These properties were proven separately by the analysis and verification of the commit blocks and their underlying Petri net templates, and presented in Chapters 4 and 5. Thus the analysis of the Petri net controllers in Chapter 6 uses a modular approach, that allows the management of the size of the Petri net model and its state space. Some form of modular analysis is required as the size of the reachable state-space of the composed system, with the protocols state space fully enumerated, would become unmanageable. The WFMB approach allows the modular specification and manageable analysis for the controller model and the protocol models sub-net. This is required as even using automated tools the state explosion can render reachability based analysis unmanageable for real-world systems [Zhou & DiCesare 93].

The main advantage of the formal verification performed (presented in sections 6.2.4 & 6.3.4) is the high level of confidence it provides in the controller's design. However, the complex nature of the proof system leaves it prone to error and requires a high level of skill and understanding on the part of the designer.

The design and analysis procedure used in this thesis is novel in that the controller and the protocol are synthesised from verified Petri net templates that are structured in a similar manner to the *well formed blocks* defined in [Valette 79]. The WFMB differ from these, in that they allow the structuring of multiple interacting processes, whose numbers may change without causing deadlock of the underlying nets. They also allow the modelling of several possible outcomes for the blocks underlying nets behaviour.

The use of WFMBs allowed a move towards a modular approach to be taken with analysis based on the use of these structures. The modular construction of designs allowed the system to be built from well understood sub-systems, for both the distributed controllers and the commit protocols. The composition procedure can offer an intuitive guide to the modelling and design process, as well as offering real benefits in the analysis and verification through the reduction in the reachable state space. By observing specified constraints on a WFMBs interconnection, properties of the WFMBs underlying net structure can be inherited in composed designs.

7.2 Summary of Contributions

The major contributions of the research described in this thesis are:

i)The development of a responsive commit protocol design, based on the E2PC protocol, using Petri nets.

ii) The development of a structuring approach (WFMB) for a Petri net based design and analysis, that is suited to distributed control systems, as the structuring is based on inter-process communication. The approach is used to manage the size and complexity of the design and to overcome the problem of state space explosion in analysis.

iii) The development of a responsive commit protocol design, equivalent to the standard Petri net version previously developed, using the WFMB approach.

iv) An illustration of the approach developed by application to two industrial applications.

v) The partition of centralised synchronisation logic designs to yield an accurate distributed control logic. This used the commit blocks to deliver a design equivalent to the original centralised controllers.

vi) The formal verification of the distributed control logic designs, developed using the WFMB approach, that incorporate the responsive commit protocols developed.

7.3 Further Work

A number of directions for future research are identified as a result of the work developed in this thesis, and these are:

i) The development of a methodology for the application of WFMBs. This methodology should provide a systematic procedure for synthesising a controller design from the specification of the system, along with techniques for analysing the resulting controller. The WFMB approach developed in this Thesis could form the basis for this methodology.

ii) The application of model checking [Clarke & Kurshan 96] to the type of commit protocols and distributed control problems investigated in this Thesis. This would involve a similar approach to that used in this research as the reachable state space of designs are required by the analysis techniques. Model checking could be used to automate the proofs done by hand in this thesis. The use of an (LTL) model checker to automate proofs would allow the approach to be applied to larger industrial applications.

iii) An investigation of the combination of high level Petri nets and temporal logic to the types of applications addressed in this thesis, as the management of size and complexity of design and analysis is of prime concern in the application of this research.

References

[Akatsu 91b]
M.Akatsu Tomohiro.Murata K.Kurihara "Verification of Error Recovery Specification for Distributed Data by Using Colored Petri Nets", IEICE Transactions, V.E-74 N.10, pp3159-3167, Oct 1991

[Alpern & Schneider 85] B. Alpern and F.B. Schneider, "Defining liveness", Information Processing Letters, Vol. 21, No. 4, 1985, pp 181 - 185.

[Anderson & Lee 81]
T.Anderson & P.A.Lee "Fault Tolerance; principles and practice", Prentice/Hall Int, 1981

[Anttila *et al* 83]
M. Anttila, H. Erikson, H., and J. Ikonen, "Tools and studies of formal techniques-Petri nets and temporal logic", In H. Rudin and C.H. West (Eds), 'Protocol Specification Testing and Verification', III, Elsevier Science Pub. B.V., North Holland, 1983, pp 139 - 148.

[Ardis *et al* 96]
M.A.Ardis, J.A.Chaves, L.J.Jagadeesan, P.Mataga, C.Puchol, M.G.Staskauskas & J.Von Olnhausen "A Framework for Evaluating Specification Methods foe Reactive Systems", IEEE Trans S/W Eng, Vol.22, No.6 June 1996, pp378389

[Azzopardi 96]
D.Azzopardi "A Methodology for Analysis and Control of Discrete Event Dynamic Systems", PhD thesis, Aston University, Birmingham, UK, 1996

[Avizienis 85]
A.Avizienis "The N-Version approach to fault-tolerant software",IEEE Trans Software Engineering,V.SE-11 N.12,pp1491-1501,Dec 1985

[Bennett 88]
S.Bennett "Real-time computer control", Prentice-Hall, 1988

[Bernstein *et al* 87]
P.A.Bernstein, V.Hadzilacos & N.Goodman "Concurrency Control & Recovery in Database Systems", Addison-Wesley. M.A., USA, 1987

[Berthelot & Terrat 82]
G. Berthelot and R. Terrat, "Petri nets for the correctness of protocols", In R.P. van de Riet and W. Litwin (Eds), 'Distributed Data Sharing Systems', North-Holland Pub. Co., 1982, pp 23 - 43.

[Berthomieu & Diaz 91]
B.Berthomieu & M.Diaz "Modelling and verification of time dependent systems using time Petri nets", IEEE Trans S/W Eng, Vol. 17, No. 3, March 1991, pp259-273

[Bouricius 71]
W.G.Bouricius *et al* "Reliability Modelling Techniques for Fault Tolerant Computers" IEEE Transactions on Computers, V.20 N.11, pp1306-1311, November 1971

[Bucci & Vicario 95]
G.Bucci & E.Vicario "Compositional validation of time-critical systems using communicating time Petri nets", IEEE Trans S/W Eng, Vol. 21, No. 12, Dec 1995, pp969-992

[Buchholz 94]
P.Buchholz "Hierachical high level Petri nets for complex system analysis", in R.Valette (ed.) Application and Theory of Petri Nets 94, LNCS 815, Springer-Verlag 1994, pp119-138

[Carpenter 92]
G.F.Carpenter "Tolerating Communication Failures", Microsystems and Microprogramming, Gran Canaria 1992, pp 394-400.

[CCITT 92]
"Specification and Description Language", Z.100, International Consultative Committee on Telegraphy and Telephony, Geneva, 1992

[Chellas 80]
B.F. Chellas, 'Modal logic: an introduction', Cambridge University Press, 1980.

[Christensen & Hansen 94]
S.Christensen & N.D.Hansen "Coloured Petri nets extended with channels for synchronous communication", in R.Valette (ed.) Application and Theory of Petri Nets 94, LNCS 815, Springer-Verlag 1994, pp159-178

[Christensen & Petrucci 95]
S.Christensen & L.Petrucci "Modular state space analysis of Coloured Petri nets ", Proc.16th ICATPN, Torino , Italy, June 1995

[Clarke & Kurshan 96]
E.M.Clarke & R.P.Kurshan "Computer-aided verification", IEEE Spectrum, June 1996, pp61-67

[Cristain 89]
F.Cristian "A probabilistic approach to clock synchronisation",9th Int Conf Distributed Computing Systems,pp288-295,IEEE Computer Soc.Press, June 1989

[Cristian 93]
F.Cristian "Automatic reconfiguration in the presence of failures", Software Engineering Journal, March 93, pp 53-60

[David 91]
R. David, "Modelling of dynamic systems by Petri nets", Proc. of European Control Conference, Grenoble, France, 2 - 5 July 1991, pp 136 -147.

[David & Alla 92]
R. David & H.Alla, "Petri nets and Grafcet: Tools for Modelling Discrete Event Systems", Prentice-Hall, 1992.

[Davidson *et al* 89a]
S.Davidson, I.Lee & V.Wolfe, "Language Constructs for Timed Atomic Commitment", Proc. 19th Int. Symp. on Fault Tolerant Computiing, FTCS-19, IEEE, 1989, pp470-477

[Davidson *et al* 89b]
S.Davidson, I.Lee & V.Wolfe, "A Protocol for Timed Atomic Commitment", 9th Int. Conf. on Distributed Computing, IEEE Comp Soc, 1989, pp199-206

[Davies & Schneider 89]
J. Davies and S. Schneider, 'An Introduction to Timed CSP', PRG Technical Monograph, No. 75, Oxford University, 1989.

[Desai & Boutros 96]
B.C.Desai & B.S.Boutros "Performance of a two-phase commit protocol", Information & Software Technology, Vol.38 No.9, 1996, pp581-599

[Dwork & Skeen 83]
C.Dwork & D.Skeen "The inherent cost of non-blocking commitment", Proc. Symp. Principles of Distributed Computing, ACM, 1983, pp1-11

[Dwyer *et al* 95]
M.B.Dwyer, L.A.Clarke & K.A.Nies "A compact Perti net representation for concurrent programs", 17th Int Conf Software Engineering, April 23-30 1995, Seattle, pp147-157

[Dwyer & Clarke 96]
M.B.Dwyer & L.A.Clarke "A compact Petri net representation and its implications for analysis", IEEE Trans. Software Eng., Vol. 22, No. 11, Nov 1996, pp794-811.

[Ehrig & Mahr 85]
H.Ehrig & B.Mahr "Fundamentals of Algebraic Specification 1, EATCS Monographs on Theoretical Computer Science, 6, Springer-Verlag, Berlin, Germany, 1985

[Emerson 86]
E.A. Emerson and J.Y. Hailpern, "Sometime and not never revisited: on branching versus linear time temporal logic", Journal of ACM, Vol. 33, No. 1, 1986, pp 515 - 178.

[Fischer 90]
M.J.Fischer "A Theoreticians view of Fault Tolerant Distributed Computing", Lecture Notes in Computer Science 448, Springer-Verlag, pp1-9, 1990

[Garcia-Molina 82]
H.Garcia-Molina "Elections in a Distributed Computing System", IEEE Trans. Computers., Vol. C-31, No. 1, Jan 1982, pp48-59.

[Genrich & Lautenbach 81]
H.J.Genrich and K.Lautenbach,"System modelling with high-level Petri nets", Theoretical Computer Science, Vol. 13, 1981, pp 109 - 136.

[Genrich 87]
H.J. Genrich, "Predicate/transition nets", In W. Brauer, W. Reisig and G. Rozenburg (Eds), 'Petri nets: Central Models and their Properties', Lecture Notes in Computer Science, Vol. 254, 1987, pp 207 - 247.

[Gheith & Schwan 89]
A.Gheith & K.Schwan, "CHAOS^{art}: Support for Real-Time Atomic Transactions", (FTCS-19), IEEE, 1989, pp462-469

[Ghezzi *el al* 91]
C. Ghezzi, D. Mandrioli, S. Morasca and M. Pezze, "A Unified High-Level Petri Net Formalism for Time-Critical Systems", IEEE Trans. Software Eng., Vol. 17, No. 2, 1991, pp 160 - 172.

[Gouda & Multari 91]
M.G.Gouda & N.J.Multari "Stabilizing Communication Protocols", IEEE Trans. Computers., Vol. 40, No. 9, 1990, pp448-458.

[Gray 78]
J.N.Gray "Notes on database operating systems", In R.Bayer *et al.*, editors, *Operating Sysatem - An Advanced Course*, Vol.60 of Lecture Notes in Computer Science, Springer-Verlag, 1978, pp393-481.

[Gray & Reuter 93]
J.N.Gray & A.Reuter, "Transaction Processing: Concepts & Techniques", Morgan Kaufman, 1993

[Harel 87]
D. Harel, "State charts: a visual formulism for complex systems", Science of Computer Programming, Vol. 8, No. 3, 1987, pp 231 - 274.

[Harel *et al* 90]
D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "STATEMATE: A working environment for the development of complex reactive systems", IEEE Trans. Software Eng., Vol. 16, No. 4, April 1990, pp 403 - 414.

[He & Lee 90]
X. He, and J.A.N. Lee, "Integrating Predicate Transition nets with first order temporal logic in the specification and verification of concurrent systems", Formal Aspects of Computing, Vol. 2, 1990, pp 226 - 246.

[Heitmeyer 94]
C.Heitmeyer, NRL (Naval Research Laboratory) "The state of the art in the application of formal methods to the specification and verification of real-time systems.", 2nd IEEE Workshop on Real-Time Apllications, Washington, D.C. July 21-22 1994

[Hill & Holding 90]
M.R.Hill & D.J.Holding "The modelling, simulation and analysis of commit protocols in distributed computing systems", Proc. UKSC conf., Brighton, Sept 1990

[Hill 90]
M.R.Hill "The design of robust protocols for distributed real-time systems", PhD Thesis, Aston University, Sept 1990

[Hoare 85]
C.A.R.Hoare "Communicating Sequential Processes", Prentice-Hall International, U.S.A. 1985

[Holding *et al* 95]
D.J.Holding, J.Jiang *et al* "A rule-ased approach to the software synchronisation of intermittently synchronised drives", Proc IFAC Workshop on Motion control, Munich, Germany, pp461-468, 1995

[Holzmann 92]
G.J.Holzmann "Protocol Design: Redefining the State of the Art", IEEE Software, Jan 1992, pp17-22

[Hoogeboom & Halang 92]
H.Hoogeboom & W.A.Halang "The concept of time in the specification of real-time systems", Real-Time Systems Engineering and Applications, Eds. M.Schiebe & S.Pferrer, Kluwer Academic, 1992, pp11-40

[Hopcroft & Ullman 79]
J.E.Hopcroft & J.D.Ullman "Intoduction to Automata Theory, Languages and Computation", Reading, Mass, Addison-Wesley, 1979

[ISO 89a]
"Information Processing Systems - Open Systems Interconnection - ESTELLE - A Formal Description technique based on an Extended State Transition Model", ISO/IEC 9074, International Organisation for Standardisation, geneva, 1989

[ISO 89b]
"Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description technique based on the Temporal Ordering of Observational Behavior", ISO/IEC 9074, International Organisation for Standardisation, Geneva, 1989

[ISO 92]
ISO/IEC "Information Technology - Open System Interconnection - Distributed Transaction Processing - Part 1: OSI TP Model", ISO/IEC IS 10026-1: 1992

[Jahanian & Mok 86]
F.Jahanian & A.K.Mok "Safety Analysis of Timing Properties in Real-Time Systems", IEEE Trans on Software Eng,V.SE-12 N.9,pp190-204,Sept 1986

[Jahanian *et al* 88]
F. Jahanian, R. Lee and A.K. Mok, "Semantics of Modechart in real-time logic", Proc. of 21st Annual Hawaii Int. Conf. on 'System Sciences', Kailula-Kona, HI, USA, 5-8 January 1988, pp 479 - 489.

[Jiang 95]
J.Jiang "Development of real-time process control systems using formal techniques", PhD thesis, Aston University, Birmingham, UK, 1995

[Jeng & DiCesare 1993]
M.D.Jeng & F.DiCesare "A review of synthesis techniques for Petri Nets with applications to automated manufacturing systems", IEEE Trans on Systems, Man & Cybernetics, V.23 N.1, pp301-312, 1993

[Jensen 90]
K.Jensen "Coloured Petri nets: A high level language for system design and analysis", in G.Rozenberg (ed.), Advances in Petri Nets 90, LNCS 383, Springer-Verlag 1990, pp.342-416

[Kakuda *et al* 92]
Y.Kakuda, T.Kikuno, M.Malek & H.Saito "A unified approach to the design of responsive protocols" , Proc. the 1992 IEEE Workshop on fault tolerant parallel and distributed systems, July 1992, pp8-15

[Kakuda *et al* 94]
Y.Kakuda, T.Kikuno & K.Kawashima "Automated verification of responsive protocols modelled by extended finite state machines", Real-Time Systems , Vol. 7, No. 3, Kluwer, 1994, pp275-289

[Khalifa & Nketsa 96]
N.B.Khalifa, A.Nketsa "Speeding up conservative distributed simulation by means of efficient deadlock detection and recovery: Application to high level Petri nets simulation", Symp. Discrete Events & Manufac Sys, CESA'96 IMACS multiconference, pp450-454

[Kindler 97]
E.Kindler "A compositional partial order semantics for Petri net components", ICATPN'97, Ed. P.azema & G.Balbo, 23-27 June 1997, pp235-252

[King 91]
P.W.King "Formalization of Protocol Engineering Concepts", IEEE Trans on Computers, V.40 N.4, pp387-403, July 1990

[Kopetz & Ochsenreiter 87]
H.Kopetz & W.Ochsenreiter "Clock synchronisation in distributed real-time systems", IEEE Trans Computers, Vol.C-36 No.8, Aug 1987, pp933-940

[Kopetz & Grunstedl 94]
K. Kopetz & G Grunstedl "TTP - A protocol for fault-tolerant real-time systems" , IEEE Computer , January 1994, pp14-23

[Krishna & Shin 97]
C.M.Krishna & K.G.Shin "Real-Time Systems", McGraw-Hill, U.S.A., 1997

[Lamport 77]
L. Lamport, "Proving the correctness of multiprocess programs", IEEE Trans. Software Eng., Vol. SE-3, No. 2, 1977, pp 125 - 143.

[Lamport *et al* 82]
L.Lamport, R.Shostak & M.Pease "The byzantine generals problem", ACM Trans Programming Languages and Systems,Vol.4 No.3,July 1982, pp382-401

[Lamport &Melliar-Smith 85]
L.Lamport & P.M.Melliar-Smith "Synchronising clocks in the presence of faults", Journal of Association of Computing Machinery, Jan 1985

[Laprie 85]
J-C.Laprie "Dependable Computing and Fault Tolerance: Concepts and Terminology", 15th Annual International Symposium on Fault-Tolerant Computing, Ann Arbor, MI , pp 2-11,June 1985

[Laprie *et al* 90]
J-C.Laprie, J.Arlat, C.Beounes & K.Kanoun "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architecture", IEEE Computer, V.23 N.7, pp 39-51, July 1990

[Laprie 92]
J-C.Laprie, (Ed.). "Dependability: basic concepts and terminology", Vienna, Springer-Verlag, 1992

[Lautenbach & Schmid 74]
K. Lautenbach and H. Schmid, "Use of Petri nets for proving correctness of concurrent process systems", In 'Information Processing 74', North-Holland Pub. Co., 1974, pp 187 - 191.

[Lee & Gehlot 85]
I. Lee and V. Gehlot, "Language constructs for distributed real-time programming", Proc. of IEEE Symposium on 'Real-time Systems', December 1985, pp 57 - 66.

[Leveson & Harvey 83]
N.G.Leveson & P.R.Harvey "Analyzing software safety",IEEE Trans Software Engineering, V.SE-9 N.5,pp569-579,Sept 1983

[Leveson & Stolzy 85]
N.G. Leveson and J.L. Stolzy, "Analysing safety and fault-tolerance using time Petri nets", TAPSOFT: Joint Conference on 'Theory and Practice of Software Development', 25 - 29 March, 1985, Berlin, pp 339 - 355.

[Leveson 86]
N.G. Leveson, "Reliability and safety in real-time systems", IEEE Trans. Software Eng., Vol. SE-12, No. 9, 1986, pp 877 - 878.

[Leveson & Stolzy 87]
N.G. Leveson and J.L. Stolzy, "Safety analyses using Petri nets", IEEE Trans. Software Eng., Vol SE-13, No. 3, March 1987, pp 386 - 397.

[Levi & Agrawala 94]
S-T.Levi & A.K.Agrawala "Fault Tolerant System Design", McGraw-Hill, 1994

[Levy *et al* 91]
E.Levy, H.F.Korth & A.Silberschatz "An optimistic commit protocol for distributed transaction management", SIGMOND Records, May 1991, pp88-97

[Littlewood 79]
B.Littlewood "How to Measure Software Reliability and How Not To", IEEE Transactions on Reliability, V.28 N.2, pp103-110, June 1979

[Lynch 96]
N.A.Lynch "Distributed Algorithms", Morgan Kaufmann, 1996

[Mahony & hayes 92]
B.P.Mahony & I.J.Hayes "A case-study in times refinement: a mine pump", IEEE Trans on S/W Eng, Vol. SE-18, No.9, Sept 19902, pp817-825

[Malek 90]
M.Malek "Responsive Systems: A challenge for the ninties", In Proc. Euromicro 90, 16th Symp. on Microprocessing and Microprogramming. North Holland, August 1990

[Manna & Pnueli 83a]
Z. Manna and A. Pnueli, "Verification of concurrent programs: the temporal framework", In R.S. Boyer and J.S. Moore (Eds), 'The Correctness Problem in Computer Science', Mathematical Center Tracts, 159, Amsterdam, 1983.

[Manna & Pnueli 83b]
Z.Manna and A.Pnueli, "Verification of concurrent programs: a temporal proof system", In 'Foundations of Computer Science', IV, Amsterdam, Mathematical Center Tracts, 1983, pp 163 - 225.

[Manna & Pnueli 83c] Z. Manna and A. Pnueli, "How to cook a temporal proof system for your pet language", Proc. Symposium on 'Principles of Programming Languages', Austin, Texas, Jan. 1983, pp 141 - 154.

[Manna & Pnueli 92]
Z.Manna & A.Pnueli "The Temporal Logic of Reactive and Concurrent Systems", Springer-Verlag, 1992.

[Menasche & Berthomieu 83]
M.Menasche & B.Berthomieu "Time Petri nets for analysing and verifying time dependent communications protocols" , Protocol specification, testing, and verification, III, H.Rudin & C.H.West (eds), IFIP, 1983

[Merlin & Farber 76]
P.M.Merlin & D.J.Farber "Recoverability of communications protocols - Implications of a theoretical study", IEEE Trans Comms, Vol. COM-24, No. 9, Sept 1976, pp1036-1043

[Meyer & Pham 93]
J.F.Meyer & H.Pham "Fault-Tolerant Software- Guest Editors' Prolog", IEEE Trans on Reliability, V.42 N.2, June 1993

[Milner 80]
R.G.Milner "A Calculus of Communicating Systems", LNCS V.92., Springer-Verlag, 1980

[Milner 89]
R.G.Milner "Communication and Concurrency", Addison -Wesley, Massachusetts, U.S.A., 1989

[Motus 92]
L.Motus "Time concepts in real-time software", IFAC, Bruges, Belgium, pp1-10, 1992

[Murata 89]
Tadao.Murata "Petri Nets: Properties, Analysis and Applications", Proc of IEEE, Vol.77 No.4, April 1989, pp541-580

[Ostroff 89]
J.S.Ostrof "Temporal Logic for Real-Time Systems", Research Studies Press, 1989

[Ostroff 89]
J.S.Ostrof "Formal methods for the specification and design of real-time safety critical systems", Journal of Systems Software, 18, 1992, pp33-60

[Peterson 81]
J.L.Peterson "Petri Net Theory and the Modelling of Systems", Prentice-Hall, 1981

[Petri 66]
C. A. Petri, "Kommunikation mit automaten", Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIm No. 2. English translation: "Communication with automata", Tech. Report RADC-TR-65-377, Vol. 1, Suppl 1, Applied Data Research, Princeton, NJ, 1966.

[Pfluegl & Blough 91a]
M.J.Pfluegl & D.M.Blough "Evaluation of a new algorithm for fault-tolerant clock synchronisation",Pacific Rim Int Symp on Fault Tolerant Systems,IEEE Computer Soc.Press,Sep 1991, pp38-43

[Pfluegl & Blough 91b]
M.J.Pfluegl & D.M.Blough "A new model and simulation tool for fault-tolerant clock synchronisation",22nd Pittsburgh Conf on Modelling and Simulation,V.22 N.3, Instrument Soc of America,May 1991, pp1604-1611

[Pfluegl & Blough 92]
M.J.Pfluegl & D.M.Blough "Modelling and simulation of the sliding window algorithm for fault-tolerant clock synchronisation",IEICE Trans.Inf.& Syst., V.E75-D N.6,Nov 1992, pp793-796

[Pfluegl 92]
M.J.Pfluegl "Clock synchronisation in fault tolerant systems (distributed systems)" Ph.d. Theses, University of California, Irvine, USA, 1992

[Pham 92]
H.Pham (Ed.) "Introduction - Fault-Tolerant Software Systems:Techniques and Applications", IEEE Computer Society Press, pp 1-4, 1992

[Pnueli 77]
A. Pnueli, "Temporal logic of programs", Proc. of 18th Symposium on 'The Foundations of Computer Science', Nov. 1977, pp 46 - 57.

[Pnueli 85] A. Pnueli, "In transition from global to modular temporal reasoning about programs", In K.R. Apt (Ed), 'Logics and Models of Concurrent Systems', NATO ASI Series, Vol. F13, Springer-Verlag, 1985, pp 123 - 144.

[Pnueli 86]
A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends", In J. de Bakker, W.P. de Roever and G. Rozenburg (Eds), 'Current Trends in Concurrency', Lecture Notes in Computer Science, Vol. 244, Springer-Verlag, 1986, pp 510 - 584.

[Pnueli & Harel 88]
A. Pnueli and E. Harel, "Applications of temporal logic to the specification of real-time systems", In M. Joseph (Ed.), 'Formal Techniques in Real-time and Fault-tolerant Systems', Lecture Notes in Computer Science, Vol. 331, Springer-Verlag, 1988, pp 84 - 98.

[Powell *et al* 91]
D.Powell, M.Chereque & D.Drackley "Fault-Tolerance in DELTA-4", Operating Systems Review,Vol.25 I.2, April 91

[Ramaswamy & Valavanis 94]
S.Ramaswamy & K.P.Valavanis, "Modelling, analysis and simulation of failures in a materials handling system with extended Petri nets", IEEE Trans on Systems, Man & Cybernetics, Vol.24, No.9, Sept 1994

[Ramaswamy & Valavanis 96]
S.Ramaswamy & K.P.Valavanis, "Hierachical time-extended Petri nets (H-EPNs) based error identification and recovery for multilevel systems", IEEE Trans on Systems, Man & Cybernetics, Vol.26, No.1, Jan 1996

[Ramchandani 74]
C.Ramchandani "Analysis of asynchronous concurrent systems by timed Petri nets", Massachusetts Inst. Technol., Project MAC, Tech. Rep. 120, Feb. 1974

[Razouk & Phelps 85]
R.R.Razouk & C.V.Phelps "Performance analysis using timed Petri nets", Protocol specification, testing, and verification, IV, Y.Yemeni, R.Strom & S.Yemeni (eds), IFIP, 1985, pp561-576

[Reisig 85]
W.Reisig "Petri Nets: Introduction", Springer-verlag, 1985

[Reisig 88]
W. Reisig, "Temporal logic and causality in concurrent systems", In F.H. Vogt (Eds), 'Concurrency 88', Lecture Notes in Computer Science, Vol. 335, Springer-Verlag, 1988, pp 121 - 139.

[Rescher & Urquhart 71]
N. Rescher and A. Urquhart, 'Temporal logic', Springer-Verlag, 1971.

[Sagoo & Holding 90]
J.S.Sagoo & D.J.Holding "The specification and design of hard real-time systems using timed and temporal Petri nets", Microprocessing and Microprogramming, (30), 1990, pp389-396

[Sagoo & Holding 91]
J.S. Sagoo and D.J. Holding, "The use of temporal Petri nets in the specification and design of systems with safety implications", Proc. of 1st IFAC Workshop on 'Architectures and algorithms for real-time control', IFAC Workshop Series, No. 4, Pergamon Press, 1991, pp 231 - 236.

[Sagoo 92]
J.S.Sagoo "The development of hard real-time systems using a formal approach", PhD thesis, Aston University, Birmingham, UK, 1992

[Samaras *et al* 95]
G.Samaras, K.Britton, A.Citron & C.Mohan, "Two-Phase Commit Optimizations in a Commarcial Distributed Environment", Distributed and Parrallel Databases, 3, Kluwer Academic Publishers, Boston, 1995, pp325-360

[Scholefield 90]
D.J. Scholefield, "The formal development of real-time systems: a review", University of York, Computer Science, 145, 1990.

[Schwartz & Melliar-Smith 82]
R.L.Schwartz & P.M.Melliar-Smith "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", IEEE Trans on Communications, Vol.COM-30, No.12, Dec 1982, pp2486-2496

[Shanker & Lam 87]
A.U. Shanker and S.S. Lam, "Time-dependent distributed systems: proving safety, liveness and real-time properties", Distributed Computing, Vol. 2, 1987, pp 61 - 79.

[Sharp 94]
R.Sharp "Principles of Protocol Design", Prentice Hall Int., U.K. 1994

[Shaw 92]
A.C.Shaw "Communicating Real-Time State Machines", IEEE Transactions on Software Engineering, V.SE-18 N.9, Sept 1992, pp805-816

[Shieh 89]
Y-B.Shieh *et al* "Application of Petri Net Models for the Evaluation of Fault Tolerant Techniques in Distributed Systems", 9th Int Conf on Distributed Computing Systems, pp151-159, IEEE Comp Soc Press, 5-9 June 1989

[Shieh *et al* 90]
Y-B.Shieh, D.Ghosal, P.R.Chintamaneni & S.K.Tripathi, "Modelling of hierachical distributed systems with fault-tolerance", IEEE Trans on S/W Eng, Vol. SE-16, No.4, Apr 1990, pp444-457

[Sifakis 78]
J.Sifakis "Performance evaluation of systems using nets", LNCS Net Theory and Applications, Vol.84, pp307-319, 1978

[Skeen & Stonebraker 83]
D.Skeen & M.Stonebraker, "A formal model of crash recovery in a distributed system", IEEE Trans, Software Eng, Vol SE-9 N.3, May 1983, pp69-81

[Sloman 87]
M.Sloman & J.Kramer "Distributed systems and computer networks",Prentice-Hall, U.K., 1987

[Soparkar *et al* 91]
N.Soparkar, H.F.Korth & A.Silerschatz "Failure-resilient transaction management in multidatabases", IEEE Computers, Dec 1991, pp28-36

[Spivey 88]
J.M. Spivey, "An introduction to Z and formal specifications", Software Eng. Journal, Vol. 4, No. 1, 1988, pp 40 - 50.

[Srinivasan & Jafari 93]
V.S.Srinvasan & M.A.Jafari "Fault detection/monitoring using time Petri nets", IEEE Trans on Systems, Man & Cybernetics, V.23 N.4, July/Aug 1993, pp301-312

[Suzuki & Murata 83]
I.Suzuki, & T.Murata "A method for stepwise refinements and abstarctions of Petri nets", Journal of Computer and Systems Science, No.27, pp51-76, 1983

[Suzuki 85]
I. Suzuki, "Fundamental properties and application of temporal Petri nets", Proc. of 9th Annual Conf. on 'Inform. Sci. Syst.' , John Hopkins Univ., Baltimore, MD, March 1985, pp 641 - 646.

[Suzuki & Lu 89]
I. Suzuki and H. Lu, "Temporal Petri nets and their application to modelling and analysis of a handshake daisy chain arbiter", IEEE Trans. Computers., Vol. 38, No. 5, 1989, pp 696 - 704.

[Suzuki 90]
I. Suzuki, "Formal analysis of the alternating bit protocol using temporal Petri nets", IEEE Trans. Software Eng., Vol. 16, No. 11, 1990, pp 1273 - 1281.

[Suzuki *et al* 90]
T.Suzuki, S.M.Shatz & T.Murata "A protocol modeling and verification approach based on specification languge and Petri nets", IEEE Trans S/W Eng, Vol.16, No.5, May 1990, pp523-536

[Tel 94]
Gerard Tel "Introduction to distributed algorithms", Cambridge University Press, U.K., 1994

[Triantafillou 96]
P.Triantafillou, "Independent Recovery in Large-Scale Distributed Systems", IEEE Trans S/W Eng, Vol.22 No.11, Nov 1996, pp812-826

[Turner 93]
K.J.Turner (editor)"Using Formal Description Techniques: An introduction to ESTELLE, LOTOS and SDL", John Wiley & Sons Ltd, 1993, U.K.

[Valette 79]
R.Valette, "Analysis of Petri nets by stepwise refinement", Journal of Computer & System Sciences, No.18, 1979, pp35-46

[Valmari 91]
A.Valmari "Stubborn Sets for Coloured Petri Nets", Proc 12th Int Conf Application and Theory of Petri Nets, Aarhaus, 1991

[Valmari 93]
A.Valmari "Compositional State Space Generation", in G.Rozenberg (ed.), Advances in Petri Nets 93, LNCS 674, Springer-Verlag 1993, pp.427-457

[Valmari 94]
A.Valmari "Compositional analysis with place bordered subnets", iin R.Valette (ed.) Application and Theory of Petri Nets 94, LNCS 815, Springer-Verlag 1994, pp531-547

[Wing & Nixon 89]
J. M. Wing and M.R. Nixon,"Extending Ina Jo with temporal logic", IEEE Trans. Software Eng., Vol. 15, No. 2, 1989, pp 181 - 197.

[Woodcock & Davies 96]
J.Woodcock & J.Davies "Using Z: Specification, Refinement and Proof", Prentice-Hall Int, 1996

[Wu & Murata 83]
Z. Wu and T. Murata, "A Petri net model of a starvation-free solution to the dining philosophers problem", IEEE workshop on 'Languages for Automation', Chicago, USA, 7 - 9 Nov. 1983, pp 192 - 195.

[Xu & Parnas 93]
J,Xu & D.L.Parnas "On Satisfying Timing Constraints in Hard Real-Time Systems", IEEE Trans Software Engineering,V.SE-19 N.1, Jan 1993, pp70-84

[X/Open 91]
X/Open Company Ltd, "X/Open Distributed Transaction Processing: The XA Specification, CAE Specification", Dec 1991

[Yoo & Kim 95]
H-D.Yoo & M.Kim "A reliable global atomic committment protocol for distributed multidadabase systems", Information science, Elsevier science inc, Vol 83, Part 2, 1995 , pp49-76

[Yuan & Agrawala 88]
S-M.Yuan & A.K.Agrawala "A class of optimal decentralised commit protocols", Proc. 8th Int Conf. Computing Systems, San Jose, CA, June 1988, pp234-241

[Yuan & Jalote 89]
S-M.Yuan & P.Jalote "Fault tolerant commit protocols", 5th Int Conf on Data Engineering, IEEE Comput Soc, 6-10 Feb 1989, pp280-286

[Zuberek 80]
W.M. Zuberek, "Timed Petri net and preliminary performance evaluation", Proc. of 7th Annual Symposium on 'Computer Architecture', 1980, pp 88-96.

[Zurawski & Zhou 94]
R.Zurawski & M.Zhou "Petri nets and Industrial Applications: A Tutorial", IEEE Transactions on Industrial Electronics, V.41 N.6, SDec 1994, pp567-583,

Appendix A.

A.1. 3-Party Communication Block

The following are the reachability analysis for a 3-party communication block. The net is *live* as all transitions fire, and *safe* as all places are 1-bound. The reachability tree has 74 nodes. This Petri net is given as Fig.5.7. in section 5.2.2.

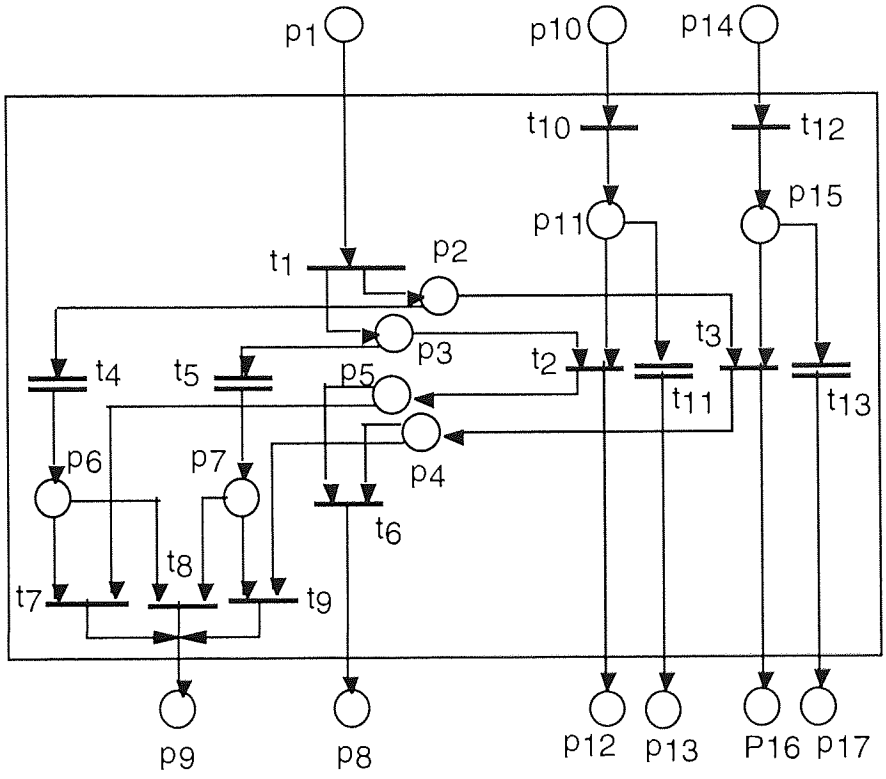


Fig.A.1. 3-party communication block

A.1.1. Petri net structure

places=17 transitions=13
transition 1: input places { 1 }: output places { 2 3 };
transition 2: input places { 3 11 }: output places { 5 12 };
transition 3: input places { 2 15 }: output places { 4 16 };
transition 4: input places { 2 }: output places { 6 };
transition 5: input places { 3 }: output places { 7 };
transition 6: input places { 4 5 }: output places { 8 };
transition 7: input places { 5 6 }: output places { 9 };
transition 8: input places { 6 7 }: output places { 9 };
transition 9: input places { 4 7 }: output places { 9 };
transition 10: input places { 10 }: output places { 11 };
transition 11: input places { 11 }: output places { 13 };
transition 12: input places { 14 }: output places { 15 };
transition 13: input places { 15 }: output places { 17 };
M_0 = { 1 10 14 }, initial marking.

A.1.2. Concurrency set

place	concurrency set
Place 1	{ 10, 11, 13, 14, 15, 17 }
Place 2	{ 3, 5, 7, 10, 11, 12, 13, 14, 15, 17 }

Place 3 { 2, 4, 6, 10, 11, 13, 14, 15, 16, 17 }
Place 4 { 3, 5, 7, 10, 11, 12, 13, 16 }
Place 5 { 2, 4, 6, 12, 14, 15, 16, 17 }s
Place 6 { 3, 5, 7, 10, 11, 12, 13, 14, 15, 17 }
Place 7 { 2, 4, 6, 10, 11, 13, 14, 15, 16, 17 }
Place 8 { 12, 16 }
Place 9 { 10, 11, 12, 13, 14, 15, 16, 17 }
Place 10 { 1, 2, 3, 4, 6, 7, 9, 14, 15, 16, 17 }
Place 11 { 1, 2, 3, 4, 6, 7, 9, 14, 15, 16, 17 }
Place 12 { 2, 4, 5, 6, 8, 9, 14, 15, 16, 17 }
Place 13 { 1, 2, 3, 4, 6, 7, 9, 14, 15, 16, 17 }
Place 14 { 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13 }
Place 15 { 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13 }
Place 16 { 3, 4, 5, 7, 8, 9, 10, 11, 12, 13 }
Place 17 { 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13 }

A.1.2. Reachability Tree

The tree deadlocks at nodes { 9, 17, 31, 45 }, these represent the following final states;
Node 9, all communications failed and all timeouts occurred.
Node 17, communication link coordinator - process 2 failed, associated timeouts occurred.
Node 31, communication link coordinator - process 1 failed, associated timeouts occurred.
Node 45, all communications succeeded.

Node Index = 1
Marking = { p1 p10 p14 }
Enabled = 1 10 12
Pre_nodes = 0
Pre_trans = 0
Post_nodes = 2 55 70
Post_trans = 1 10 12

Node Index = 2
Marking = { p1 p10 p15 }
Enabled = 1 10 13
Pre_nodes = 1
Pre_trans = 12
Post_nodes = 3 24 47
Post_trans = 1 10 13

Node Index = 3
Marking = { p1 p10 p17 }
Enabled = 1 10
Pre_nodes = 2
Pre_trans = 13
Post_nodes = 4 19
Post_trans = 1 10

Node Index = 4
Marking = { p1 p11 p17 }
Enabled = 1 11
Pre_nodes = 3 24
Pre_trans = 10 13
Post_nodes = 5 11
Post_trans = 1 11

Node Index = 5
Marking = { p1 p13 p17 }
Enabled = 1
Pre_nodes = 4 25
Pre_trans = 11 13
Post_nodes = 6
Post_trans = 1

Node Index = 6
Marking = { p2 p3 p13 p17 }
Enabled = 4 5
Pre_nodes = 5 11 26
Pre_trans = 1 11 13
Post_nodes = 7 10
Post_trans = 4 5

Node Index = 7
Marking = { p2 p7 p13 p17 }
Enabled = 4
Pre_nodes = 6 12 27
Pre_trans = 5 11 13
Post_nodes = 8
Post_trans = 4

Node Index = 8
Marking = { p6 p7 p13 p17 }
Enabled = 8
Pre_nodes = 7 10 13 28
Pre_trans = 4 5 11 13
Post_nodes = 9
Post_trans = 8

(nodes 9 to 70 not shown)

Node Index = 71
Marking = { p2 p7 p10 p14 }
Enabled = 4 10 12
Pre_nodes = 70
Pre_trans = 5
Post_nodes = 48 63 72
Post_trans = 4 10 12

Node Index = 72
Marking = { p6 p7 p10 p14 }
Enabled = 8 10 12
Pre_nodes = 71 74
Pre_trans = 4 5
Post_nodes = 49 64 73
Post_trans = 8 10 12

Node Index = 73
Marking = { p9 p10 p14 }
Enabled = 10 12
Pre_nodes = 72
Pre_trans = 8
Post_nodes = 50 65
Post_trans = 10 12

Node Index = 74
Marking = { p3 p6 p10 p14 }
Enabled = 5 10 12
Pre_nodes = 70
Pre_trans = 4

Post_nodes = 53 66 72 Post_trans = 5 10 12

A.2. Restricted 3-Party Communication Block

This is a form of the 3-party communication block, as used in the E2PC for the *prepare* and *acknowledgement* blocks (although the *acknowledgement* block differs in that there is atomic feedthrough for the participant input places). The dynamic behaviour of the net is restricted using the interlock places { p18, p19, p20, p21 }, based on the assumption of a single link failure, and timeout settings being such that all possible communication can take place. Therefore the transition (t8) that represents the coordinator timing out both participants is not required,. Places p18(19) and p20(21) are interlock places, such that timeout transitions t5(4) and t11(13) are only fireable once communication transition t3(2) has fired.

A.2.1. Petri net structure

places=21 transitions=12
transition 1: input places { 1 }: output places { 2 3 };
transition 2: input places { 3 11 }: output places { 5 12 19 21 };
transition 3: input places { 2 15 }: output places { 4 16 18 20 };
transition 4: input places { 2 21 }: output places { 6 };
transition 5: input places { 3 20 }: output places { 7 };
transition 6: input places { 4 5 18 19 20 21 }: output places { 8 };
transition 7: input places { 5 6 }: output places { 9 };
transition 9: input places { 4 7 }: output places { 9 };
transition 10: input places { 10 }: output places { 11 };
transition 11: input places { 11 18 }: output places { 13 };
transition 12: input places { 14 }: output places { 15 };
transition 13: input places { 15 19 }: output places { 17 };
M_0 = { 1 10 14 }

A.2.2. Concurrency set

place	concurrency set
Place 1	{ 10, 11, 14, 15 }
Place 2	{ 3, 5, 10, 11, 12, 14, 15, 17, 19, 21 }
Place 3	{ 2, 4, 10, 11, 13, 14, 15, 16, 18, 20 }
Place 4	{ 3, 5, 7, 10, 11, 12, 13, 16, 18, 19, 20, 21 }
Place 5	{ 2, 4, 6, 12, 14, 15, 16, 17, 18, 19, 20, 21 }
Place 6	{ 5, 12, 14, 15, 17, 19 }
Place 7	{ 4, 10, 11, 13, 16, 18 }
Place 8	{ 12, 16 }
Place 9	{ 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 }
Place 10	{ 1, 2, 3, 4, 7, 9, 14, 15, 16, 18, 20 }
Place 11	{ 1, 2, 3, 4, 7, 9, 14, 15, 16, 18, 20 }
Place 12	{ 2, 4, 5, 6, 8, 9, 14, 15, 16, 17, 18, 19, 20, 21 }
Place 13	{ 3, 4, 7, 9, 16, 20 }
Place 14	{ 1, 2, 3, 5, 6, 9, 10, 11, 12, 19, 21 }
Place 15	{ 1, 2, 3, 5, 6, 9, 10, 11, 12, 19, 21 }
Place 16	{ 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 18, 19, 20, 21 }
Place 17	{ 2, 5, 6, 9, 12, 21 }

Interlock places
Place 18 { 3, 4, 5, 7, 9, 10, 11, 12, 16, 19, 20, 21 }
Place 19 { 2, 4, 5, 6, 9, 12, 14, 15, 16, 18, 20, 21 }
Place 20 { 3, 4, 5, 10, 11, 12, 13, 16, 18, 19, 21 }

Place 21 { 2, 4, 5, 12, 14, 15, 16, 17, 18, 19, 20 }

A.2.3. Reachability Tree

The tree deadlocks at nodes { 8, 12, 17 }, these represent the following final states;
Node 8, communication link coordinator - process 2 failed, associated timeouts occurred.
Node 16, communication link coordinator - process 1 failed, associated timeouts occurred.
Node 12, all communications succeeded. The Petri net is live and 1-bound (safe). The reachability tree has 28 nodes.

Node Index = 1
Marking = { p1 p10 p14 }
Enabled = 1 9 11
Pre_nodes = 0
Pre_trans = 0
Post_nodes = 2 23 28
Post_trans = 1 9 11

Node Index = 2
Marking = { p1 p10 p15 }
Enabled = 1 9
Pre_nodes = 1
Pre_trans = 11
Post_nodes = 3 19
Post_trans = 1 9

Node Index = 3
Marking = { p1 p11 p15 }
Enabled = 1
Pre_nodes = 2 23
Pre_trans = 9 11
Post_nodes = 4
Post_trans = 1

Node Index = 4
Marking = { p2 p3 p11 p15 }
Enabled = 2 3
Pre_nodes = 3 19 24
Pre_trans = 1 9 11
Post_nodes = 5 13
Post_trans = 2 3

Node Index = 5
Marking = { p3 p4 p11 p16 p18 p20 }
Enabled = 2 5 10
Pre_nodes = 4 20
Pre_trans = 3 9
Post_nodes = 6 9 11
Post_trans = 2 5 10

Node Index = 6
Marking = { p3 p4 p13 p16 p20 }
Enabled = 5
Pre_nodes = 5
Pre_trans = 10
Post_nodes = 7
Post_trans = 5

Node Index = 7
Marking = { p4 p7 p13 p16 }
Enabled = 8
Pre_nodes = 6 9
Pre_trans = 5 10
Post_nodes = 8
Post_trans = 8

Node Index = 8
Marking = { p9 p13 p16 }
Enabled =
Pre_nodes = 7 10
Pre_trans = 8 10
Post_nodes =
Post_trans =

Node Index = 9
Marking = { p4 p7 p11 p16 p18 }
Enabled = 8 10
Pre_nodes = 5 21
Pre_trans = 5 9
Post_nodes = 7 10
Post_trans = 8 10

Node Index = 10
Marking = { p9 p11 p16 p18 }
Enabled = 10
Pre_nodes = 9 22
Pre_trans = 8 9
Post_nodes = 8
Post_trans = 10

(nodes 11 to 22 not shown)

Node Index = 23
Marking = { p1 p11 p14 }
Enabled = 1 11
Pre_nodes = 1
Pre_trans = 9
Post_nodes = 3 24
Post_trans = 1 11

Node Index = 24
Marking = { p2 p3 p11 p14 }
Enabled = 2 11
Pre_nodes = 23 28
Pre_trans = 1 9
Post_nodes = 4 25
Post_trans = 2 11

Node Index = 26
Marking = { p5 p6 p12 p14 p19 }
Enabled = 7 11
Pre_nodes = 25
Pre_trans = 4
Post_nodes = 17 27
Post_trans = 7 11

Node Index = 27
Marking = { p9 p12 p14 p19 }
Enabled = 11
Pre_nodes = 26
Pre_trans = 7
Post_nodes = 18
Post_trans = 11

Node Index = 28
Marking = { p2 p3 p10 p14 }
Enabled = 9 11
Pre_nodes = 1
Pre_trans = 1
Post_nodes = 19 24
Post_trans = 9 11

A.3. Block Structured E2PC Protocol (CB1), locking of all input places.

This Petri net can be seen in chapter 5, Fig.5.14. and Appendix.B.2, the net is live, 1-bound and deadlock free. Transitions t18 & t19 are the environment transitions of the commit protocols WFMB, t18 firing represents the action of the environment in the commit case, and t19 in the abort case. Transitions t16 & t17, and places 13..19, 29 &37..39 are used only to keep the notation for each process distinguished. The reachability tree has 89 nodes.

A.3.1. Petri net structure

places=46 transitions=40
transition 1: input places { 1 23 }: output places { 2 24 };
transition 2: input places { 23 }: output places { 22 };
transition 3: input places { 2 28 }: output places { 3 20 };
transition 4: input places { 2 22 }: output places { 4 20 };
transition 5: input places { 1 22 }: output places { 4 20 };
transition 6: input places { 5 31 }: output places { 6 33 };
transition 7: input places { 31 }: output places { 32 };
transition 8: input places { 6 36 }: output places { 7 30 };
transition 9: input places { 6 32 }: output places { 8 30 };
transition 10: input places { 5 32 }: output places { 8 30 };
transition 11: input places { 9 41 }: output places { 10 43 };
transition 12: input places { 41 }: output places { 42 };
transition 13: input places { 10 46 }: output places { 11 40 };
transition 14: input places { 10 42 }: output places { 12 40 };
transition 15: input places { 9 42 }: output places { 12 40 };
transition 16: input places { 29 }: output places { 37 38 39 };
transition 18: input places { 3 7 11 }: output places { 1 5 9 };
transition 19: input places { 4 8 12 }: output places { 1 5 9 };
transition 20: input places { 20 30 40 }: output places { 21 31 41 };
transition 21: input places { 20 30 40 }: output places { 22 31 42 };
transition 22: input places { 20 30 40 }: output places { 22 32 41 };
transition 30: input places { 21 33 43 }: output places { 23 34 44 };
transition 31: input places { 21 33 43 }: output places { 22 34 42 };
transition 32: input places { 21 33 43 }: output places { 22 32 44 };
transition 33: input places { 21 33 40 }: output places { 22 34 40 };
transition 34: input places { 21 33 40 }: output places { 22 32 40 };
transition 35: input places { 21 30 43 }: output places { 22 30 44 };
transition 36: input places { 21 30 43 }: output places { 22 30 42 };
transition 37: input places { 20 33 40 }: output places { 20 32 40 };
transition 38: input places { 20 30 43 }: output places { 20 30 42 };
transition 39: input places { 21 30 40 }: output places { 22 30 40 };
transition 40: input places { 24 34 44 }: output places { 25 35 45 };
transition 41: input places { 24 34 44 }: output places { 26 35 42 };
transition 42: input places { 24 34 44 }: output places { 27 32 45 };
transition 43: input places { 20 34 44 }: output places { 20 32 42 };
transition 44: input places { 20 34 40 }: output places { 20 32 40 };
transition 45: input places { 20 30 44 }: output places { 20 30 42 };
transition 46: input places { 26 35 }: output places { 22 32 };
transition 47: input places { 27 45 }: output places { 22 42 };
transition 48: input places { 25 35 45 }: output places { 28 36 46 };
M_0 = { 1 5 9 20 30 40 }

A.3.2. Concurrency set

Place 1 { 5, 6, 8, 9, 10, 12, 20, 21, 22, 23, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44 }
Place 2 { 6, 7, 8, 10, 11, 12, 22, 24, 25, 26, 27, 28, 30, 32, 34, 35, 36, 40, 42, 44, 45, 46 }

Place 3 { 6, 7, 10, 11, 20, 21, 22, 30, 31, 32, 36, 40, 41, 42, 46 }
Place 4 { 5, 6, 8, 9, 10, 12, 20, 21, 22, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44 }
Place 5 { 1, 4, 9, 10, 12, 20, 21, 22, 30, 31, 32, 40, 41, 42, 43 }
Place 6 { 1, 2, 3, 4, 9, 10, 11, 12, 20, 21, 22, 23, 24, 25, 26, 27, 28, 32, 33, 34, 35, 36, 40, 41, 42, 43, 44, 45, 46 }
Place 7 { 2, 3, 10, 11, 20, 21, 22, 28, 30, 31, 32, 40, 41, 42, 46 }
Place 8 { 1, 2, 4, 9, 10, 12, 20, 21, 22, 27, 30, 31, 32, 40, 41, 42, 43, 44, 45 }
Place 9 { 1, 4, 5, 6, 8, 20, 21, 22, 30, 31, 32, 33, 40, 41, 42 }
Place 10 { 1, 2, 3, 4, 5, 6, 7, 8, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46 }
Place 11 { 2, 3, 6, 7, 20, 21, 22, 28, 30, 31, 32, 36, 40, 41, 42 }
Place 12 { 1, 2, 4, 5, 6, 8, 20, 21, 22, 26, 30, 31, 32, 33, 34, 35, 40, 41, 42 }
Place 20 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 34, 36, 40, 41, 42, 43, 44, 46 }
Place 21 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 40, 41, 42, 43 }
Place 22 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44 }
Place 23 { 1, 6, 10, 34, 44 }
Place 24 { 2, 6, 10, 34, 44 }
Place 25 { 2, 6, 10, 35, 45 }
Place 26 { 2, 6, 10, 12, 35, 40, 42 }
Place 27 { 2, 6, 8, 10, 30, 32, 45 }
Place 28 { 2, 6, 7, 10, 11, 30, 36, 40, 46 }
Place 30 { 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 20, 21, 22, 27, 28, 40, 41, 42, 43, 44, 45, 46 }
Place 31 { 1, 3, 4, 5, 7, 8, 9, 10, 11, 12, 20, 21, 22, 40, 41, 42, 43 }
Place 32 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 21, 22, 27, 40, 41, 42, 43, 44, 45 }
Place 33 { 1, 4, 6, 9, 10, 12, 20, 21, 22, 40, 41, 42, 43 }
Place 34 { 1, 2, 4, 6, 10, 12, 20, 22, 23, 24, 40, 42, 44 }
Place 35 { 2, 6, 10, 12, 25, 26, 40, 42, 45 }
Place 36 { 2, 3, 6, 10, 11, 20, 28, 40, 46 }
Place 40 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 20, 21, 22, 26, 28, 30, 31, 32, 33, 34, 35, 36 }
Place 41 { 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 20, 21, 22, 30, 31, 32, 33 }
Place 42 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 21, 22, 26, 30, 31, 32, 33, 34, 35 }
Place 43 { 1, 4, 5, 6, 8, 10, 20, 21, 22, 30, 31, 32, 33 }
Place 44 { 1, 2, 4, 6, 8, 10, 20, 22, 23, 24, 30, 32, 34 }
Place 45 { 2, 6, 8, 10, 25, 27, 30, 32, 35 }
Place 46 { 2, 3, 6, 7, 10, 20, 28, 30, 36 }

A.3.3. Reachability Tree

Node Index = 1
Marking = { p1 p5 p9 p20 p30 p40 }
Enabled = 19 20 21
Pre_nodes = 0 64 6
Pre_trans = 0 17 18
Post_nodes = 2 10 23
Post_trans = 19 20 21

Node Index = 2
Marking = { p1 p5 p9 p22 p32 p41 }
Enabled = 5 10 11 12
Pre_nodes = 1 7 65
Pre_trans = 21 18 17
Post_nodes = 3 83 87 89
Post_trans = 5 10 11 12

Node Index = 3
Marking = { p1 p5 p9 p22 p32 p42 }
Enabled = 5 10 15
Pre_nodes = 2 8 10 66
Pre_trans = 12 18 7 17
Post_nodes = 4 20 82
Post_trans = 5 10 15

Node Index = 4
Marking = { p1 p5 p12 p22 p32 p40 }
Enabled = 5 10
Pre_nodes = 3 11
Pre_trans = 15 7
Post_nodes = 5 16
Post_trans = 5 10

Node Index = 5
Marking = { p1 p8 p12 p22 p30 p40 }
Enabled = 5
Pre_nodes = 4 27 29 38 82
Pre_trans = 10 31 9 14 15
Post_nodes = 6
Post_trans = 5

Node Index = 6
Marking = { p4 p8 p12 p20 p30 p40 }
Enabled = 18 19 20 21
Pre_nodes = 5 14 16 21 39 55
Pre_trans = 5 9 10 15 14 4
Post_nodes = 1 7 9 22
Post_trans = 18 19 20 21

Node Index = 7

Marking = { p4 p8 p12
p22 p32 p41 }
Enabled = 12 18
Pre_nodes = 6
Pre_trans = 21
Post_nodes = 2 8
Post_trans = 12 18

Node Index = 8
Marking = { p4 p8 p12
p22 p32 p42 }
Enabled = 18
Pre_nodes = 7 9
Pre_trans = 12 7
Post_nodes = 3
Post_trans = 18

Node Index = 9
Marking = { p4 p8 p12
p22 p31 p42 }
Enabled = 7 18
Pre_nodes = 6
Pre_trans = 20
Post_nodes = 8 10
Post_trans = 7 18

Node Index = 10
Marking = { p1 p5 p9
p22 p31 p42 }
Enabled = 5 6 7 15
Pre_nodes = 9 67 1
Pre_trans = 18 17 20
Post_nodes = 3 11 17 19
Post_trans = 5 6 7 15

(*node 11 to 85 not shown*)

Node Index = 86
Marking = { p4 p5 p10
p20 p32 p43 }
Enabled = 10
Pre_nodes = 83 89
Pre_trans = 5 11
Post_nodes = 85
Post_trans = 10

Node Index = 87
Marking = { p1 p8 p9
p22 p30 p41 }
Enabled = 5 11 12
Pre_nodes = 2
Pre_trans = 10
Post_nodes = 82 84 88
Post_trans = 5 11 12

Node Index = 88
Marking = { p4 p8 p9
p20 p30 p41 }
Enabled = 11 12
Pre_nodes = 87 89
Pre_trans = 5 10
Post_nodes = 21 85
Post_trans = 11 12

Node Index = 89
Marking = { p4 p5 p9
p20 p32 p41 }
Enabled = 10 11 12
Pre_nodes = 2
Pre_trans = 5
Post_nodes = 20 86 88
Post_trans = 10 11 12

A.4. Block Structured E2PC Protocol (CB2), locking of participant & atomic coordinator.

This Petri net can be seen in Appendix.B.2 , the net is live, 1-bound and deadlock free. Transitions t18 & t19 are an abstraction of the commit protocols environment, t18 firing represents the action of the environment in the *commit* case, and t18 in the *abort* case. Transitions t4, t16 & t17, and places 2, 13..19, 29 &37..39 are used only to keep the notation for each process distinguished. The reachability tree has 86 nodes.

A.4.1. Petri net structure

places=46 transitions=39
transition 1: input places { 1 23 }: output places { 1 24 };
transition 2: input places { 23 }: output places { 22 };
transition 3: input places { 1 28 }: output places { 3 20 };
transition 5: input places { 1 22 }: output places { 4 20 };
transition 6: input places { 5 31 }: output places { 6 33 };
transition 7: input places { 31 }: output places { 32 };
transition 8: input places { 6 36 }: output places { 7 30 };
transition 9: input places { 6 32 }: output places { 8 30 };
transition 10: input places { 5 32 }: output places { 8 30 };
transition 11: input places { 9 41 }: output places { 10 43 };
transition 12: input places { 41 }: output places { 42 };
transition 13: input places { 10 46 }: output places { 11 40 };
transition 14: input places { 10 42 }: output places { 12 40 };
transition 15: input places { 9 42 }: output places { 12 40 };
transition 16: input places { 2 29 }: output places { 37 38 39 };
transition 18: input places { 3 7 11 }: output places { 1 5 9 };
transition 19: input places { 4 8 12 }: output places { 1 5 9 };
transition 20: input places { 20 30 40 }: output places { 21 31 41 };
transition 21: input places { 20 30 40 }: output places { 22 31 42 };

transition 22: input places { 20 30 40 }: output places { 22 32 41 };
transition 30: input places { 21 33 43 }: output places { 23 34 44 };
transition 31: input places { 21 33 43 }: output places { 22 34 42 };
transition 32: input places { 21 33 43 }: output places { 22 32 44 };
transition 33: input places { 21 33 40 }: output places { 22 34 40 };
transition 34: input places { 21 33 40 }: output places { 22 32 40 };
transition 35: input places { 21 30 43 }: output places { 22 30 44 };
transition 36: input places { 21 30 43 }: output places { 22 30 42 };
transition 37: input places { 20 33 40 }: output places { 20 32 40 };
transition 38: input places { 20 30 43 }: output places { 20 30 42 };
transition 39: input places { 21 30 40 }: output places { 22 30 40 };
transition 40: input places { 24 34 44 }: output places { 25 35 45 };
transition 41: input places { 24 34 44 }: output places { 26 35 42 };
transition 42: input places { 24 34 44 }: output places { 27 32 45 };
transition 43: input places { 20 34 44 }: output places { 20 32 42 };
transition 44: input places { 20 34 40 }: output places { 20 32 40 };
transition 45: input places { 20 30 44 }: output places { 20 30 42 };
transition 46: input places { 26 35 }: output places { 22 32 };
transition 47: input places { 27 45 }: output places { 22 42 };
transition 48: input places { 25 35 45 }: output places { 28 36 46 };
M_0 = { 1 5 9 20 30 40 }

A.4.2. Concurrency set

Place 1 { 5, 6, 7, 8, 9, 10, 11, 12, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 40, 41, 42, 43, 44, 45, 46 }
Place 3 { 6, 7, 10, 11, 20, 21, 22, 30, 31, 32, 36, 40, 41, 42, 46 }
Place 4 { 5, 6, 8, 9, 10, 12, 20, 21, 22, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44 }
Place 5 { 1, 4, 9, 10, 12, 20, 21, 22, 30, 31, 32, 40, 41, 42, 43 }
Place 6 { 1, 3, 4, 9, 10, 11, 12, 20, 21, 22, 23, 24, 25, 26, 27, 28, 32, 33, 34, 35, 36, 40, 41, 42, 43, 44, 45, 46 }
Place 7 { 1, 3, 10, 11, 20, 21, 22, 28, 30, 31, 32, 40, 41, 42, 46 }
Place 8 { 1, 4, 9, 10, 12, 20, 21, 22, 27, 30, 31, 32, 40, 41, 42, 43, 44, 45 }
Place 9 { 1, 4, 5, 6, 8, 20, 21, 22, 30, 31, 32, 33, 40, 41, 42 }
Place 10 { 1, 3, 4, 5, 6, 7, 8, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46 }
Place 11 { 1, 3, 6, 7, 20, 21, 22, 28, 30, 31, 32, 36, 40, 41, 42 }
Place 12 { 1, 4, 5, 6, 8, 20, 21, 22, 26, 30, 31, 32, 33, 34, 35, 40, 41, 42 }
Place 20 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 34, 36, 40, 41, 42, 43, 44, 46 }
Place 21 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 40, 41, 42, 43 }
Place 22 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44 }
Place 23 { 1, 6, 10, 34, 44 }
Place 24 { 1, 6, 10, 34, 44 }
Place 25 { 1, 6, 10, 35, 45 }
Place 26 { 1, 6, 10, 12, 35, 40, 42 }
Place 27 { 1, 6, 8, 10, 30, 32, 45 }
Place 28 { 1, 6, 7, 10, 11, 30, 36, 40, 46 }
Place 30 { 1, 3, 4, 5, 7, 8, 9, 10, 11, 12, 20, 21, 22, 27, 28, 40, 41, 42, 43, 44, 45, 46 }
Place 31 { 1, 3, 4, 5, 7, 8, 9, 10, 11, 12, 20, 21, 22, 40, 41, 42, 43 }
Place 32 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 21, 22, 27, 40, 41, 42, 43, 44, 45 }
Place 33 { 1, 4, 6, 9, 10, 12, 20, 21, 22, 40, 41, 42, 43 }
Place 34 { 1, 4, 6, 10, 12, 20, 22, 23, 24, 40, 42, 44 }
Place 35 { 1, 6, 10, 12, 25, 26, 40, 42, 45 }
Place 36 { 1, 3, 6, 10, 11, 20, 28, 40, 46 }
Place 40 { 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 20, 21, 22, 26, 28, 30, 31, 32, 33, 34, 35, 36 }
Place 41 { 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 20, 21, 22, 30, 31, 32, 33 }
Place 42 { 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 21, 22, 26, 30, 31, 32, 33, 34, 35 }
Place 43 { 1, 4, 5, 6, 8, 10, 20, 21, 22, 30, 31, 32, 33 }
Place 44 { 1, 4, 6, 8, 10, 20, 22, 23, 24, 30, 32, 34 }

Place 45 { 1, 6, 8, 10, 25, 27, 30, 32, 35 }
Place 46 { 1, 3, 6, 7, 10, 20, 28, 30, 36 }

A.4.3. Reachability Tree

Node Index = 1
Marking = { p1 p5 p9 p20 p30 p40 }
Time= 0
Enabled = 18 19 20
Pre_nodes = 0 61 6
Pre_trans = 0 16 17
Post_nodes = 2 10 23
Post_trans = 18 19 20

Node Index = 2
Marking = { p1 p5 p9 p22 p32 p41 }
Time= 0
Enabled = 4 9 10 11
Pre_nodes = 1 7 62
Pre_trans = 20 17 16
Post_nodes = 3 80 84 86
Post_trans = 4 9 10 11

Node Index = 3
Marking = { p1 p5 p9 p22 p32 p42 }
Time= 0
Enabled = 4 9 14
Pre_nodes = 2 8 10 63
Pre_trans = 11 17 6 16
Post_nodes = 4 20 79
Post_trans = 4 9 14

Node Index = 4
Marking = { p1 p5 p12 p22 p32 p40 }
Time= 0
Enabled = 4 9
Pre_nodes = 3 11
Pre_trans = 14 6
Post_nodes = 5 16
Post_trans = 4 9

Node Index = 5
Marking = { p1 p8 p12 p22 p30 p40 }
Time= 0
Enabled = 4
Pre_nodes = 4 27 29 38 79
Pre_trans = 9 30 8 13 14
Post_nodes = 6
Post_trans = 4

Node Index = 6
Marking = { p4 p8 p12 p20 p30 p40 }
Time= 0
Enabled = 17 18 19 20
Pre_nodes = 5 14 16 21 39
Pre_trans = 4 8 9 14 13
Post_nodes = 1 7 9 22
Post_trans = 17 18 19 20

Node Index = 7
Marking = { p4 p8 p12 p22 p32 p41 }
Time= 0
Enabled = 11 17
Pre_nodes = 6
Pre_trans = 20
Post_nodes = 2 8
Post_trans = 11 17

Node Index = 8
Marking = { p4 p8 p12 p22 p32 p42 }
Time= 0
Enabled = 17
Pre_nodes = 7 9
Pre_trans = 11 6
Post_nodes = 3
Post_trans = 17

Node Index = 9
Marking = { p4 p8 p12 p22 p31 p42 }
Time= 0
Enabled = 6 17
Pre_nodes = 6
Pre_trans = 19
Post_nodes = 8 10
Post_trans = 6 17

Node Index = 10
Marking = { p1 p5 p9 p22 p31 p42 }
Time= 0
Enabled = 4 5 6 14
Pre_nodes = 9 64 1
Pre_trans = 17 16 19
Post_nodes = 3 11 17 19
Post_trans = 4 5 6 14

(nodes 11 to 81 not shown)

Node Index = 82
Marking = { p4 p8 p10 p20 p30 p43 }
Time= 0
Enabled = 29
Pre_nodes = 81 83 85
Pre_trans = 4 9 10
Post_nodes = 39
Post_trans = 29

Node Index = 83
Marking = { p4 p5 p10 p20 p32 p43 }
Time= 0
Enabled = 9
Pre_nodes = 80 86
Pre_trans = 4 10
Post_nodes = 82
Post_trans = 9

Node Index = 84
Marking = { p1 p8 p9 p22 p30 p41 }
Time= 0
Enabled = 4 10 11
Pre_nodes = 2
Pre_trans = 9
Post_nodes = 79 81 85
Post_trans = 4 10 11

Node Index = 85
Marking = { p4 p8 p9 p20 p30 p41 }
Time= 0
Enabled = 10 11
Pre_nodes = 84 86
Pre_trans = 4 9
Post_nodes = 21 82
Post_trans = 10 11

Node Index = 86
Marking = { p4 p5 p9 p20 p32 p41 }
Time= 0
Enabled = 9 10 11
Pre_nodes = 2
Pre_trans = 4
Post_nodes = 20 83 85
Post_trans = 9 10 11

A.5. 6-Axis Machine

This Petri net can be seen in chapter 6, Fig.6.5. , the net is live, 1-bound (safe) and deadlock free. The reachability tree has 79 nodes.

A.5.1. Petri net structure
places=30 transitions=19
transition 1: input places { 2 8 21 }: output places { 3 9 22 };
transition 2: input places { 3 9 21 }: output places { 4 10 21 };
transition 3: input places { 5 13 25 }: output places { 6 14 26 };
transition 4: input places { 6 14 25 }: output places { 1 15 25 };
transition 5: input places { 11 15 29 }: output places { 12 16 30 };
transition 6: input places { 12 16 29 }: output places { 7 13 29 };
transition 7: input places { 1 }: output places { 2 };
transition 8: input places { 4 }: output places { 5 };
transition 9: input places { 7 }: output places { 8 };
transition 10: input places { 10 }: output places { 11 };
transition 11: input places { 17 }: output places { 18 };
transition 12: input places { 18 22 }: output places { 19 22 };
transition 13: input places { 18 21 }: output places { 20 21 };
transition 14: input places { 20 22 }: output places { 19 22 };
transition 15: input places { 19 22 }: output places { 17 21 };
transition 16: input places { 23 26 }: output places { 24 26 };
transition 17: input places { 24 26 }: output places { 23 25 };
transition 18: input places { 27 30 }: output places { 28 30 };
transition 19: input places { 28 30 }: output places { 27 29 };
M_0 = { 2 8 15 18 21 23 25 27 29 }

A.5.2. Concurrency set
Place 1 { 7, 8, 15, 17, 18, 20, 21, 23, 25, 27, 29 }
Place 2 { 7, 8, 15, 17, 18, 20, 21, 23, 25, 27, 29 }
Place 3 { 9, 15, 17, 18, 19, 20, 21, 22, 23, 25, 27, 29 }
Place 4 { 7, 8, 10, 11, 12, 13, 15, 16, 17, 18, 20, 21, 23, 25, 27, 28, 29, 30 }
Place 5 { 7, 8, 10, 11, 12, 13, 15, 16, 17, 18, 20, 21, 23, 25, 27, 28, 29, 30 }
Place 6 { 7, 8, 14, 17, 18, 20, 21, 23, 24, 25, 26, 27, 29 }
Place 7 { 1, 2, 4, 5, 6, 13, 14, 15, 17, 18, 20, 21, 23, 24, 25, 26, 27, 29 }
Place 8 { 1, 2, 4, 5, 6, 13, 14, 15, 17, 18, 20, 21, 23, 24, 25, 26, 27, 29 }
Place 9 { 3, 15, 17, 18, 19, 20, 21, 22, 23, 25, 27, 29 }
Place 10 { 4, 5, 15, 17, 18, 20, 21, 23, 25, 27, 29 }
Place 11 { 4, 5, 15, 17, 18, 20, 21, 23, 25, 27, 29 }
Place 12 { 4, 5, 16, 17, 18, 20, 21, 23, 25, 27, 28, 29, 30 }
Place 13 { 4, 5, 7, 8, 17, 18, 20, 21, 23, 25, 27, 29 }
Place 14 { 6, 7, 8, 17, 18, 20, 21, 23, 24, 25, 26, 27, 29 }
Place 15 { 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 17, 18, 19, 20, 21, 22, 23, 25, 27, 29 }
Place 16 { 4, 5, 12, 17, 18, 20, 21, 23, 25, 27, 28, 29, 30 }
Place 17 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 }
Place 18 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 }
Place 19 { 3, 9, 15, 22, 23, 25, 27, 29 }
Place 20 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 }
Place 21 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 23, 24, 25, 26, 27, 28, 29, 30 }
Place 22 { 3, 9, 15, 17, 18, 19, 20, 23, 25, 27, 29 }

Place 23 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 25, 26, 27, 28, 29, 30 }
Place 24 { 6, 7, 8, 14, 17, 18, 20, 21, 26, 27, 29 }
Place 25 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 27, 28, 29, 30 }
Place 26 { 6, 7, 8, 14, 17, 18, 20, 21, 23, 24, 27, 29 }
Place 27 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30 }
Place 28 { 4, 5, 12, 16, 17, 18, 20, 21, 23, 25, 30 }
Place 29 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27 }
Place 30 { 4, 5, 12, 16, 17, 18, 20, 21, 23, 25, 27, 28 }

A.5.3. Reachability Tree

Node Index = 1
Marking = { p2 p8 p15 p18 p21 p23 p25 p27 p29 }
Enabled = 1 13
Pre_nodes = 0 42 47 66
Pre_trans = 0 7 9 11
Post_nodes = 2 68
Post_trans = 1 13

Node Index = 2
Marking = { p2 p8 p15 p20 p21 p23 p25 p27 p29 }
Enabled = 1
Pre_nodes = 1 19 24
Pre_trans = 13 7 9
Post_nodes = 3
Post_trans = 1

Node Index = 3
Marking = { p3 p9 p15 p20 p22 p23 p25 p27 p29 }
Enabled = 14
Pre_nodes = 2
Pre_trans = 1
Post_nodes = 4
Post_trans = 14

Node Index = 4
Marking = { p3 p9 p15 p19 p22 p23 p25 p27 p29 }
Enabled = 15
Pre_nodes = 3 68
Pre_trans = 14 12
Post_nodes = 5
Post_trans = 15

Node Index = 5
Marking = { p3 p9 p15 p17 p21 p23 p25 p27 p29 }
Enabled = 2 11
Pre_nodes = 4
Pre_trans = 15
Post_nodes = 6 54
Post_trans = 2 11

Node Index = 6
Marking = { p3 p9 p15 p18 p21 p23 p25 p27 p29 }
Enabled = 2 13
Pre_nodes = 5
Pre_trans = 11
Post_nodes = 7 31
Post_trans = 2 13

Node Index = 7
Marking = { p3 p9 p15 p20 p21 p23 p25 p27 p29 }
Enabled = 2
Pre_nodes = 6
Pre_trans = 13
Post_nodes = 8
Post_trans = 2

Node Index = 8
Marking = { p4 p10 p15 p20 p21 p23 p25 p27 p29 }
Enabled = 8 10
Pre_nodes = 7 31
Pre_trans = 2 13
Post_nodes = 9 30
Post_trans = 8 10

Node Index = 9
Marking = { p4 p11 p15 p20 p21 p23 p25 p27 p29 }
Enabled = 5 8
Pre_nodes = 8 32
Pre_trans = 10 13
Post_nodes = 10 25
Post_trans = 5 8

(nodes 10 to 76 not shown)

Node Index = 77
Marking = { p4 p7 p13 p17 p21 p23 p25 p27 p29 }
Enabled = 8 9 11
Pre_nodes = 76
Pre_trans = 6
Post_nodes = 51 60 78
Post_trans = 8 9 11

Node Index = 78
Marking = { p4 p8 p13 p17 p21 p23 p25 p27 p29 }
Enabled = 8 11
Pre_nodes = 77
Pre_trans = 9
Post_nodes = 52 61
Post_trans = 8 11

Node Index = 79
Marking = { p5 p10 p15 p17 p21 p23 p25 p27 p29 }
Enabled = 10 11
Pre_nodes = 54
Pre_trans = 8
Post_nodes = 53 56
Post_trans = 10 11

A.6. Drum & Sliders Mechanism

This Petri net can be seen in chapter 6, Fig.6.9 , the net is live, 1-bound and deadlock free. This structure has minimal p-invariants of { 1, 2, 3 }, { 4, 5, 6 }, { 12, 13 } and { 7, 8, 9, 10, 11 }. The reachability tree has 25 nodes.

A.6.1. Petri net structure

places=13 transitions=10
transition 1: input places { 1 }: output places { 2 };
transition 2: input places { 4 }: output places { 5 };
transition 3: input places { 11 }: output places { 7 };
transition 4: input places { 7 }: output places { 8 };
transition 5: input places { 2 5 12 }: output places { 3 6 13 };
transition 6: input places { 3 6 12 }: output places { 1 4 12 };
transition 7: input places { 8 13 }: output places { 9 13 };
transition 8: input places { 8 12 }: output places { 10 12 };
transition 9: input places { 10 13 }: output places { 9 13 };
transition 10: input places { 9 13 }: output places { 11 12 };
M_0 = { 2 5 8 12 }

A.6.2. Concurrency set

Place 1 { 4, 5, 7, 8, 10, 11, 12 }
Place 2 { 4, 5, 7, 8, 10, 11, 12 }
Place 3 { 6, 7, 8, 9, 10, 11, 12, 13 }
Place 4 { 1, 2, 7, 8, 10, 11, 12 }
Place 5 { 1, 2, 7, 8, 10, 11, 12 }
Place 6 { 3, 7, 8, 9, 10, 11, 12, 13 }
Place 7 { 1, 2, 3, 4, 5, 6, 12, 13 }
Place 8 { 1, 2, 3, 4, 5, 6, 12, 13 }
Place 9 { 3, 6, 13 }
Place 10 { 1, 2, 3, 4, 5, 6, 12, 13 }
Place 11 { 1, 2, 3, 4, 5, 6, 12, 13 }
Place 12 { 1, 2, 3, 4, 5, 6, 7, 8, 10, 11 }
Place 13 { 3, 6, 7, 8, 9, 10, 11 }

A.6.3. Reachability Tree

Node Index = 1
Marking = { p2 p5 p8 p12 }
Enabled = 5 8
Pre_nodes = 0 12 13 15
Pre_trans = 0 1 2 4
Post_nodes = 2 17
Post_trans = 5 8

Node Index = 2
Marking = { p2 p5 p10 p12 }
Enabled = 5
Pre_nodes = 1 10 11
Pre_trans = 8 1 2
Post_nodes = 3
Post_trans = 5

Node Index = 3
Marking = { p3 p6 p10 p13 }
Enabled = 9
Pre_nodes = 2
Pre_trans = 5
Post_nodes = 4
Post_trans = 9

Node Index = 4
Marking = { p3 p6 p9 p13 }
Enabled = 10
Pre_nodes = 3 17
Pre_trans = 9 7
Post_nodes = 5
Post_trans = 10

Node Index = 5
Marking = { p3 p6 p11 p12 }
Enabled = 3 6
Pre_nodes = 4

Pre_trans = 10
Post_nodes = 6 23
Post_trans = 3 6

Node Index = 6
Marking = { p1 p4 p11 p12 }
Enabled = 1 2 3
Pre_nodes = 5
Pre_trans = 6
Post_nodes = 7 19 22
Post_trans = 1 2 3

Node Index = 7
Marking = { p1 p4 p7 p12 }
Enabled = 1 2 4
Pre_nodes = 6 23
Pre_trans = 3 6
Post_nodes = 8 14 18
Post_trans = 1 2 4

Node Index = 8
Marking = { p1 p4 p8 p12 }
Enabled = 1 2 8
Pre_nodes = 7 24
Pre_trans = 4 6
Post_nodes = 9 12 13
Post_trans = 1 2 8

Node Index = 9
Marking = { p1 p4 p10 p12 }
Enabled = 1 2
Pre_nodes = 8 25
Pre_trans = 8 6
Post_nodes = 10 11
Post_trans = 1 2

Node Index = 10
Marking = { p1 p5 p10 p12 }
Enabled = 1
Pre_nodes = 9 12
Pre_trans = 2 8
Post_nodes = 2
Post_trans = 1

Node Index = 11
Marking = { p2 p4 p10 p12 }
Enabled = 2
Pre_nodes = 9 13
Pre_trans = 1 8
Post_nodes = 2
Post_trans = 2

Node Index = 12
Marking = { p1 p5 p8 p12 }
Enabled = 1 8
Pre_nodes = 8 14
Pre_trans = 2 4
Post_nodes = 1 10
Post_trans = 1 8

Node Index = 13
Marking = { p2 p4 p8 p12 }
Enabled = 2 8
Pre_nodes = 8 18
Pre_trans = 1 4
Post_nodes = 1 11
Post_trans = 2 8

Node Index = 14
Marking = { p1 p5 p7 p12 }
Enabled = 1 4
Pre_nodes = 7 19
Pre_trans = 2 3
Post_nodes = 12 15
Post_trans = 1 4

Node Index = 15
Marking = { p2 p5 p7 p12 }
Enabled = 4 5
Pre_nodes = 14 18 20
Pre_trans = 1 2 3
Post_nodes = 1 16
Post_trans = 4 5

Node Index = 16
Marking = { p3 p6 p7 p13 }
Enabled = 4
Pre_nodes = 15 21
Pre_trans = 5 3
Post_nodes = 17
Post_trans = 4

Node Index = 17
Marking = { p3 p6 p8 p13 }
Enabled = 7
Pre_nodes = 16 1
Pre_trans = 4 5
Post_nodes = 4
Post_trans = 7

Node Index = 18
Marking = { p2 p4 p7 p12 }
Enabled = 2 4
Pre_nodes = 7 22
Pre_trans = 1 3
Post_nodes = 13 15
Post_trans = 2 4

Node Index = 19
Marking = { p1 p5 p11 p12 }
Enabled = 1 3
Pre_nodes = 6
Pre_trans = 2
Post_nodes = 14 20
Post_trans = 1 3

Node Index = 20
Marking = { p2 p5 p11 p12 }
Enabled = 3 5
Pre_nodes = 19 22
Pre_trans = 1 2
Post_nodes = 15 21
Post_trans = 3 5

Node Index = 21
Marking = { p3 p6 p11 p13 }
Enabled = 3
Pre_nodes = 20
Pre_trans = 5
Post_nodes = 16
Post_trans = 3

Node Index = 22
Marking = { p2 p4 p11 p12 }
Enabled = 2 3
Pre_nodes = 6
Pre_trans = 1
Post_nodes = 18 20
Post_trans = 2 3

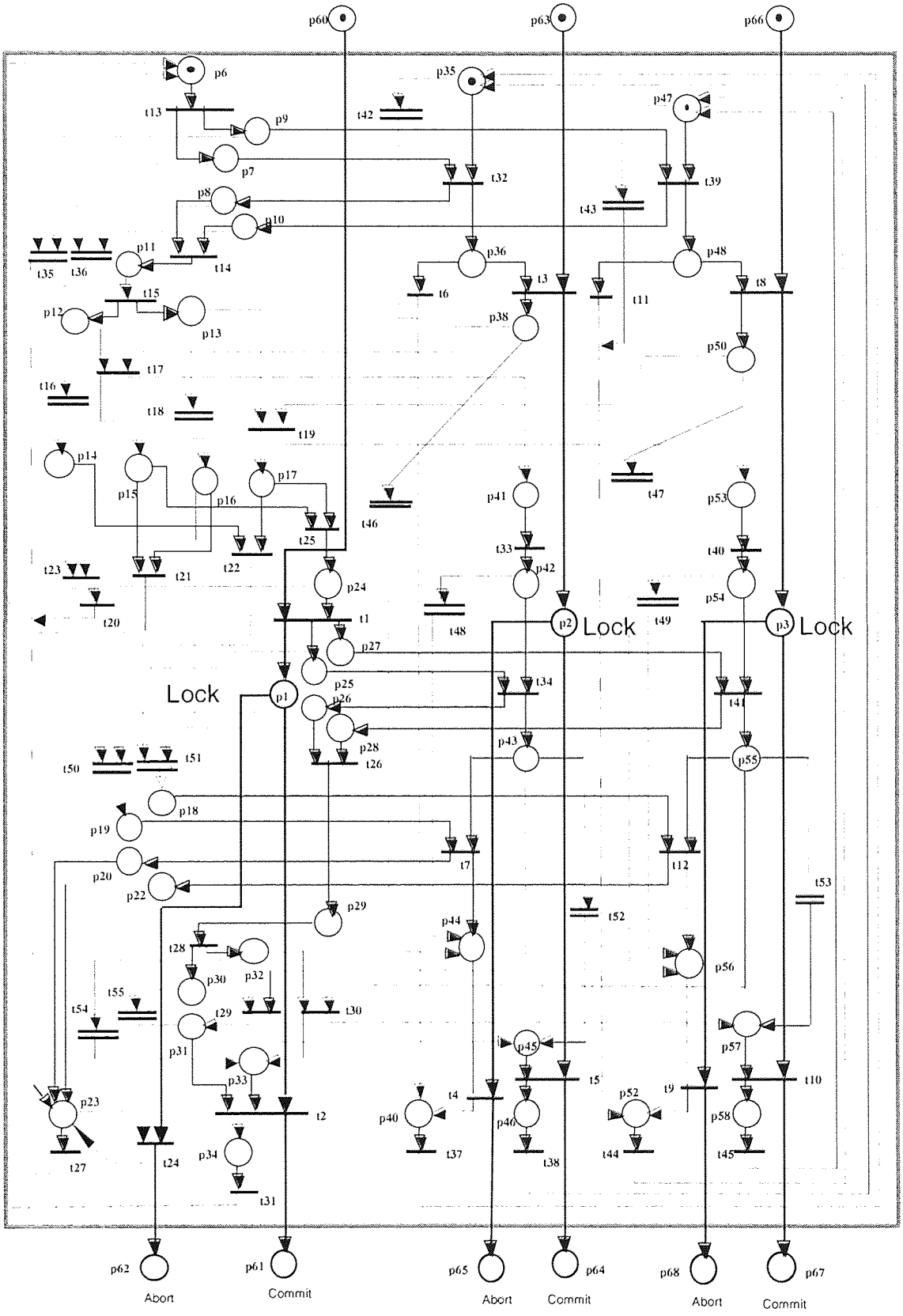
Node Index = 23
Marking = { p3 p6 p7 p12 }
Enabled = 4 6
Pre_nodes = 5
Pre_trans = 3
Post_nodes = 7 24
Post_trans = 4 6

Node Index = 24
Marking = { p3 p6 p8 p12 }
Enabled = 6 8
Pre_nodes = 23
Pre_trans = 4
Post_nodes = 8 25
Post_trans = 6 8

Node Index = 25

Marking = { p3 p6 p10
p12 }
Enabled = 6
Pre_nodes = 24
Pre_trans = 8
Post_nodes = 9
Post_trans = 6

Appendix B.
B.1. Responsive E2PC Protocol

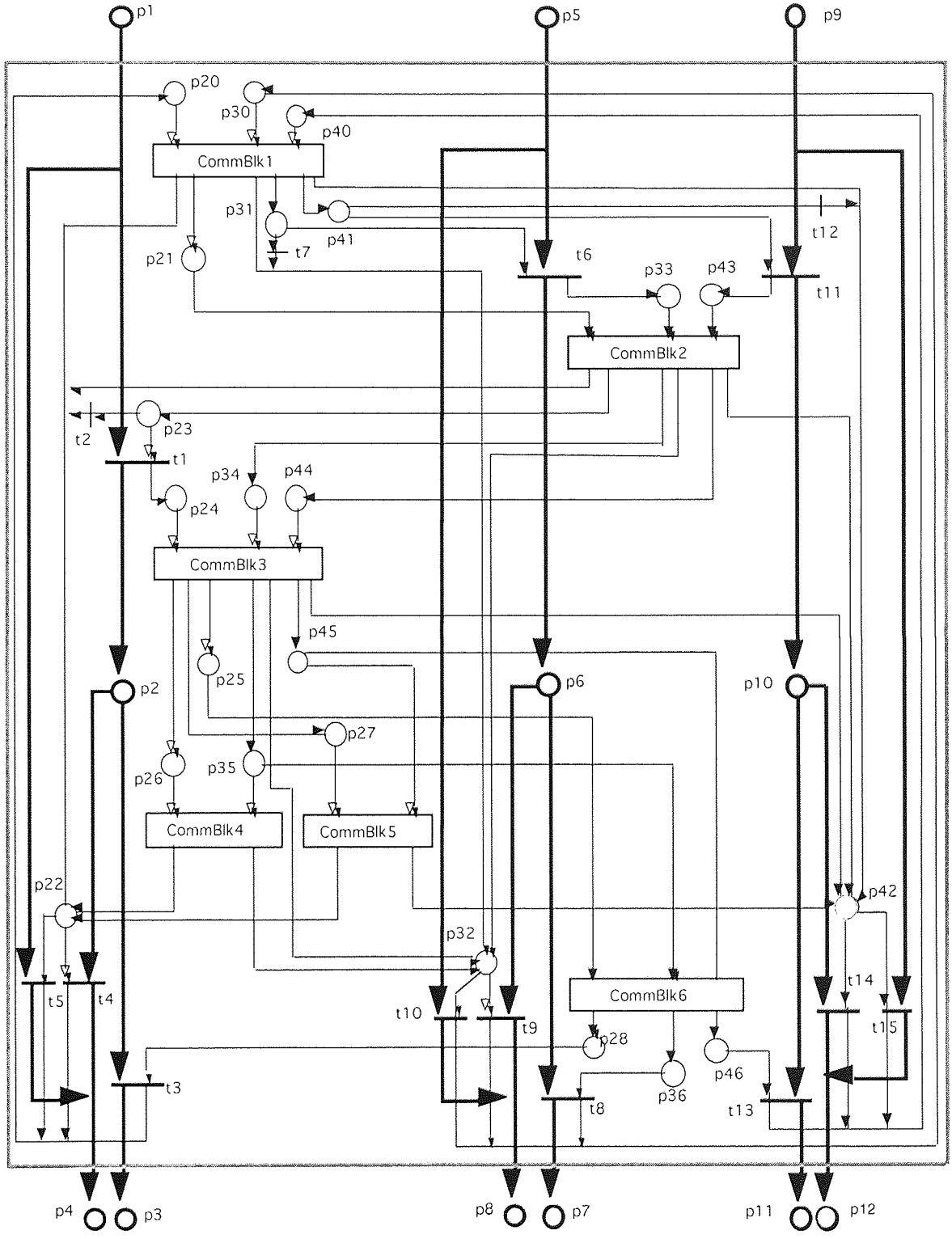


Locking of coordinator and participants input place tokens.

B.2. Hybrid Design E2PC Protocol (CB1)

Commit block 1. - CB1

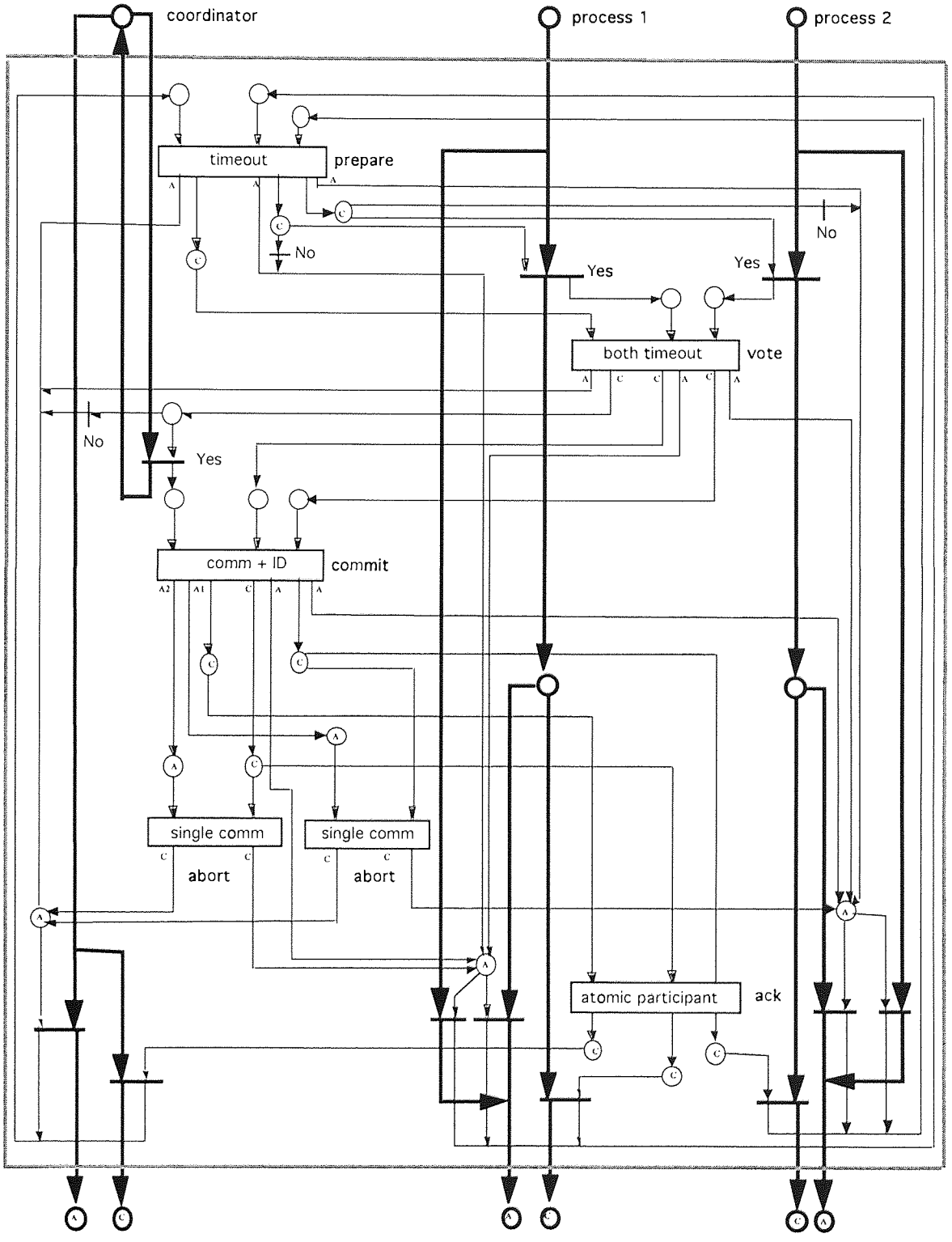
Locking of environment input place tokens, for coordinator and participants.



B.3. Hybrid Design E2PC Protocol (CB2)

Commit block 2. - CB2

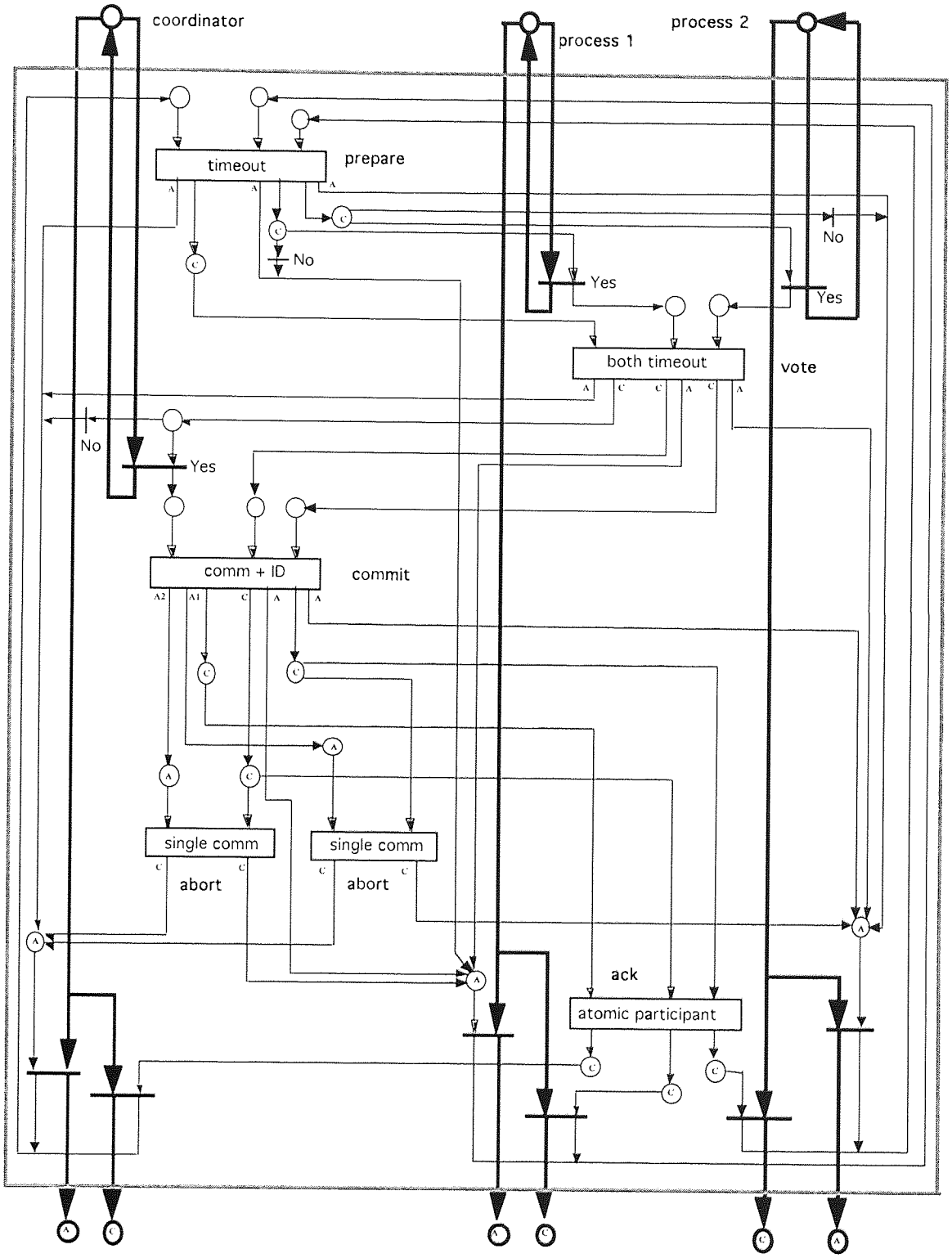
Atomic feedthrough of coordinators input place token, and locking of participant input place tokens.



B.4. Hybrid Design E2PC Protocol (CB3)

Commit block 3. - CB3

Atomic feedthrough of coordinator participants input place tokens.



Appendix.C.

C.1.

The following illustrates a proof of a property for the Prepare communication block using Temporal Petri net analysis. The Petri net for this communication block is shown in Fig.5.9., and its set of input and output places are shown in Table 5.2.

Property 5.7

Let TN_1 be a temporal Petri net for the prepare communication block as shown in Fig.5.9., which has an initial marking $M_0 = \{ p_1, p_{10}, p_{14} \}$. The property of TN_1 that will be proved is that:

Whenever M_0 of TN_1 becomes tokenised, then eventually either one of the following markings is reachable:

(a) $M_1 = \{ p_8, p_{12}, p_{16} \}$
(b) $M_2 = \{ p_9, p_{12}, p_{17} \}$
(c) $M_3 = \{ p_9, p_{13}, p_{16} \}$

This property can be formalised as:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_1 \vee M_2 \vee M_3)] \qquad (5.7)$$

This property can be interpreted as the behaviour for the prepare block when the coordinator attempts to send the prepare message to participant-1 and participant-2. The outcome is either that (a)both participants receive the message, (b) participant-1 receives the message while the coordinator and participant-2 timeout communications, due to a link failure, or (c) participant-2 receives the message while the coordinator and participant-1 timeout communications, due to a link failure.

Proof

At M_0, transitions t_1, t_{10} and t_{12} are fireable, this can be formalised as:

1. $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(t_1(ok) \wedge t_{10}(ok) \wedge t_{12}(ok))]$

From an observation of the Petri net of Fig.5.9. and its associated reachability graph (shown in Appendix.A.2), it can be seen that t_1, t_{10} and t_{12} can fire independently. Hence firing each transition yields the following sub-markings:

2. $<M_0,\alpha> \models \Box [t_1 \Rightarrow (p_2 \wedge p_3)]$

3. $<M_0,\alpha> \models \Box [t_{10} \Rightarrow p_{11}]$

4. $<M_0,\alpha> \models \Box [t_{12} \Rightarrow p_{15}]$

5. By Proposition 1 the following can be asserted about the firability of transitions t_1, t_{10} and t_{12}:

(i) $<M_0,\alpha> \models \Box [t_1(ok) \Rightarrow \Diamond t_1]$

(ii) $<M_0,\alpha> \models \Box [t_{10}(ok) \Rightarrow \Diamond t_{10}]$

(iii) $<M_0,\alpha> \models \Box [t_{12}(ok) \Rightarrow \Diamond t_{12}]$

Combining (i) - (iii) with 1. yields:

(iv) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond t_1]$

(v) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond t_{10}]$

(vi) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond t_{12}]$

Combining (iv) - 2. yields:

(vii) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_2 \wedge p_3)]$

Combining (v) - 3. yields:

(viii) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_{11})]$

Combining (vi) - 4. yields:

(ix) $<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(p_{15})]$

Combining (vii), (viii) and (x) via conjunction (because t_1 and t_{10} and t_{12} can fire)

(x) $<M_0,\alpha> \models M_0 \Rightarrow \Diamond(p_2 \wedge p_3 \wedge p_{11} \wedge p_{15})$

Let $M_4 = \{ p_2, p_3, p_{11}, p_{15} \}$, at this marking only the following sequence of transitions are allowed to fire.

(xi) $< t_2, t_3 >$,

(xii) $< t_2, t_4, t_{13} >$,

(xiii) $< t_3, t_5, t_{11} >$,

The following will consider the consequences of firing the transitions detailed in 5.(xi) - (xiii)

6. considering case 5(xi), the firing of t_2 and t_3 yields the following:

(i) $<M_0,\alpha> \models \Box [(t_2 \Rightarrow (p_5 \wedge p_{12})) \wedge (t_3 \Rightarrow (p_4 \wedge p_{16}))]$

which can be stated as $\Box [t_2 \wedge t_3 \Rightarrow (p_4 \wedge p_5 \wedge p_{12} \wedge p_{16})]$

(ii) From the structure of TN_1 it can be seen that:

$<M_0,\alpha> \models \Box [(p_4 \wedge p_5) \Rightarrow t_6(ok)]$

(iii) Using proposition 1 on 6(ii), produces:

$<M_0,\alpha> \models \Box [(p_4 \wedge p_5) \Rightarrow \Diamond t_6]$

(iv) From TN_1, it can see that

$<M_0,\alpha> \models \Box [t_6 \Rightarrow p_8]$

(v) Combining 6(ii) -6(iv), we have

$<M_0,\alpha> \models \Box [(p_4 \wedge p_5) \Rightarrow \Diamond p_8]$

(vi) Combining 6(v) - 6(i)

$<M_0,\alpha> \models \Box [(t_2 \wedge t_3) \Rightarrow \Diamond(p_8 \wedge p_{12} \wedge p_{16})]$

The consequences of 6(vi) represents M_1 above.

7. Considering case 5(xii)

(i) Firing t_2 produces:

$<M_0,\alpha> \models \Box [t_2 \Rightarrow (p_2 \wedge p_{15}) \wedge (p_5 \wedge p_{12})]$

(ii) But From TN_1, it can be seen that when sub-marking $(p_2 \wedge p_{15})$ occurs, t_4 and t_{13} can fire independently. This can be formalised as:

$<M_0,\alpha> \models \Box [(p_2 \wedge p_{15}) \Rightarrow \Diamond(t_4 \wedge t_{13})]$

(iii) Firing t_4 and t_{13} independently produces

$<M_0,\alpha> \models \Box [(t_4 \wedge t_{13}) \Rightarrow (p_6 \wedge p_{17})]$

(iv) Combining 7(ii) -7(iii), we have

$<M_0,\alpha> \models \Box [(p_2 \wedge p_{15}) \Rightarrow \Diamond(p_6 \wedge p_{17})]$

(v) Combining 7(iv) -7(i), we have

$<M_0,\alpha> \models \Box [t_2 \Rightarrow \Diamond(p_5 \wedge p_6 \wedge p_{12} \wedge p_{17})]$

(vi) But From TN_1, it can be seen that when sub-marking $(p_5 \wedge p_6)$ occurs, t_7 becomes fireable. When t_7 fires (using Proposition 1) p_9 becomes tokenised. This can be formalised as:

$<M_0,\alpha> \models \Box [(p_5 \wedge p_6) \Rightarrow \Diamond p_9]$

(vii) Combining 7(v) -7(vi), we have

$<M_0,\alpha> \models \Box [t_2 \Rightarrow \Diamond(p_9 \wedge p_{12} \wedge p_{17})]$

The consequences of 7(vii) represents M_2 above.

8. Considering case 5(xiii)

(i) Firing t_3 produces:

$<M_0,\alpha> \models \Box [t_3 \Rightarrow (p_4 \wedge p_{16}) \wedge (p_3 \wedge p_{11})]$

(ii) But From TN_1, it can be seen that when sub-marking $(p_3 \wedge p_{11})$ occurs, t_5 and t_{11} can fire independently. This can be formalised as:

$<M_0,\alpha> \models \Box [(p_3 \wedge p_{11}) \Rightarrow \Diamond(t_5 \wedge t_{11})]$

(iii) Firing t_5 and t_{11} independently produces:

$<M_0,\alpha> \models \Box [(t_5 \wedge t_{11}) \Rightarrow (p_7 \wedge p_{13})]$

(iv) Combining 8(ii) -8(iii), we have

$<M_0,\alpha> \models \Box [(p_3 \wedge p_{11}) \Rightarrow \Diamond(p_7 \wedge p_{13})]$

(v) Combining 8(iv) -8(i), we have

$<M_0,\alpha> \models \Box [t_3 \Rightarrow \Diamond(p_4 \wedge p_7 \wedge p_{16} \wedge p_{13})]$

(vi) But From TN_1, it can be seen that when sub-marking ($p4 \wedge p7$) become tokenised, $t9$ becomes fireable. When $t9$ fires (using Proposition 1) $p9$ becomes tokenised. This can be formalised as:

$$<M_0,\alpha> \models \Box [(p4 \wedge p7) \Rightarrow \Diamond p9]$$

(vii) Combining 8(v) -8(vi), produces

$$<M_0,\alpha> \models \Box [t3 \Rightarrow \Diamond(p9 \wedge p13 \wedge p16)]$$

The consequences of 8(vii) represents M_3 above.

Now combing 6(vi), 7(vii) and 8(vii) with 5. we have that:

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond[(p8 \wedge p12 \wedge p16) \vee (p9 \wedge p12 \wedge p17) \vee (p9 \wedge p13 \wedge p16)]]$$

Since $M_1 = \{ p8, p12, p16 \}$, $M_2 = \{ p9, p12, p17 \}$ and $M_3 = \{ p9, p13, p16 \}$ we have that

$$<M_0,\alpha> \models \Box [M_0 \Rightarrow \Diamond(M_1 \vee M_2 \vee M_3)]$$ ■
