# Information and Computer Science

UNIVERSITY OF CALIFORNIA

Irvine

Modeling and Analysis of Concurrent Systems

using Contour/Transition-Nets

Technical Report #245

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

## TECHNICAL REPORT


## UNIVERSITY OF CALIFORNIA
## IRVINE

UNIVERSITY OF CALIFORNIA

Irvine

# Modeling and Analysis of Concurrent Systems using Contour/Transition-Nets

Technical Report #245

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Information and Computer Science

by

Marshall Toufic Rose

Committee in charge:

Professor Rami R. Razouk, Chair

Professor David J. Farber

Professor Nancy G. Leveson

Professor Richard N. Taylor

1984

## Dedication

To my parents,
Max and Jeanette Rose

# Contents

# List of Tables

# List of Figures

# Acknowledgements

The author thanks his dissertation committee, and in particular the chair of the committee, Professor Rami R. Razouk, for their guidance during the preparation of this dissertation, and for their thoughtful criticism.

The author also thanks Einar A. Stefferud, for his invaluable mentorship during the early phases of the author's graduate studies.

The author sincerely appreciates the tremendous support of certain key members of the *ICS Computing Support Group* at U.C. Irvine. In particular, E. Timothy Morgan, an underpaid but highly skilled TEXnicial wizard, deserves special thanks for developing the "dissertation style" for AMS-TEX which provided the TEX format for this dissertation. In addition, John L. Romine, deserves thanks for maintaining a smooth running network facility which permitted the author great connectivity with his committee members, advisors, and peers. Without the professional dedication of these individuals, the author would have spent much more time preparing this dissertation. Their efforts have not gone un-noticed by those who appreciate the value of computing support.

This work was supported in part by the *USC/Information Sciences Institute*, which provided computing resources for text editing and TEX formatting. The author gratefully acknowledges this support, as without the kind help of *USC/ISI*, this dissertation would have taken *much, much* longer to complete. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of *USC/ISI*.

This work was also supported in part by the *Trilog Corporation*, which generously made possible the *Joseph F. Fischer Memorial Endowed Fellowship in Information and Computer Science*. This fellowship supported the educational expenses of the author during his final year of graduate studies. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the *Trilog Corporation*.

The summary of reachability analysis for the "readers and writers" problem in Chapter 2 was generated using reachability analysis tools developed by the Distributed Systems Group at the University of California, Irvine.

Finally, the author acknowledges the motion picture character, *"Dirty" Harry*. Inspector Callahan and his trusty .44 symbolize a purity of motivation and an intense single-mindedness which is not often found in our society. The author praises this fictional character, and the actor who portrays the character, Clint

Eastwood, for being able to tell a story simply as it is, without distractions or politics of any kind. It is encouraging to know that there exist individuals who know that "compromise is unacceptable" on certain issues, and who know that "some things cannot be talked away." It is unfortunate that nearly all of these characters are found in works of popular fiction. The reader should note that this last acknowledgement has nothing at all to do with the author's dissertation.

## Curriculum Vitae
### Marshall Toufic Rose

| | |
|---|---|
| 1981 | B.S. in Information and Computer Science<br>University of California, Irvine |
| 1983 | M.S. in Information and Computer Science<br>University of California, Irvine |
| 1984 | Ph.D. in Information and Computer Science<br>University of California, Irvine<br>Dissertation: *Modeling and Analysis of Concurrent Systems using Contour/Transition-Nets* |

# Abstract of the Dissertation

Modeling and Analysis of Concurrent Systems

using Contour/Transition-Nets

by

Marshall Toufic Rose

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1984

Professor Rami R. Razouk, Chair

This dissertation introduces a model for parallel computation, the *Contour/Transition model*, which blends the strengths of transition-based systems (concurrency, conflict, and synchronization) with the strengths of contour-based systems (control and data abstraction). By using contour/transition-nets to model computer communication protocols, a basis is provided for the formal modeling and analysis of concurrent systems.

The contour/transition model adds three notions to Petri net theory: *invocations* which correspond to Petri net procedures; *colors* which model multiple instantiations of nets; and *contours* which permit scoping mechanisms. This dissertation develops an extension to Keller's invariant-method for parallel computation that is useful for proving properties of contour/transition-nets. The usefulness of contour/transition-nets is demonstrated by a model of initial connection handling (a three-way handshake) for a protocol that offers virtual-circuit service (the DoD Transmission Control Protocol).

# CHAPTER 1
# Introduction

Computer communication networks are an important type of distributed system. The services that these networks can offer to their users are quite powerful and extensive: virtual terminal emulation, remote command execution, file transfer facilities, electronic mail, and so forth. The rapid proliferation of these networks into the research, office, and industrial environments demonstrates that computer communication networks have a wide-range of communication services that are useful to a variety of clients.

All of these services are based on a well defined communications hierarchy that allows processes on different hosts in a network or inter-network to interact in a useful fashion. At the very heart of this hierarchy is the notion of a protocol-entity.

## Protocols

Computer communication networks are often viewed as consisting of protocol-entities or (N)-peers. Computer network protocols are viewed as existing between peers in a layered system. A particular layer is described by four specifications: the (N)-service specification, which describes the services that the layer offers to the layer above; the (N)-interface specification, which describes the rules to be used to access these services; the (N-1)-service specification, which describes what services are expected from the layer below; and, the (N)-protocol specification, which describes how the peers at that level co-operate. This dissertation focuses on this final specification: the rules by which the protocol-entities interact.

In this hierarchy, the (N)-peer can be viewed as providing a set of services to the (N+1)-peer above, using two resources: the services provided by the (N-1)-peer, and a protocol to direct interaction with one or more other (N)-peers. From an alternate perspective, the (N)-protocol is used to achieve the services listed in the (N)-service specification by using the services provided in the (N-1) service specification and by co-operating with one or more (N)-peers. From a

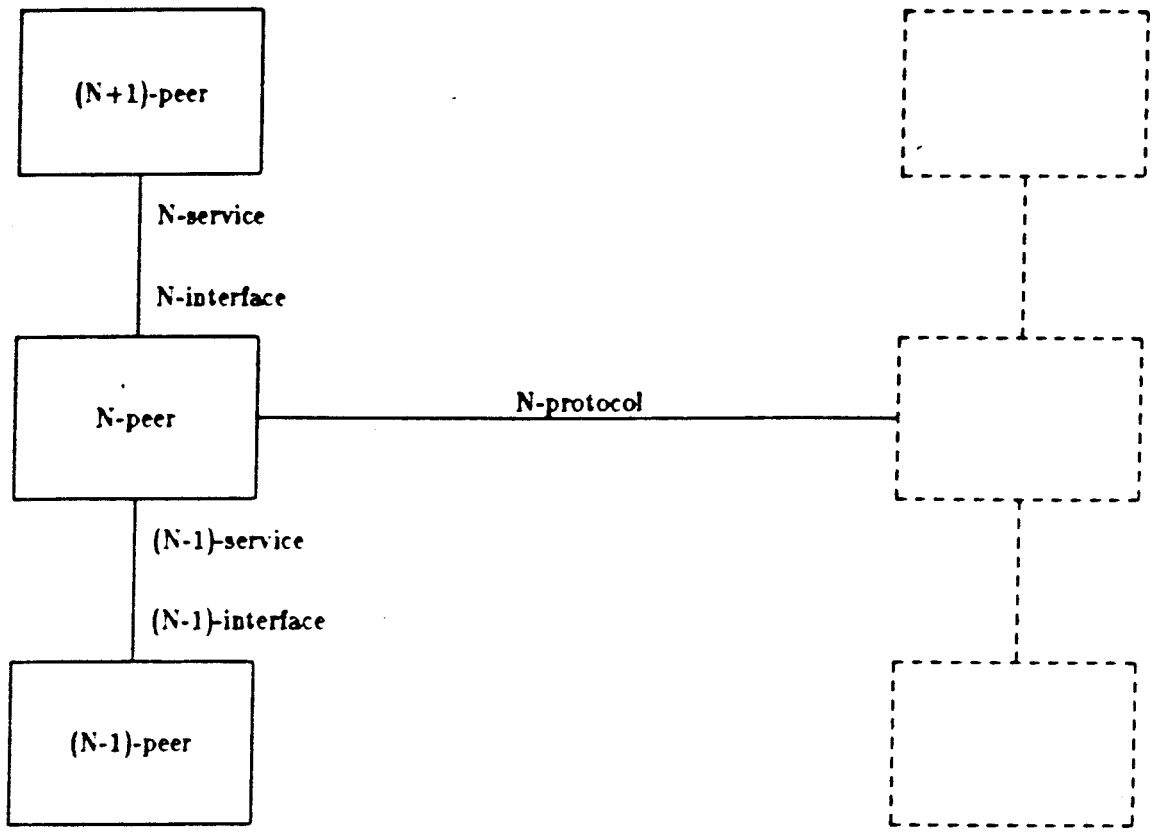



**Figure 1**

Protocol-Entities

third perspective, each layer in the system can be considered an abstract machine: the (N)-service specification defines the semantics of the services performed by the machine, the (N)-interface specification defines the syntax of those services, and the (N)-protocol defines its (internal) operating behavior. Figure 1 illustrates this hierarchy of protocol-entities. This hierarchical approach is widely-used. For example, the International Standards Organization's reference model of Open Systems Interconnection[OSI] is a protocol hierarchy defined in seven layers, while the ARPAnet reference model[PADL82] does not assign fixed layers. Cohen and Postel[COHE83] demonstrate that the number of layers in a system is neither bounded above or below the seven OSI layers, and argues instead for emphasis on the merits of layering (logical decomposition of functionality) rather than leveling (arbitrary decomposition into a fixed number of layers).

Extending these perspectives further, the end-users can themselves be seen as entities in the system, which are using the services provided to them by

other protocol-entities. Since the users of computer communication networks find the services offered by these networks to be useful, the protocols that make these services possible are considered to be important as they make possible useful services. In particular, note that for networks and hosts to inter-operate and provide useful services, different implementations of protocol-entities must correctly communicate. It is hoped that by clearly describing the behavior of a protocol different implementors will build protocol-entities that behave properly. In order to achieve this, accurate methods of describing protocols and verifying their behavior must be derived.

*Specification of Protocols*

Our focus now narrows to examining how these protocols are described, an activity known as protocol specification.

The nature of specification is based on two primary tenets, which are always in conflict with each other. At one extreme, a specification should unambiguously convey the characteristics of the system it is describing. It should state how the system acts, or say "what to do." At the other extreme, a specification should not restrict any ensuing implementations of the system it is describing. It should not state how the system is implemented, or say "how to do it." The justification for these goals is beyond the scope of this dissertation. It should be noted though, that from our perspective, the "bottom-line" is that it must be guaranteed that all implementations of a given protocol can "talk" to each other, and that they co-operate in the way in which the protocol designer expects them to. In this sense, an *unambiguous* description that *does not restrict implementation* is thought to be the best way to achieve our objective.

In addition to these two primary objectives, there are several other qualities that a specification technique should exhibit, several of which receive brief mention here. One important consideration is that the specifications made using a particular technique be *readable*. Although the readability of a given specification may largely depend on the protocol designer, a specification technique should be capable of expressing the nature of the protocol in a straightforward and "natural" way. Clearly the capability to express protocol behavior in a *concise* fashion is desirable as a part of this, although a concise method may very well reduce the readability of a specification. Related to this, a technique should be *writable* as well. A protocol

designer should find the technique easy to use and easy to write specifications with. An automated, or semi-automated, method of proving that the protocol specification can achieve the service specification based on the underlying services specification would be useful in *verifying* that the protocol specification is correct and complete. Similarly, an automated, or semi-automated, method of producing an *implementation* from the specification would be advantageous. Following along these lines, another desirable capability would be the ability to *execute* the specification prior to its implementation. This would allow the designer to observe the behavior of the protocol as it has been specified, and hopefully permit a greater understanding. Similarly, it would also be desirable to be able to provide some semi-automatic method for *testing* specifications written using the technique.

### *Formal Specification of Protocols*

It is important that formal specification techniques be used when describing the behavior of protocol-entities[SUNS79]. Without formal methods, it is not possible for designers to achieve many of these desired objectives in their specifications. As previously noted, a protocol-entity may be implemented by different organizations and individuals for use at different sites and it is unlikely that without a rigorous method of describing the behavior of a protocol that these different implementations will be able to co-operate correctly. Hence, a formal method is sought which embodies the characteristics tersely presented above: a method that can describe a protocol-entity unambiguously, in a readable but concise and straightforward way, which is easy for a protocol designer to use to capture the spirit of the protocol, which produces descriptions that can be analyzed and tested prior to implementation, and which does not hinder the implementation of the protocol-entity. It is an unfortunate circumstance that the techniques currently in use are sorely inadequate in most (or all) of these concerns. The most common technique, natural language is inherently ambiguous and poorly suited to automatic processing. As a result, a more formal method is required to specify protocols, and this area remains open to new research.

One approach to developing a formal method for the description of a protocol is to develop a representation language that is powerful enough to capture the essence of the protocol-entity's behavior and to present that behavior to the reader in such a way as to be consistent with the specification objectives described above.

Communication protocols present several challenges for a representation language. Traditional methods of specification, intended primarily for the design of sequential software, do not address the issues of timing, uncertainty, failure, and recovery which are the central focus of many parallel applications, and in particular, network protocols.

## Related Research

The breadth of methods proposed to specify protocols can be seen as ranging from state-encoded techniques at one end, to history-encoded techniques at the other[SCHWAR82]. State-encoded techniques tend to simplify the bridge to implementation, while history-encoded methods tend to permit more elegant proofs of correctness. Rather than viewing these techniques as alternatives to one another, many of the techniques are viewed as complementing each other in going from more abstract specifications to implementation.

History-encoded methods concern themselves with *sequences* of events and their relation in time. An example of a history-encoded method is temporal logic[SCHWAR81]. In contrast, state-encoded methods concern themselves with the context of a system in terms of the values of variables, simply described as *state* information. The history of the system is encoded as state information. At a particular instant, for example, each node in the system may have a different understanding of the system, based on the contents of its memory component. Examples of state-encoded methods include finite state automata, Petri nets, and algorithmic representation languages. Note that at the state-encoded extreme, there is no concept of the ordering of states with respect to time. Below, these techniques are briefly discussed.

### *Finite State Automata*

The use of finite state machines to represent protocol-entities is common. In the general case, a state machine is used to represent each protocol-entity participating, and possibly, an additional machine is used to represent the underlying layer. In concept, a machine reacts to an event (receiving an input) in a particular context (state) by producing a response (generating an output) and updating its context (state). Danthine[DANT80] presents a discussion of the use of finite state machines for protocol modeling.

The technique may be significantly augmented by supporting hierarchy with the addition of named machines, with single initial and final states. Furthermore, state encoded information may be replaced with contextual information by introducing simple data types which are manipulated during transitions between states. Bochmann[BOCH80B] presents an example of such an extended technique.

The basic finite state model is not suited for specifying complicated protocols due to its simplicity. When extensions are added, the technique is more than adequate in terms of readability. Several extensions to the approach have been suggested (e.g., Simon and Kaufman[SIMO82]), and many of these have proven successful. The CCITT X.21 interface[X.21], which is a reasonably complex entity, can be described using a method which is an extended finite state technique[WEST78]. Without extensions to permit hierarchical decomposition, the finite state technique suffers greatly when it attempts to promote abstraction. Fortunately, the use of named machines and associated data handling power permits the designer to hide details. Finally, there have been extensive research efforts to devise methodologies for the synthesis of protocols using finite state machines (e.g., Zafiropulo, et. al.[ZAFI80]).

### *Petri Nets*

A Petri net[PETR62] is another specification method that is based on transitions. A Petri net is a directed graph populated with two types of nodes: places and transitions. Places hold tokens, which represent abstract resources in the system, while transitions, which map input places (by absorbing tokens) to output places (by producing tokens), represent events in the system. The fundamental basis of Petri net theory is that transitions *fire* atomically, and if more than one transition is enabled, the choice as to which transition actually gets to fire is made non-deterministically.

As with the finite state approach, many advantages can be seen for the use of Petri nets in specification. The configuration of tokens on the net at a given instant, the Petri net's marking, clearly denotes the state of a system or entity. Enabled transitions correspond to events which may occur at the next instant in time. Finally, the ease with which such phenomena as resource contention and concurrent activity can be described makes Petri nets a powerful method for specifying a peer in a distributed system.

Unaugmented Petri net models suffer from some major drawbacks however. First, data manipulation is difficult. Since the fundamental unit in the net is the token, arithmetic and boolean computations are very tedious to represent. Second, since these tokens are considered to be of indistinguishable identity, and have a very simple semantic value, the net cannot take advantage of having enabling/firing rules which make decisions on the basis on a given token's value. Third, as a result of both of these, the only information which can be encoded in a Petri net is that information which can be represented in a marking at a given instant. Many augmentations have been suggested for Petri nets, both to solve these problems and to extend the specification power of unaugmented Petri nets. One particular Petri net extension, is the numerical Petri net[SYMO80A]. By adding processors and memory, numerical Petri nets are able to easily perform most types of data manipulation, and can be seen to have uses for protocol design[SYMO80B].

In the next chapter, the Petri net approach is examined in greater detail.

*Programming Languages*

Another popular specification technique which has received wide use for the presentation of protocol-entities, is the algorithmic approach. This method uses a programming language, usually one specifically designed for representing concurrent processes, as the specification language.

The algorithmic approach has some difficultly in accurately representing the true nature of a protocol-entity, depending on the programming language selected. Ease of use also depends quite heavily on the language used for the specification. Verification for the algorithmic approach can take several forms, including symbolic execution[BRAN82], and path analysis[SCHU80], to note two common techniques. Hailpern[HAIL81] presents a good survey of efforts in this area.

It is usually very easy to go from a system described using the algorithmic approach to an implementation. After all, the specification usually *is* an implementation. As a result, the algorithmic approach may be accurately referred to as "specification by example." A common criticism of the algorithmic approach is that it shifts the emphasis of design from the protocol to the nuances of the programming language. For these reasons, designers using the algorithmic approach produce specifications that tend to suffer from over-specification.

*Temporal Logic*

Temporal logic is a relatively new logic system which is well-suited to proofs involving sequences of events and their relation in time[PNUE77]. In addition to the standard operations of conjunction, disjunction, and negation, temporal logic introduces the notion of the *henceforth*, *eventually*, and *until* operations (to mention the most common).

There are several ways to proceed when using temporal logic as the specification technique, depending on the amount of state information one wishes to introduce into the specification. Schwartz and Melliar-Smith[SCHWAR82] presents several specifications using the temporal logic technique, which develop different specifications (and verification proofs) based on the desired amount of state information.

It is difficult to evaluate the success that temporal logic has in explaining the actions of the protocol. In terms of presenting a set of rules which describe the time relations of actions in the protocol, temporal logic does very well.

Verification is the strong-point for the temporal logic technique. Typically, one presents a temporal logic specification in two parts — the safety part and the liveness part. Safety properties focus on undesirable behavior (such as deadlock), while liveness properties deal with desirable behavior (such as progress). This means that the initial work common to verification efforts for most other specification techniques is in fact already done during specification. This method of design also raises an interesting point: Designers using temporal logic must view a protocol in an entirely different way from designers using other techniques. Temporal logic forces the designer to pay careful attention to the definition of correctness of the protocol, since that definition forms the basis of the specification. One might view temporal logic as an agent that helps the designer express the nature of the protocol in such a way so as to closely embody the service-specification. The actual verification of specifications using the temporal logic technique is straight-forward, although one may need an extensive background in logic in order to prove some of the assertions.

Temporal logic has been suggested and used for concurrent systems[PNUE81] in general and for protocol specification[SCHWAR81]. Kurose[KURO82] uses temporal logic to specify and verify a connection establishment protocol, while Sabnani and

Schwartz[SABN82] make use of temporal logic to present and analyze a multi-destinational protocol.

## Method of Attack

It is important to emphasize the difference between *specification* methods and *design* methods. Although the discussion thus far has concentrated on specification techniques, it is necessary to view things in a somewhat larger picture in order to appreciate the scope of this dissertation. In the broadest sense, the path that a protocol-entity takes from its initial conceptualization to a final implementation is a continuum. During this process, models are used to represent the protocol-entity at various levels of abstraction. At first, when the protocol-entity is abstractly presented, the protocol might be represented using a requirements definition language. Later on, when an implementation of the protocol is completed, a programming language is used as the language for representing the protocol-entity. In both cases the description of the protocol-entity forms a model of the protocol. This work concerns itself with a middle ground between these two extrema. In particular, the models which are interesting are those that are used to represent protocol-entities (and other types of concurrent software) when they are sufficiently concrete so as to be expressed unambiguously, but prior to their realization in an implemented form. That is, it is desirable to be able to model a protocol when it can be expressed formally as a design.

This dissertation focuses on the state-encoded approach to the modeling of protocol-entities and other forms of concurrent systems, and in particular the Petri net model[PETE77]. This approach was chosen for two reasons: first, state-encoded techniques tend to ease the bridge to implementation, and, second, Petri nets are well suited for representing powerful concurrency notions with simple constructs and semantics thereby.

Petri nets have been used successfully for describing hardware systems[MISU73] and in modeling many types of simple concurrent software systems[AGER79]. Petri nets and their derivatives have long been considered useful in describing and verifying communications protocols[POST74]. Because of their theoretical foundations, Petri nets tend to lend themselves well to automated analysis.

Many authors have proposed extensions to the basic Petri net model. Memory and processors have been added to the Petri net models: by Mekly and Yau[MEKL80] to introduce AP-nets, a restricted form of Petri net; by Vernon[VERN82] for use with the UCLA Graph Model of Behavior; by Symons[SYMO80A] to introduce numerical Petri nets; by Azema, et. al.[AZEM84] for use with Prolog interpreted Petri nets; and by Yau and Caglayan[YAU83] for use with "modified" Petri nets. Timing has been added to the Petri net models: by Ramchandani[RAMC74] to introduce fixed timing delays; by Merlin[MERL74] to introduce ranges of delays to model recoverable systems; by Sifakis[SIFA77] to introduce processing time to places; and by Razouk and Phelps[RAZO83A] to introduce a fixed enabling delay and a firing delay to each transition. Researchers have further demonstrated the nature of correctness proofs of systems described by Petri nets. For example, the research presented by Berthomieu and Menasche[BERTHO83] concerned itself with systems modeled with the time Petri nets presented by Merlin and Farber[MERL76B]. Many authors have demonstrated the use of various models to achieve performance analysis: by Ramamoorthy[RAMAMO80] to analyze the performance of a restricted subset of the timed Petri nets of Ramchandani[RAMC74]; by Zuberek[ZUBE80] to analyze the performance of a larger class of nets; and by Razouk and Phelps[RAZO83A] to analyze the performance of timed Petri nets, a variation of those described by Zuberek[ZUBE80].

When viewed with these extensions in mind, Petri nets continue to suffer from some inherent weaknesses. First, Petri nets are not hierarchical in nature, and do not decompose into smaller, more manageable and less cluttered structures. Traditionally, the only unit of representation is a single net. Hence, the notion of abstraction of detail is lost. As a result, Petri nets generally require a strict one-to-one mapping between nets and components in the system, e.g., if you wish to model several nodes, then usually a single, separate net is required for each node, along with some interconnection structure.

Second, Petri nets are a very *static* representational technique. Without augmentations, they are very good at describing systems with small state spaces, and very bad at describing dynamic systems with a potentially infinite number of complex, intertwined states. The concepts of scoping, procedure calls, and recursion, which are taken completely for granted by software designers, are

completely absent from the Petri net model. This is a particularly unfortunate situation as all of these are useful mechanisms of abstraction.

It is the position of this dissertation that these weaknesses can be addressed by the introduction of contour/transition-nets and that this extension provides a useful method of modeling concurrent systems. The contour/transition model introduces three notions towards these ends: the notion of *invocations* which correspond to Petri net procedures; the notion of *color* which model multiple instantiations of nets; and the notion of *contours* which permit modeling dynamic scoping mechanisms.

It is the hypothesis of this work that the methods traditionally used for the modeling and design of communications protocols rely too much on an underlying *sequential* model of computation. In contrast, the Petri net method is inherently well-suited for representing concurrent systems, but, due to some unfortunate notational weaknesses, is unable to represent concurrent software well. Contour/transition-nets are proposed as a useful hybrid which allows concurrent software to be modeled, while maintaining a strong basis in Petri net theory. It must be noted that the contour/transition model is not proposed as a specification technique per se. Rather, contour/transition-nets are suggested as being useful for modeling the behavior of a concurrent system at various levels of abstraction and detail.

## Contributions

The foremost contribution of this dissertation is the development of the contour/transition model. Contour/transition-nets are able to represent complex concurrency mechanisms using conceptually simple constructs, while at the same time permitting conventional data manipulation capabilities.

### *The Contour/Transition Model*

One contribution is to introduce and develop the theoretical foundation of contour/transition-nets. This dissertation presents a definition of the semantics of the contour/transition model and a methodology for using contour/transition-nets to model concurrent systems. This methodology identifies what are believed to be the best uses of the added extensions.

### *Analysis of Contour/Transition-Nets*

Proving a system expressed with the contour/transition model correct should be no *more* difficult than proving a concurrent program correct. Although methods and tools exist for verifying and testing concurrent software, and other methods and tools also exist for analyzing simple Petri net models, there are no such techniques developed for hybrid approaches such as the contour/transition model. This dissertation presents a hierarchically based proof methodology for analyzing properties of contour/transition-nets which is based, in part, on the work presented by Keller[KELL76] and Razouk[RAZO81].

### *Experimental Evaluation*

Although not a contribution per se, this dissertation contains an experimental evaluation of the technique. A model is built of the connection establishment phase of the Transmission Control Protocol, which represents a significant portion of a complicated protocol. Researchers in the field have used other techniques to model the three-way handshake used by the Transmission Control Protocol (e.g., Umbaugh, et. al.[UMBA82B]).

## Outline of the Dissertation

This dissertation is organized into six parts: this first part, which motivates the reader as to why the problem should be examined and solved has been presented. Next, in Chapter 2, a survey of related work in the problem area is given. In Chapter 3, a detailed definition of the contour/transition model is presented. In Chapter 4, a proof method for analyzing properties of contour/transition-nets is developed. In Chapter 5, a model of initial connection establishment in TCP is given. Finally, in Chapter 6 a summary of possible future research to be done with the contour/transition model and in the problem area is given.

# CHAPTER 2
# Related Research

Various models have been proposed to represent concurrent processes. The focus in this chapter is primarily on those methods which are based on transition systems.

## Petri Net Models

In order to present the reader with a consistent naming scheme, a single terminology is used, i.e., that of Diaz[DIAZ82]. Initially, the simplest Petri net model, place/transition-nets, is described. Then the various extensions that have been made to this simple model are discussed. This chapter concludes by discussing analysis methods which can be applied to the various models.

### *Place/Transition-Nets*

A Petri net[PETR62] is a transition-based approach to modeling. The net itself is a directed graph with two types of nodes: *places* and *transitions*. The arcs in the graph connect nodes of different types. Arcs which start at a place are called *input arcs*, while those which start at a transition are called *output arcs*. Tokens move along the graph from place to place according to strict enabling and firing rules. A transition is enabled (said to "have concession") when each of the places on its input arcs has at least one token. When a transition fires, a single token is placed on each of the places on the transition's output arcs and a single token is removed from each of the places on the transition's input arcs. In addition, only one transition in the net may fire during a given instant. If more than one transition is enabled (i.e., can fire), then the choice of which of the transitions does actually fire is non-deterministic.

The marking of a Petri net at a given instant is the distribution (location and number) of tokens on the net's places. The *initial state* of a Petri net is called its initial marking. A given marking, M, is considered to be a home-state if for all markings reachable from the initial state, M is reachable from those markings.

Conceptually, a home-state corresponds to a *final state* in a finite state automaton. A given marking, M, is considered to be live if for all markings reachable from M, at least one transition may fire. The Petri net itself is considered live if the initial marking is live. A given place in a Petri net is said to be bounded at $k$ if the place can never contain more than $k$ tokens for every marking reachable from the initial marking. The notion of a given place being safe can be thought of as that place being bounded at unity. Along these lines, a safe net is one in which each place in the net is safe. A similar definition exists for a bounded net.

A large amount of research has been devoted to analyzing Petri nets and various restricted classes of Petri nets[PETE81], which has led to a number of powerful analysis techniques for systems represented by place/transition-nets. In particular, a system represented by a place/transition-net can be expressed in terms of an *incidence matrix* which, after the proper linear algebraic manipulation, yields invariants as to the behavior of the system. This capability is a very attractive feature of unaugmented Petri net model.

Unfortunately, place/transition-nets are not particularly well-suited for representing concurrent systems as they lack several important features that have been found to be useful for modeling software. In response to these weaknesses, many extensions have been proposed to the Petri net model. The remainder of this chapter presents a representative sample of these extensions.

*Place/Coloured-Nets*

Coloured Petri nets[SCHI78] (or place/coloured-nets in the terminology of Diaz[DIAZ82]) are one attempt to increase the modeling ease of place/transition-nets while retaining the analysis results mentioned previously. In short, tokens are said to have a colour, which identifies them as a member of a set of tokens. The system consists of a number of sets. In the solution to the "dining philosophers'" problem given by Jensen[JENS81], for example, the system contains two sets: the set of philosophers and the set of forks. Each philosopher and fork in the system is represented by a single token. Furthermore, places are labelled with the types of colors that they can hold, while transitions are labelled with a mapping function describing the type of colors acceptable on each input arc, the relation that must hold between these tokens for the transition to fire, and the colors generated on each output arc. Hence, in the place/coloured-net model, colors interact. Finally,

the totality of each set of colors must be fully identified (e.g., for the dining philosophers' problem, five philosophers and forks are identified).

Place/coloured-nets do not have additional representational power than place/transition-nets, but they generally produce a more compact model of the system. For example, given 5 philosophers, the place/transition-net solution to the problem has 15 places and 10 transitions with a complicated interconnection structure. In contrast the coloured Petri net solution has three places and 2 transitions with a very simple set of arcs connecting the places and transitions.

*Predicate/Transition-Nets*

The predicate/transition-net model[GENR81, GENR78] has been introduced in the hope of increasing the modeling ease of place/transition-nets while retaining previously achieved analysis results. Predicate/transition-nets are concise abbreviations for place/transition-nets: a place in a predicate/transition-net corresponds to a set of places in a place/transition-net; and, a transition in a predicate/transition-net corresponds to a set of transitions in a place/transition-net. Tokens are multi-valued, containing an ordered tuple of parameters. Transitions map tuples on their input arcs to tuples on their output arcs, but do so in a wholly local context (which directly results from the fact that predicate/transition-nets are abbreviations for place/transition-nets). The facilities provided by the resulting economy of expression can be quite helpful in modeling complex systems. Voss[Voss80], for example, models a database management system using predicate/transition-nets.

As with the place/coloured-net model, once the totality of the values that tuples can take on is known, there is a straightforward mapping between place/transition-nets and predicate/transition-nets. Hence, predicate/transition-nets do not have additional modeling power, though like place/coloured-nets, they permit more concise descriptions of concurrent systems.

*The Graph Model of Behavior*

Another model of concurrent computation, similar in graphic nature to Petri nets, though developed independently, is the UCLA Graph Model of Behavior[POST74]. A control structure in the GMB (sometimes referred to as a UCLA graph, or a complex bi-logic directed graph) is, in many senses, an inverted

place/transition-net. Tokens flow along multi-arcs, which feed and are fed by nodes. The nodes in the GMB correspond to transitions where activity occurs, while the multi-arcs, which hold tokens, correspond to places in place/transition-nets. In addition, a complex boolean decision logic is permitted to exist between nodes and multi-arcs. Hence, a GMB representation of a system is often more concise than a representation using a place/transition-net.

When only the control structure of the GMB is viewed, the model is identical in power to place/transition-nets. Several extensions have been made to the GMB though to extend the model to encompassing three domains[RAZO81]: a *control* domain (the control portion previously discussed), a *data* domain, and a *interpretation* domain. The data domain describes the flow of data in the system by identifying the relation between nodes and the type of access they have to datasets (i.e., read, write, or read/write). The interpretation domain determines the data-dependent paths taken by the control domain and also performs transformations on the data domain. For example, if an OR output logic is present between a node and two multi-arcs, the interpretation domain decides which multi-arc of the data-dependent path receives a token based on the value of the datasets readable by the multi-arc. With these (and other) powerful extensions, the GMB is well-suited for simulating hardware and software systems. For example, in order to allow the GMB and SARA to better model software systems, Penedo[PENE81] presented a module interconnection language to support instantiations of processes and data types. In other work, Razouk and Estrin[RAZO80A] uses the GMB and the SARA system to model and analyze X.21. This analysis, like the results of other researchers (e.g., Umbaugh and Liu[UMBA82A], and West and Zafiropulo[WEST78]), found several errors in the CCITT X.21 specification[X.21].

*AP-Nets*

Mekly and Yau[MEKL80] introduce Abstract Process networks (AP-nets) as a solution to some of the problems found in software development. In particular, the AP-net is proposed as the basis of the design phase, and it is further proposed that these AP-nets act as engineering blueprints for software designs.

An abstract process schema is represented as a 5-tuple consisting of an initial state set, an input set, a process function, an output set and a final state set. Mekly and Yau[MEKL80] formally develop an analysis showing how abstract

processes can be used as building blocks for the control constructs of structured programming, and demonstrate sequential, selective, and iterative derivations. Finally, construction and implementation techniques are discussed. AP-nets are graphically realized as Petri nets with the restriction that safeness (no more than one token in any place) and liveness (at least one transition has concession) always be preserved. Furthermore, the nets are required to always be conflict-free (no two transitions in the net may have their input arcs originating at the same place).

AP-nets have several good qualities. They are easily decomposed into smaller AP-nets. This hierarchical structure makes possible the derivation of compact algebraic summaries of an AP-net. The analysis of AP-nets is straightforward. Since AP-nets are free of conflicts, the majority of problems found in examining more complicated forms of Petri nets either are not present or are reduced to much simpler proportions. AP-nets bear a close resemblance to other forms of graphical representation and can be used to represent concurrent processes.

AP-nets also have several disadvantages. Intuitively, it is difficult to see how AP-nets can be used to model complicated decision structures if the resulting Petri net representations are forced to be free of conflict. While simple algorithms can probably be demonstrated using the AP-net approach, there is doubt as to the usefulness of AP-nets in complex designs. Other Petri net based methods, such as numerical Petri nets are much more robust in the control structures that are permitted to be represented. AP-nets, in contrast, seem determined to minimize the non-deterministic nature of Petri nets. Furthermore, one issue that is relatively untouched by AP-nets are the considerations given to the scoping of data components. Despite the good hierarchical structure enjoyed by the control logic portion of an AP-net, no consideration is given to a similar semantics for variables. Finally, although AP-nets are highly decomposable, no provision for the support of invocations (and related issues, such as parameters) is made.

*"Modified" Petri nets*

Another approach to software design of concurrent processes is the "modified" Petri nets as suggsted by Yau and Caglayan[YAU83]. A "modified" Petri net consists of three components: a set of control state variables (similar to nodes in a GMB control structure), a set of data objects which contain objects defined as abstract data types (similar to the data domain of the GMB), and a set of software

components which can be viewed as *non-primitive* transitions, but which are, in fact, delimited sub-graphs. The interconnection structure for a "modified" Petri net denotes the input/output relations between the places (control state variables) and the data objects, using a notation similar to the complex bi-logic notation used in the GMB.[1] Unlike the GMB however, there is nothing equivalent to an interpretation domain in the "modified" Petri net model. That is, although a software component in a "modified" Petri net may produce a token on one of two output arcs, the effect of the data objects read by the software component upon making this decision are *not* represented.

Although the "modified" Petri net theory does introduce a distinction between the external and internal view of a software component, note that true hierarchy, in the sense of general control abstraction (e.g., recursion), is not possible since the external input/output specification of a software component refers to the places and data objects of the "outer" software component which passed control to it. Hence, a software component can not really be used by more than one "outer" software component, nor can it be used by that one "outer" component more than once.

Yau and Caglayan show how the concise bi-logic relation between places and software components can be transformed into a place/transition-net representation and then discusses how "modified" Petri nets can be analyzed using standard place/transition-net analysis (which is discussed in some detail in a later chapter) to determine the nature of the control aspect of the system. They concede that the harder questions concerning the data aspect of the system and the additional complexity that arises when timing augmentations are also considered, are not presently addressed by this type of analysis. In contrast, note that in terms of the GMB, the SARA system has achieved considerable success in examining these issues.

### *Numerical Petri nets*

Numerical Petri nets[SYMO80A] extend the place/transition-net model by adding some interesting features. First, tokens may be unique and may be identified as such in both enabling conditions and firing rules. Enabling conditions

[1] As a result, the author speculates that if one were to coalesce the arcs and places into multi-arcs, a graph intertwining the control and data domains of the GMB would result.

## Analysis

The survey of analysis methods begins by discussing the properties which might be interested in proving about a protocol. Petri net properties are then reviewed and an examination is made as to how these theoretical properties relate to protocol properties.

### *Protocol Specifications*

In the largest sense, protocol verification refers to proving that the specification of a protocol is able to achieve the properties listed in its associated service specification given a set of properties listed in an underlying service specification. It should be noted that although proving properties of the service specification is the end-goal of the specification process, this activity is not directly addressed. Instead, procedures are presented to prove that the protocol specification can achieve the service specification. No statement is made as to the correctness, completeness, or consistency of the service specification itself.

The properties that verification methods seek to establish fall into two categories: *safety* and *liveness*[BOCH80A]. Safety properties ensure that any actions taken by the system are correct actions while liveness properties ensure that the system will eventually perform these actions. For example, in the case of a connection establishment protocol, safety properties state that *if* the protocol reaches its "established" state, then both end-points of the connection are synchronized and cognizant of the state of the connection. In contrast, liveness properties state that the protocol *will* actually reach the "established" state.

There are several other related properties (to name a few): *completeness*, which deals with the ability of the protocol to cope with all possible situations (usually stated as "all possible inputs"); *deadlock-freeness*; and *progress*, which ensures that the protocol advances towards a goal state. The first two properties involve safety, while the latter property is concerned with liveness. In addition, properties specific to the protocol being examined may be verified as well. For example, in the case of a file transfer protocol, a property to be proven might include that translations between ASCII and EBCDIC encodings of characters occur correctly. As expected, analysis of these properties require an in-depth knowledge of the protocol and can not be so easily verified.

Finally, it should be pointed out that regardless of the protocol specification, it may not be possible to achieve the service specification due to an incorrect service specification or a faulty underlying service specification. For example, in the case of a packet voice system, if the underlying transport specification does not guarantee delivery of packets within a certain time, then a protocol relying on this underlying service will not be able to provide a high-quality of voice transmission.

As pointed out by Bochmann and Sunshine[BOCH82B], protocol specification techniques tend to fall into two types of representation: *state-transition* models and *programming language* models (algorithmic methods). Not surprisingly, approaches to protocol verification also fall into two categories: *reachability analysis* for transition-based techniques and *program proofs* for algorithmic techniques.

Verification by program proofs is achieved by deriving assertions about the operation of the protocol in terms of the state variables which compose the programs that specify the protocol. This approach to analysis can be used to prove statements about specific characteristics of the protocol, provided that concise assertions representing those characteristics can be stated. There has been some success in automating program proofs (e.g., GYPSY[GOOD78]). This reduces the time required to perform such proofs and, to a lesser extent, reduces the amount of designer guidance required during the proof process. There has also been considerable success in automating proofs based on the axiomatic properties of abstract data types in order to perform theorem proving (e.g., AFFIRM[SUNS82A]).

Verification by reachability analysis is achieved by building a graph of all states reachable for the protocol (known as state-space exploration) and is useful for verifying both safety and liveness properties. With the entire reachability graph built, the protocol can be checked for completeness by examining the graph for absence of states that could arise based on an input (e.g., message arrival, user request, or time-out). Similarly, terminal states in the graph can be marked as either final states (correct termination) or deadlock states. If no deadlock states are found, then the system is live. Similarly, if no states in the reachability graph represent undesirable states, then the system is "safe". Hence, reachability analysis is useful for verifying the general characteristics of a protocol. In order to perform complete analysis of this type however, the entire reachability graph must be

constructed. This introduces the problem of *state-space explosion*, which can make analysis quite expensive.

Several "short-cuts" have been suggested to reduce the state-space explosion problem. One approach is simply to specify less of the protocol and perform analysis on that smaller specification. In a similar vein, a knowledgeable protocol designer might personally direct the state-space exploration by indicating to a semi-automated verification system (such as SARA[RAZO81]) a particular path of the graph to explore. These approaches suffer from the drawback that they do not verify properties across the entire specification but, with proper designer guidance, can perhaps verify important properties along critical paths. Of course, this leads us to the second drawback, namely that verification tools using this style of reducing the size of the state-space can never be fully automated as they require an intimate knowledge of the behavior of the protocol.

A second approach is to group states and hence reduce the resulting state-space. If the specification technique supports *explicit hierarchy* (e.g., some form of control abstraction), then this grouping can occur quite naturally. For example, the CCITT often presents its specifications as finite state machines in which a single node may actually refer to another automaton. In contrast to the previous style of attacking the state-space explosion problem, this approach makes use of the (hopefully) natural organization of the protocol specification and can be more fully automated. A possible disadvantage though is that any tool which uses this approach must support a hierarchically based proof mechanism. In addition, depending on the structure of the specification, the grouping present may not be useful for determining certain properties of the system. That is, although properties local to a particular group may be easier to prove, nothing may be gained in terms of proving global properties.

If explicit hierarchy is not present, then the protocol designer must indicate which states can be coalesced into larger states (or sets of states) for the purpose of analyzing a particular property. It might be noted that states might be combined differently depending on the property to be verified. One approach for the designer to take, in order to aid the analysis process, is to group states together based on an associated global assertion. That is, the criterion for a state being added to a set of states is whether the assertion should be true when the protocol is in that

state. As expected, this can significantly reduce the state-space but again requires a great deal of knowledge of the protocol's behavior.

*Properties of Petri net models*

As suggested by Peterson[PETE81], there are six general properties to consider when analyzing a Petri net:

1. *Boundedness*

   If one views a correspondence between conditions in a system and places in a Petri net, such that a token resting in a place indicates that a certain condition is true, then it often makes no sense to state that more than one token can rest in that place. The property of $k$-boundedness is used to ensure that the number of tokens found in any given place at no time exceeds $k$, where $k$ is usually 1.

   It is an unfortunate coincidence that a $1$-bounded place is in Petri net terminology is called a *safe* place. This type of safeness has little to do with *safety* in the verification sense. To avoid confusion, this dissertation uses the term boundedness to refer to properties of Petri nets and the term safety to refer to verification properties.

   In terms of verification properties, boundedness of a place (or, in general, a net) is a correctness issue. If one chooses to model conditions as places, then boundedness is important if those conditions are to have meaning.

2. *Conservation*

   If one views a correspondence between resources in a system and tokens in a Petri net, then the number of tokens are often constant since the corresponding resources often are. To illustrate how boundedness differs from conservation, consider two opposing examples: In the first, a transmission medium may be bounded in the number of messages that it can have in transit simultaneously (the resources in the system are finite, and hence bounded). In contrast, consider a system which buffers information. In this case, a buffer is either free or busy and does not "disappear" (the resources in the system are constant, and hence conserved).

   As with boundedness, conservation is a correctness issue of the specification.

3. *Reachability*

   When fully analyzing a system, it is useful to know if it possible to reach a particular marking from a given initial marking. From our perspective, it might be instructive when trying to understand the nature of the protocol to be able to ascertain the reachability of one marking from another.

   Reachability of the net is related to the safety of the system. By examining the states that the system can reach, all states which violate safety properties of the system can be found. Hence, knowing the reachability of the net can aid in proving safety properties of the system.

4. *Liveness*

   It is important that a net always be live with respect to its initial marking. A *dead* transition, one which will never fire regardless of the sequence of transitions which fire in the system from an initial marking, represents an anomaly in the system. A marking for a net in which all transitions are dead represents a deadlock state, as discussed earlier. In addition, note that dead transitions may point to potentially serious design errors.

   Again, an unfortunate collision occurs between terms in verification and Petri net theory. To compound matters, liveness in the sense of a Petri net is strongly related to safety in the sense of verifying the behavior of the system. Early research in Petri net theory has shown that the problem of determining the reachability of states in a system is equivalent to the problem of determining the liveness of the net. Hence, knowing the liveness of the net can aid in proving safety properties of the system.

5. *Firing Sequences*

   If one considers the sequence in which transitions fired, rather than the sequences in which states are entered, then it is useful to ask if a given sequence of transition firings is possible.

   Having information as to the sequence of firings in the net can be useful in proving liveness properties of the system. For instance, if a cyclic firing sequence can be shown that always returns the net to a particular marking, then that marking can be viewed as a ***homing-state*** for the system.

6. *Equivalence*

It might be very useful to perform transformations on contour/transition-nets in order to achieve optimization with respect to a particular set of criteria while stilling retain certain key properties. The capability to perform such transformations might prove useful for validating implementations or experimenting with slightly differing specifications of the same protocol. In both of these cases, it may be desirable to prove that only certain properties remain consistent across the transformation.

Finally, it should be noted that nothing has been said about how these properties are proven. As suggested earlier, the standard approach is to build a reachability graph and then examine that graph for the desired information or behavior.

## Methods of Analysis

Several different methods of analyzing concurrent systems are now reviewed. In the survey and discussion that follows all analysis techniques are not exhaustively examined. Rather, focus is on those techniques which have been found to be particularly suitable for transition-based systems. In particular, the techniques chosen are:

- State-Space Analysis
- Structural Analysis
- Inductive Analysis
- Hybrid Approaches

To highlight the differences between the techniques discussed, a common example is used, i.e., a small example of a concurrent system which enforces two access mechanisms. Given $n$ co-operating processes, two types of access to a common resource are permitted: *shared* access, in which more than one process may manipulate the resource simultaneously; and *exclusive* access, in which only one process may manipulate the resource. This is, of course, the "readers and writers" problem.

In the simple solution presented, the issues of *indefinite postponement* and *fairness* are ignored. Rather, the properties to be proven are:

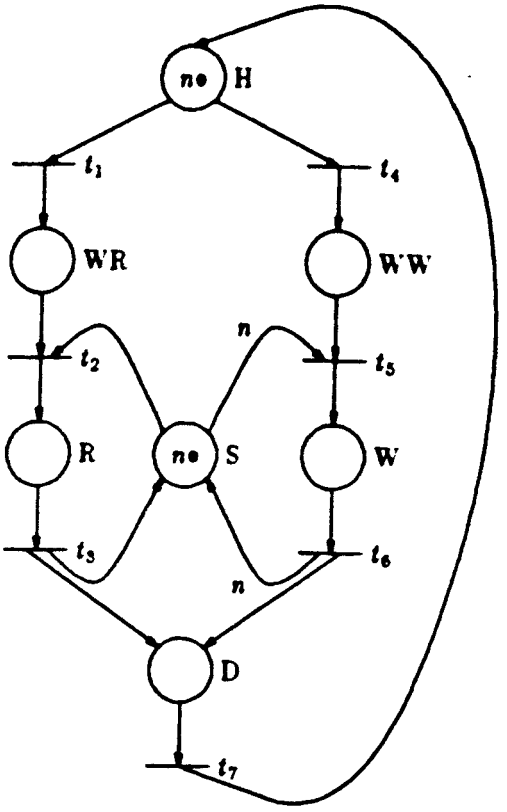



**Figure 2**
Place/Transition-net

- If at least one reader is active, then any processes wishing to read may immediately do so, and all processes wishing to write are blocked.
- If a writer is active, then all processes wishing to read or wishing to write are blocked.
- Providing that all processes spend a finite amount of time either reading or writing, the system will not deadlock.

A representation of the system modeled with a place/transition-net is shown in Figure 2. From the initial marking, each of the $n$ tokens in place H (the home state for the process) represents a process in the system. A token which fires transition $t_1$ is designated as a reader and proceeds to place WR (waiting to read). When at least one token is free in place S, then $t_2$ can fire and the process enters the reading state at place R. Sometime later the process completes its reading and fires transition $t_3$. As a result of firing $t_3$, a token is returned to place S, and the

process enters the done state (place D). This eventually leads back to the home state. Similarly, a token which fires transition $t_4$ is designated as a writer and proceeds to place WW (waiting to write). When all $n$ tokens are present in place S, then $t_5$ can fire absorbing $n$ tokens and placing the process in the writing state at place W. When the process finishes writing, it fires transition $t_6$ which returns the $n$ tokens to place S and causes the process to enter the done state. This simple net can model the minimal readers and writers system sketched above. It should be noted though that for this to be a "true" place/transition-net, $n$ should be replaced by a constant.

### *1. State-Space Analysis*

The survey begins by examining techniques that explore the state-space of the system. Two examples are presented, one applied to place/transition-nets, and the other applied to a "hybrid" model.

#### Place/Transition-Nets

Reachability analysis[PETE77] consists of building a graph of all states that can be reached by the system. In place/transition nets, the *explicit* marking (token distribution) of the net is the state of the system. Hence, the nodes of the reachability graph for a net containing $n$ places can be seen as $n$-tuples. Construction of the reachability graph begins with the initial marking of the net (represented as a single node). For each transition that is enabled in this state, a new edge carrying the name of the transition is added to the graph. The value of the node at the end of each edge, called *frontier* nodes, is the representation of the net's marking after the transition has fired. Before adding the node, a test must be made to determine if it already exists elsewhere in the graph. If so, then an edge is added to the already existing node (a frontier node is not introduced). States from which no transition is enabled are *terminal* nodes of the graph. This process is repeated for each frontier node created.

Hopefully, this process will stop when no frontier nodes remain. If the state-space for the net is *infinite*, then the reachability graph is infinite. Such nets are not $k$-bounded for arbitrarily large $k$. Unbounded places can be identified while building the graph. An incomplete reachability graph can still be built by replacing the unbounded component of the $n$-tuple with the special symbol $\omega$.

| $n$ | Number of States | CPU Seconds |
|---|---|---|
| 1 | 6 | 0.3 |
| 2 | 19 | 0.4 |
| 3 | 45 | 0.6 |
| 4 | 90 | 1.0 |
| 5 | 161 | 2.0 |
| 6 | 266 | 5.1 |
| 7 | 414 | 12.0 |
| 8 | 615 | 26.6 |
| 9 | 880 | 54.3 |
| 10 | 1221 | 106.9 |

**Table 1**

Summary of Reachability Analysis for Figure 2

Once a finite reachability graph has been constructed, properties of the place/transition net (and hence the system represented by the net) can be analyzed. Boundedness can be determined by examining each node for a component with value $\omega$. If no such node (state) exists, the net is known to be $k$-bounded for the largest value of $k$ found while traversing the tree. Conservation questions can be answered by comparing the total number of tokens in each state to the number of tokens in the initial state.

Of course, the most interesting properties are liveness and reachability. Providing that the value $\omega$ does not occur as a component of a node, then the complete state-space is known for the place/transition-net being analyzed. Terminal nodes can then be examined to determine if they represent proper termination of the system or if they represent deadlocks. Similarly, if no branches in the tree are labeled with the name of a given transition, then that transition will never fire. This usually indicates a design problem in the system represented by the net. Since the presence of $\omega$ indicates an infinite number of states which are represented by a single node, there is not a one-to-one correspondence between the state-space of the net and its corresponding reachability tree. In such cases, the complete state-space for the net is not known, and some reachability questions are undecidable.

In order to build a reachability graph for the model shown above, the value of $n$ must be fixed. Table 1 shows how the number of states grows as a function of $n$. For large values of $n$, the reachability graph, although finite, is unmanageable.

| | Incidence Matrix | | | | | | | | Invariants | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $M_0$ | $i_1$ | $i_2$ |
| H | -1 | | | -1 | | | 1 | $n$ | 1 | |
| WR | 1 | -1 | | | | | | | 1 | |
| R | | 1 | -1 | | | | | | 1 | 1 |
| WW | | | | 1 | -1 | | | | 1 | |
| W | | | | | 1 | -1 | | | 1 | $n$ |
| D | | | 1 | | | 1 | -1 | | 1 | |
| S | | -1 | 1 | | $-n$ | $n$ | | $n$ | | 1 |

**Table 2**

Incidence Matrix and Invariants for Figure 2

*2. Structural Analysis*

The next method examined does not enumerate the state-space of the system but rather attempts to derive invariants about the system's behavior from its structure. First, matrix analysis is presented and applied to place/transition-nets. Then it is show how this technique can be applied to various Petri net models. Finally, ways to transform the structure of a system in order to facilitate the analysis is discussed.

**Place/Transition-Nets**

A place/transition-net can be represented as an *incidence* matrix. The rows of the matrix refer to places in the net while the columns refer to transitions in the net. An entry in the incidence matrix, $[P_i,t_j]$, indicates the change in the number of tokens present in place $P_i$ when transition $t_j$ fires. A weakness of this representation is that *self-loops* can not be shown in the incidence matrix. A self-loop is a transition which is connected to a place by *both* input and output arcs. The input and output arcs negate each other, usually resulting in a zero entry in the matrix. A net with self-loops can be correctly represented by the incidence matrix if additional places and transitions are added to the original net.

The incidence matrix in Table 2 corresponds precisely to the net in Figure 2. Table 2, in addition to containing the incidence matrix for the net in Figure 2, also has a column (labeled $M_0$) to indicate the initial marking of the system along with two columns marked "Invariants." After the incidence matrix has been constructed, invariant aspects of the behavior of the system can be determined by applying some

straightforward linear algebra manipulations to the matrix. Two invariants, $i_1$, and $i_2$, can be derived.[2] By forming the product of the initial marking, $M_0$, with each invariant, equations can be derived which describe the invariant behavior of the system. In expressing invariant equations, $m(P_i)$ refers to the marking of place $P_i$. In the case of place/transition-nets, $m(P_i)$ is a non-negative integer referring to the number of tokens in $P_i$. In Petri net models in which tokens are distinguishable, $m(P_i)$ is the orthogonal sum of the tokens present it $P_i$, and $|m(P_i)|$ refers to the ordinality of the tokens in place $P_i$.

The first invariant,

$$m(H) + m(W'R) + m(R) + m(W'W') + m(W') + m(D) = n, \qquad (i1)$$

is interpreted as meaning that there are always $n$ processes in the system. The second invariant,

$$m(R) + n * m(W') + m(S) = n, \qquad (i2)$$

has a more detailed meaning: if there are no processes writing, then up to $n$ processes can be reading; otherwise, if one process is writing, then no other process can be reading or writing. To complete the proof, the system must be shown to be free from deadlock. Using the two invariants, the proof is quite simple: Consider the sum

$$m(H) + m(R) + m(W') + m(D).$$

There are two cases. For the first case,

case 1: $$m(H) + m(R) + m(W') + m(D) > 0,$$

at least one of transitions $t_1$, $t_3$, $t_4$, $t_6$, or $t_7$ is firable. For the second case,

case 2: $$m(H) + m(R) + m(W') + m(D) = 0,$$

from equation $(i1)$, it is known that

$$m(W'R) + m(W'W') = n,$$

and from equation $(i2)$, it is known that

$$m(S) = n.$$

[2] Actually, any linear combination of these two invariants is an invariant of the system.

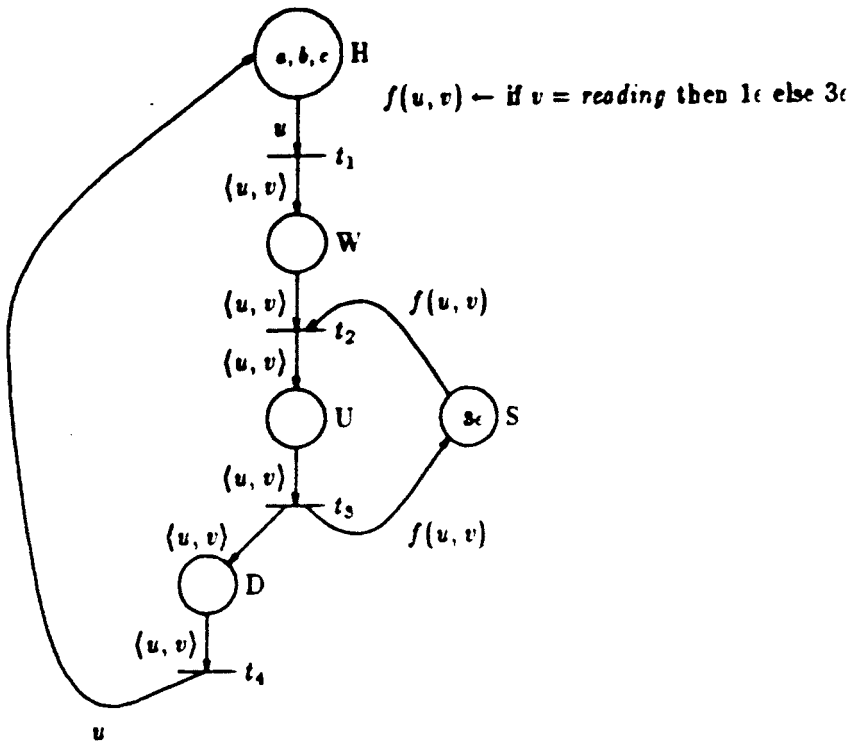



**Figure 3**

Predicate/Transition-net

These two conditions imply that either transition $t_2$ or transition $t_5$ is firable. Hence, regardless of the state of the system, at least one transition is firable which means that the system cannot deadlock.

### Predicate/Transition-Nets

Genrich and Lautenbach[GENR81] derives an approach using *S-invariants* based on linear algebra techniques to prove invariants about predicate/transition-nets. The generality of the method is established by applying it to successively more detailed predicate/transition-net representations of the same system. In short, it is demonstrated how to introduce variables representing sets of values and quantifiable variables into the equivalent incidence matrix representation of a Petri net model and how invariants of the net can be derived based on the more complicated incidence matrix.

To return to our short example, let us now suppose that three specific processes are to be modeled, *a*, *b*, and *c*, which are reading and writing. The predicate/transition-net in Figure 3 models the system using two quantities: *u*

| | Incidence Matrix | | | | | Invariants | |
|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $M_0$ | $j_1$ | $j_2$ |
| H | $-u$ | | | $u$ | $a+b+c$ | $\langle u,v\rangle$ | |
| W | $\langle u,v\rangle$ | $-\langle u,v\rangle$ | | | | $u$ | |
| U | | $\langle u,v\rangle$ | $-\langle u,v\rangle$ | | | $u$ | $f(u,v)$ |
| D | | | $\langle u,v\rangle$ | $-\langle u,v\rangle$ | | $u$ | |
| S | | $-f(u,v)$ | $f(u,v)$ | | $3\epsilon$ | | $\langle u,v\rangle$ |

**Table 3**

Incidence Matrix and Invariants for Figure 3

which refers to a particular process ($a$, $b$, or $c$), and $\langle u,v\rangle$ which refers to a particular process that has decided to use the access denoted by $v$ (either *reading* or *writing*). Each edge of the net is labeled with one of these quantifiers. From the initial marking, all three processes are in place H. When one of these processes wishes to either read or write, transition $t_1$ fires, binding an access-type to $v$ in the pair $\langle u,v\rangle$, and the process enters the wait state (place W). When transition $t_2$ fires, the process proceeds to place U (*reading* or *writing*, depending on $v$). In order for $t_2$ to fire, however, sufficient tokens must be present in place S. The number of tokens in S required to make $t_2$ firable depends on the binding of $v$ for the token in place W. If the process wants to read, then a single token (denoted by $\epsilon$) will suffice, otherwise $3\epsilon$ tokens (all the tokens that could ever be present in place S) are required. When the process is done *reading* or *writing*, transition $t_3$ fires which returns the appropriate number of tokens to place S. The process enters the done state (place D), which leads back to the home state, and in so doing, discards the binding for $v$.

Table 3 shows the incidence matrix for the net in Figure 3. Since the semantics of the net does not deal with scalar place variables (tokens are distinguishable), the marking function $m$ does not return a scalar, but rather a combination of the three quantities: $u$, $\langle u,v\rangle$, or $\epsilon$ (that is, $m(S)$ may return $3*\epsilon$). Furthermore, the $+$ and $*$ operators do not have the same meaning as far as $\langle u,v\rangle$ and $a$, $b$, or $c$ are concerned. $*$ is thought of as acting as a quantification operator and $+$ as acting as a set union operator. The invariant equations are

$$\langle u,v\rangle * m(H) + u * m(W) + u * m(U) + u * m(D) = \langle u,v\rangle * (a+b+c), \quad (j1)$$

and

$$f(m(U)) + m(S) = 3\epsilon. \qquad (j2)$$

The first invariant is interpreted as meaning that each of the processes, $a$, $b$, and $c$, are in one of the places H, W, U, or D, and when they are in the home state, any (future) binding for $v$ is valid. The second invariant is more complicated: regardless of the processes involved, if no processes have $v$ bound to *writing* in place U, then any process may have $v$ bound to *reading* in place U; otherwise, if a process has $v$ bound to *writing* in place U, then no other process can be present in place U, regardless of the binding it has for $v$. To show freeness from deadlock, the same cases discussed above are considered for the sum

$$m(H) + m(U) + m(D).$$

For the first case,

case 1: $$m(H) + m(U) + m(D) > 0,$$

at least one of the transitions $t_1$, $t_3$, or $t_4$ is firable. For the second case,

case 2: $$m(H) + m(U) + m(D) = 0,$$

from equation $(j1)$, it is known that

$$u * m(W) = \langle u, v \rangle * (a + b + c).$$

and from equation $(j2)$, it is known that

$$m(S) = 3\epsilon.$$

These two conditions imply that transition $t_2$ is firable, and hence the system is free from deadlock.

### Place/Coloured-Nets

Jensen[JENS81], introduces a method for proving invariants for coloured Petri nets. The method is similar in spirit to the invariant method reported by Genrich and Lautenbach[GENR81], but differs in one important respect: the incidence matrix and resulting invariants are expressed in terms of explicit sets of tokens (e.g., *colours*) rather than individual groupings of particular types of

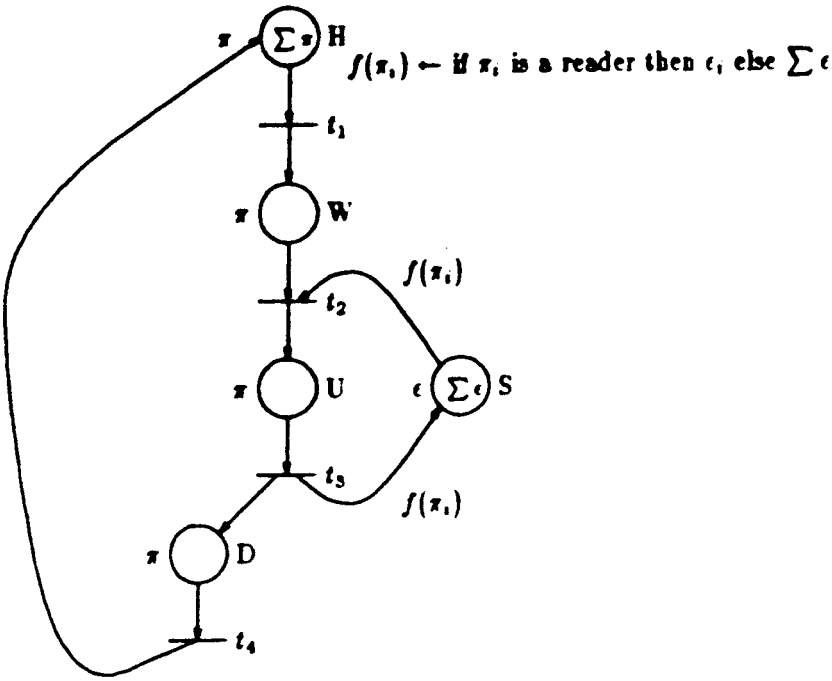



**Figure 4**

Place/Coloured-net

tokens. This is, of course, the fundamental difference between coloured Petri nets and predicate/transition-nets.

To return to our short example, let us suppose that two kinds of processes are to be modeled: readers and writers. The coloured Petri net in Figure 4 can be constructed to model the system using two colour sets: $\sum\pi$ which refers to the set of all processes, and $\sum\epsilon$ which refers to the resources (counters in our context) in the system. The further distinction will be made that each member of the set $\sum\pi$ has an attribute, *ID*, which indicates if the process is a reader or a writer. Finally, suppose that there be a one-to-one correspondence between the set $\sum\pi$ and the set $\sum\epsilon$ such that given any member of $\sum\pi$, say $\pi_i$, one can identify the corresponding member of $\sum\epsilon$, say $\epsilon_i$. Each place of the net is labeled with one of these colour sets.[3] From the initial marking, all processes are in place H (the home state for the process). When one of these processes becomes active, transition $t_1$ fires, and the process enters the wait state (place W). When transition $t_2$ fires, the process proceeds to place U and begins to read or write, depending on its *ID*.

[3] This label occurs to the left of each place in Figure 4.

There are two cases. For the first case,

case 1: $$m(H) + m(U) + m(D) > 0,$$

at least one of the transitions $t_1$, $t_3$, or $t_4$ is firable. For the second case,

case 2: $$m(H) + m(U) + m(D) = 0,$$

from equation (*k1*), it is known that

$$m(W) = \sum \pi,$$

and from equation (*k2*), it is known that

$$m(S) = \sum \epsilon.$$

These two conditions imply that transition $t_2$ is firable. Since at least one transition is firable, regardless of the state of the system, the system is free from deadlock.

### Graph Model of Behavior

The application of structural analysis to the UCLA Graph Model of Behavior is considered. Much work has been devoted to proving formal properties of systems represented by the GMB. The majority of previous research has centered on proving aspects of the control portion of systems represented by the GMB. The usual method relies in expressing the control graph as a system of transformation expressions and then applying a powerful reduction algorithm to the expressions. The control set of the system is then derived from the reduced set of transformation expressions. This latter activity is called *control-flow analysis*.

To return to our short example, the place/transition-net in Figure 2 is converted to a GMB control structure by following a straightforward procedure: each place in the net (and the arcs connected to it) is replaced with a *multi-arc*, and each transition is replaced with a *node*. The resulting control graph (also known as a UCLA graph) is shown in Figure 5.[4]

[4] The reader should note that Figure 5 does not conform to the method used by most authors when drawing the control graph in a GMB representation: normally the nodes are labeled with names of the form *Ni*, as in N1, and the arcs are labeled with names of the form *Aj*, as in A1. In addition, the input and output arcs for the graph are usually labeled S and X, respectively. In order to aid the reader in understanding the mapping between the structures in Figures 2 and 5, the names of places/transitions and multi-arcs/nodes have been preserved.

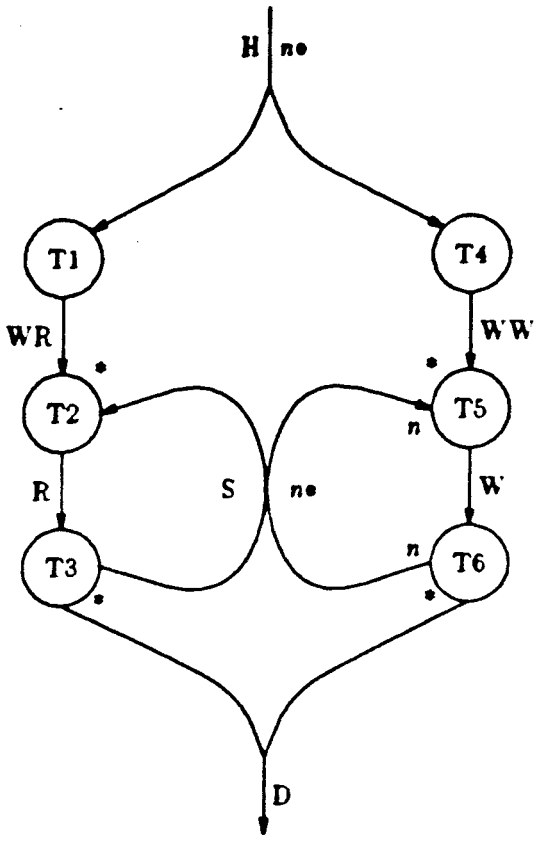



**Figure 5**
Graph Model of Behavior

To verify that the system is free from deadlock, it must be shown that the system is *properly terminating* (PT) when $n$ tokens are introduced onto the starting arc for Figure 5. In order to show that the control graph in Figure 5 is PT(n), It is first necessary to prove that the transformation

$$H(n), S(n) \rightarrow D(n), S(n)$$

holds for the control graph in Figure 5. This transformation expression indicates that all $n$ processes will terminate (being free from deadlock), and that the number of internal tokens is preserved (a weaker statement than the actual properties to be proven). The set of transformation expressions representing the structure in

Figure 5 is:

$$\begin{aligned} H &\rightarrow W'R \\ W'R, S &\rightarrow R \\ R &\rightarrow S, D \\ H &\rightarrow W'W' \\ W'W', S(n) &\rightarrow W' \\ W' &\rightarrow S(n), D \end{aligned} \qquad (l1)$$

Rather than prove that the control graph in Figure 5 is PT(n), it is transformed into an equivalent SESX structure which is then proven to be PT(1). First, the interior tokens from the system are removed by adding two transformation expressions:

$$\begin{aligned} S' &\rightarrow H(n), S(n) \\ D(n), S(n) &\rightarrow X' \end{aligned} \qquad (l2)$$

and the I/O assertion

$$S' \rightarrow X'$$

must be proved.

Second, a series of *strong reductions*[RAZ081, pp 37–43] is performed on our set of transformation expressions in $(l1)$. This yields:

$$\begin{aligned} S' &\rightarrow H(n), S(n) \\ H &\rightarrow W'R \\ W'R, S &\rightarrow S, D \\ H &\rightarrow W'W' \\ W'W', S(n) &\rightarrow S(n), D \\ D(n), S(n) &\rightarrow X' \end{aligned} \qquad (l3)$$

This set of transformation expressions cannot be further reduced. If the set of transformation expressions in $(l1)$ and $(l2)$ could be reduced to the single expression

$$S' \rightarrow X',$$

then our proof would have been done.

Since this is not the case, an alternate approach is to derive some invariants, based on the transformation expressions in $(l1)$. In particular, consider

$$n * S' + H + W'R + R + WW' + W + D + n * X' = n, \qquad (l4)$$

and

$$n * S' + R + n * W + S + n * X' = n, \qquad (l5)$$

which are quite similar to the invariants $(i1)$ and $(i2)$ with the additions of $n * S'$ and $n * X'$ to reflect our augmented set of transformation expressions. Examining the transformation expressions in $(l1)$ and $(l2)$, shows that if $(l4)$ holds prior to the application of any transformation expression in $(l1)$, then $(l4)$ holds after the application of that transformation expression. The same is true for $(l5)$. Hence, these are invariants on the number of tokens on the multi-arcs in Figure 5.

Two points are of interest here. First, the correspondence between places/transitions in place/transition-nets and multi-arcs/nodes in the GMB allows us to use invariants derived from one model with the other. To the author's knowledge, a general invariant technique on transformation expressions has not been developed with the same power as the linear algebra techniques on incidence matrices. This is not really a shortcoming, though, as each transformation expression corresponds to a column in the incidence matrix for the set of transformation expressions describing a particular control structure in a GMB. Second, these invariants (or derivatives of them) could not have been derived for the transformation expressions in $(l3)$. Although the systems represented by both sets of transformation expressions are identical, there is a loss of information when going from $(l1)$ to $(l3)$. Hence, although reduction is very useful if one is applying control flow analysis, it may actually *hinder* proofs that make use of invariant properties of the control graph.

### *3. Inductive Analysis*

An alternative to automatic derivation of invariants is to use invariants supplied by the system's designer. First, this method is applied to a very general model of concurrent computation and then the method is used to augment reachability analysis.

It is important to emphasize the difference between invariants that can be derived automatically from the structure of a net and those invariants which a system designer suggests. In particular, the former type of invariants describe general properties of the system (e.g., boundedness and possibly freedom from deadlock) and are strictly limited to equalities that can be derived from the

incidence matrix. In contrast, the invariants proposed by the designer are usually system-specific and can include inequalities as well (not being limited to the incidence matrix). In most cases, these properties, owing to their specialized nature, do not fall out of the topology of the net.

**Predicate/Action-Nets**

Keller[KELL76] presents two models of parallel computation: a *conceptual* model and a *presentational* model. The conceptual model is a transition system which uses a binary relation to map from one state to another. Although not useful for modeling particular systems, the conceptual model has sufficient generality to be useful for high-level reasoning about general concurrent systems. In contrast, the presentational model, or predicate/action-net in the terminology of Diaz[DIAZ82], resembles an augmented Petri net and can be used for representing a concurrent system. The presentational model presented by Keller bears a strong resemblance to the contour/transition model. In particular, since colorful tokens in contour/transition-nets represent execution contexts, just as place variables represent multiple instruction pointers in the presentational model, the two models are very similar. This view is strengthened when the presentational model is augmented with "local variables." Although the contour/transition model has strict rules regarding data access (contour scoping), the same notion is being addressed by the two models.

To prove properties of the presentational model, Keller uses an *induction principle*. Put simply, if a predicate holds in every state reachable from the initial state, $q_0$, of the system, then that predicate is said to be $q_0$-*invariant*. Furthermore, a predicate is said to be $q_0$-*inductive*, if:

1. a predicate holds in the initial state of the system;
2. the predicate holds in any given state of the system implies that the predicate holds in any state immediately reachable from that given state.

Finally, Keller shows that if a predicate is $q_0$-*inductive*, then it is $q_0$-*invariant*. This means that the invariance of a property can be proven via an inductive proof method. This result is important since it means that the entire reachability graph of a system need not be constructed in order to prove properties of the system.

The principal drawback of the invariant method is that the designer must explicitly determine what properties should be invariant in the system: *the construction of invariants requires intimate knowledge of the system.*

One important augmentatiton of predicate/action-nets, which is now considered is the "distinct process" extension to the presentational model. This extension allows the predicate/action-net to model systems involving unique processes with local variables. Keller purposely avoids all issues of scoping and allocation of variables and simply classifies variables as one of two types: *global* variables which are accessible by all processes present in the system; and *local* variables which are identically named by all processes present in the system but which exist uniquely for each process. Furthermore, the assumption is made that a single token exists for each process in the system (process creation and termination are not considered). Invariants are now expressed in terms of three quantifiers: global variables, local variables in the context of a particular process, and tokens representing particular processes. Proving properties using this technique follows along the usual lines: first, specific properties of the system are isolated to be proven; second, these properties are expressed as invariants; and third, these invariants are then proven by showing that the invariant holds in the initial marking for the net, and, for each transition in the net, if the invariant holds prior to that transition firing, then the invariant holds after that transition has fired.

Since analysis using the presentational model is discussed in greater depth in Chapter 4, the short example of the readers and writers is not presented again here.

**Predicate/Transition-Nets**

Berthelot and Terrat[BERTHE82] use predicate/transition-nets to model parts of the European Computer Manufacturer Association transport protocol[ECMA]. A set of *invariant assertions* are presented for control states in the net, which, if proven, show that the net is free from deadlock and that data is transmitted from one peer to another without corruption, loss, duplication, or re-ordering. These properties are then proven by: first, deriving a set of global predicates (*invariant assertions*) for the system, second, demonstrating that the predicates hold for the initial state; and third, showing for each transition in the system that the

transition does not alter the predicate and then using the predicates to show that the invariants hold at the appropriate control points.

The analysis presented follows the nature of predicate/transition-nets very well: the range of values that each variable may take is fully declared, and, since the predicate/transition net is actually a highly-compacted place/transition net, the values of "variables" can be considered only with respect to each transition that refers to those variables on its input or output arcs.

### *4. Hybrid Approaches*

Combinations of these methods have been applied to models which are more computationally powerful that place/transition-nets. Two of these methods are examined here.

#### Predicate/Action-Nets

Bochmann and Gecsei[BOCH77] attempt to unify reachability analysis and the assertions method by augmenting Keller's the presentational model. In essence, the presentational model is extended by permitting decomposition of a system represented by the model into separate *subsystems* (one for each peer in the system) and by requiring that each subsystem consider data in a context independent from all other subsystems (i.e., each subsystem has its own "private" collection of global variables). To achieve communication between subsystems, the notion of the *distantly initiated action* is presented. Bearing a shadowy reflection of boundary transitions in the contour/transition model, the distantly initiated action is a form of message passing, similar in spirit to a remote procedure call. The specifics of distantly initiated actions is quite simple: as a part of the action of a transition firing, the execution of a distantly initiated action, specifying the name of the action and any value-passed parameters, is requested; the action is then executed for the other peer some finite time later, with the one provision that the execution of each distantly initiated action is considered an atomic operation. Note that although requests to execute distantly initiated actions are queued in some sense, no temporal relation exists between requests in the queue: *distantly initiated actions may be executed in a different order than the order in which they were queued.*

The analysis technique reported by Bochmann and Gecsei seeks to prove the full correctness of systems by establishing the absence of deadlocks, liveness, cyclic behavior, and partial correctness properties of the system. To ascertain the first three properties, reachability analysis is performed to construct a single reachability structure for the system. The analysis is based on the control structure of each subsystem, intimate knowledge as to the order in which certain transitions can be fired and distantly initiated actions can be executed, and also a small number of assertions on program variables. In short, the reachability analysis is guided by a knowledge of the expected behavior of the system. Correctness of the system is expressed as an assertion on the state variables of the system when the system is in a particular control state. Partial correctness is proven when the existence of a *complete* state (one which implies the correctness assertion) is shown to exist, and full correctness is proven when it can be shown that a complete state can be reached in a finite amount of time.

At this point, note that the subsystem approach suggested by Bochmann and Gecsei may not be particularly well-suited to modeling our short example of the readers and writers, since the author can not think of a simple way of using distantly initiated actions to help represent the system. As a result, the short example of the readers and writers is not presented again here.

### Graph Model of Behavior

Razouk[RAZO81] presents a lengthy discussion of analysis of systems represented with the GMB, in particular computer communication protocols, with a heavy emphasis on using SARA to mix control-flow analysis with execution (simulation) of the system represented by the GMB. By using control-flow analysis, it is possible to verify proper termination (to a limited extent), boundedness, and liveness properties of GMB systems. Control-flow analysis suffers from the same restrictions as reachability analysis since it does not consider the relation between the control and interpretation domains of the GMB. In short, if control-flow analysis declares a graph to be properly terminating, then the interpretation domain is inconsequential to this property. Otherwise, if control-flow analysis declares a graph to lack this property, then a more comprehensive proof is required which takes the interpretation domain into account.

To meet this problem, Razouk[RAZO81] presents an invariant method based upon the general method reported by Keller: first, critical transitions, which represent the system going from a desirable to an undesirable state, are identified; second, this knowledge is used to form invariants which, if proven, show that these critical transitions can not occur in the system; and third, these invariants are then proven by showing that the invariants holds for the initial state of the GMB, and, that at each point of the interpretation domain, considered atomically, if the invariant holds prior to the interpretation being applied, then the invariant holds after the interpretation has been applied. Razouk then shows that by clever use of the topology of the data domain, the nodes in the GMB which do not affect the invariant can be isolated and removed from the invariant analysis. This allows partial correctness proofs to be more easily formulated since the time to perform the proofs has been reduced.

In contrast to the work in Razouk's work, Shapiro[SHAP83] attempts to extend analysis of the GMB to include the data and interpretation domains more directly. To achieve these extensions, a system represented by the GMB is represented by an equivalent GMB in which all iteration and parallelism present in the interpretation domain is shifted to the control domain. This process is straightforward and can be done without loss of generality or descriptive power in the GMB. Furthermore, timing considerations are ignored in the interpretation domain. With these two limitations, it is possible to view each statement in the interpretation domain as an indivisible action, and by doing so, Shapiro is able to claim the important advantage of true parallelism being re-introduced into the control domain for the purposes of analysis, inasmuch as more than one node in the control domain may be simultaneously active. Previous work on verifying properties of systems represented by the GMB that made use of the invariant method required the limitation that only a single node in the control domain be active at a given instant (i.e., that nodes fire atomically).

The argument is made that using inductive invariants on the control graph alone is insufficient for complete verification and that a proof method which includes the data and interpretations domains is required. To introduce a basis for this, the use of predicate transformations are proposed. In short, assertions are placed at various control points in a system represented by the GMB. The parts of

the interpretation domain associated with those control points are then examined, using symbolic execution, to see how they transform the predicate. To make this type of analysis less expensive, Shapiro proposes to exploit the natural separation of the control and interpretation domains in the GMB.

### Remarks

One theme which emerges from this discussion is that of the trade-off between modeling power and ease of analysis. By using place/transition-nets to model a system, very powerful invariants of that system can be automatically generated. In contrast, using numerical Petri nets allows very complex systems with complicated decisions structures to be represented, at the expense of introducing tremendous complexity in terms of analysis. This phenomenon is due to the fact that models such as numerical Petri nets and predicate/action-nets are equivalent in computational power to Turing machines. Fortunately, the work of Keller presents a method for proving invariants of concurrent systems expressed as predicate/action-nets, although, unlike the analysis methods for place/transition-nets, these invariants must be generated by the designer of the system.

Despite their differences in computational power, the models examined share some common weaknesses which are summarized here. A central weakness is that these models are not able to easily represent software systems since the models are generally static in nature while the systems are dynamic. (In contrast, these models, and particularly Petri nets, enjoy good success in representing hardware systems which are more static in nature.) Most of this inability arises from a lack of good abstraction facilities. Mechanisms for abstraction of control, such as recursion and hierarchical ordering, are not present. Similarly, mechanisms for abstraction of data, such as scoping (and in particular "well-disciplined" scoping), are also lacking in these models.

Having presented some of the models proposed for representing concurrent systems and their associated analysis techniques, a new transition-based model is now introduced, the *Contour/Transition Model*. This model combines a powerful modeling capability along with control and data abstractions to allow concise representations and yet is still amenable to analysis techniques.

# CHAPTER 3
# The Contour/Transition Model

The contour/transition model combines the data-handling features of programming languages (the contour model[JOHN71]) with the control-flow features of transition-nets (Petri net theory[PETR62]) in a hierarchical and ordered fashion.

## Contour/Transition-Nets

A contour/transition-net is (yet) another extension to the place/transition-net model.[5] Our discussion begins by describing the elements that compose each net and follow this by describing other aspects of the contour/transition model.

### *Topology*

A contour/transition-net is a directed bi-partite graph populated with four types of nodes: ***places***, which hold tokens; ***transitions***, which absorb and produce tokens; ***named nets***, which instantiate the execution of another contour/transition-net; and ***named subnets***, which denote the substitution of another contour/transition-net. Arcs starting from a place or named net and leading to a transition are called *input* arcs, while arcs starting from a transition and leading to a place, named net, or subnet are called *output* arcs.

### *Tokens*

Tokens traverse the net. The tokens found in the contour/transition model are not unique. Instead, all tokens have a single attribute — a *color*. All tokens of the same color are indistinguishable from each other. Each color is mapped

[5] In earlier research, the author used the term *structured Petri net* to refer to the transition-based model described in this chapter. After much thought, the term contour/transition model was adopted instead. This change is not meant to slight Petri nets in any fashion. Quite the contrary: the author has great admiration for the simple yet powerful ways that place/transition-nets use to represent concurrency, conflict, and synchronization. However, owing to the large number of transition-based models that incorporate the term "Petri net" into their name, the author felt that it would be a good idea to avoid confusing the literature with another model claiming to be a close relative of the place/transition-net model.

to a contour block, which represents a private resource space for each execution (instantiation) of a named net.

Contour blocks contain bindings for variables and a static link which is a pointer to a previous (scoping) contour. At this point, the restriction is made that all colors in the system are independent: there are no shared variables or shared contours between named nets or instantiations of named nets. Hence each static link is null. When searching for a variable in the context of a particular token, the contour block associated with the token's color is examined. If the variable is present, then the binding for the variable has been ascertained. Otherwise, the variable is undefined in the context of the token.

*Colors*

The color attribute of a token has an ordinal value from the set of all colors. The above restriction may now be relaxed by stating that each N-peer has its own color space from which colors may be generated. To permit this, each N-peer has an associated color-generator which produces new colors for the N-peer when the generator is *incremented.* The generator has a *current color value* which is set to the color produced by the last increment operation. The color-generator is accessible by all contour/transition-nets executing for a particular N-peer.

In addition to the colors that may be produced by the generator, there is a special color — the *blank* color. Tokens with the blank color differ from other tokens in one important way: *each blank token has a set of variables associated with it and is unique.* The same variable may have different values, depending on which blank token is being used to delimit the context. The static link in the context of a blank token is always null.

*Enabling Predicates*

Each transition has associated with it an enabling predicate which specifies the condition under which that transition is permitted to fire. Only one transition in a contour/transition-net may fire at a given instant. If more than one transition is enabled, then a non-deterministic choice is made as to which transition is actually permitted to fire.

A transition's enabling predicate is a single, possibly very complex, boolean expression. The enabling predicate may specify the number of tokens required

on each input arc (usually 1). Only tokens with the same color are considered when the expression is evaluated. After evaluating the enabling predicate of all transitions in the context of all colors, a set of enabled transitions is generated which contains pairings between particular transitions and those colors which satisfy the enabling predicate of the transition. A non-deterministic choice is then made as to which transition will fire and which color is permitted to initiate the firing of that transition. Tokens which are chosen to initiate the firing of a transition are known as *eligible* tokens.

In addition to considering the marking of the net, the enabling predicate is allowed to reference variables in the context of the eligible tokens. No memory, other than those variables considered in the context of the eligible tokens, may be referenced by the enabling predicate. Furthermore, testing of the enabling predicate must not promote any sort of change in state (i.e., it must be free of side-effects). This requirement permits the enabling predicate for all transitions in the system to be examined simultaneously, or in any order or combination, with the same final outcome.

#### *Firing Actions*

Each transition has associated with it a set of firing actions. These actions may be ordered to allow sequential processing. Although several activities occur when a transition fires, the totality of a transition firing is considered to be atomic (i.e., once a transition begins to fire, no other transition is considered to be enabled).[6]

The firing actions of a transition compose a set of operations that perform four tasks: first, a set of eligible tokens is removed from their input places; second, a determination is made as to the number of tokens (zero or more) that are to be placed on each output arc; third, these tokens are *introduced* onto the selected output arcs; and, fourth, variables in the context of the introduced tokens are modified, and the introduced tokens are moved to their output places.

### Removal of Eligible Tokens

The very first action to occur is the removal of the eligible tokens from their input places. Although these tokens are now inaccessible to all other transitions,

[6] **Timing considerations, discussed momentarily, may modify this perception somewhat.**

during the remainder of the firing of the transition they may be referenced by the firing actions.

**Selection Rules**

After the eligible tokens have been removed, the selection rule is consulted. The selection rule is an expression which is evaluated once, in the context of the eligible tokens. This rule specifies which of the output arcs receive introduced tokens and the number of tokens which are introduced. Associated with each output arc is a predicate, the *selection predicate*, and a constant. The selection rule is composed of the selection predicates and associated constant for each output arc of the transition. Normally, this is used to provide decision-logic in a net. In addition to opposing predicates (i.e., $\rho$ and $\neg\rho$) that specify which of two output arcs will receive a token, the selection rule is easily able to support a switch-decision capability, including a default path (if no cases satisfied), providing that the selection rule evaluates all cases consistently (i.e., without side-effects).

**Construction Rules**

After the distribution of the introduced tokens has been specified by the evaluation of the selection rule, these introduced tokens are *constructed*, according to a construction rule, and placed on the appropriate output places. At present, the restriction is made that the construction rules require that the introduced tokens have the same color as the eligible tokens. Furthermore, if the transition is fed by an entry place for a named net, the construction rules are also permitted to introduce variables into the new contour. When transitions types are discussed, the notion of the construction rule will be developed in greater detail.

**Manipulation Rules**

After the introduced tokens have been constructed, they are manipulated. This means that the manipulation rules, a set of program statements, are executed to change the value of variables in the context of the introduced tokens.[7] These statements are allowed to reference variables in the context of the eligible tokens. No memory, other than those variables considered in the context of the eligible or introduced tokens, may be referenced by the firing actions.

[7] **Each output arc has its own set of manipulation rules. If a transition has more than one set of non-empty manipulation rules, then regardless of the order of execution of each set, the state of the introduced tokens should be consistent.**

After the manipulation rules have been completed, the introduced tokens are moved from the output arcs to the corresponding output places, and the transition is considered to have fully fired.

*Timing*

The contour/transition model supports temporal augmentations which further modify the behavior of transitions. In addition to an enabling predicate and some firing actions determining if and how a transition fires, each transition has an *enabling time* and a *firing time.*

**Enabling Times**

In order for a transition to be ready to fire, it must have its enabling predicate constantly satisfied for the amount of time indicated by its enabling time. The transition must be continuously enabled during this period. Once the enabling predicate fails to be satisfied, the transition is disabled. After the enabling time has expired, the transition is ready to fire. This capability allows the notion of time-outs and similar concepts to be represented in the contour/transition model. Most transitions will have an enabling time of zero, i.e., as soon as the enabling predicate is satisfied, the transition may fire.

**Firing Times**

After executing the manipulation rules, but prior to the movement of the introduced tokens to their output places the firing time of the transition must expire. This requirement forces all of the critical activities of the firing actions (in particular the execution of the manipulation rules) to occur atomically (before other transitions may fire and modify the state), but still permits the use of a delay before the introduced tokens are considered part of the state of the contour/transition-net. Most transitions will have a firing time of zero which makes the entire firing process appear to be atomic.

To permit more interesting simulations, the enabling time and firing time of a transition need not be constant throughout the execution of the system. Rather, the enabling time for a particular set of eligible tokens may be calculated when the transition is first enabled. Furthermore, once the enabling time expires and the transition begins to fire, the firing time may be calculated immediately after the execution of the transition's manipulation rules.

### *Transition Types*

The preceeding discussion did not allow colors to be created or destroyed nor did it permit colors to interact. Special transitions are now introduced which enforce rigorous rules for allowing these types of activities. These restrictions are necessary in order to make analysis and verification possible.

There are five types of transitions used in contour/transition-nets: ***normal*** transitions, ***entry*** transitions, ***exit*** transitions, ***boundary*** transitions, and ***split*** transitions.

A normal transition is a transition which adheres exactly to the description of the enabling predicates and firing actions described above. Other transition disciplines which modify these rules somewhat are described below. Although it is possible to combine the attributes of two or more of these types of transitions to achieve a hybrid, in the interests of clarity the discussion that follows considers no such possibility.

### Entry Transitions

An entry transition is viewed as the preparation that occurs prior to the call of a procedure or function in an algorithmic language. Topologically, entry transitions usually have a single output arc which is connected to an named net (see Figure 6). The selection rule of a entry transition is constrained to introduce exactly one token on this arc. At the point where the token is introduced, the color generator is incremented and the resulting current color value of the generator is used as the color of the introduced token. The static link of this new color is considered null. The construction rules may specify variables to be defined in the new context, and the manipulation rules may initialize these variables. This achieves parameter-passing for the named net. All parameters are required to be passed entirely by value. Finally, the eligible tokens are removed and the introduced token is placed on the named net which causes the named net to be instantiated.

### Exit Transitions

An exit transition is viewed as the clean-up that occurs after the return of a call to a procedure or function. Topologically, exit transitions have a single input arc which must be connected to an named net (see Figure 6). The enabling

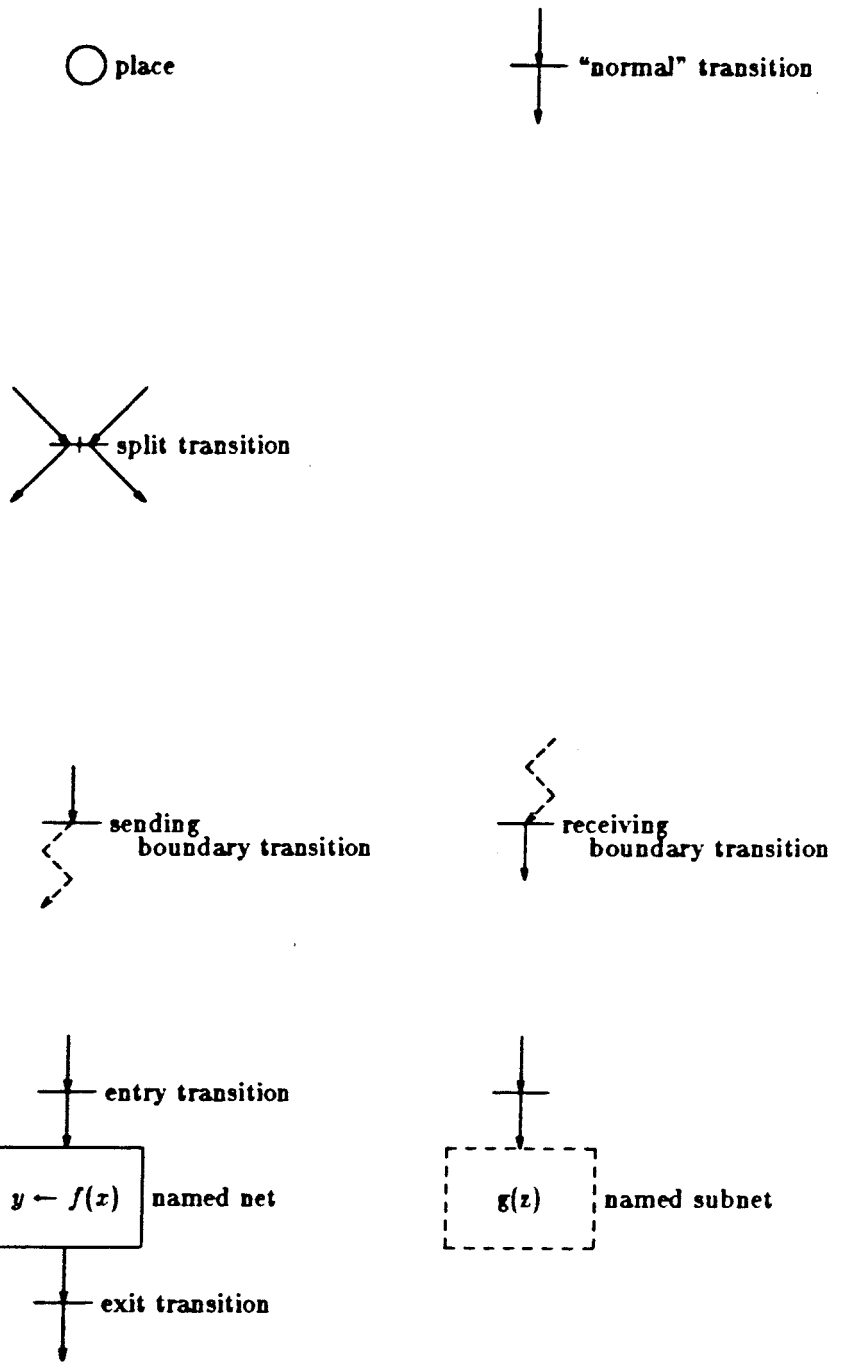



**Figure 6**
Building Blocks for a Contour/Transition-Net

predicate for an exit transition must require the presence of one token on its input

place. The mechanism by which an instantiation of a named net returns control to the correct exit transition of the calling net is discussed later. By using the context of the eligible and introduced tokens, the manipulation rules may make use of a return value.

**Boundary Transitions**

A boundary transition is viewed as allowing the passing of information from one N-peer to another. There are two sub-types of boundary transitions: *sending* and *receiving*. Topologically, sending boundary transitions have a single output arc which is graphically depicted with a dashed rather than solid line (see Figure 6). The selection rule of a boundary transition is constrained to introduce exactly one token on this arc, and the construction rules are required to specify that this token have the blank color (in violation of the rules presented above). The manipulation rules are required to completely specify the variables which one N-peer communicates to the other. These variables have their values fully copied to the introduced token. As a result, any information passed from one N-peer to another is transferred to the latter's private resource space, ensuring that each N-peer's resource space is private. Topologically, receiving boundary transitions have a single input arc which is graphically depicted with a dashed rather than solid line (see Figure 6). Each receiving boundary transition indicates the type of information that it expects to receive.

From a practical basis, the reader should understand that boundary transitions rely on an underlying addressing and "delivery" mechanism to achieve their semantics. From a mechanical basis, a sending boundary transition takes a colorful token, maps it into a colorless token, and passes that token to the underlying delivery mechanism. Similarly, a receiving boundary transition accepts a colorless token from the underlying delivery mechanism. Usually, information from the colorless token is passed to a particular execution context through the use of a split transition. From a conceptual basis, two boundary transitions for two different peers may be thought of as "touching" at the point where the blank token is passed, since they model a message passing mechanism. This touching is guaranteed to limit interference and to allow the sending action to be separated from the receiving action.

**Split Transitions**

A split transition is viewed as a local synchronization method for processes residing on the same processor. Split transitions are allowed to violate a key tenet of the contour/transition model philosophy: they may consider different colors when evaluating their enabling predicates. This lapse is permitted under a very strict condition: although different colors may be considered, only one color at a particular input arc is considered. Topologically, split transitions have the same number of input arcs and output arcs and must have at least two of each. Graphically, a split transition has a "notch" between each of its input arcs and between each of its output arcs to differentiate it from a transition which has multiple input arcs but which considers tokens of the same color when evaluating its enabling predicate (see Figure 6). While the enabling predicate for a split transition must require the presence of one (or more) tokens on each input place, the selection rule is constrained to introduce at least one token on each output arc. Each introduced token on a given output arc must have the same color as the eligible token on the corresponding input arc.

It must be granted that the "rendezvous" semantics of the split transition is meaningful only when the peers participating in its firing are resident in a tightly coupled system (e.g., the same host). Any activity more complicated than this must be performed using boundary transitions, which do not suffer this restriction.

***Named nets***

Named nets are references to other contour/transition-nets. Topologically, they are similar to places but are represented with a labeled square instead of circle (see Figure 6). For representational convenience, in addition to labeling a named net with its name, the parameters used when the named net is invoked may appear also within the labeled square using the traditional parenthesized notation.

Named nets may have more than one entry place. In this case, each entry place is named, and the contour/transition-net which instantiates the named net must specify (in the labeled square) which entry place is to be used. A named net needs only a single exit place although it may contain more. This is for representational convenience only. Multiple exit places should be thought of as leading to a single, actual exit place.

The underlying mechanics which implement the named net semantics may be viewed in the following fashion. A colorful token actually contains three components. The first two, variable bindings and a static link, have already been discussed. The third is a return pointer. When an entry transition fires, the color generator is incremented, and the resulting context is manipulated. In addition to "loading" parameters and making the static link empty, the return pointer of the introduced token is initialized. This pointer consists of two components: a *location* and a *context*. The location of the pointer is set to the corresponding exit transition for the entry transition that is firing, while the context of the return pointer is set to the color of the eligible token. Again, the reader will observe that the contour/transition model continues to borrow from the contour model. The introduced token, when placed on the named net, is immediately removed and placed at the appropriate entry place for the named net. When a named net terminates (at the exit place of the contour/transition-net), the token residing at the exit place is removed and its return pointer examined. The exit transition corresponding to the entry transition that instantiated the named net fires with the correct eligible and introduced tokens. The location of the exit transition and the color of the token that will be introduced by the exit transition are found by examining the return pointer of the token removed from the exit place. The eligible token for the exit transition is, of course, the token removed from the exit place.

Since all entry transitions are required to invoke the color-generator and no transition is allowed to introduce arbitrary colors, this mechanism is fully capable of supporting the desired semantics. The reader should note that since the static link of a colorful token generated at an entry transition is null, a named net which is instantiated can not "tamper" with the context of the net which invoked it.

Named nets are traditionally thought of as "single entry, single exit" nets. As discussed above, more than one entry place and one exit place may be present. As a further enhancement, named nets may also be specified as "single entry, zero exit" nets. That is, once the net is instantiated, it never returns. In such cases, the named net need not feed an input arc to a transition and may be thought of as a "terminal" node. This is particularly useful in modeling non-terminating processes (e.g., operating systems, network servers, and so forth).

### *Named subnets*

Named subnets are references to other contour/transition-nets. Topologically, they are similar to places but are represented with a labeled square with a dashed outline instead of a circle (see Figure 6).

Subnets are used as a representational convenience only. Subnets never connect to input arcs, only to output arcs. When a transition introduces a token for a named subnet, that token is placed at the entry place of the corresponding contour/transition-net. Hence, subnets differ from named nets because no contour is saved or restored and token colors are preserved.

It might be noted that subnets correspond to "gotos" in the programming language sense. In response, subnets are intended to a convenient way to generalize the state of a particular N-peer. Rather than using "tail-recursion" to denote state changes (where a particular state is represented by a contour/transition-net), subnets provide a more concise description and simpler technique.

## Other Aspects

Now that the basic operations of the contour/transition model have been described, additional aspects can be considered.

### *Initial Markings*

Each contour/transition-net has an initial marking. If the net is the main instantiation of an N-peer, then no restrictions are made on this initial marking. At the beginning of execution, the N-peer initializes its color generator, and the first color is produced. The net is assigned a global contour which is given to all tokens in the initial marking.

The initial marking of an named net is always a single token appearing on one of the named net's entry places. Naturally, if a named net is intended to return control to the net that instantiated it, the former should be produced in such a way that it (eventually) produces one token on its exit place after a token appears on its entry place.

### *Graphical Conventions*

The convention for drawing contour/transition-nets differs somewhat from the standard notation used for Petri nets. When drawing a transition, text

appearing to the left of the transition is interpreted as the enabling predicate for the transition. Similarly, text appearing to the right of the transition is interpreted as the firing actions for the transition. Since the firing actions consist of three components (selection rules, construction rules, and manipulation rules), individual components are specified by prefixing them with their name or a short abbreviation (e.g., "sr:" for selection rule). In addition, the enabling time for the transition may appear to the left of the transition under the enabling predicate (e.g., prefixed with "et:"). Similarly, the firing time for the transition may appear to the right of the transition, under the firing actions (e.g., prefixed with "ft:"). As a short-hand notation, if the text appearing to the right of a transition is not prefixed, then the text is presumed to be the manipulation rules for the transition.

The defaults for unspecified predicates and actions are as follows:

- enabling predicate — 1 token on each input arc (AND input logic)
- enabling time — 0 time
- selection rule — 1 token on each output arc (each selection predicate is TRUE)
- construction rule — none
- manipulation rule — no change of state in the context of the introduced tokens
- firing time — 0 time

For those places which feed input arcs leading to more than one transition, a "tilde" symbol (e.g., '~') may be used as the enabling predicate for one of the transitions. This is a short-hand expression meaning that none of the other transitions being fed by the place have their enabling predicates satisfied. Similarly, for those transitions with more than one output arc, a tilde symbol may be used as the selection rule for one of the arcs. As expected, this is a short-hand expression meaning that none of the selection rules associated with the other output arcs introduce any tokens.

By clever use of these conventions, the graphical representations of contour/transition-nets can be presented in a concise fashion and kept relatively free from clutter.

### *Semantic Issues*

This discussion has ignored many of the semantic issues that can arise when constructing a system out of contour/transition-nets. In particular, no requirement is made as to the programming language used in the manipulation rules of each transition. As long as the firing of a transition can occur atomically, no further requirements are needed.

### *Conversion to GMB style*

The extensions introduced herein can be easily applied to the Graph Model of Behavior. Contour/transition-nets can be transformed into a GMB-like control structure by following the rules used to translate place/transition-nets into a GMB: each place and its associated arcs are coalesced into multi-arcs, and each transition becomes a node. Named nets (and named subnets) are still represented as a (dashed) square.

There is no semantic difference between the two structures, only a difference in presentation. Since this is largely a matter of personal taste as to which form is the most readable, the actual form used when presenting contour/transition-nets is unimportant.

## Some Examples

Below a few examples of simple concurrent systems are presented which are described using the contour/transition model. The solutions to these simple problems demonstrate most of the fundamental concepts of the contour/transition model.

### *The Sieve of Eratosthenes*

First, a system is presented which generates prime numbers by sifting a sequence of ascending natural numbers through a set of processes as filters. Each process in the filter has associated with it a prime number. When it receives a number to consider, the process checks if that number is evenly divisible by the prime number associated with it. If so, the number can not be prime and is filtered out. If not, then the process passes the number to the next process in the filter (which is responsible for the next larger prime number). If there is not another process, then the number has successfully passed through all prime numbers known

to the system and must therefore be prime.[8] Stated more precisely, the composition of the system is:

1. There are two types of processes in the system, a *driver* process, and many *filter* processes. The driver process is the top-level process that is instantiated to guide the system. Each filter process represents a particular prime number.

2. A driver process begins by starting the first of the filter processes, with the prime number 2. Then, the driver perpetually iterates, by incrementing the number and communicating it to its child, so as to enumerate 3, 4, ..., and so on.

3. Each filter process begins by printing out the number with which it was invoked. It then enters a perpetual loop. The loop begins by receiving a number from the parent of the process. If this number is evenly divisible by the prime number associated with the process, then this new number is not a prime and may be discarded. If the number is not evenly divisible, the filter checks to see if it has started a successor. If there is a successor, the filter simply communicates that number to its successor. If not, the number is a prime, so the filter starts another filter process with the new number.

Three contour/transition-nets are used to model the system. The MAIN net (Figure 7) is the driver process. When instantiated, it begins by creating a contour for two local variables: *x*, which is the number currently being enumerated, and *id*, which will be used to identify processes. Control then forks. One fork instantiates the PNF (Prime Number Filter) net. Parameters *x* and *id + 1* are passed. An important convention is that a child always has an *id* equal to *1* plus the *id* of the parent. Since PNF is a non-terminating process (a "single entry, zero exit" net), no exit transition is required after the named net. The other fork begins the enumeration loop, *x* is incremented, and then the SYNC net is instantiated at the C entry point in order to communicate *x* to the immediate child of the driver process.

The PNF net (Figure 8) models a filter process. When instantiated, it begins by creating a contour for two local variables: *i*, a boolean which indicates if a child has been created for this process, and *x*, which is used to hold communicated

[8] This conclusion is true since ascending numbers, starting at 3, are given to the filter which initially contains only one process, which is associated with the prime number 2.

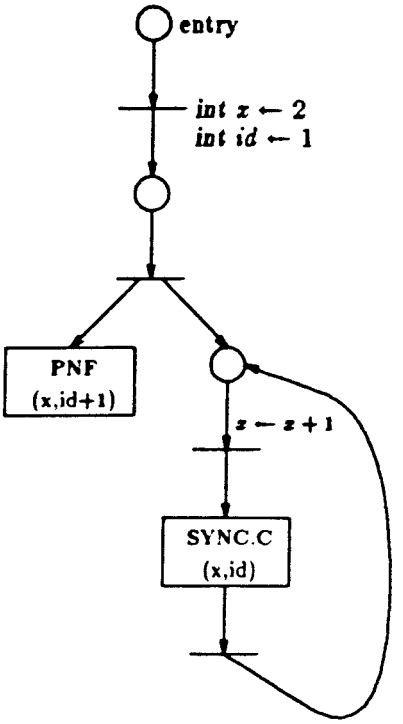



**Figure 7**
Driver for Sieve: MAIN

numbers. As a part of the manipulation rules, the statement *print*($p$) is called. This procedure prints out the value of the variable $p$ in the current context (recall that PNF has two formal parameters, $p$ and $n$). The SYNC net is now instantiated at the P entry point, to retrieve the next number from the parent of this process. When the invocation returns, its return value is copied into the variable $x$. This value is compared against $p$ (the prime number associated with this process). If $x$ is evenly divisible by $p$, then control loops back. Otherwise, if $i$ is *FALSE*, then $i$ is set to *TRUE* and control forks. One fork instantiates the PNF net with parameters $x$ and $id + 1$. The other fork loops back. If $i$ is *TRUE*, then the SYNC net is instantiated at the C entry point, to communicate $x$ to the immediate child of this filter process.

The SYNC net (Figure 9) is the net that synchronizes the processes. The SYNC net has two entry points P (synchronize with parent) and C (synchronize with child) and consists of a single split transition. The enabling conditions for this transition specify that the value of the variable *id* in the context of a token from

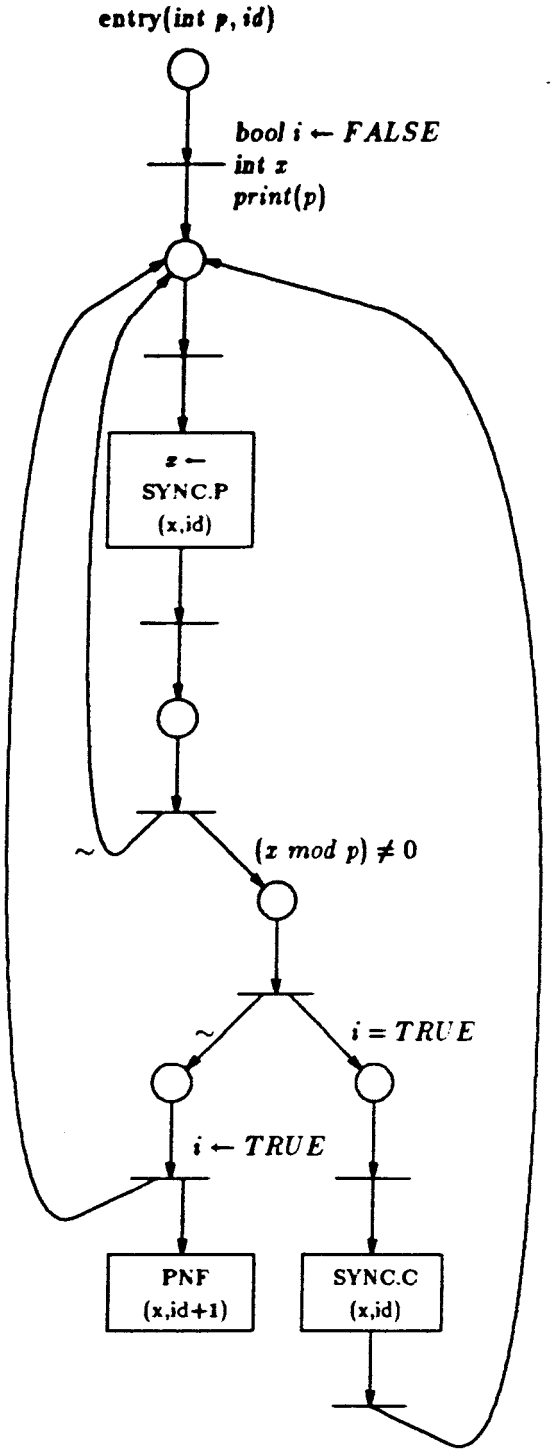
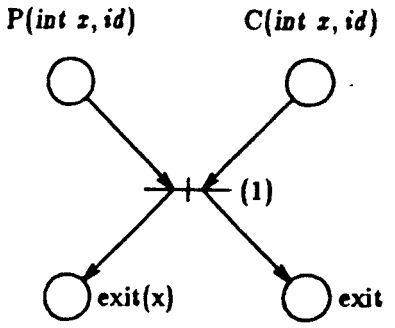



**Figure 8**

Prime Number Filter: PNF

place P must be equal to one more than the value of the variable *id* in the context




(1) ep: $P.id = C.id + 1$
mr: $P.x \leftarrow C.x$

**Figure 9**

Rendezvous of PNFs: SYNC

of a token from place C. When the transition fires, the value of the variable *id* in the context of the eligible token from place P is set to the value of the variable *id* in the context of the eligible token from place C. After this manipulation, each token reaches an exit place. For the token that entered at place P (a child wishing to synchronize with its parent), the value of the communicated number is returned at the exit place. Described simply, the SYNC net synchronizes a parent and its child and has the parent pass an integer to the child.

This example demonstrates an interesting property of the contour/transition model, i.e., the ability to perform *horizontal-* and *vertical-multiplexing*. Although several PNF processes may be executing, only one contour/transition-net is required to represent them since the colorful tokens contain all of the state information. This is an example of horizontal-multiplexing. There are several advantages to this type of capability. For example, when specifying a system, it is not necessary to know before-hand the number of processes that will be running in the system. Instead the structure of each type of process must be detailed exactly once. In contrast, an example of vertical-multiplexing might be a system modeled by several named nets which have instantiated each other according to some functional hierarchy. In this case, only one process which has been divided into several functional units is modeled.

### *The Dining Philosophers*

Next, a system in which an arbitrary number of philosophers attempt to think, sleep, and eat in a co-operative fashion is presented. Although any philosopher may think and sleep without regard for what other philosophers are doing, the activity of the other philosophers may interfere with a given philosopher who wishes to eat. The $n$ philosophers eat at a circular table. Each philosopher has an assigned seat and cannot sit elsewhere. Between every two philosophers is a fork. Unfortunately, a philosopher must eat with the two adjacent forks. Therefore when one philosopher is eating, neither of the philosopher's two neighbors may be eating as well. Naturally, if all the philosophers were to sit down to eat at the same time and each grabbed the fork to the left, then they will all starve as a deadlock has developed. The system to be modeled will have to avoid this undesirable situation. A better system would address the problem of indefinite postponement, in which a given philosopher could starve because at no time would both of his neighbors not be eating (philosophers are a conspiritous bunch). Stated more precisely, the composition of the system is:

1. There are three types of entities in the system: a *driver* process, $n$ *philosophers*, and an equal number of *forks*. The driver process initializes the system. Each philosopher and fork has an *id* which indicates where that entity resides at the table.
2. A philosopher endlessly cycles through the loop: think, eat, and sleep.
3. A fork is either free or busy. Modulo $n$ arithmetic (represented by the $\oplus$ operator) is used to relate a philosopher and two adjacent forks. For a given philosopher with *id* of $i$, the fork to left has an *id* of $i$, and the fork to the right has an *id* of $i \oplus 1$.

Two contour/transition-nets are used to model the system. The MAIN net (Figure 10) is the driver process. When instantiated with parameter $n$, it begins by creating a contour for the local variable $i$, which it uses as a counter. It then creates $n$ philosophers and forks by instantiating PHILOS.P and PHILOS.F with an *id* of *1* to $n$, and then terminates control by advancing to an exit place.

The PHILOS net (Figure 11) models a common control structure for philosophers and forks. When instantiated at the P entry place, a philosopher is modeled. An instantiation at the F entry place starts a fork. Initially, a philosopher is thinking (at place T), and a fork is free (at place F). When the first

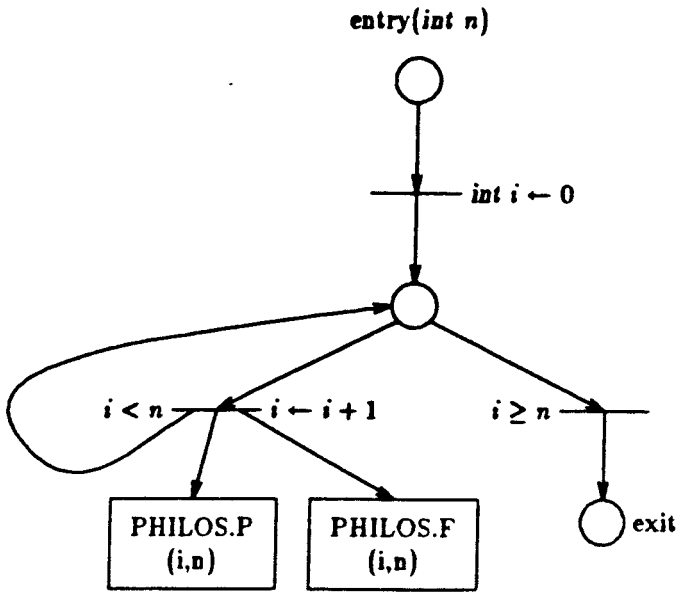



**Figure 10**

Driver for the Philosophers: MAIN

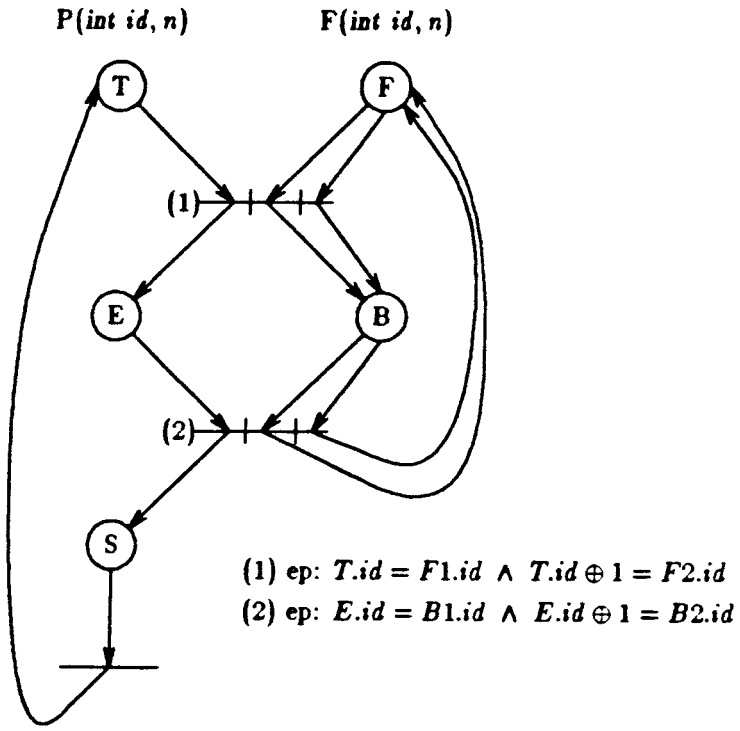



**Figure 11**

The Dining Philosophers: PHILOS

split transition fires, indicating that both of the philosopher's forks are free, the

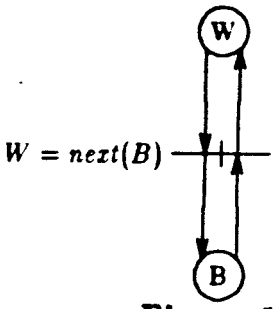



**Figure 12**

Behavior of Stations on a Ring Network

philosopher enters the eating state (place E), and the two forks enter the busy state (place B). When the second split transition fires, the philosopher enters the sleeping state (place S), and the two forks enter the free state. Sometime later, another transition fires, and the philosopher begins thinking again.

The solution to the problem of deadlock (though not indefinite postponement) is achieved through the use of a single split transition which requires as an enabling condition that both of a philosopher's forks be free. Hence, a philosopher will not be able to start eating until this condition is met, and the philosopher *cannot* grab one fork and then wait for the other, (which could lead to a deadlock situation).

*Token Ring Protocol*

Finally, a model of the behavior of a number of stations on a ring network is presented. In short, an arbitrary number of stations are arranged along a token passing network with a ring topology. At any given time, one of the stations has the token and is allowed to transmit a packet on the wire. After the packet is sent, the token proceeds to the next station in the ring.

For the purposes of this example, only the relation between the $n$ stations will be modeled and the token (to avoid confusion between the token in the ring network and the tokens in the contour/transition-nets of our model, the former object will be called the *marker*). Intuitively, one can imagine some type of behavior as described in Figure 12. Initially, one station has the marker, and its corresponding token is in place B (the BUSY state). All stations awaiting possession of the marker so they can transmit a packet on the network have a corresponding token in place W (the WAIT state). The *next*() function must enforce some sort of

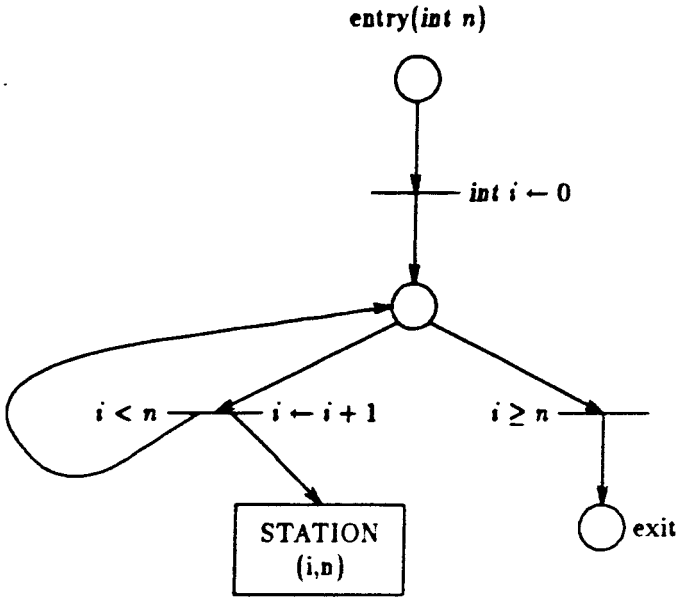



**Figure 13**

Driver for the Stations: MAIN

one-to-one mapping between stations, in order to ensure consistent passing of the marker from station to station for all iterations along the network.

Our model of the system will use modulo *n* arithmetic to construct the function *next*(). The assumption is made that the number of stations in the system is conserved at exactly *n*. The MAIN net (Figure 13) is the driver process. When instantiated with parameter *n*, it begins by creating a contour for the local variable *i*, which it uses as a counter. It then creates *n* stations, by instantiating STATION with an *id* of *1* to *n*, and then terminates control by advancing to an exit place.

The STATION net (Figure 14) models the common control structure for all stations. When instantiated with parameters *id* and *n*, it determines if this station should be given the marker first. If so, control proceeds to place B. Otherwise, control proceeds to place W. Once both places W and B have tokens, the split transition may fire. The enabling predicate states that *id* in the context of the token from place W must be *1* greater (modulo *n*) than *id* in the context of the token currently. Hence the sequence of *id*'s chosen from place W to satisfy the enabling predicate form a ring (not surprisingly).

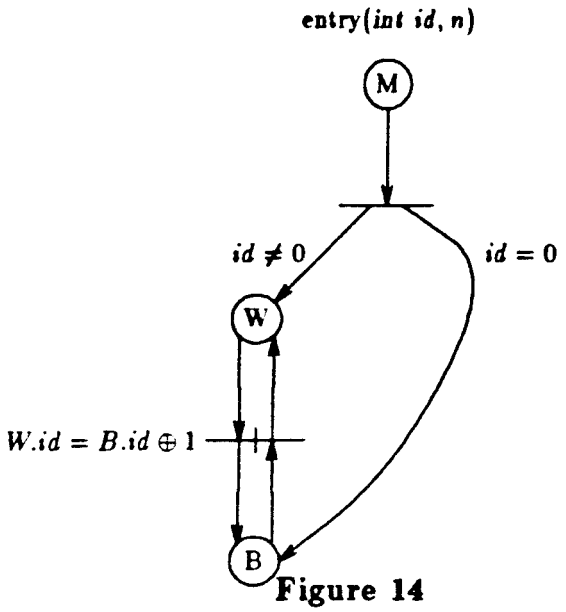



**Figure 14**

The Stations on a Ring Network: STATION

## Extensions to the Model

The previous sections in this chapter have introduced parts of the contour/transition model that form the basis of the model. This section suggests some extensions to the basis which may prove useful in modeling systems. A detailed investigation of the merits of these extensions is left to future research.

### *Block Structuring*

The base model allows only entry and exit transitions to have construction rules. One obvious extension is to allow normal transitions to also have construction rules. In this way, a pair of matched transitions can introduce a "begin-end" block into the context of a net. Of course, a correct design must guarantee that the transitions are in fact matched. If this extension is permitted, then the construction rules may perform two types of operations. In the first, the color generator is incremented, the new current color value is used as the color of the introduced tokens, and the static link of the introduced tokens is set to the color of the eligible tokens. In the second type of operation, the color of the introduced tokens is set to the static link of the eligible tokens. The former operation ***pushes*** a next context while the latter ***pops*** the current context and restores the immediately scoping one.

It should be noted that, with this extension, a named net need not exercise great care in removing any contours introduced during its execution. When the

exit place is reached, the underlying mechanics will notice that the eligible token does not have an initialized return pointer and may consecutively discard contours until one is found.

With this extension, searching for a variable in the context of a token is somewhat more complicated. The token's contour block is examined. If the variable is not present, then the previous contour (found by using the static link) is examined. This process continues until either the binding for the variable is found or there is no previous contour (a null static link is encountered). The latter case indicates that the variable is undefined in the context of the original token.

Finally, regardless of the introduction of this extension, it should be noted that the construction rules are not allowed to specify arbitrary colors for the introduced tokens — *this is strictly prohibited.*

*Dynamic Scoping*

A further extension is to modify the actions taken by the construction rules of entry transitions. In particular, if the static link of the introduced token is not required to be null but instead set it to the context of the eligible token, then a dynamic scoping mechanism is enforced when a named net is invoked. This means that during the execution of that net in the "new" context, the "old" context may be modified by following static links. Although this form of scoping is found to be extremely useful in some applications, permitting this type of data access between instantiated nets is thought to be very harmful for several reasons. Perhaps the most important of these is the difficulty that is introduced in the analysis of contour/transition-nets when *contextual interference* between instantiated nets becomes possible.

*Remarks*

It should be noted that the lack of these extensions in the base model do not reduce the power of the invocation mechanism. Rather, they enforce a particular modeling discipline upon the designer. Forcing the use of return values and value-parameter passing permits a simpler analysis later.

## Comparison to Other Petri Net Models

This chapter is concluded by comparing the contour/transition model and the other Petri net extensions discussed in the previous chapter. As the next chapter deals with issues of analysis, only modeling power and ease of use is considered here.

Comparisons are made with respect to the control and data aspects of the system are represented and what interactions occur between control and data.

### *Place/Transition-Nets and Abbreviations*

Neither place/transition-nets nor place/coloured-nets nor predicate/transition-nets attain any level of hierarchy or modularization. In all three models, a single net is used to represent the entire system. Since the contour/transition model introduces named nets and named subnets into the topology of the net, these useful control abstraction mechanisms can be achieved by contour/transition-nets.

In terms of the data aspect of the system, place/transition-nets use colorless ("grey") tokens, place/coloured-nets use color sets, and predicate/transition-nets use well-defined tuples. In the first case, any data aspect of the system must be represented as a part of the control aspect (i.e., the marking of the net), while in the latter two cases, the data aspect can be simulated by a priori delimiting the totality of the values taken on by the data aspect. As a result, data abstraction is not possible.

In terms of the interpretation aspect of the system, the mapping functions associated with transitions in place/coloured-nets and predicate/transition-nets do resemble the enabling predicates and firing actions of contour/transition-nets to some degree. However, note that in the other models, the mapping functions represent an abbreviation of the net and not a true interaction between the control and data aspects of the system. In contrast, the use of enabling predicates and firing actions really does establish an interaction and makes contour/transition-nets Turing equivalent. This has a significant effect on analysis, since invariants are not automatically generated from the topology of a contour/transition-net in contrast to place/transition-nets, place/coloured-nets, and predicate/transition-nets.

### *AP-Nets and "modified" Petri nets*

As discussed in the previous chapter, the restricted nature of AP-nets prevents control structures of a complex nature. In contrast, "modified" Petri nets and contour/transition-nets do not suffer from this restriction. All three models permit some degree of hierarchy. But AP-nets, as defined, cannot support recursion or concurrent instantiation, and "modified" Petri nets, while permitting abstraction in the input/output specification sense, do not support recursion or concurrent instantiation.

Neither AP-nets nor "modified" Petri nets pay much attention to the data aspect of the system. The "modified" Petri net model introduces abstract data types to define the organization of the data manipulated by the system. It also introduces data objects to delimit the relations between components in the system and the affected data. Owing to the static nature of both of these models, scoping and related issues are not addressed directly. The contour/transition model purposely avoids the issues of data types, relying on the semantics of the language used for the manipulation rules to resolve them. The merits of this approach is open to discussion.

AP-nets indirectly solve the problem of control and data interactions by requiring that all transitions be conflict-free. "Modified" Petri nets do not address this issue at all.

### *The Graph Model of Behavior*

From the aspect of control, as the discussion above indicated, there is little difference between the two approaches for graphically depicting the control structure of the system. In particular, since contour/transition-nets introduce selection rules into the net, a contour/transition-net representation of the system is likely to be as concise as the corresponding GMB control structure. In addition, the GMB can support control abstraction to some extent, through the use of *single entry, single exit* (SESX) UCLA graphs. One extension to the GMB permits replacing a safe/live node with a safe/live SESX control structure and maintaining safeness and liveness properties. With this extension, systems may be modularized. Owing to the static nature of the data domain, however, recursion is not supported.

The GMB uses two graphs to represent the control and data domains of the system and a third segment, in the form of program code, to represent the interpretation domain. The contour/transition model uses one graph which explicitly represents the control and interpretation aspects. The data aspect is represented implicitly with contours. This has both advantages and disadvantages. To its credit, this permits recursion and concurrent instantiation of a net in a simple fashion. Unfortunately, it does not allow us to graphically delimit the effects of a transition on a particular data set (though the underlying contour model does allow us discipline in accessing data). Although the ramifications of this distinction are not presently clear, it appears to the author that it would be useful to be able to combine these two facilities: data abstraction and data delimitation.

*Predicate/Action-Nets and Numerical Petri nets*

In terms of the control aspect, predicate/action-nets and numerical Petri nets can not permit hierarchy without further extensions (the issues of scoping and allocation must be addressed). Furthermore, note that the proof method for the distinct process extension relies on the assumption that a single token exists for each process in the system, while contour/transition-nets do not have this restriction. Finally, the contour/transition model addresses the issues of process creation and termination, while the distinct process extension does not.

In terms of the data aspect, both models, and in particular the numerical Petri net model, treat all data as globally accessible to all transitions. In contrast, contour/transition-nets denote the data aspect in terms of contours. Tokens of the same color (i.e., contour) are identical and co-operate to give transitions concession (allow them to fire). This permits several instantiations of the same net to be concurrently active in different execution contexts. Hence, it is easy for the contour/transition model to achieve data abstraction facilities while the other models would have to be re-structured in order to do so.

Although the distinct process extension does allow a simple two-level discipline of data access, the contour/transition model permits arbitrarily complex contours to be constructed. In addition, the distinct process extension does not consider interaction between different processes in order to enable a transition, or to exchange information as during the firing of a transition; usually, processes communicate indirectly through global variables. With split and boundary

transitions, the contour/transition model allows processes to cleanly interact in order to synchronize and communicate.

Hence, there are significant differences between the models, even when considering the distinct process extensions to the presentational model. It is emphasized that these differences will have a great bearing on our extensions to the invariant-method.

# CHAPTER 4
# Analysis of Contour/Transition-Nets

Although the verification of specifications using simple Petri nets is well understood, contour/transition-nets are sufficiently more complex to prevent direct application of previous analysis results. In this chapter, extensions to previous analysis methods are presented to enable the derivation of proofs about systems represented with the contour/transition model.

First, the problems in using existing analysis methods on systems modeled with contour/transition-nets are examined.

Then, a different approach is introduced to form the basis of a set of analysis concepts useful for proving statements about contour/transition-nets. The approach takes advantage of the hierarchical nature of the contour/transition model and certain key restrictions in the way colors interact which permit the use of invariants in deriving proofs for systems represented by contour/transition-nets.

Next, the approach is applied to a relatively simple system represented by the contour/transition model. Finally, the success of the approach is evaluated and areas where this method should be further researched are identified.

## Methods of Analysis and the Contour/Transition Model

Let us consider how the different analysis methods surveyed in Chapter 2 can be applied to the contour/transition model. To refresh the reader's memory, the techniques to be considered are:

- State-Space Analysis
- Structural Analysis
- Inductive Analysis
- Hybrid Approaches

### *1. State-Space Analysis*

As has been demonstrated, strong statements may be made about a system represented with the Petri net model due to information presented by the reachability tree for the system.

Although Petri nets are able to present concise descriptions of concurrency, they nevertheless lack expressive power. Recall that one way of viewing the state of a system is to consider it as being composed of two related portions: a *control* aspect and a *data* aspect. Since the entire state of the system is encoded in its marking, both the control and data components of that state must be present in the marking. Tokens in place/transition-nets are identical and do not carry data values. Hence, the control portion of the state can usually be represented well, usually quite well, owing to the graphical nature of the net. The same is not true of the data portion. Since all the designer has to work with are "grey" tokens, encoding data components of the state, even simple components such as counters or sequence numbers, becomes quite difficult. To make matters worse, none of the powerful control and data abstraction facilities found in modern programming languages are to be found in place/transition-nets.

To meet this deficiency, contour/transition-nets enhance the Petri net model by adding such features as colorful tokens, enabling predicates, and firing actions. Since tokens have an associated data context, and transitions are able to access this contextual information in a controlled fashion, the protocol designer is able to capitalize on both powerful control representations and powerful data representations. But consider that, with these additions to the model, the construction of the reachability tree has become more difficult. In fact, the addition of enabling predicates makes reachability questions undecidable.

Recall now that in the context of the place/transition-net, the reachability tree encapsulates the totality of the state-space and this state-space is expressed as the number of tokens residing at each place in the net. Reachability analysis, as it is currently used, does not consider tokens as having an identity and a well-defined relation to other tokens which regulates how they interact. In the context of the contour/transition-model, a named net may appear to have incorrect behavior if it is instantiated more than once, since the analysis technique can not distinguish between tokens belonging to different instantiations. Yet, if a named net is being

used as a *monitor*, it may may be desirable to be able to analyze how different execution contexts behave as they call upon the monitor at its various entry places. As such, it must be possible to consider all tokens present in the net, regardless of their context, while taking into account the rules that guide the interaction between tokens of different colors.

This is but a minor problem though. Regardless of the method that is used to access the data aspect of the system (e.g., contours, global data store), the computational power of our model is being raised to that of a Turing machine by the addition of enabling predicates which may prevent transitions from firing. For a short (and informal) proof, consider that adding an enabling predicate to a transition is equivalent to adding an *inhibitor* arc to that transition (an arc that prevents the transition from firing as long as there is at least one token in the place at the other end of the arc). As discussed by Peterson[PETE81], if the Petri net model is extended to permit testing a single place for the presence of *no* tokens, then the model has been extended to be equivalent in power to a Turing machine. Since reachability is undecidable for Turing machines in general, reachability analysis is not suitable for contour/transition-nets.

Considering that the goal is to use the contour/transition model as a representation technique which can be analyzed, this is particularly ironic. As Peterson[PETE77, p 238] points out:

> "Although some work on design with Petri nets ... and implementation of Petri nets ... has been done, it has been limited in scope, presumably because its success hinges on the existence of effective analysis techniques."

Very clearly, if the contour/transition model is going to be successful as a tool for protocol modeling, methods for analyzing properties of contour/transition-nets must be developed.

Finally, note that there are other properties of extended Petri net models that should be considered by reachability analysis. In particular, the use of enabling and firing times for transitions, which is present in the timed Petri net model[RAZO83A] (and in the contour/transition model) adds additional complexity to the analysis.

### 2. *Structural Analysis*

Although the results of structural analysis are quite attractive (e.g., the automatic generation of system invariants), the additional computational power of the contour/transition model makes this method unsuitable for analysis.

In particular, observe that the analysis via the incidence matrix of a net considers variables in the context of places and transitions rather than in the context of tokens. This difference in perspective between place/transition-nets (and its abbreviated forms) and contour/transition-nets is *critical*, and explains quite well why, in general, an invariant method based on the structural properties (i.e., solely the control aspect) of the system can not prove all of the interesting properties. This deficiency in the incidence matrix analysis, while making it unsuitable for *some* of our purposes, does not make less the value of the invariant method for use on simpler Petri net models.[9] Rather, it merely highlights how the philosophical differences between the two models make possible the use of different analysis techniques. More accurately, it should be noted that contour/transition-nets, unlike predicate/transition-nets or place/coloured-nets, are not simply concise abbreviations for place/transition-nets, but instead are capable of greater descriptive power.

### 3. *Inductive Analysis*

Although more difficult to apply, inductive analysis appears to be suitable, with some modifications, for use on contour/transition-nets. Both predicate/action-nets and contour/transition-nets are equivalent to Turing machines in power and quite similar in nature, particularly when the distinct process extension to predicate/action-nets are considered.

Note an important observation by Keller[KELL76, p 381] concerning the distinct process extension:

> **"The technique described appears to have the very desirable property that the effort required for proofs of systems with multiple processes increases only with the size of the program rather than the number of processes executing the program."**

---

[9] **In fact, incidence matrix analysis is useful for boundedness proofs of systems using contour/transition-nets and, as such, forms one component of the proposed proof method.**

Providing a sufficiently powerful conceptualism can be developed to allow us to frame contour/transition-nets in terms of the presentational model, then an even larger proof savings may be achieved since the natural hierarchy of the contour/transition model may be exploited in order to produce proofs dealing with smaller nets. Naturally, our approach must take into consideration the interaction between colors, the various transition types, and also be able to deal with the characteristic of contour/transition-nets which allow more than one colorful token of the same context to exist in the system.

*4. Hybrid Approaches*

When proving properties about the total GMB representation of a system (control, data, and interpretation domains), similar problems arise to those found when analyzing contour/transition-nets. Note though that in the context of Shapiro's work, non-atomic firing is not relevant and thus those results appear not to be germane to our analysis. Furthermore, recall that the contour/transition model blends the control, data, and interpretation domains into a single graph. Hence, although a contribution to the analysis of systems represented by the GMB, the work by Shapiro[SHAP83] does not appear to have a direct bearing on our efforts to find a proof methodology for systems represented by contour/transition-nets. It is important to emphasize, however, that if delays are permitted in the systems being analyzed, then the issues and methods reported by Shapiro must definitely be considered.

## A Method for Analyzing Properties of Contour/Transition-Nets

The distinct process extension to the presentational model[KELL76, p 381] is extended by considering each process index, $\pi$, as referring to a unique color in the system.

For each color present in the net, $\pi$, the state of the net is given by:

- the context (variable bindings) of $\pi$
- the marking of the $\pi$-colored tokens in the net

Observe that this differs from the distinct process extension in three important respects: first, since variable bindings for different $\pi$ are immune from *contextual*

*interference* (a term to be fully defined further on), there are no global variables — all variables are local relative to their execution context, $\pi$; second, unlike the distinct process extension, more than one token may execute on behalf of $\pi$ (hence there is both *inter-process* concurrency and *intra-process* concurrency); and, third, only tokens associated with the same $\pi$ may co-operate to enable a transition.[10] In this sense, variables in the context of a particular $\pi$ are global to all tokens executing in that context.

It should be noted that if split and boundary transitions are ignored, then since tokens with different $\pi$ do not interact in any way whatsoever in either the control aspect or the data aspect of the net, each $\pi$ executes in its own isolated net. Each net is therefore equivalent to the unextended presentational model of Keller:

- all variables are global in the context of each net
- the state of the net is given by its marking and the global data store

This result provides a basis (but, as shall be seen, not a perfect one) for analyzing properties of systems represented by the contour/transition model.

Now that a mapping, of a sort, has been demonstrated between a contour/transition-net and a net in the presentational model, it is possible to fully explain the analysis and to examine the key assumption of freedom from contextual interference.

### *The Method Revealed*

Boundedness, deadlock freeness, and system-specific properties of a system represented by contour/transition-nets are proven by starting with the MAIN net of the system and proceeding recursively to consider if those properties hold for the descendants of MAIN. Boundedness is proven by verifying invariants involving place-variables of the net. Next, $q_0$-invariants of the net are derived and verified in order to prove system-specific properties. Finally, freedom from deadlock is proven by verifying the existence of a ***homing-state*** for the net. It should be noted that although reference is made to the properties of "the net", if the net in question has more than one entry place, then reference is actually being made to the properties

[10] As mentioned earlier, interaction between different $\pi$ to enable transitions, is not addressed by Keller.

of a particular entry place of that net. That is a net with multiple entry places is considered to be more than one net.

### *Restrictions of the Method*

Some restrictions must be enforced to make the contour/transition-net being considered more suitable for analysis. For the purposes of our discussion let the symbol $\pi$ represent a particular context of execution (i.e., color) in the system. Four sets of restrictions are placed on the contour/transition-nets to be analyzed:

First, only entry transitions may introduce new contexts and only exit transitions discard contexts (which is consistent with the base model described in Chapter 3). Hence, although named nets may be invoked from the net, additional scoping contexts may not be constructed. This does not reduce the control abstraction facilities of the net (e.g., recursion), rather it ensures that a unique $(color, net)$ pair is associated with a particular $\pi$.

Second, the selection rules for each transition in the net must introduce the same number of tokens on each output arc each time the transition fires. That is, a transition may introduce two tokens on one output arc and one token on another, but it must do so consistently each time it fires. It should not introduce two tokens at one firing and one token at a later time. Only transitions that have a selection predicate as a part of their selection rules exhibit this behavior. In order to make analysis easier, such nefarious transitions can be transformed into a set of transitions (Appendix A discusses how this can be performed).

Third, that no OR input logic can be present in the net. In order to make analysis easier, such transitions are transformed into a set of transitions (Appendix A discusses how this can be performed). The reader should note that this transformation or the preceeding one does not change the semantics of the system or result in a loss of representational power, rather it elaborates the control aspect of the net somewhat.

Fourth, the use of named subnets in the contour/transition-net is prohibited. In addition, all temporal attachments to the net (the enabling time and firing time of each transition) are ignored. The issue of timing delays is left as an open area of research. This work concerns itself only with the ordering of events and not their duration.

One aspect of the contour/transition model which is examined in detail is the split transition. In fact, split transitions highlight an inconsistency in the simple viewpoint presented earlier concerning a unique net for each $\pi$: *split transitions allow interactions between different $\pi$ in aspects of both control and data.*

Each $\pi$ in a contour/transition-net meeting the above restrictions executes without fear of contextual interference with two exceptions:

- named nets introduce vertical (recurrent) interference by allowing control to pass to another $\pi$ and return some time later
- split and boundary transitions introduce horizontal (concurrent) interference by allowing two or more different $\pi$ to interact

Hence it is these two exceptions that separate the restricted form of contour/transition-nets described above from the unextended presentational model.

### *Boundedness Properties*

The analysis begins by examining the boundedness of the net through constructing incidence-invariants for the control aspect of the net. In short, the restricted contour/transition-net is treated as if it were a place/transition-net: the enabling predicates and firing actions of the net are ignored; each named net is considered as a simple place, after proving that it is 1-bounded and firable at its entry place; split transitions are considered as being multiple transitions; and, allowances are made for boundary transitions.

To begin, note that considering the data aspect reduces the number of reachable markings that the net can achieve (via the interaction of enabling predicates and firing actions). Hence, if it is possible to show that the place/transition-net is bounded, then no possible influence from the data aspect (or from temporal aspects) can make the net unbounded.[11] In general, any invariants derived from the incidence matrix are valid regardless of the data aspect; but properties which can not be derived from the incidence matrix may or may not be invariant — sufficient information is lacking, without consulting the data aspect, to know one way or the other. Unfortunately, the same argument cannot be made for liveness (or non-termination) since it is possible that the reduced set of reachable

[11] **It should be noted however, that if a net appears unbounded when the data aspect is not considered, then it is possible that consideration of the data aspect will show the net to be bounded.**

markings derived when considering the data aspect fully (i.e., the inclusion of enabling predicates into the model) may actually exclude those markings which are termed free from deadlock.

If it can be proven a given net is 1-bounded and free from deadlock at an entry place, then a named net can be substituted inside another contour/transition-net while preserving the boundedness and deadlock freeness properties of the second net. This is similar in nature to Razouk's[RAZ081, p 49] observation that a safe/live single-entry/single-exit graph can be substituted for a place in another safe/live graph without loss of safeness/liveness in the second graph. Intuitively, it is clear from the semantics of entry and exit transitions that if a token fires an entry transition for a named net and that net is 1-bounded and live when invoked at its entry place, then it will eventually produce a single token at one of its exit places. This will result in a single token appearing sometime later at the appropriate exit transition. This notion is expressed by saying that a named net which meets these criteria is *invocation-safe/live*. In practice, it is not possible to immediately prove each named net present in the contour/transition-net being examined to be invocation-safe/live. This is because in order to prove liveness, it is often necessary to rely on various system-specific properties that are proven in the next step. Hence, our boundedness results often require using the assumption that the named nets are in fact invocation-safe/live, where this property is actually verified later on in the proof.

Note that split transitions have little effect on boundedness (though a large effect on freedom from deadlock) when considered from the perspective of a particular $\pi$. In short, in the context of any particular $\pi$, split transitions absorb one eligible token and output one introduced token. Hence, in the context of a given $\pi$, replacing a split transition with a normal one does not affected the boundedness of the net. Therefore, when interpreting the incidence matrix of the net, the portion of the split-transition which is considered is that which applies to the path taken by the entry place for the net being considered.

Furthermore, note that boundary transitions have a small effect on boundness. Two sub-types of boundary transitions can be distinguished: *sending* and *receiving*. As discussed previously, a sending boundary transition takes a colorful token, maps it into a colorless token, and passes that token to the underlying delivery

mechanism. In contrast, a receiving boundary transition accepts a colorless token from the underlying delivery mechanism. In terms of boundedness analysis, sending boundary transitions are considered to absorb all tokens which enable them and receiving boundary transitions are considered to be analogous to an entry place in a named net.

*System-specific Properties*

The designer of the system must specify the invariants which should be proven about the behavior of the system. These invariants are proven using an extension to the Keller's invariant-method: if the net does not contain enabling predicates, then the incidence-invariants (or a modified form of them) derived in the preceeding step can be used; otherwise, the state-space is augmented to reflect the fact that multiple $\pi$ are executing, the appropriate initial state, $q_0$ is defined, and then the designer-supplied predicates are proven to be $q_0$-invariant.

Note that if none of the transitions in the restricted contour/transition-net have enabling predicates, then the incidence-invariants, derived from the structural properties of the net can be used, to prove the system-specific properties as well as the boundedness properties. This results from the fact that the net is simply a place/transition-net (or perhaps a place/action-net in the terminology of Diaz[DIAZ82]) which may contain named nets. Split and boundary transitions make the topology of the restricted contour/transition-net more complex, perhaps making a single net represent multiple nets joined at these special transitions. However, these additional semantics do not make the analysis sufficiently more difficult to mandate the use of an inductive invariant approach — the structural properties of the net can still be useful in deriving invariants and system-specific properties.

If different execution contexts can interact in the net (e.g., split transitions are present), then it is often useful to convert the incidence-invariants into equations involving place-variables in order to state and prove the system-specific properties. Before discussing this, it is necessary to introduce the notion of a ***hierarchy vector.*** The reader may find it useful to peruse the following two examples before reading the following description of the proof process.

Instead of considering the state-space to be a single vector containing values for $v_i$ for each $P_i$, and values for each component of the data aspect, let the state-space be represented by a hierarchy vector, where $v_{ij}$ is the place variable valued as the marking of place $P_i$ in the context of $\pi_j$ (i.e., $m_j(P_i)$) and $v_i$ is the place variable valued as the marking of place $P_i$ in the context of all $\pi$ executing (i.e., $\sum_\pi m(P_i)$, or $|m(P_i)|$). One state vector is assigned for each $\pi$ that is active in the net and can interfere with another $\pi$. Hence, for a net with a split transition permitting the interaction of two $\pi$, the state-space contains two state vectors. If multiple instantiations of a net at a given entry place are expected, a state vector may be given a superscript to indicate that more than one instance of the vector may be concurrently active.

Once a hierarchy vector has been constructed for the system, the invariants derived from the incidence matrix may be converted into equations involving place-variables, and the system-specific properties to be proven may be stated. Usually, it is necessary to prove certain relations between the components of the hierarchy vector in order to demonstrate the invariance of the system-specific properties. This activity proceeds in a two-level fashion: first, properties local to particular context are proven; second, properties between different contexts are proven. After proving these properties, the place-variable equations may often be simplified.

Now, if no enabling predicates are present, the system-specific properties can usually be proven directly from the place-variable equations. If not, the invariant-method of Keller is used to demonstrate the invariance of the system-specific properties. Let $q_0$, the initial state, be the state of the system when a token is placed on the appropriate entry place. After forming the hierarchy vector to represent the state-space of the net, and choosing the appropriate value for $q_0$, the invariants are proven first by showing that they hold for $q_0$, and second by showing for each transition that if the invariants hold prior to that transition firing, then the invariants hold after that transition has fired.

As with our analysis of boundedness, a distinction is made between sending and receiving boundary transitions. If convenient to the analysis, corresponding pairs of sending and receiving boundary transitions may be coalesced into a single transition which maps a colorful token into a colorless token. This often has the

effect of joining two named nets together since a given pair of sending and receiving boundary transitions often reside in different nets. Regardless, for each receiving boundary transition in the net, a separate component (state vector) is added to the hierarchy vector for the net to represent the colorless execution context constructed when the transitions fires. Appendix B contains a short proof of a system which uses boundary transitions.

### *Deadlock Freeness*

The method of analysis similar to that described for system-specific properties is applied: if the net does not contain enabling predicates then the incidence-invariants are used which were derived when boundedness was analyzed; otherwise, a homing-state is derived and then the homing state is shown to be $q_0$-reachable.

Again, Note that if none of the transitions in the restricted contour/transition-net have enabling predicates, then the incidence-invariants which are derived from the structural properties of the net along with any system-specific properties, can be used to prove freedom from deadlock. As before, if different contexts may interact in the net, it is useful to use equations involving place-variables to demonstrate that the net is free from deadlock.

If enabling predicates are present, then the same hierarchy vector and the same $q_0$ are used that were used was used to prove system-specific properties. Using the same designer knowledge that was required for $q_0$, $q_H$ (the homing-state) is defined, and the homing-state logic of Keller is applied to prove that $q_H$ is $q_0$-reachable. In short, the procedure is to prove the existence of a ***norm with zero-state*** $q_H$, $n(\vec{q})$ a function on the state-space, which shows that ***homing-***$q_0$ is $q_0$-invariant for the net.

## An Example

Now let us re-consider the readers and writers example. Contour/transition-nets are used to model the system and then its properties are analyzed using the methods described above.

Figure 15 describes the nature of the system. It states that the named net MAIN.M is instantiated once (think of this invocation as a call to initialize a monitor), and then the MAIN net gets instantiated an arbitrary number of times

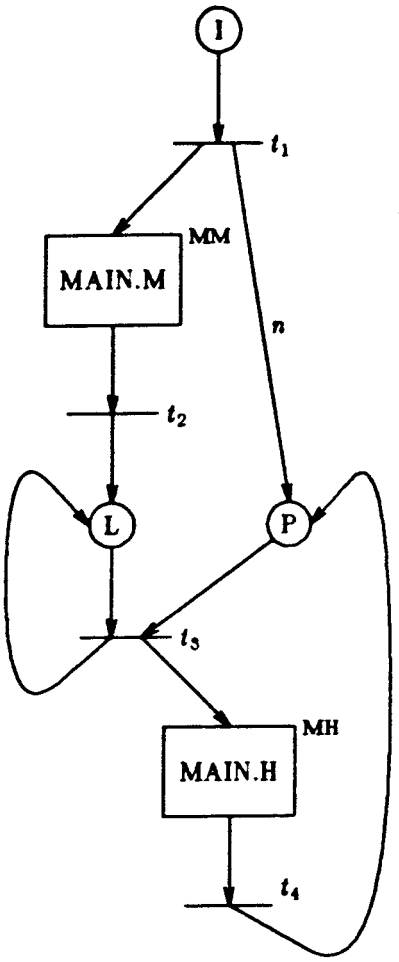



**Figure 15**
Overview of the System

(up to $n$ times concurrently) at the entry place H (once for each reader/writer in the system).

The MAIN net in Figure 16 has two entry places. At the M entry place, control forks. One fork generates $n$ (identical) tokens for place S. The other fork returns control to the net that instantiated MAIN.M. At place S, the system waits to service requests to read (a token appears in place WR) or requests to write (a token appears in place WW). H is the second entry place, which is instantiated an arbitrary number of times by the driver process. Each instance is characterized by a token of a unique color. A token which fires transition $t_1$ is designated as a reader and proceeds to place WR. When at least one token is free in place S, then

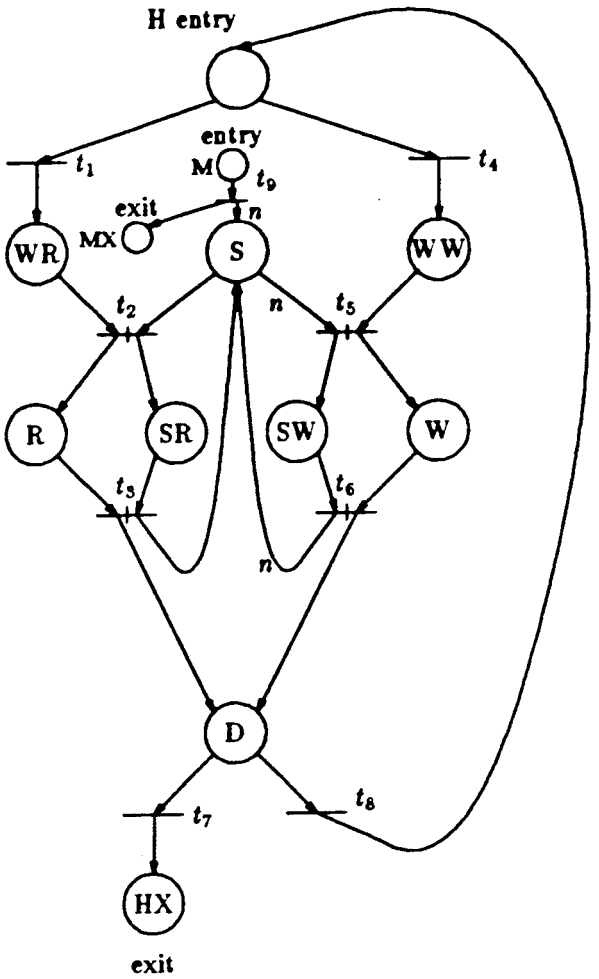



**Figure 16**

Contour/Transition-Net: MAIN

the split transition $t_2$ can fire and the process enters the reading state at place R (a token from the context of place S also goes to place SR). Sometime later the process completes its reading and fires transition $t_3$. As a result of firing $t_3$, a token is returned to place S and the process enters the done state (place D) which eventually leads back to the home state. Similarly, a token which fires transition $t_4$ is designated as a writer and proceeds to place WW (waiting to write). When all $n$ tokens are present in place S, then $t_5$ can fire absorbing $n$ tokens and placing the process in the writing state at place W (a token from the context of place S also goes to place SW). When the process finishes writing, it fires transition $t_6$ which returns the $n$ tokens to place S and causes the process to enter the done state.

| | Incidence Matrix | | | | | | | | | Markings | | Invariants | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $M_M$ | $M_H$ | $m_3$ | $m_4$ | $m_5$ |
| M | | | | | | | | | -1 | 1 | | 1 | $n$ | |
| S | | -1 | 1 | | $-n$ | $n$ | | | $n$ | | | | 1 | |
| SR | | 1 | -1 | | | | | | | | | | 1 | |
| SW | | | | | 1 | -1 | | | | | | | $n$ | |
| MX | | | | | | | | | 1 | | | 1 | | |
| H | -1 | | | -1 | | | | 1 | | | 1 | | | 1 |
| WR | 1 | -1 | | | | | | | | | | | | 1 |
| R | | 1 | -1 | | | | | | | | | | | 1 |
| WW | | | | 1 | -1 | | | | | | | | | 1 |
| W | | | | | 1 | -1 | | | | | | | | 1 |
| D | | | 1 | | | 1 | -1 | -1 | | | | | | 1 |
| HX | | | | | | | 1 | | | | | | | 1 |

**Table 5**

Incidence Matrix and Invariants for Figure 16

The properties of the system are now analyzed. Initially, note that the contour/transition-nets in Figures 15 and 16 already meet all of the restrictions discussed previously. The analysis begins with the initial state of $m(I) = 1$.

### *Boundedness Properties*

THEOREM 1. *The total system, as described in Figure 15, is bounded.*

From Figure 15, two invariants are found by inspection,

$$m(I) + m(MM) + m(L) = 1, \qquad (m1)$$

and

$$n * m(I) + m(P) + m(MH) = n. \qquad (m2)$$

These invariants can be automatically generated or proven using induction. These are the highest-level invariants of the system.

Before proving THEOREM 1, two lemmas concerning the named nets invoked by the system are stated and proven.

LEMMA 1. *MAIN.H is invocation-safe.*

Table 5 shows the incidence matrix for the contour/transition-net in Figure 16. Three invariants are present: two for the M entry place,

$$m(M) + m(MX) = 1, \qquad (m3)$$

and

$$n * m(M) + m(S) + m(SR) + n * m(SW) = n, \qquad (m4)$$

and one for the H entry place,

$$m(H) + m(WR) + m(R) + m(WW) + m(W) + m(D) + m(HX) = 1. \qquad (m5)$$

Note that invariant (*m5*), despite the fact that there are $n$ readers and writers in the system, equates to *1* and not $n$. This results from the fact that as far as boundedness analysis is concerned, the individual readers and writers (each $\pi$) do not interact. Second, note that the three invariants were derived in a slightly different way than the standard incidence matrix method: invariants (*m3*) and (*m4*) were derived by considering the incidence matrix as being composed only of rows for places M, S, SR, SW, and MX (the top half of Table 5). Similarly, invariant (*m5*) was derived by considering the incidence matrix as being composed only of rows for places H, WR, R, WW, W, D, and HX (the bottom half of Table 5). Hence, for our boundedness proof, MAIN is actually two nets which share four common transitions.

PROOF OF LEMMA 1. Directly from (*m5*): the control path along the H entry place in MAIN is 1-bounded, and the associated exit place can hold at most 1 token.

LEMMA 2. *MAIN.M is invocation-safe.*

PROOF OF LEMMA 2. Directly from (*m3*): the control path along the M entry place in MAIN is 1-bounded, and the associated exit place can hold at most 1 token.

PROOF OF THEOREM 1. Given LEMMA 1 and LEMMA 2, the proof of THEOREM 1 follows directly from (*m1*) and (*m2*). That is, the two invariants state that the net in Figure 15 is bounded provided that any named nets that it instantiates are

invocation-safe. Since MAIN.H and MAIN.M are the only named nets instantiated by the net in Figure 15, the two lemmas are sufficient to complete the proof. ∎

*System-specific Properties*

Let the hierarchy vector, $\vec{q}$, used to represent the state-space of the system, be of the form $\langle \vec{i}, \langle \vec{m}^j, \vec{h}^k \rangle \rangle$ where

$$\vec{i} = \langle S, MM, L, P, MH \rangle,$$

$$\vec{m} = \langle M, S, SR, SW, MX \rangle,$$

and

$$\vec{h} = \langle H, WR, R, WW, W, D, HX \rangle.$$

(To avoid clutter, the name of the place (e.g., $P$) is used instead of $m(P)$ in the three vectors above.)

System-specific invariants can now be proven about MAIN. In particular, it is necessary to ensure that the two properties

($i$): $v_R * v_W = 0$, and

($ii$): $v_W \leq 1$.

hold regardless of the state of the system.

THEOREM 2. *Properties ($i$) and ($ii$) are invariant in the system.*

The structural properties of the net will continue to be used to prove the system-specific properties. To begin, the incidence-invariants are translated into equations involving place-variables. For the net in Figure 15, since there is only one instance of the net executing, the equations are:

$$v_I + v_{MM} + v_L = 1, \qquad (n1)$$

and

$$n * v_I + v_P + v_{MH} = n. \qquad (n2)$$

For the net in Figure 16, the equations are:

$$v_M + v_{MX} = j, \qquad (n3)$$

$$n * v_M + v_S + v_{SR} + n * v_{SW} = n * j, \qquad (n4)$$

and

$$v_B + v_{WR} + v_R + v_{WW} + v_W + v_D + v_{BX} = k. \qquad (n5)$$

To show the relation between $\vec{i}$ and $(\vec{m}^j, \vec{h}^k)$.

LEMMA 3. *$j$ is at most 1 in the system.*

PROOF OF LEMMA 3. Directly from $(n1)$. Only one of $v_I$, $v_{MM}$, and $v_L$ can be greater than zero (and equal to 1), and the other two are zero. Furthermore, from the topology of the contour/transition-net in Figure 15, it follows that once $v_L = 1$, it remains so.

Hence, if $j$ refers to the number of times that MAIN.M is instantiated by the system, then $v_L = 1 \longrightarrow j = 1$. As a result of this, invariants $(n3)$ and $(n4)$ simplify to

$$v_M + v_{MX} = 1, \qquad (n3)$$

and

$$n * v_M + v_S + v_{SR} + n * v_{SW} = n. \qquad (n4)$$

LEMMA 4. *$k$ is at most $n$ in the system.*

PROOF OF LEMMA 4. Directly from $(n2)$, using a similar line of analysis as in the proof of LEMMA 3.

The relation between $\vec{m}$ and $\vec{h}^k$ is now considered. From the incidence matrix in Table 5, it is straightforward to show that the two properties

$(iii)$: $v_R = v_{SR}$, and

$(iv)$: $v_W = v_{SW}$.

are always true by using an inductive-invariant method: initially, $v_R = v_{SR} = 0$, and $v_W = v_{SW} = 0$, so $(iii)$ and $(iv)$ hold; furthermore, if $(iii)$ and $(iv)$ hold prior to the firing of any transition, then they hold after that transition fires. For the sake of brevity, the detailed proof is omitted. *It should be emphasized that these properties relate execution contexts in different nets.*

Hence, the system-specific properties which actually need to be proven are:

$(i')$: $v_{SR} * v_{SW} = 0$, and

$(ii')$: $v_{SW} \leq 1$.

PROOF OF THEOREM 2. Consider invariant $(n4)$. There are two cases to consider for $v_{SW}$. For the first case,

case 1: $$v_{SW} = 0,$$

from $(n4)$ it is known that up to $n$ readers may be active and both $(i')$ and $(ii')$ are satisfied. For the second case,

case 2: $$v_{SW} > 0,$$

from $(n4)$ it is known that $v_{SW} = 1$ and that $v_{SR} = 0$, which also satisfies $(i')$ and $(ii')$. ∎

*Deadlock Freeness*

To show freedom from deadlock, let the initial state be $m(L) = 1$.

THEOREM 3. *The total system, as described in Figure 15, is free from deadlock.*

Before proving THEOREM 3, a lemma concerning the named nets invoked by the system is stated and proven.

LEMMA 5. *MAIN.H and MAIN.M are invocation-live.*

PROOF OF LEMMA 5. Consider the sum

$$v_H + v_D + v_{HX}.$$

There are two cases. For the first case,

case 1: $$v_H + v_D + v_{HX} > 0,$$

and thus transitions $t_1$, $t_4$, $t_7$, and $t_8$ are firable for any $\pi$ with $m(H) + m(D) + m(HX) > 0$. For the second case,

case 2: $$v_H + v_D + v_{HX} = 0,$$

and

$$v_{WR} + v_R + v_{WW} + v_W = k,$$

which comes from $(n5)$. Now, given this equation, consider the sum

$$v_R + v_W.$$

There are again two cases. For the first case,

case 1: $$v_R + v_W > 0,$$

and it follows as a result of the analysis of system properties that $v_R = v_{SR}$ and $v_W = v_{SW}$. Hence, either $t_3$ or $t_6$ is firable for any $\pi$ with $m(H)+m(D)+m(HX) = 0$. For the second case,

case 2: $$v_R + v_W = 0,$$

and

$$v_{WW} + v_{WR} > 0$$

from $(n4)$ it is known that $v_S = n$. Hence, either transition $t_2$ or transition $t_5$ is firable for any $\pi$ with $m(H) + m(D) + m(HX) = 0$.

PROOF OF THEOREM 3. Consider the sum

$$v_I + v_{MM} + v_{MH}.$$

There are two cases. For the first case,

case 1: $$v_I + v_{MM} + v_{MH} > 0,$$

then transitions $t_1$, $t_2$ and $t_4$ are firable. For the second case,

case 2: $$v_I + v_{MM} + v_{MH} = 0,$$

from $(n1)$ it is known that $v_L = 1$ and from $(n2)$ it is known that $v_P = n$. Hence $t_3$ is firable. The proof of THEOREM 3 follows directly from this case analysis and LEMMA 5. That is, the case analysis shows that the net in Figure 15 is free from deadlock provided that any named nets that it instantiates are invocation-live. Since MAIN.H and MAIN.M are the only named nets instantiated by the net in Figure 15, the LEMMA 5 is sufficient to complete the proof. ∎

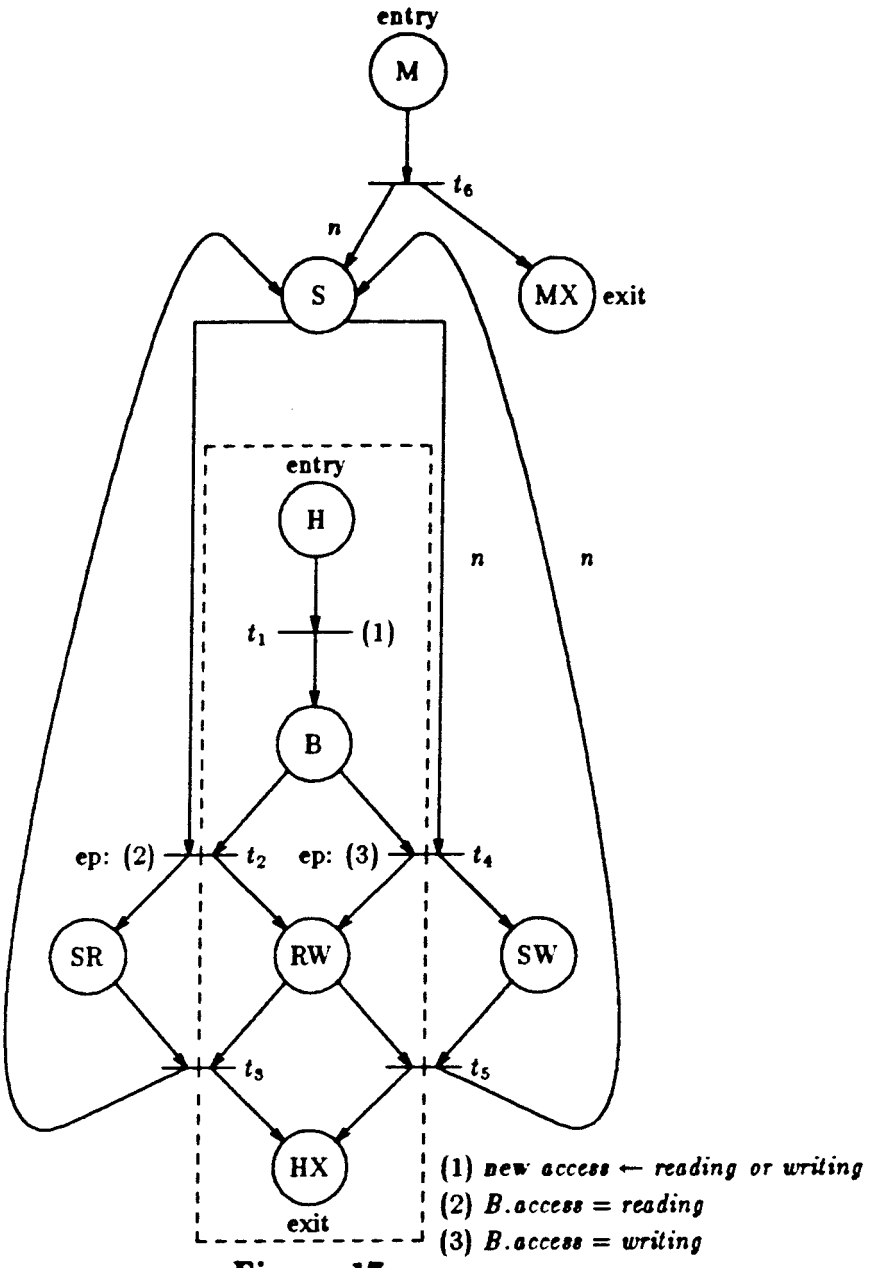



**Figure 17**

Revised Readers and Writers: MAIN

## Another Example

Our example is now modified to shift some state information from the control aspect to the data aspect.

As with the previous example, Figure 15 describes the nature of the system. A different net is used for MAIN, however. The MAIN net in Figure 17 has two entry places, just like the net in Figure 16. At the M entry place, control forks.

One fork generates $n$ (identical) tokens for place S. The other fork returns control to the net that instantiated MAIN.M. At place S, the process waits to service requests to read or write (a token appears in place B). Thus, MAIN.M in the net in Figure 17 is very similar to its counterpart in Figure 16. The other entry place is H, which is instantiated an arbitrary number of times by the driver process. (In the interests of readability, a dashed box appears around the control path for the MAIN.H control path in Figure 17). Each instance is characterized by a token of a unique color. When MAIN.H is instantiated, a new context is created and a new variable, *access*, is initialized as transition $t_1$ fires. The variable is set to either *reading* or *writing*, in a random fashion. A token with *access* = *reading* is designated as a reader and proceeds to place B. When at least one token is free in place S, then the split transition $t_2$ can fire and the process begins reading at place RW (a token from place S also goes to place SR). Sometime later the process completes its reading and fires transition $t_3$. As a result of firing $t_3$, a token is returned to place S, and the process reaches an exit place (place HX). Similarly, a token with *access* = *writing* is designated as a writer and proceeds to place B. When all $n$ tokens are present in place S, then $t_4$ can fire, absorbing the $n$ tokens, and the process begins writing at place RW (a token from place S also goes to place SW). When the process finishes writing, it fires transition $t_5$ which returns the $n$ tokens to place S and the process reaches an exit place.

Properties of this system are now analyzed. Initially, note that the contour/transition-nets in Figures 15 and 17 already meet all of our restrictions discussed previously. The analysis begins with the initial state of $m(I) = 1$.

### *Boundedness Properties*

THEOREM 4. *The total system, as described in Figure 15, is bounded.*

As the net in Figure 15 has not changed, the incidence-invariants of that net can be restated:

$$m(I) + m(MM) + m(L) = 1, \qquad (o1)$$

and

$$n * m(I) + m(P) + m(MH) = n. \qquad (o2)$$

Again, these are the highest-level invariants of the system.

| | Incidence Matrix | | | | | | Markings | | Invariants | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $M_M$ | $M_H$ | $o_3$ | $o_4$ | $o_5$ |
| M | | | | | | -1 | 1 | | 1 | $n$ | |
| S | | -1 | 1 | $-n$ | $n$ | $n$ | | | | 1 | |
| SR | | 1 | -1 | | | | | | | 1 | |
| SW | | | | 1 | -1 | | | | | $n$ | |
| MX | | | | | | 1 | | | 1 | | |
| H | -1 | | | | | | | 1 | | | 1 |
| B | 1 | -1 | | -1 | | | | | | | 1 |
| RW | | 1 | -1 | 1 | -1 | | | | | | 1 |
| HX | | | 1 | | 1 | | | | | | 1 |

**Table 6**

Incidence Matrix and Invariants for Figure 17

Before proving THEOREM 4, two lemmas concerning the named nets invoked by the system are stated and proven.

LEMMA 6. *MAIN.H is invocation-safe.*

Table 6 shows the incidence matrix for the contour/transition-net in Figure 17. Three invariants are present: two for the M entry place,

$$m(M) + m(MX) = 1, \tag{o3}$$

and

$$n * m(M) + m(S) + m(SR) + n * m(SW) = n, \tag{o4}$$

and one for the H entry place,

$$m(H) + m(B) + m(RW) + m(HX) = 1. \tag{o5}$$

Again, note that invariant (*o5*), despite the fact that there are $n$ readers and writers in the system, equates to *1* and not $n$. Invariants (*o3*) and (*o4*) were derived by considering the incidence matrix as being composed only of rows for places M, S, SR, SW, and MX (the top half of Table 6). Similarly, invariant (*o5*) was derived by considering the incidence matrix as being composed only of rows for places H, B, RW, and HX (the bottom half of Table 6).

PROOF OF LEMMA 6. Directly from (*o5*): the control path along the H entry place in MAIN is 1-bounded, and the associated exit place can hold at most 1 token.

LEMMA 7. *MAIN.M is invocation-safe.*

PROOF OF LEMMA 7. Directly from (*o3*): the control path along the M entry place in MAIN is 1-bounded, and the associated exit place can hold at most 1 token.

PROOF OF THEOREM 4. Given LEMMA 6 and LEMMA 7, the proof of THEOREM 4 follows directly from (*o1*) and (*o2*). ∎

*System-specific Properties*

Let the hierarchy vector, $\vec{q}$, used to represent the state-space of the system, be of the form $\langle \vec{i}, \langle \vec{m}^j, \vec{h}^k \rangle \rangle$ where

$$\vec{i} = \langle S, MM, L, P, MH \rangle,$$

$$\vec{m} = \langle M, S, SR, SW, MX \rangle,$$

and

$$\vec{h} = \langle H, B, RW, HX, access \rangle.$$

To prove system-specific invariants about MAIN, let $\pi_r$ denote an instance of a $\vec{h}$ context such that

$$access = reading \wedge m(RW) > 0,$$

and $|\sum \pi_r|$ denotes the ordinality of all such processes; and, let $\pi_w$ denote an instance of a $\vec{h}$ context such that

$$access = writing \wedge m(RW) > 0,$$

and $|\sum \pi_w|$ denotes the ordinality of all such processes. It is necessary to ensure that the two properties

(*i*): $|\sum \pi_r| * |\sum \pi_w| = 0$, and

(*ii*): $|\sum \pi_w| \leq 1$.

hold regardless of the state of the system.

THEOREM 5. *Properties (i) and (ii) are invariant in the system.*

The structural properties of the net are again used to prove the system-specific properties. To begin, the incidence-invariants are translated into equations involving place-variables. As the net in Figure 15 has not changed, those equations can be restated as:

$$v_I + v_{MM} + v_L = 1, \tag{p1}$$

and

$$n * v_I + v_P + v_{MH} = n. \tag{p2}$$

For the net in Figure 17,

$$v_M + v_{MX} = j, \tag{p3}$$

$$n * v_M + v_S + v_{SR} + n * v_{SW} = n * j, \tag{p4}$$

and

$$v_H + v_B + v_{RW} + v_{HX} = k. \tag{p5}$$

To show the relation between $\vec{i}$ and $(\vec{m}^j, \vec{h}^k)$.

LEMMA 8. *$j$ is at most 1 in the system.*

PROOF OF LEMMA 8. By the proof made for LEMMA 3.

Hence, invariants $(p3)$ and $(p4)$ simplify to

$$v_M + v_{MX} = 1, \tag{p3}$$

and

$$n * v_M + v_S + v_{SR} + n * v_{SW} = n. \tag{p4}$$

LEMMA 9. *$k$ is at most $n$ in the system.*

PROOF OF LEMMA 9. By the proof made for LEMMA 4.

The relation between $\vec{m}$ and $\vec{h}^k$ is now considered. In particular, the following properties are invariant:

(*iii*): $|\sum \pi_r| = v_{SR}$, and

(*iv*): $|\sum \pi_w| = v_{SW}$.

Initially, $\vec{m} = \vec{0}$ and $\vec{h}^k = \vec{0}^k$, so (*iii*) and (*iv*) hold. For each transition in MAIN, note that if (*iii*) and (*iv*) hold prior to the firing of any transition, then they hold after that transition fires. For example, when transition $t_2$ fires, both $|\sum \pi_r|$ and $v_{SR}$ increase by 1. Similar observations hold for transitions $t_3$, $t_4$, and $t_5$. Transitions $t_1$ and $t_6$ do not affect places RW, SR, or SW.

Finally, note that the following property is also invariant:

(*v*): let $\pi_h$ denote an instance of a $\vec{h}$ context, then *access* = *reading* or *access* = *writing* in that context.

For each transition in MAIN, if (*v*) holds prior to the firing of any transition, then it holds after that transition fires: only transition $t_1$ modifies *access*, setting it to *reading* or *writing*. Although other transitions may examine the value of *access*, they do not modify the value.

By combining (*i*), (*ii*), (*iii*), and (*iv*), the system-specific properties which actually need to be proven are:

(*i'*): $v_{SR} * v_{SW} = 0$, and

(*ii'*): $v_{SW} \leq 1$.

PROOF OF THEOREM 5. The proof is similar to the proof for THEOREM 2. Consider invariant (*p4*). There are two cases to consider for $v_{SW}$. For the first case,

case 1: $$v_{SW} = 0,$$

from (*p4*) that up to $n$ readers may be active and both (*i'*) and (*ii'*) are satisfied. For the second case,

case 2: $$v_{SW} > 0,$$

from (*p4*) that $v_{SW} = 1$ and that $v_{SR} = 0$, which also satisfies (*i'*) and (*ii'*). ∎

*Deadlock Freeness*

To show freedom from deadlock, let the initial state be $m(L) = 1$.

THEOREM 6. *The total system, as described in Figure 15, is free from deadlock.*

Before proving THEOREM 6, a lemma concerning the named nets invoked by the system is stated and proven.

LEMMA 10. *MAIN.H and MAIN.M are invocation-live.*

PROOF OF LEMMA 10. Since MAIN contains enabling predicates, the existence of a homing state, $q_H$, for $\vec{h}^k$ is shown. Let $n(\vec{q})$ be a norm with zero-state $q_0$, defined as

$$4 * v_H + 3 * v_B + 2 * \left( \left| \sum \pi_r \right| + \left| \sum \pi_w \right| \right) + v_{HX}.$$

First, note that $n(\vec{q}) = 0$ if and only if $\vec{q} = q_0$.

Second, for any state, $\vec{q}_x$, reachable from the initial marking, if $n(\vec{q}_x) > 0$, then there exists a state, $\vec{q}_y$, reachable from $\vec{q}_x$, such that $n(\vec{q}_x) > n(\vec{q}_y)$.

case 1: $|\sum \pi_r| > 0$
from (*iii*), $v_{SR} > 0$, so $t_3$ fires, reducing $n$.

case 2: $|\sum \pi_w| > 0$
from (*iv*), $v_{SW} > 0$, so $t_5$ fires, reducing $n$.

case 3: $|\sum \pi_r| + |\sum \pi_w| = 0 \wedge v_H > 0$
$t_1$ fires, reducing $n$.

case 4: $|\sum \pi_r| + |\sum \pi_w| = 0 \wedge v_B > 0$
from (*v*), either $t_2$ or $t_4$ may fire, each reducing $n$.

case 5: $|\sum \pi_r| + |\sum \pi_w| = 0 \wedge v_{HX} > 0$
place HX is an exit place so the exit transition in the net that invoked this instance of MAIN.H will fire, removing the token from HX and reducing $n$. In this case, the value of $k$ in (*p5*) decreases by 1 since there is one less instantiation of $\vec{h}$ in the system.

These five cases cover the entire state-space for $\vec{h}^k$. Homing-$q_0$ has been shown to be $q_0$-invariant for the net since the existence of a norm for $n(\vec{q})$ has been proven. Hence, the net is free from deadlock.

PROOF OF THEOREM 6. Consider the case analysis presented in the proof of THEOREM 3. The proof of THEOREM 6 follows directly from this analysis and LEMMA 10. ∎

## Evaluation

This chapter has evaluated the applicability of various analysis methods to the contour/transition model. In summary, incidence matrix analysis, with some minor extensions, is quite suitable for proving properties about contour/transition-nets that do not involve enabling predicates. The incidence-invariants that are derived from the incidence matrix permit demonstration system-specific and deadlock freeness properties. If enabling predicates are present, then the invariant-method is particularly well-suited for contour/transition-nets. After extending the invariant-method, it is possible to achieve results for the contour/transition model similar to those that Keller showed for the presentational model.

In comparing our method to the proof method for the distinct process extension to the presentational model, note that the hierarchy vector is a more general mechanism for representing the state space of the system than the two-level access scheme used by the former method. Of particular promise is the way in which the hierarchy vector allows us to state the relationships between different contents. With both approaches, the same properties are proven, using very similar methods. Furthermore, note that the proof process is simplified with both methods by isolating where contextual interference can occur. In the contour/transition model, this is limited to split and boundary transitions and named nets, while in the distinct process extension to the presentational model, each transition is queried for its affect on global variables and variables local to an individual $\pi$.

Note that the proof method discussed is not complete, as it restricts the use of certain features of the contour/transition model. Clearly, a future area of research is to generalize the augmentations and further extend them to remove the restrictions mentioned above. Such extensions include addressing issues of timing so that the analysis technique may take into account enabling and firing delays in the transitions. In addition, it would be interesting to investigate the use of multi-level homing-state proofs in which the relation between different contexts in the hierarchy vector is further exploited.

# CHAPTER 5
# A Lengthy Example

This chapter presents an experimental evaluation of the contour/transition model. Contour/transition-nets are used to model the handling of initial connection establishment by the DOD Transmission Control Protocol.

Initially, a terse discussion of the problems of establishing connections in a network takes place. This discussion centers on the use of the three-way handshake, which is used by TCP as a solution for many of these problems.

Next, the description of the three-way handshake used in TCP is made. The description is presented in three sections: first, a general set of notes concerning the nature of this particular description is discussed; second, the data definitions of the description are given; and, third, the actual nets themselves are presented.

Finally, the model is contrasted with other formal specifications of the three-way handshake in TCP and then the success of contour/transition-nets in modeling this problem isi evaluated.

## Connection Establishment

The problems of providing reliable virtual-circuit service over a potentially unreliable packet-switched network are discussed in great detail by Sunshine and Dalal[SUNS78B]. A central problem in this area is ensuring that a connection established between two processes in the network becomes synchronized and remains so. Each peer of the N-protocol that provides reliable communications to these processes must agree as to the state of the connection and then update that state as changes occur.

A connection may be viewed as traversing through a number of states at each end. These states trace the activity of the connection from non-existence to establishment, then to data transfer, and finally to closing. Consider the means by which two N-peers establish a connection. Initially, a connection may be viewed as being closed. When a process indicates that it is willing to accept a connection

from another process, the connection for that process enters a listening state where the process waits for another process to complete the connection. This is known as a *passive open*. After this point, another process may attempt to complete the connection by performing an *active open*. Providing that all of the appropriate conditions are met, the connection progresses to the established state at both ends. Alternately, both processes may try to actively open the connection simultaneously, and the N-peers must be able to handle this situation correctly.

Connections occur between two sockets in the network. A socket is a pairing between the address for a host in the network and a local port number for that host. A connection may be uniquely specified by listing the two sockets participating in the connection, as a socket uniquely identifies the end-point of a particular conversation. Usually, for passive opens, a process requests a local port number and does not specify a foreign socket. In contrast, for active opens, a process requests both a local port number and a foreign socket.

Information to be sent from one process to another is first given to the local N-peer which then encapsulates the information in a *segment*. In addition to containing the data to be transmitted, the segment contains control information for the use of the N-peers. For our purposes, three control bits that can be found in the segments that the N-peers exchange are interesting. The SYN bit indicates that an N-peer is requesting initial synchronization. The ACK bit indicates that an N-peer is acknowledging a previously received segment. The RST bit indicates that an N-peer is demanding that the connection should be reset.

At some time, a segment containing a SYN arrives. When the foreign and local sockets specified in the segment match the socket pairing specified by a process' open, a connection begins. To synchronize the connection, initial sequence numbers are exchanged between the two peers. These sequence numbers impose an ordering on the data exchanged by the peers. Selection of the initial sequence number is a tricky business, as one must take great care to ensure that segments floating about from instantiations of previous connections have sequence numbers outside the range of legitimate sequence numbers. Once both peers have selected an initial sequence number, informed the other peer of the number, and received an acknowledgement, the connection becomes established. If something goes amiss in the connection establishment, a peer sends a segment

containing a RST to the other peer. This has the effect of removing all traces of the connection, with the appropriate information returned to the process associated with the peer receiving the RST. Other information may be exchanged during this synchronization, including process data (to be passed up only when the connection is fully established), but these considerations are not germane to the focus of the discussion.

This method of synchronization is referred to as a *three-way handshake* since the simplest case of its operation can be summarized as follows: Process A performs a passive open. Some time later, process B performs an active open. This results in process B's TCP choosing an initial sequence number and sending a segment containing a SYN and the sequence number to process A's TCP. Process A's TCP receives the segment, examines it, chooses an initial sequence number of its own, and responds by sending a segment containing a SYN, the sequence number, and an ACK of the incoming segment. Process B's TCP receives the segment, examines it, and decides that the connection is established. In addition, process B's TCP sends a segment acknowledging the incoming segment. Upon receipt of this segment, process A's TCP also decides that the connection is established.

The three-way handshake is able to successfully deal with a large number of variations and exceptions, including such events as simultaneous active opens, duplicate segments, segments from other instantiations, and half-open connections (which occur when one host crashes during the synchronization activity and loses all knowledge of a connection). Hence, it is a good initial connection protocol for use by a reliable virtual-circuit service.

### Description Notes

This section provides a general explanation of the description that follows.

First, it must be noted that the following sections do not provide a complete description of TCP. Only those aspects of TCP that deal with connection establishment are examined. Furthermore, some aspects of TCP that play a minor role in connection establishment are abstracted to avoid unnecessary detail. Components of TCP which are given little attention are: TCP options and option handling, urgent data handling, windowing, user time-outs, and precedence and security/compartment considerations. The description presented pays varying

(small) degrees of attention to these aspects of TCP. In contrast, the TCP specification[TCP] fully considers all of these issues in its discussion of initial connection establishment.[12]

Second, the description uses short mnemonics to represent error conditions which may be raised and given to the user process. Their meanings are:

*REFUSED* — the connect was refused by the foreign peer

*RESET* — the connection was reset by the foreign peer

*ISCONN* — the connection is already established

*NOBUFS* — insufficient resources to service request

*NOPEER* — no foreign socket specified in an active *open*() call

*DENIED* — the user process is not allowed to specify this type of *open*() call

Third, the underlying (N-1)-service is presumed to be the DOD Internet Protocol[IP]. The discussion of the (N-1)-interface describes the type of service expected.

### Description Data Definitions

This section describes the data structures used by the description. Three major structures are explicitly used the *tcb*, *ip_type*, and *segment_type* structures.

The *tcb* structure contains all of the state information for a connection. In addition to the local and foreign sockets, all sequencing and windowing information, precedence and security/compartment information, and so forth are all kept in this structure. In the description that follows, a *tcb* completely contains all known information about a connection for an N-peer. A *tcb* is in fact its own contour and as a result has its own unique color. In the next section, the reader should be able to appreciate the advantages and disadvantages that this interpretation permits. Unlike the original TCP specification, the *tcb* does not have its state encoded as a variable. Instead, contour/transition-nets are used to denote the state of a *tcb*.

---

[12] There are two widely distributed versions of the specification of TCP. This chapter is based on the earlier version, [TCP]. The other document is the MILSTD specification of TCP, known as *MIL-STD-1778*. To the author's understanding, these two documents describe precisely the same protocol, although in different formats. The former is a natural language treatise, the latter consists mainly of an algorithmic specification. At the time this section of the dissertation was prepared, only the earlier specification was available.

The *ip_type* structure is the data type passed to/from the (N-1)-layer. This structure specifies the source and destination addresses, precedence and security/compartment information, a protocol code (which for our purposes is always the code for TCP), a time-to-live value, and a segment to be communicated. The *ip_type* structure is used as a part of the (N-1)-interface definition.

The *segment_type* structure is passed between N-peers as their means of communication. The definition of the *segment_type* follows fairly closely the definition given in the TCP specification, with a few exceptions. Each segment has associated with it a length. This is not kept explicitly in the segment, but instead is calculated based upon the values of various components in the segment. In order to make this description more clear, the length is treated as an explicit component of a *segment_type*. In addition, although the source and destination addresses are not present in each segment, they have been made explicit components of a *segment_type* as well. Similarly, pointers to the urgent, data, and options portions of the *segment_type* have been abstracted somewhat.

All of the major data structures used in the description are presented in Figure 18. Figure 18 presents these in a rough C-like syntax, and a complete description of the semantics of these structures is not presented here.

### Description Nets

This section presents the actual description itself. A series of contour/transition-nets are presented, along with additional explanatory text.

Some conventions are used in the drawing of these nets which should be noted. First, the graphical conventions presented earlier in this dissertation are followed. The primary motivation for this is to reduce clutter and make the nets appear more readable. As expected, this results in the loss of some simplicity. For example, some of the transitions presented have more than one output arc. One of these arcs may lead to a named net while another may not. This means that the transition also serves as an entry transition if the selection rule for the output arc feeding the named net produces a token. Hybrid constructs of this sort are meant as a convenient short-hand notation. In the interests of further brevity, one additional liberty has been taken in drawing these nets: the entry places do not declare the types of the formal parameters. This information has been excluded in

```
struct  tcb {
        socket_type       lsock, fsock;
        precedence_type   prc;
        security_type     sec;
        boolean           active_open;
        sequence_type     iss, irs;
        window_type       wnd;
        s_wnd_type        snd;
        r_wnd_type        rcv;
        usr_sig_type      timeout;
        segment_type      msg, seg;        /* not state information */
        segment_queue     retransmit;
};
struct  ip_type {
        addr_type         saddr, daddr;
        protocol_type     proto;
        precedence_type   prc;
        security_type     sec;
        time_type         ttl;
        segment_type      data;
};
struct  segment_type {
        addr_type         saddr, daddr;    /* not actually in the segment */
        port_type         sport, dport;
        seg_flags_type    ctl;
        integer           len;             /* not actually in the segment */
        sequence_type     seq, ack;
        window_type       wnd;
        seg_ptr           up, data, options;
        integer           data_offset;
        checksum_type     cksum;
};
struct  s_wnd_type {
        sequence_type     una, nxt, up, wl1, wl2;
        window_type       wnd;
};
struct  r_wnd_type {
        sequence_type     nxt, up;
        window_type       wnd;
};
struct  socket_type {
        addr_type         addr;
        port_type         port;
};
struct  seg_flags_type {
        boolean           urg, ack, psh, rst, syn, fin;
};
```

**Figure 18**

Data Definitions

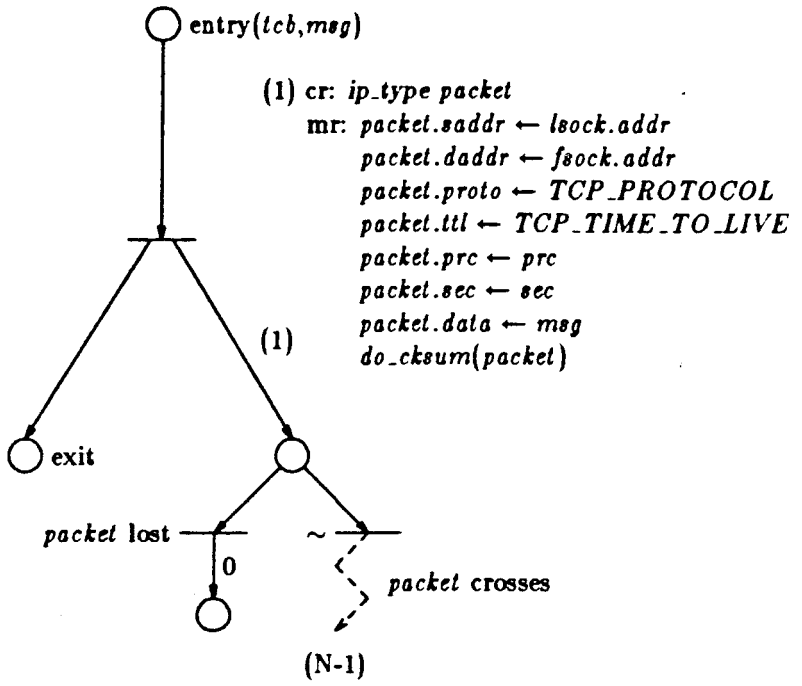



**Figure 19**

(N-1)-interface: SNDPKT

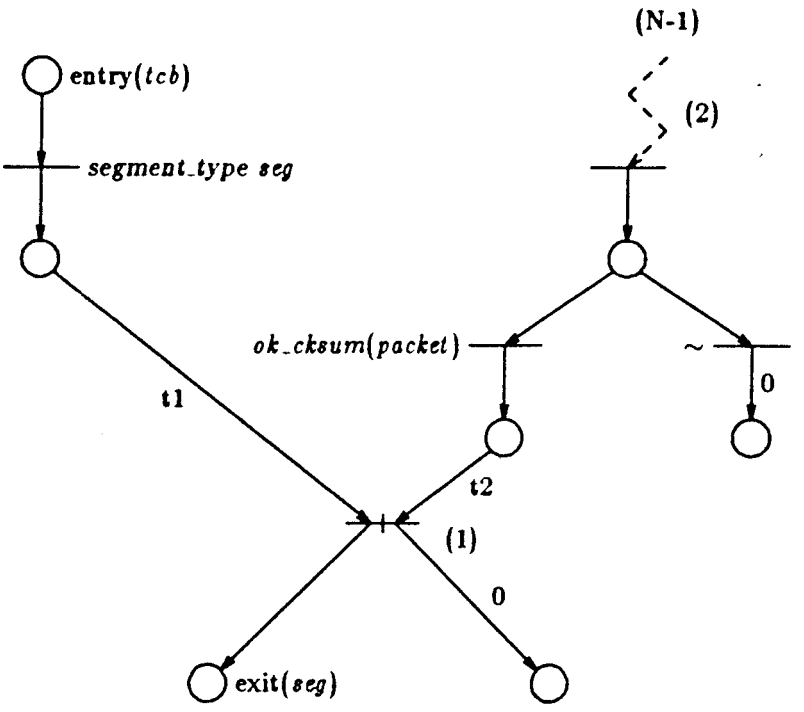
order to avoid certain space limitations in preparing this dissertation. In all cases though, the type of the formal parameters is easily determined by inspection.

Second, the entire description is not composed entirely of nets. Very often, the enabling predicate or firing actions of a transition may reference an external routine or predicate to provide a value. These functions are assumed to be called in the current context. An example of an enabling predicate so specified would be one that uses the expression *P/S okay*, which appears throughout the description and asks if the precedence and security/compartment conditions properly met. An example of a routine appearing in the firing actions for a transition would be *new_iss*() which generates a new initial sequence number for a connection.

*(N-1)-interface*

The (N-1)-interface description is achieved through two nets, SNDPKT and RCVPKT.

The SNDPKT net (Figure 19) loads the appropriate information for the (N-1)-layer into an *ip_type* structure and then forks control. One fork leads to the net's exit place. The other leads to a place which feeds one of two transitions. One




(1) ep: t2's *packet* is for t1's connection
mr: *t1.seg* ← *t2.packet.data*
(2) *ip_type packet* crosses

**Figure 20**

(N-1)-interface: RCVPKT

transition consumes the eligible token and does not introduce a token on its output arc. This represents the packet being lost by the (N-1)-layer. The other transition is a boundary transition. The firing actions for this transition state that the data structure known as *packet* crosses into the (N-1)-media.

Recall that firing actions, while indivisible, execute in an ordered fashion. Hence, the manipulation rules, after loading each field of the *ip_type* structure, calls *do_chksum*() to calculate the TCP checksum for the segment and the (imaginary) IP header. This corresponds to an interesting nuance in the TCP specification that actually includes information not found in the TCP segment into the checksum stored in the segment. Note that the although the selection rules for the transition introduced a token on both output arcs, only one arc had construction rules and manipulation rules to execute. Since only one set of manipulation rules is present, the order of execution is unimportant.

The RCVPKT net (Figure 20) is instantiated to accept information from the (N-1)-media. When a token enters the net, it waits for a split transition to be enabled. This transition will be enabled when a token is present on the other input arc which contains a packet for the connection represented by the entry token. That is, control will block until an incoming packet for the connection arrives at the other input place. Independently of this, whenever any packet arrives, an *ip_type* structure is given to the N-peer by the (N-1)-layer, and placed in a blank token by the boundary transition. The routine *ok_chksum*() is called to verify the checksum of the incoming packet (using the same algorithm which checks information in both the segment and the (imaginary) IP header). If the checksum found in the segment is incorrect, the blank token containing the *ip_type* structure is dropped. Otherwise, the token proceeds to a place to wait for the N-peer to request the next segment. When this split transition is enabled, the firing actions specify that the incoming packet is copied into the entry token's variable *packet* and that *seq* ← *packet.data* in the context of this token. After firing, as with all split transitions, control forks. The fork corresponding to the entry token exits, which returns control to the caller. The other fork, which corresponds to the blank token that was introduced by the boundary transition, terminates control.

It should be noted that the topology of Figure 20 does not enforce an ordering on the incoming packets. If two packets with a correct checksum arrive before the split transition fires, then the choice as to which packet is chosen as eligible is non-deterministic. The Internet Protocol, which is the (N-1)-service, is datagram oriented and may deliver packets out of order. Figure 20 demonstrates very simply that the order of incoming packets is unimportant to the description.

These two nets compose the entire (N-1)-interface description. Conceptually, one could view the two transition boundaries presented in these nets as being joined between two N-peers, as these are the only transition boundaries in the description.

### *N-interface*

Only a partial description of the N-interface is made. In particular, of the several calls that processes may make upon TCP in the TCP specification, only one call, the *open*() call is specified, as this is the only call that actually deals with connection establishment. The form of the *open*() call described here is

somewhat more limited than that presented in the original TCP specification. The description herein permits a user process to issue an open request only for those connections which are in the CLOSED state.

The *open*() call takes the following input parameters:

- *port_type lport*, which indicates the local port that the user process wants to use as its end-point in a connection.
- *socket_type fsok*, which indicates the foreign socket that the user process wants to connect with. If this is omitted or partially given, then any foreign socket matching the requirements can establish a connection.
- *boolean active*, which indicates if an active or passive open should be done.
- *usr_sig_type timeout*, which indicates, if given, a user signal handling routine to be notified when a segment is not be acknowledged within a certain time limitation.
- *precedence_type prc*, which indicates the precedence level that the connection should have.
- *security_type sec*, which indicates the security/compartment level that the connection should have.
- *options_type options*, which specifies any TCP options to be used.

When a user process issues an *open*() call, an *open request* is issued on behalf of the user process. This results in the OPEN net (presented momentarily) being instantiated, given the parameters specified by the user process for the *open*() call. During the execution of OPEN, either an error code or a connection handle is returned to the user process. If a connection handle is returned, then an associated *tcb* has been instantiated for the user process.

### *N-protocol*

The actions of the N-protocol for a particular connection start with the OPEN net. Before describing OPEN, it is necessary to digress and introduce another net, MAIN.

If a segment arrives for a connection that does not exist, some method for rejecting the segment must exist. This generally requires some global knowledge of all of the connections that are known on the local host. Since RCVPKT receives

all incoming packets and only known connections instantiate RCVPKT to fetch packets. packets arriving for non-existent connections will "stack-up" at the place feeding the split transition in Figure 20. Clearly, this is incorrect behavior. Hence, Figure 20, although correct on a per-connection basis and correct as a description of part of the (N-1)-interface, is not actually invoked by the system. Rather, connections wishing to fetch the next packet meant for them instantiate the MAIN net at the RCVPKT entry place.

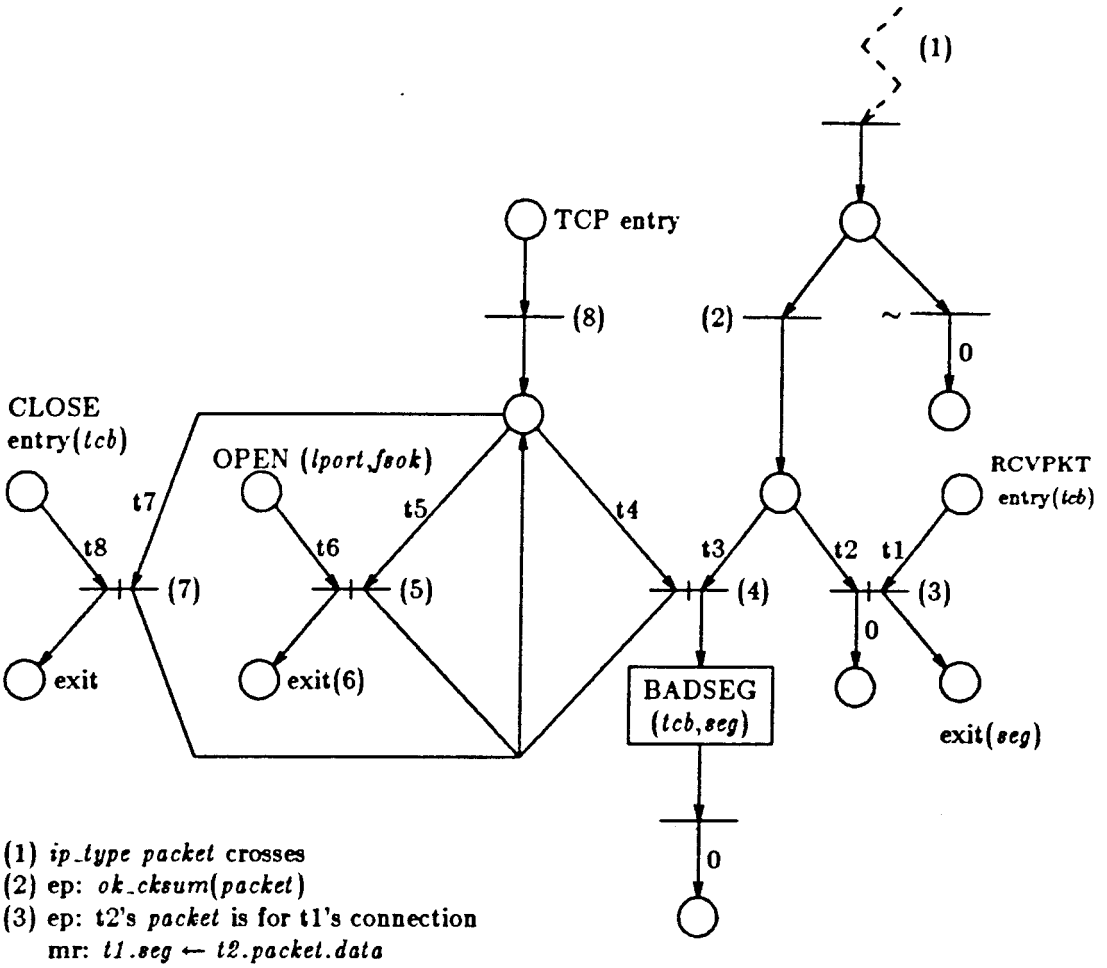
The MAIN net (Figure 21) specifies several parts of the system. When the system starts, it instantiates MAIN at the TCP entry place. MAIN.TCP is viewed as a "single entry, zero exit" net. In short, an instantiation of MAIN.TCP never returns. The TCP entry place begins by establishing a new context containing a list of connections which is initially empty. It then proceeds to a place where it can enable any of three split transitions. Once any of these transitions fire, control loops back.

If the OPEN entry place is instantiated in MAIN, a split transition fires which checks if the connection specified by the parameters *lport* and *fsok* is present in the connection list. If so, the *ISCONN* error is raised, and this is returned to the caller. Otherwise, a connection handle is associated with a new connection for *lport* and *fsok*. This connection is added to the list of known connections, and the connection handle is returned to the caller.

If the CLOSE entry place is instantiated in MAIN for a connection, a split transition fires which removes the connection from the list of known connections and flushes any queues and releases any resources associated with the connection.

Independent of the instantiation of the OPEN and CLOSE entry places, if a packet arrives for a connection that is not present in the list of known connections, a third split transition fires which invokes the BADSEG net (presented later). BADSEG, described later, rejects the segment by constructing and sending an appropriate segment.

The remainder of the MAIN is essentially the RCVPKT net. The RCVPKT entry place in MAIN corresponds to the single entry place in the RCVPKT net,




(1) *ip_type packet* crosses
(2) ep: *ok_cksum(packet)*
(3) ep: t2's *packet* is for t1's connection
mr: *t1.seg* ← *t2.packet.data*
(4) ep: t3's *packet* is not for any connection known in the context of t4
cr: *tcb* initialized for dummy connection
(5) mr: if the connection specified by t6's *lport* and *fsok*
is known in the context of t5, then raise *ISCONN*
otherwise, associate a connection handle in the context of t6
and add the connection to the list in the context of t5
(6) return error or connection handle
(7) mr: remove t8's connection from t7's list of connections
flush any queues and release any resources associated with t8's connection
(8): cr: new list of connections, initially empty

**Figure 21**

N-protocol: MAIN

and the split transition has the same enabling predicate and firing actions.[13] The interesting interaction, which was the motivation for merging the RCVPKT net into MAIN, is that the place which supplies incoming packets to the RCVPKT

[13] The observant reader will notice a slight deficiency in Figure 21. Due to unfortunate space limitations, a transition is missing immediately after the RCVPKT entry place. This transition creates the *segment_type seg* which is returned. The author apologizes for this inconsistency.

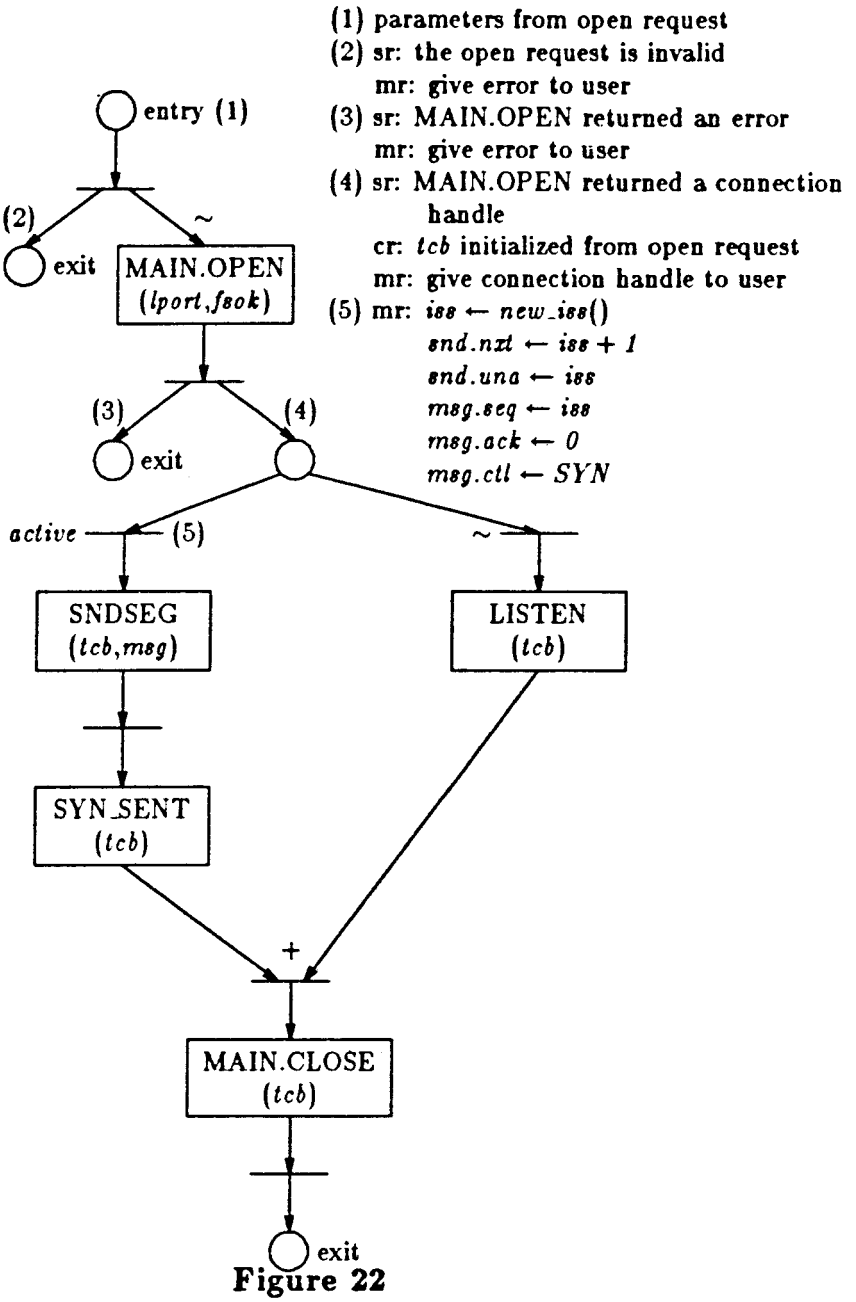



**Figure 22**

N-protocol: OPEN

control path can also supply incoming packets to the BADSEG control path. Since this place feeds both transitions, either may make use of incoming packets depending on how well their enabling predicates are satisfied.

Now that MAIN has been discussed, the parts of the description dealing with individual connections may be presented. As discussed above, an *open request*

results in the OPEN net (Figure 22) being instantiated with the appropriate parameters. OPEN first checks the validity of the request.

This check can be summarized as:

1. If the user process is not allowed to access *lport*, or is not allowed to use the indicated *prc* and *sec* levels, then the error *DENIED* should be raised.
2. If *active* is set, but *fsok* is not fully specified, then the error *NOPEER* should be raised.
3. If there are insufficient resources to handle this request, then the error *NOBUFS* should be raised.
4. Otherwise, the request is considered valid.

If the request is not valid, the appropriate error is given to the user process, and control returns as well. Otherwise, the MAIN net is instantiated at the OPEN entry place. If MAIN.OPEN returned an error, the error is given to the user process and control returns. Otherwise, the connection handle returned by MAIN.OPEN is given to the user process. Next, the OPEN net checks to see if the user process wanted an active open. If so, a SYN segment is sent, and the connection enters the SYN_SENT state (by instantiating the SYN_SENT named net). If, instead, a passive open was requested, the connection enters the LISTEN state. Eventually, the path taken returns and the MAIN net is instantiated at the CLOSE entry place. When MAIN.CLOSE returns, control returns as the connection has now entered the CLOSED state and no longer exists.

The LISTEN named net (Figure 23) is used to process a connection that is in the LISTEN state. First, RCVSEG is instantiated to await and return an incoming segment for this connection. A RST is explicitly checked for. If present, LISTEN ignores the segment. If an ACK is present, the segment is rejected. If a SYN is not present, the segment is ignored. If a SYN is present, but precedence and security/compartment considerations are not satisfied, then the segment is rejected. Otherwise, the connection enters the SYN_RCVD state, and a SYN/ACK is sent as the second part of the three-way handshake. If more of TCP were being considered, other parts of the segment might be processed prior to entering the SYN_RCVD state.

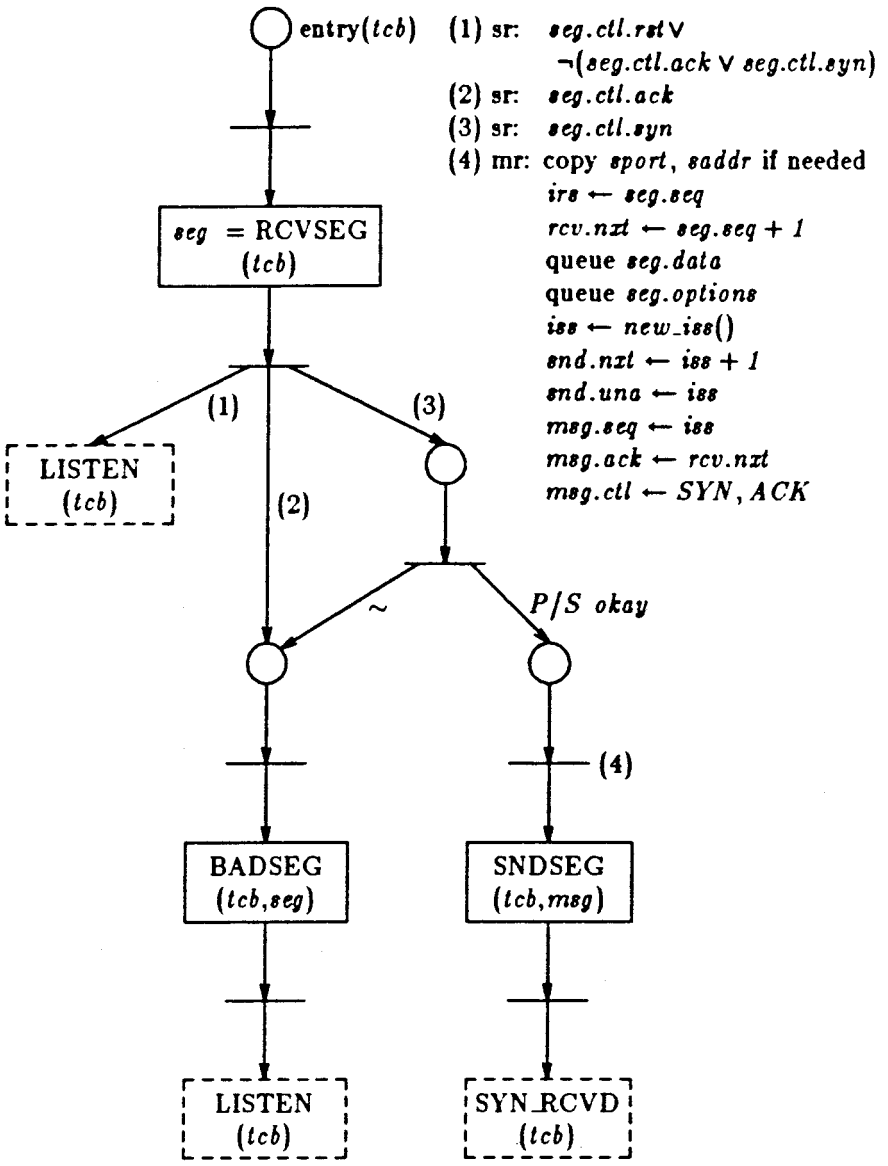



**Figure 23**

N-protocol: LISTEN

The SYN_SENT named net (Figure 24) is used to process a connection that is in the SYN_SENT state. First, RCVSEG is instantiated to await and return an incoming segment for this connection. If the segment acknowledges a segment not belonging to this instance of this connection, the segment is rejected. Otherwise the presence of a RST and a SYN is checked. If a RST is present, and this is an acknowledgement, then the user process is informed that the connection was rejected, and control returns (to the OPEN net). Otherwise, the connection

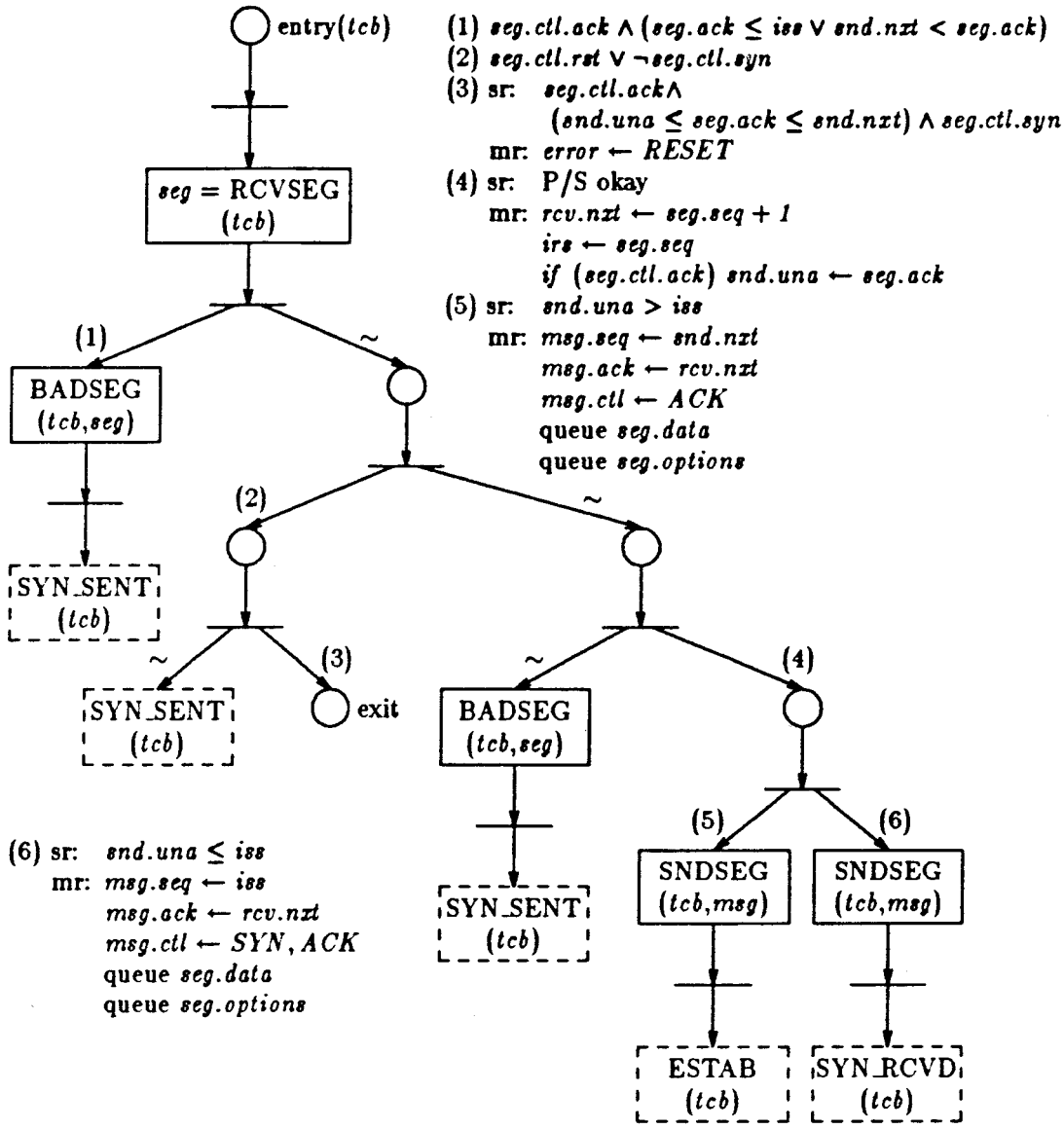



**Figure 24**

N-protocol: SYN_SENT

remains in the SYN_SENT state (the segment is ignored). Precedence and security/compartment considerations are then checked. If they are not satisfied, then the segment is rejected. If the considerations are satisfied, then a response is sent, and the connection enters either the ESTAB or SYN_RCVD state, depending on whether our SYN has been acknowledged. Again, if more of TCP were being considered, other parts of the segment might be processed prior to entering the new state.

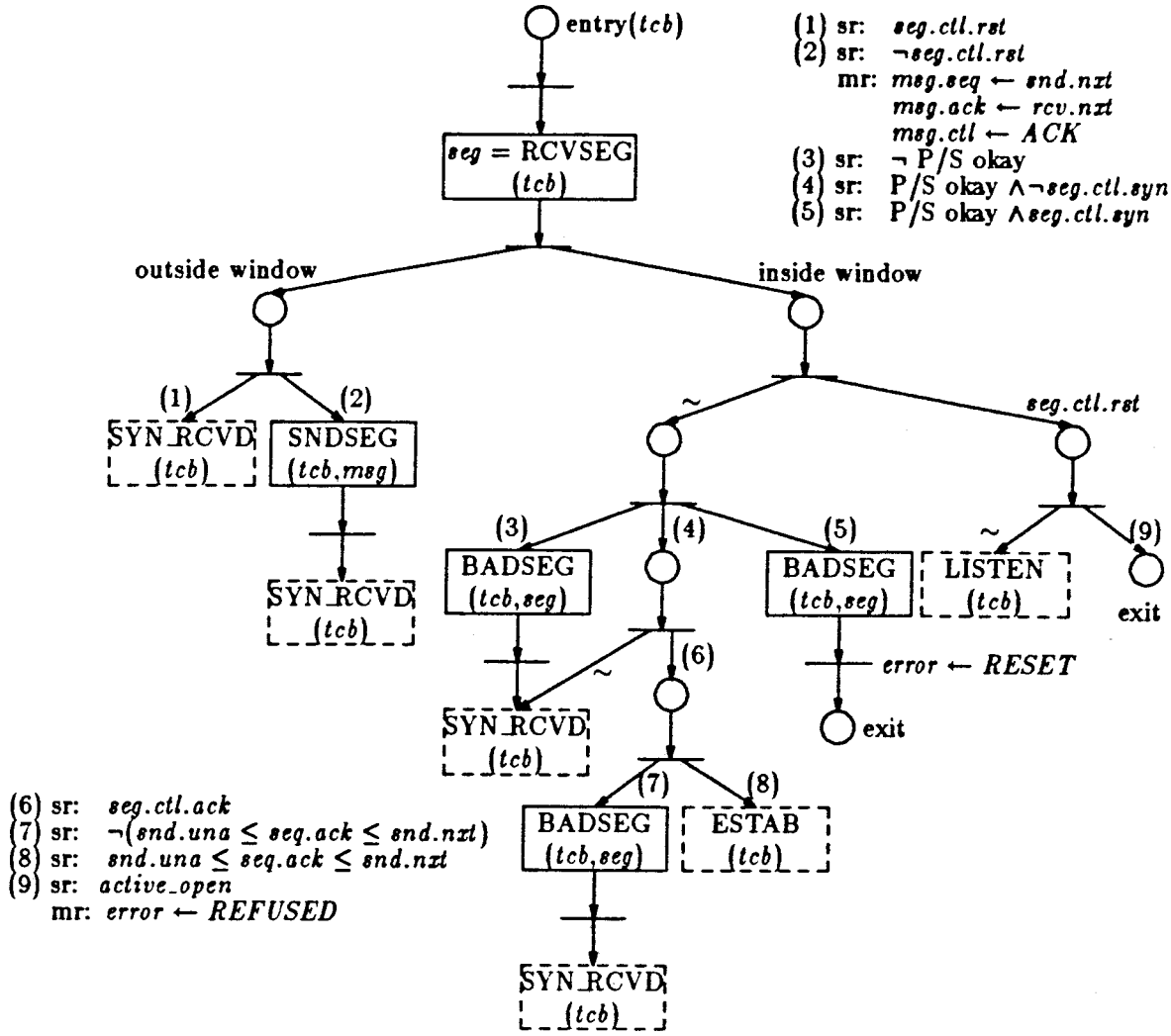



**Figure 25**

N-protocol: SYN_RCVD

Figure 24 is rather sequential in nature. This need not be the case. A single, large switch-decision could be used to remove the sequential nature of the decisions which lead to the net's actions. This was not done for reasons of clarity. Depending upon the designer's interpretation of the trade-offs, the contour/transition model could be used to remove nearly all of the sequential nature of this invocation.

The SYN_RCVD named net (Figure 25) is used to process a connection that is in the SYN_RCVD state. First, RCVSEG is instantiated to await and return an incoming segment for this connection. If the segment is outside the window, the presence of a RST is checked. If present, the segment is ignored; if not, a response is sent to force the foreign peer to re-transmit a valid segment. If the segment is inside

the window, the presence of a RST is checked. If present, and this connection was started with an active *open*(), the user process told that the connection has been refused, and control returns. Otherwise, the connection enters the LISTEN state. If a RST was not present, precedence and security/compartment considerations are checked. If they are not acceptable, the segment is rejected; otherwise the presence of a SYN is checked. If present, the segment is rejected, the connection is closed, and the user process is informed that the connection has been reset; if not, the presence of an ACK is checked. If not present, the segment is ignored. If an ACK is present, a check is made to make sure that the ACK is correctly acknowledging our SYN. If not, the segment is rejected. Otherwise the connection enters the ESTAB state. Again, if more of TCP were being considered, other parts of the segment might be processed prior to instantiating ESTAB.

The BADSEG net (Figure 26) is used to reject an incoming segment. First, BADSEG checks for a RST in the segment. If present, no response is sent. Otherwise, an ACK is checked for. Based upon the presence of an ACK, the appropriate response is set-up in *msg*, and SNDPKT is instantiated to send the response.

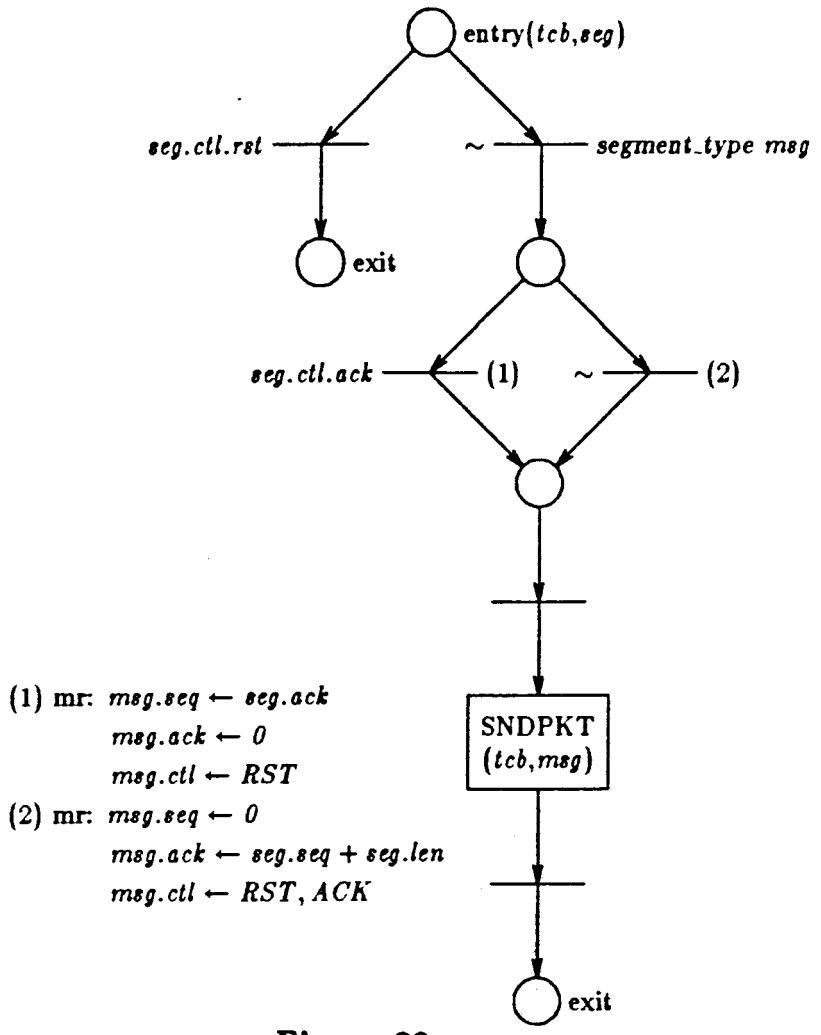



**Figure 26**

N-protocol: BADSEG

The SNDSEG net (Figure 27) is used to send a segment to the other N-peer. It instantiates SNDPKT to interact with the (N-1)-layer, and then forks control. One fork returns. The other instantiates TIMER at the ENQ entry place. TIMER.ENQ is viewed as a "single entry, zero exit" net.

The RCVSEG net (Figure 28) is used to get the next segment from the other N-peer. It instantiates MAIN.RCVPKT to get the next packet for this connection, and then forks control. One fork returns. The other instantiates TIMER at the DEQ entry place, which, like MAIN.TCP and TIMER.ENQ, does not return.

The TIMER net (Figure 29) is responsible for handling retransmission. When instantiated at the ENQ entry place, a transition fires which feeds one of

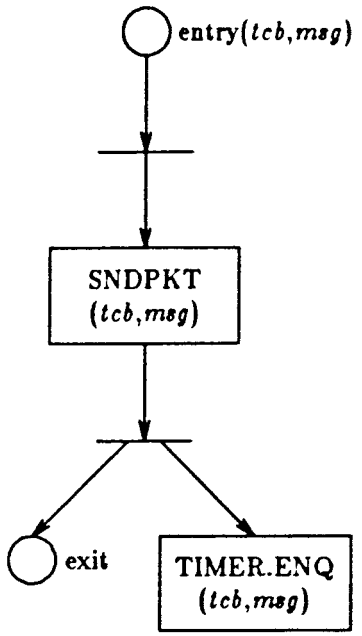



**Figure 27**

N-protocol: SNDSEG

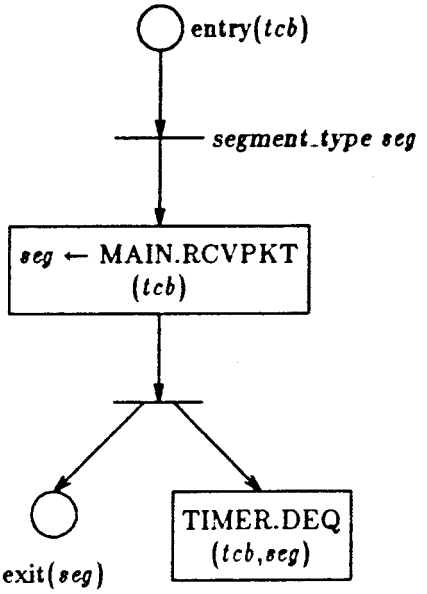
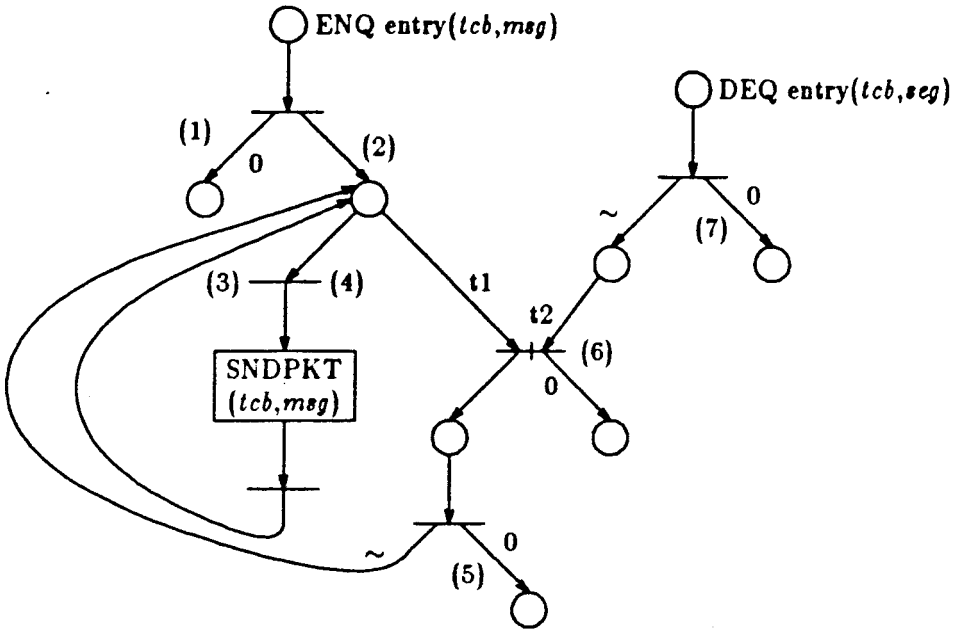



**Figure 28**

N-protocol: RCVSEG

two output arcs. If the retransmission queue is non-empty, the segment is added to the retransmission queue and control terminates. If the retransmission queue is empty, the segment is added to the retransmission queue and control proceeds to




(1) sr: retransmission queue non-empty
mr: add *msg* to the end of the retransmission queue
(2) sr: retransmission queue empty
mr: add *msg* to the end of the retransmission queue
(3) et: *TCP_TIME_OUT*
(4) mr: set *msg* to the first message in the retransmission queue
(5) sr: retransmission queue empty
(6) ep: t2's *seg* is for t1's connection
mr: in t1's context remove segments from rtq that are fully ack'd by t2's *seg*
(7) sr: retransmission queue empty $\vee \neg seg.ctl.ack$

**Figure 29**

N-protocol: TIMER

a place that can enable one of two transitions. One transition is used to model a time-out. It has its enabling time set to a non-zero time-out value. Although the transition is enabled as soon as a token appears at the place feeding its input arc, it can not begin executing its firing actions (and absorb the eligible token) until the enabling time has expired. If the time does expire, then SNDPKT is instantiated and told to send the first segment in the retransmission queue. Control then loops back. If prior to the expiration of the enabling time, the other transition, a split transition which has an enabling time of 0, can fire, it does so immediately and disables the former transition. When instantiated at the DEQ entry place, TIMER checks to see if the retransmission queue for this *tcb* is empty, or if the incoming segment does not have the ACK bit set. In either case, control terminates by firing

a null transition. If the queue is non-empty and the segment does acknowledge something, then control proceeds to the other half of the split transition mentioned previously. The split transition is enabled when a token is present in both of the places feeding its input arcs that are for the same *tcb*. When two such tokens can satisfy this predicate, the transition fires and removes any segments from the *tcb*'s retransmission queue that are fully acknowledged by the incoming segment. The half of the split transition associated with the incoming segment terminates control immediately. The other half proceeds to see if the *tcb*'s retransmission queue is empty. If so, control is terminated; otherwise control loops back to the place feeding the input arcs of the time-out transition and the split transition.

The reader should understand that TIMER is not adequate as a correct description of the actions that occur when a segment times-out. In particular, the structure of TIMER is such that control from the ENQ entry place loops forever if an acknowledgement is never received. A necessary addition should be to keep track of the number of retransmissions performed and, if the number exceeded a certain threshold, then to abort the connection and inform the user process that OPENed the connection that the connection has timed-out. Furthermore, a more powerful description would probably use a variable enabling time, based on segment-acknowledgement transit times, instead of the *TCP_TIME_OUT* constant used in TIMER.

*N-protocol primitives*

The definitions of the predicates and routines used in the enabling predicates and firing actions of some of the transitions in the net are presented here.

The routine *do_chksum*() computes the TCP checksum for a segment and associated packet, and stores the checksum in the segment's *cksum* field. Similarly, the predicate *ok_chksum*() computes the TCP checksum for a segment and associated packet, and compares it to the value of the segment's *cksum* field. If the two values do not match, the routine returns *FALSE* otherwise it returns *TRUE*.

The *new_iss*() routine calculates an initial sequence number for a *tcb*.

The predicate *P/S okay* refers to the precedence and security/compartment conditions being satisfied.

Finally, the predicates *inside window* and *outside window* check to see if a segment is inside the valid window for a connection, checking the segment's sequence number and the *s_wnd_type* structure of the *tcb*.

### Evaluation

This section makes a critical examination of the preceding description. Some general comments are made, and then the description is compared to two other formal specifications of connection establishment in TCP (other than the official DOD specification). Finally, the value of the contour/transition model as a specification tool is considered.

In comparison to the full TCP specification, several observations can be made which point to the advantages and weaknesses of the contour/transition model. In many ways, the contour/transition model is less ambiguous that its natural language counterpart. The flow of control for a particular state is more clearly defined. Despite the use of "structured" (rigorously indented) paragraphs in the original TCP specification[TCP], ambiguities do arise. These are in-escapable.[14] Although "structured", little hierarchy is present, which results in rather repetitive groups of statements throughout the specification. In contrast, the contour/transition-net description does not suffer from these problems, as transition-nets are used to convey the bulk of the meaning. Even so, the contour/transition-net description does make use of several predicates, functions, and procedures which are presented in natural language. It is emphasized that the contour/transition model does not seek to eliminate the use of natural language as a part of the specification, but rather to introduce a rigorous approach which is more capable of capturing the spirit of the protocol and is less ambiguous than less formal methods.

A difficult problem in presenting a specification is "knowing when to stop." That is, at what point has the specification given the full functional description and further discussion on the part of the specification is actually constraining possible implementations? Both descriptions do rather well in this regard, although the original specification tends to do somewhat better. In providing a less ambiguous description, the contour/transition model has taken the liberty to make several

[14] This section is not meant to be a critique of the TCP specification[TCP]. *Any* specification made using a natural language approach will suffer these problems. This was a primary motivation for the development of the MILSTD specification of TCP, *MIL-STD-1778*.

things more concrete, so as to avoid possible mis-interpretation. It is not clear if this has crossed the line from functional specification to implementational constraint.

*Comparison with Transmission Grammar*

Teng and Liu[TENG78], present a context-free grammar known as the *transmission grammar* as a method for protocol specification. The transmission grammar has been used to specify and verify initial connection handling by TCP[UMBA82B]. A key advantage to the transmission grammar approach over the use of finite state automata is the ability to compress transitions into a compact form. Most problems with state explosion can be solved by simply adding another production to the set of production rules for the grammar. In addition, completeness is handled more automatically: by continuously expanding the starting string, all possible states and outputs can be calculated. It is unfortunate that the compact nature of the transmission grammar makes conceptualization of the protocol more difficult to grasp: in the author's opinion, specifications using the transmission grammar tend to lack readability.

The specification of TCP's initial connection handling reported by Umbaugh, et. al.[UMBA82B] begins by presenting the valid messages that one TCP could send to another. From these and the standard, the production rules for the grammar were produced. Finally, a validation automaton was constructed to verify the action of the protocol.

The presentation made is specifically addressed towards verification. In terms of presenting a usable specification or a conceptual model of initial connection handling in TCP, little effort is expended. This division of attention can not be under-stated. It becomes clear that the transmission grammar, while potentially useful for verification activities, does not excel in either protocol explanation or as an aid for implementation. In contrast, the description using the contour/transition model goes to great length to present the functional basis of the protocol, and little or no attention is given to properties to be analyzed. This approach is discussed in somewhat greater detail further on.

### *Comparison with SPEX*

Schwabe[SCHWAB81A] introduced the SPEX specification language. SPEX is a transition-based system with some augmentations. A *SPEXification* (a specification made using SPEX) consists of the definitions of nodes, along with the definition of the topology and a set of properties defining liveness and safety (functional correctness). All of these definitions are given in an algorithmic form, with a few special operators that are "tailored" for protocol modeling. Node definitions consist of a description of state (local) and interface variables, the initial state, a set of events and the pre-conditions required to instantiate the events, and a procedural definition of the behavior of the node when an event occurs. There are two types of nodes: the *station* and the *medium*. Stations may be thought of as end-points in the connection, while the media may be considered the underlying communications subnet. The topology definition enumerates the nodes present in the specification and their connections.

The SPEXification of the initial connection protocol is presented by Schwabe[SCHWAB81B]. Four nodes are present: two stations, and two media. It is interesting to note that the definition of the media nodes includes the event of a message being lost. The actions of TCP are embedded in the definition of the behavior of the stations. After achieving a SPEXification, the semi-automated verification tool, AFFIRM[SUNS82A] is used. SPEXifications have a relatively straightforward transformation to an AFFIRM representation, so SPEX lends itself well to verification efforts.

A SPEXification is considerably more readable than a specification using the transmission grammar approach. With improved readability, a greater understanding of the spirit of the specification can be achieved. Unfortunately, the algorithmic nature of SPEXifications do not present the reader with a great insight into the nature of the protocol. It is true that the fundamental properties are asserted and plainly described, but this does not demonstrate the true spirit of the protocol's actions.

### *The Contour/Transition Model as a Specification Tool*

A great strength of a hybrid approach, such as the contour/transition model, is the ability to use the natural concurrency structures found in Petri nets with the abstraction facilities found in algorithmic approaches. From this vantage,

contour/transition-nets could be useful as a specification method. The standard Petri net model suffers from the inability to easily perform data manipulations, along with some problems with state explosion. Numerical Petri nets, while making data manipulation easier, suffer from a lack of discipline with regard to data access.[15] Contour/transition-nets solve this problem by introducing tokens which represent various contexts of a given peer's state. Furthermore, the hierarchy introduced by using named nets might reduce state explosion problems somewhat.

It is important to realize however that protocol specification techniques can serve three purposes:

- explanation of the protocol
- the basis for the verification of the protocol
- the basis for the implementation of the protocol

As shown with the description of connection establishment in TCP, the contour/transition model does well to demonstrate the spirit of the protocol. Given the powerful concurrency representation techniques that Petri nets provide, along with the concurrent and recurrent invocation control semantics that contour/transition-nets achieve, the contour/transition model can be seen favorably as an explanation tool.

In terms of implementation, the contour/transition model is sufficiently different from the algorithmic approach as to avoid placing many constraints on the implementation process. For example, consider the operation of Figure 20. No requirement is made as to the order that connections or incoming packets are services. Rather, the notion that incoming packets pair with known connections is put forth, with no other restraints. Some seemingly unnecessary constraints do remain however. These can be observed from the description's data definitions in particular. Fortunately, an implementation would not have to adhere to the letter of the data description providing that it did adhere to the intent of the state variables. Regardless of these concerns for the data definitions, the nature of Petri nets is sufficiently different from most traditional implementation languages so as

[15] Recall that when using the numerical Petri net approach, all variables are global to all peers. This introduces a tremendous problem for those who strive for conceptual clarity in specification techniques.

to avoid the trap of "specification by example." Finally, to enjoy much success, automatic, or semi-automatic methods for transforming contour/transition-nets into tasking programs must be present.

In terms of verification, the contour/transition-net approach is notably, and painfully weaker. Although the preceeding chapter presented sufficient results to provide a foundation for analyzing the properties of system described with contour/transition-nets, applying these results to the description presented in this chapter would be quite expensive.

A criticism made of many specification and verification efforts is the lack of ability to properly describe the behavior of the protocol when more than one connection is active. For the description presented in our discussion, the use of colors to represent connections, and the clean semantics of colors in contour/transition-nets, permits a natural explanation of the protocol's activities. For analysis purposes, if one could demonstrate that initial connection handling for a single connection was handled correctly, and one could demonstrate that the interaction between connections (colors) did not disturb this property, then one can prove that multiple connections are handled properly by the description. The interaction between colors occurs when split transitions are used, while the interaction between peers occurs when boundary transitions are used. In the description, this is limited to the SNDPKT net, the MAIN net (including MAIN.RCVPKT) and the TIMER net. It would not be difficult to show that the instantiation of this net does not cause an incorrect series of interactions between colors.

Much work remains in the area of analysis of systems represented with the contour/transition model. In terms of the other two aspects however, the contour/transition model seem to be well-suited as a specification tool.

# CHAPTER 6
# Conclusions

The results presented by this dissertation are now summarized, and directions for future research are suggested.

## Contributions

This concluding chapter begins by reviewing the results shown by this dissertation.

### *Modeling*

This dissertation has presented a graphical model for concurrent computation that combines the control aspects of Petri net theory with the abstraction facilities found in programming languages. The contour/transition model introduces abstraction facilities to both the control and data aspects in order to make its underlying transition-based skeleton more capable of modeling concurrent systems. Through the use of colorful tokens, which represent independent contexts of execution, a contour/transition-net is able to be active both concurrently and recurrently.

This dissertation has also demonstrated the modeling facilities of contour/transition-nets by modeling initial connection handling in the DOD Transmission Control Protocol. In particular, note that the contour/transition model is able to present the protocol in a concise fashion with generally unambiguous semantics. Furthermore, note that owing to the mechanism by which different execution contexts interact in the model, more than one connection establishment can be modeled by the nets presented with no additional modeling complexity.

### *Analysis*

This dissertation has demonstrated that results previous applied to predicate/action-nets can be extended to prove properties about contour/transition-nets. These extensions do not change the well-founded analysis properties

of the invariant-method and capitalize on the hierarchical nature of the contour/transition model to permit proofs of smaller constructs. As a result, the term *invocation-safe/live* is formalized to refer to the notion of proper termination for a contour/transition-net, and the term *contextual interference* to refer to aspects of a contour/transition-net that allow different contexts of execution to interact.

In addition to developing an extension to the invariant-method, this dissertation discussed when simpler, cheaper methods of analysis, such as incidence-matrix analysis can be used instead. Although not applicable in most interesting situations, some of the time required to perform the analysis could be reduced if care is taken to use the method easiest to apply to a given contour/transition-net.

## Future Research

This dissertation has left many problems unanswered or has introduced new research problems which, if contour/transition-nets are found to be useful for modeling, should prove interesting. These areas are briefly surveyed here.

### *Communication Between Colors*

The contour/transition model, as defined in this dissertation, provides two means for different contexts of execution (i.e., colors) to synchronize and communicate. The boundary transition corresponds to the facilities found in message passing systems, while the split transition corresponds to a rendezvous facility. Both of these methods have various merits, depending on the degree of connectivity found in the system being modeled.

Although the behavior of split transitions is graphically intuitive, the same is not necessarily true of boundary transitions. In particular, if a boundary transition is really used to model a message passing facility, then the issues of loss and corruption of messages and channel capacity might be addressed. Currently, our interpretation of boundary transitions avoids these characteristics. One line of research might be to be more considerate in this area and adjust the analysis method accordingly.

In addition, it might be interesting to explore other types of synchronization and communication primitives found in other languages and to see if they can be framed in terms of the two types of transitions previously described. If

not, then perhaps additional transition disciplines should be introduced into the contour/transition model.

### *Analysis*

One open area of research is the integration of timing information into the analysis method. At present, the enabling time and firing time associated with each transition is ignored in a contour/transition-net. This simple-minded view, while adequate for systems which are not time-critical, is not suitable when the goal is to model real-time systems in which both the ordering and duration of events must be analyzed in order to determine if the system behaves correctly.

In addition, it might be useful to permit transitions other than entry and exit transitions to modify the context of execution (i.e., introduce new scoping contexts). In order to adapt the analysis method presented, the hierarchy vector would need to take on a more dynamic nature in order to reflect the notion that various "begin-end" blocks have changed the context of the system. This leads to the consideration that the notion of the hierarchy vector could be extended in many ways. For example, in addition to permitting arbitrary nesting, it might also be useful to allow the analysis technique a greater level of granularity when indicating the relation between a particular execution context in the system and its corresponding component in the hierarchy vector.

Finally, in order to to be truly useful, the analysis process must be automated to free the designer from the difficult tasks of analyzing the system by hand. In part, this motivates a discussion of how contour/transition-nets might be simulated and implemented.

### *Simulation*

As a part of the design process, it is interesting to be able to *animate* a system that is modeled with contour/transition-nets. With the use of sophisticated bit-mapped, color graphics terminals, which are becoming widely available, the author can envision a scenario where a contour/transition-net is displayed on the designer's screen and colorful tokens move about on the net. Animation of this sort could be presented in a very compact representation as only one picture of the graph need be displayed, while tokens of different colors, each representing a different execution context, can occupy the same graph. The author believes that

this approach could be particularly appealing when a system has several contexts executing concurrently (as in the case of multiple connections being established, which was briefly touched upon in the previous chapter). Furthermore, this type of animation might be useful in examining how colors interact in a system.

It should be noted that the contour/transition model adds nothing to animation theory. Rather, it is felt that the animation of systems represented with the contour/transition model might compose a significant portion of a powerful design method for concurrent systems.

*Implementation*

The suitability of contour/transition-nets for semi-automatic implementation should prove to be an interesting research topic. It is desirable to have language processors take concurrent systems described using structured Petri nets and produce ADA[16] tasking programs. This appears to be a natural approach since a designer can use the simple (yet powerful) concurrency mechanisms of the contour/transition model and have this translated to the corresponding ADA rendezvous constructs. Although the ADA constructs are, in theory, as powerful as the original structured Petri net mechanisms, they are likely to be much more complicated.

In related work, Nelson, et. al.[NELS82, NELS83] have shown one method for translating annotated Petri nets into a PL/I-like language. This work suffers from several drawbacks however (e.g., because of PL/I's lack of good concurrency primitives, some Petri net (sub-)topologies are not representable). Hence, one research direction would be to explore the relation between concurrency constructs in contour/transition-nets and concurrency constructs in ADA. The author hypothesizes that, owing to the strict nature of interaction between colors (execution contexts) in the contour/transition model, an implementation of a system modeled as contour/transition-nets should be able to exploit a high degree of parallelism.

[16] ADA is a trademark of the Department of Defense (ADA Joint Program Office).

## Concluding Remarks

The author notes that previous extensions to the place/transition-net model tend to follow an "all or nothing" philosophy: either some mechanism for abbreviations are added to permit more concise representations but which do not enhance modeling power; or computational power is added to make the model Turing equivalent, with little or no consideration for data-handling facilities and useful abstraction facilities. The contour/transition model is suggested as a balanced hybrid approach that attempts to retain the control aspects of Petri nets while introducing the abstraction facilities and data manipulation capabilities found in programming languages. It is hoped that these extensions will make transition-based models a more viable method for modeling concurrent systems.

# Appendix A
# Transformation of Contour/Transition-Nets for Analysis

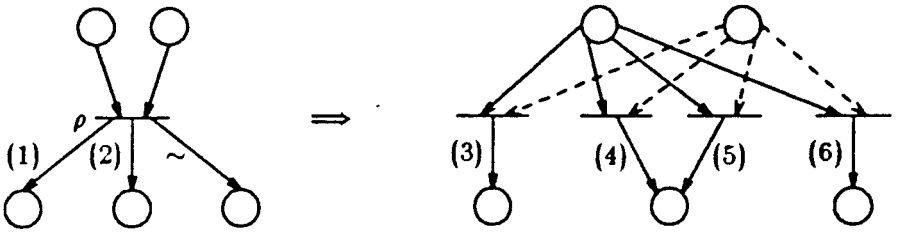
The analysis method presented in Chapter 4 placed some certain restrictions on the structure of those contour/transition-nets which could be analyzed. This appendix describes how contour/transitions-nets may be transformed in order to meet those structural requirements.

## Elimination of Selection Predicates

Removing selection predicates from a contour/transition-net is complicated since the atomicity of the transition must be preserved at all costs. The prose description is[17]:

1. Add a new transition to correspond to each output arc of the original transition. If different numbers of tokens can be introduced onto an output arc, then add a new transition for each possible distribution. This is consistent with the requirement that selection rules for transitions are composed of a predicate (evaluated in the context of the eligible tokens), and a constant.
2. For each new transition, connect a copy of each input arc for the original transition as an input arc to the new transition, and also connect an output arc corresponding to the appropriate output arc for the original transition.
3. For each new transition, copy the enabling time, construction rule, manipulation rule, and firing time from the original transition. For each new transition give it the appropriate selection rule based on its corresponding output arc on the old transition, but without any selection predicate. This means each selection rule is a simple constant (usually just 1).
4. For each new transition, declare its enabling predicate to be the logical AND of the enabling predicate of the original transition and the predicate

[17] The author sincerely apologizes for the incomprehensibility of this descriptive passage. He experimented with several wordings before choosing the the current one and placing it an appendix. Clever readers will skip the passage and study Figure 30 instead.




(1) sr: $x = \alpha \rightarrow 1$
(2) sr: $x = \beta \rightarrow 1$
$x = \gamma \rightarrow 2$

(3) ep: $\rho \wedge (x = \alpha)$
(4) ep: $\rho \wedge (x = \beta)$
(5) ep: $\rho \wedge (x = \gamma)$
sr: 2
(6) ep: $\rho \wedge \neg(x = \alpha \vee x = \beta \vee x = \gamma)$

**Figure 30**

Elimination of Selection Predicates

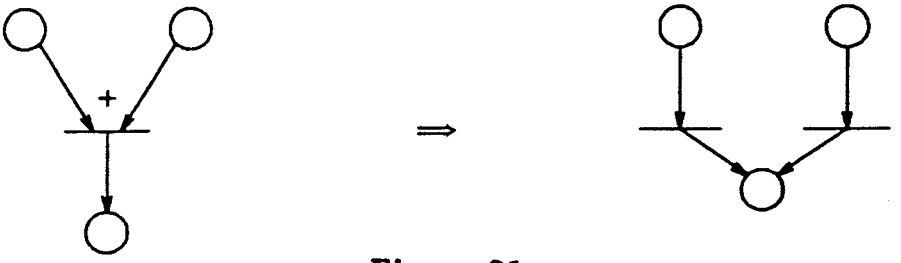



**Figure 31**

Elimination of OR Input Logic

associated with the selection rule of the corresponding output arc of the original transition. Note that in the case of the default predicate('~'), the actual selection predicate is the negation of the logical OR of all other selection predicates for the original transition.

5. Finally, remove the original transition and all of its input and output arcs.

Figure 30 demonstrates this process.

## Elimination of OR Input Logic

Although not as complicated as the previous transformations used to eliminate selection predicates, again the atomicity of the original transition must be preserved. The prose description is:

1. Add a new transition to correspond to each input arc of the original transition.

2. For each new transition, connect an input arc corresponding to the appropriate input arc for the original transition, and also connect a copy of each output arc for the original transition as an output arc to the new transition.
3. For each new transition, copy the enabling predicate, enabling time, construction rule, selection rule, manipulation rule, and firing time from the original transition.
4. Finally, remove the original transition and all of its input and output arcs.

Figure 31 demonstrates this process.

# Appendix B
# Boundary Transitions

This appendix presents an analysis of a small system which uses boundary transitions. The system presented is a very simple one in which a process communicates with another via messages passed across a boundary transition. It must be emphasized that this system is trivial in extent, as a perfect underlying medium is assumed. As a result, the system does not make use of time-outs, retransmissions, sequence numbers, windows, checksums, and so on. The example has been included to demonstrate how the analysis techniques described in Chapter 4 can be used on systems with boundary transitions.

Figure 32 represents the message writer. When invoked at its entry place, a new variable, *msg*, is declared. Next, a function is called to get the contents of the message to be sent. Control then reaches a combined transition[18]: one fork (the right arc) is a sending boundary transition; the other loops back to get the next message to be sent.

Figure 33 represents the message reader. When invoked at its entry place, a new variable, *msg*, is declared. Control now proceeds to a place which waits for a split transition to fire. The other input arc for the split transition is fed by a place connected to the output arc of a receiving boundary transition. When the boundary transition fires, a message crosses, which is placed in the blank token which arrives at place R. When the split transition fires, the message from the blank token is copied into the context of token at place CL. The blank token is discarded, the token with the message then calls a function to dispose of the contents of the message received. Control then loops back to await the next message.

[18] It should be noted that a combined transition such as the one in Figure 32 is perfectly acceptable in a contour/transition-net. The transition could be expanded into two transitions (one "normal" and the other a sending boundary transition), and an additional place, but in the interests of brevity, this particular design was chosen

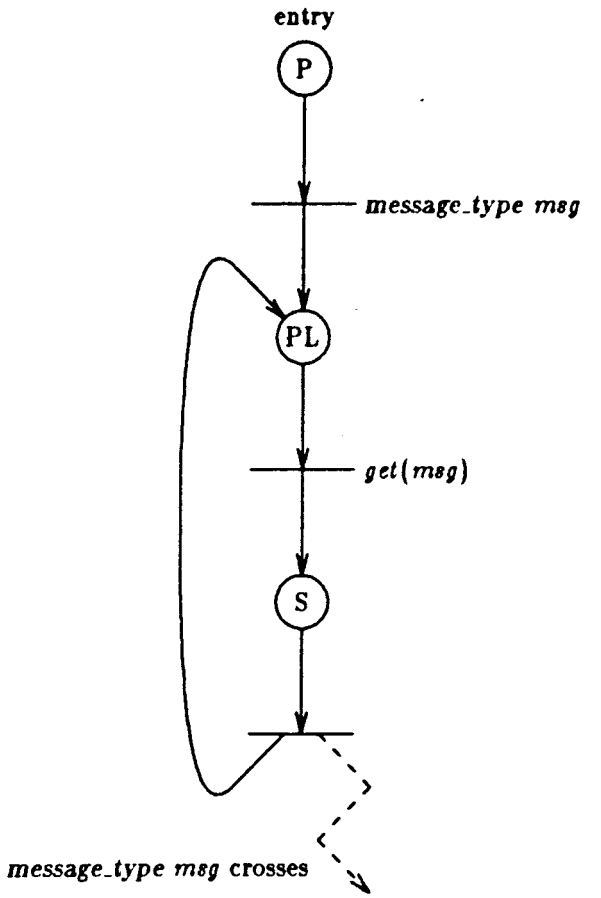



**Figure 32**
Message Writer

As discussed in Chapter 4, the sending and receiving boundary transitions in the two nets will be coalesced in order to make analysis of the system easier. This results in the single contour/transition-net shown in Figure 34. The boundary transitions are now a part of transition $t_3$, which is a combined transition (the left output arc is "normal", the right output arc corresponds to the juxtaposition of the sending and receiving boundary transitions). At this point, it is clear that the boundary transitions could be removed altogether in favor of a similar topology using a split transition. The merits of this approach are left as questions of open research: as suggested in Chapter 6, the different methods of communication between colors in the contour/transition model requires further investigation.

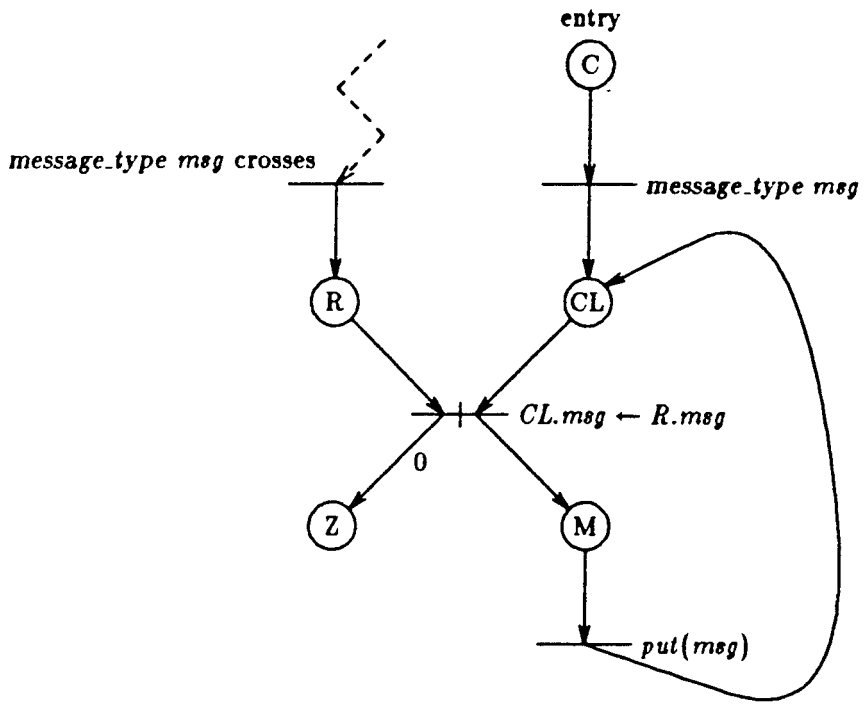



**Figure 33**
Message Reader

Table 7 shows the incidence matrix for the contour/transition-net in Figure 34. Two invariants are present, one for the P entry place,

$$m(P) + m(PL) + m(S) = 1, \qquad (q1)$$

and one for the C entry place,

$$m(C) + m(CL) + m(M) = 1. \qquad (q2)$$

As expected, these invariants were derived in a slightly different way that the standard incidence matrix method: invariant ($q1$) has calculated by considering the incidence matrix as being composed only of rows for places P, PL, and S (the top part of Table 7). Similarly, invariant ($q2$) has calculated by considering the incidence matrix as being composed only of rows for places C, CL, and M (the bottom part of Table 7). No statement can be made as to the invariance of the

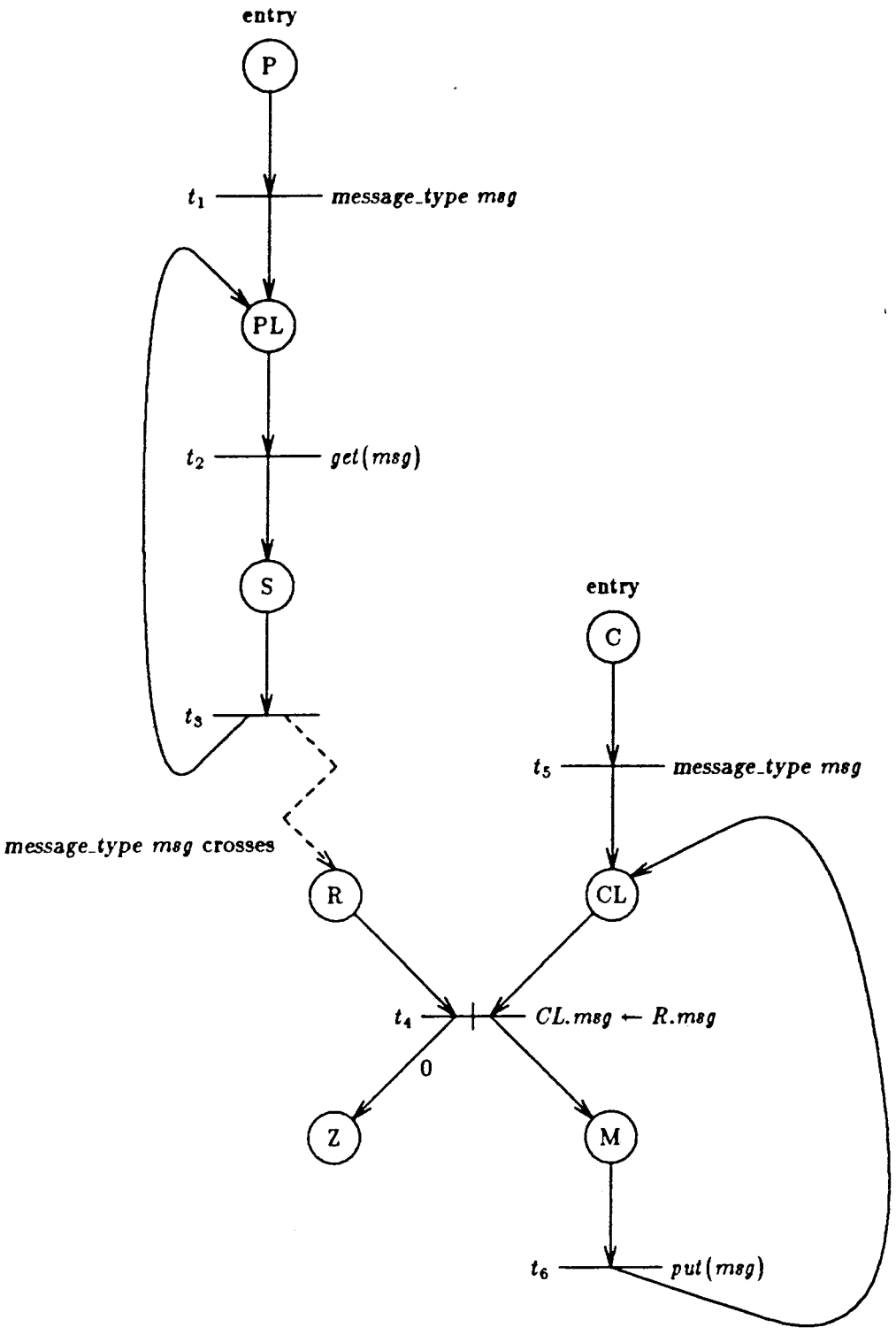



**Figure 34**
Combined System

| | Incidence Matrix | | | | | | Markings | | Invariants | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $M_P$ | $M_C$ | $r_1$ | $r_2$ |
| P | -1 | | | | | | 1 | | 1 | |
| PL | 1 | -1 | 1 | | | | | | 1 | |
| S | | 1 | -1 | | | | | | 1 | |
| R | | | 1 | -1 | | | | | | |
| Z | | | | 0 | | | | | | |
| C | | | | | -1 | | | 1 | | 1 |
| CL | | | | -1 | 1 | 1 | | | | 1 |
| M | | | | 1 | | -1 | | | | 1 |

**Table 7**

Incidence Matrix and Invariants for Figure 34

marking of place R, as the former is fed by a boundary transition. As far as the invariance of the marking of place Z, a third invariant can be derived from Table 7:

$$m(Z) = 0. \qquad (q3)$$

Since there are no exit places associated with the control path for either entry place, neither entry place is invocation/safe. However, invariants $(q1)$ and $(q2)$ do show that the control paths for both entry places are bounded.

To prove freedom from deadlock, an argument is made directly from the invariants (since each context is instantiated only once, the incidence-invariants need not be translated into equations involving place-variables). Invariant $(q1)$ tells us that transitions $t_1$, $t_2$, and $t_3$ are firable. Consider the sum

$$m(C) + m(M).$$

There are two cases. For the first case,

case 1: $$m(C) + m(M) = 1,$$

either transition $t_5$ or $t_6$ is firable. For the second case,

case 2: $$m(C) + m(M) = 0,$$

from $(q2)$ it is clear that $m(CL) = 1$. Furthermore, since $t_3$ has been shown to be firable, eventually $m(R) = 1$. Hence, $t_4$ is firable as well. Since at least

one transition can fire, regardless of the state of the net, the system is free from deadlock.

## REFERENCES


[AGER79] T. AGERWALA. Putting Petri Nets to Work. *IEEE Computer 12*, 12 (December, 1979), 325–342.

[AZEM84] P. AZEMA, G. JUANOLE, E. SANCHIS, M. MONTBERNARD. Specification and Verification of Distributed Systems Using Prolog interpreted Petri nets. Appearing in *Proceedings, Seventh International Conference on Software Engineering*, IEEE Computer Society Press, 1984, pp. 510–518.

[BERTHE82] G. BERTHELOT, R. TERRAT. Petri Net Theory for the Correctness of protocols. *IEEE Transactions on Communications COM-30*, 12 (December, 1982), 2476–2505.

[BERTHO83] B. BERTHOMIEU, M. MENASCHE. An Enumerative Approach for Analyzing Time Petri Nets. Appearing in *Proceedings, IFIP Congress*, Paris, France, 1983.

[BILL82] J. BILLINGTON. Specification of the Transport Service using Numerical Petri Nets. Appearing in *Proceedings, Second International Workshop on Protocol Specification, Testing, and Verification*, C.A. Sunshine (ed.), North-Holland Publishing Company, 1982, pp. 77–100.

[BOCH77] G.V. BOCHMANN, J. GECSEI. A Unified Method for the Specification and Verification of Protocols. Appearing in *Proceedings, IFIP Congress*, Toronto, Canada, 1977, pp. 229–234.

[BOCH80A] G.V. BOCHMANN, C.A. SUNSHINE. Formal Methods in Communication Protocol Design. *IEEE Transactions on Communications COM-28*, 4 (April, 1980), 624–631.

[BOCH80B] G.V. BOCHMANN. A General Transition Model for Protocols and Communication Services. *IEEE Transactions on Communications COM-28*, 4 (April, 1980), 643–650.

[BOCH82B] G.V. BOCHMANN, C.A. SUNSHINE. A Survey of Formal Methods. Appearing in *Computer Network Architectures and Protocols*, P.E. Green (ed.), Plenum Press, 1982, pp. 561–578.

[BRAN82] D. BRAND, W.H. JOYNER, JR. Verification of HDLC. *IEEE Transactions on Communications COM-30*, 5 (May, 1982), 1136–1142.

[COHE83] D. COHEN, J. POSTEL. The ISO Reference Model and Other Protocol Architectures. Appearing in *Proceedings, IFIP Congress*, Paris, France, 1983.

[DANT78] A.S. DANTHINE, J. BREMER. Modeling and Verification of End-to-End Transport Protocols. Appearing in *Communication Protocol Modeling*, C.A. Sunshine (ed.), Artech House, 1981.

[DANT80] A.A.S. DANTHINE. Protocol Representation with Finite State Machines. *IEEE Transactions on Communications COM-28*, 4 (April, 1980), 632–643.

[DIAZ82] M. DIAZ. Modeling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models. *Computer Networks 6*, 6 (December, 1982), 419–441.

[ECMA] *European Computer Manufacturer Association Transport Protocol*, ECMA TC24/16 document number 80, 1980.

[GENR78] H.J. GENRICH, K. LAUTENBACH. Facts in Place/Transition-Nets. Appearing in *Mathematical Foundations of Computer Science*, J. Winkowski (ed.), Springer, Berlin, Vol. *64*, 1978, pp. 213–231.

[GENR81] _____. System Modelling with High-Level Petri Nets. *Theoretical Computer Science 13* (1981), 109–136, North-Holland Publishing Company.

[GOOD78] D. GOOD, ET. AL. Report on the Language Gypsy. Technical Report ISCMA-CMP-10. University of Texas, Austin (September, 1978).

[HAIL81] B.T. HAILPERN. Specifying and Verifying Protocols Represented as Abstract Programs. Appearing in *Computer Network Architectures and Protocols*, P.E. Green (ed.), Plenum Press, 1982, pp. 607–623.

[IP] Internet Protocol. Request for Comments 791. Appearing in *Internet Protocol Transition Workbook*, Network Information Center, SRI International, 1981.

[JENS81] K. JENSEN. Coloured Petri Nets and the Invariant-Method. *Theoretical Computer Science 14* (1981), 317–336, North-Holland Publishing Company.

[JOHN71] J.B. JOHNSTON. The Contour Model of Block Structured Processes. *SIGPLAN Notices*, Proceedings of a Symposium on Data Structures in Programming Languages *6*, 2 (February, 1971), 56–82.

[KELL76] R.M. KELLER. Formal Verfication of Parallel Programs. *Communications of the ACM 19*, 7 (July, 1976), 371–384.

[KURO82] J. KUROSE. The Verification of a Connection Establishment Protocol using Temporal Logic. Appearing in *Proceedings, Second International Workshop on Protocol Specification, Testing, and Verification*, C.A. Sunshine (ed.), North-Holland Publishing Company, 1982, pp. 43–62.

[MEKL80] J.L. MEKLY, S.S. YAU. Software Design Representation Using Abstract Process Networks. *IEEE Transactions on Software Engineering SE–6*, 5 (September, 1980), 426–435.

[MERL74] P.M. MERLIN. *A Study of the Recoverability of Computing Systems*, PhD dissertation (technical report number 58), Department of Information and Computer Science, University of California, Irvine, 1974.

[MERL76B] P.M. MERLIN, D.J. FARBER. Recoverability of Communication Protocols — Implications of a Theoretical Study. Appearing in *Communication Protocol Modeling*, C.A. Sunshine (ed.), Artech House, 1981.

[MISU73] D. MISUNAS. Petri Nets and Speed Independent Design. *Communications of the ACM 16*, 8 (August, 1973), 474–481.

[NELS82] R.A. NELSON, L.M. HAIBT, P.B. SHERIDAN. Specification, Design, and Implementation via Annotated Petri Nets. Computer Science Report RC9317 (#41041). IBM Thomas J. Watson Research Center (March, 1982).

[NELS83] _____. Casting Petri Nets into Programs. *IEEE Transactions on Software Engineering SE–9*, 5 (September, 1983), 590–602.

[OSI] *Reference Model of Open Systems Interconnection*, ISO TC97/16 document number 227, 1979.

[PADL82] M.A. PADLIPSKY. A Perspective on the ARPAnet Reference Model. Request for Comments 871. Network Information Center, SRI International (September, 1982).

[PENE81] M.H. PENEDO. *The Use of a Module Interface Description in the Synthesis of Reliable Software Systems*, PhD dissertation (report CSD-810115), Computer Science Department, University of California, Los Angeles, 1981.

[PETE77] J.L. PETERSON. Petri Nets. *Computing Surveys 9*, 3 (September, 1977), 224-252.

[PETE81] _____. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[PETR62] C.A. PETRI. Communication with Automata. Report RADC-TR-65-337. Rome Air Development Center (1965). Translation: C.F. Greene.

[PNUE77] A. PNUELLI. The Temporal Logic of Programs. Appearing in *18th Annual Symposium on the Foundations of Computer Science*, 1977.

[PNUE81] _____. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science 13* (1981), 45-60, North-Holland Publishing Company.

[POST74] J. POSTEL. *A Graph Model Analysis of Computer Communication Protocols*, PhD dissertation (report ENG-7410), Computer Science Department, University of California, Los Angeles, 1974.

[RAMAMO80] C. RAMAMOORTHY, G. HO. Evaluation of Asynchronous Concurrency Systems using Petri Nets. *IEEE Transactions on Software Engineering SE-6*, 5 (September, 1980), 440-449.

[RAMC74] C. RAMCHANDANI. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*, PhD dissertation, Computer Science Department, Massachusetts Institute of Technology, 1974.

[RAZ080A] R.R. RAZOUK, G. ESTRIN. Validation of the X.21 Interface Specification Using SARA. Appearing in *Proceedings, 1980 Trends and Applications: Computer Network Protocols*, 1980, pp. 155–163.

[RAZ081] R.R. RAZOUK. *Computer-Aided Design and Evaluation of Digital Computer Systems*, PhD dissertation (report CSD–810205), Computer Science Department, University of California, Los Angeles, 1981.

[RAZ083A] R.R. RAZOUK, C.V. PHELPS. Performance Analysis Using Timed Petri Nets. Technical Report Number 206. Department of Information and Computer Science, University of California, Irvine (August, 1983).

[SABN82] K. SABNANI, M. SCHWARTZ. Verification of a Multidestination Protocol Using Temporal Logic. Appearing in *Proceedings, Second International Workshop on Protocol Specification, Testing, and Verification*, C.A. Sunshine (ed.), North-Holland Publishing Company, 1982, pp. 21–42.

[SCHI78] M. SCHIFFERS, H. WEDDE. Analyzing Program Solutions of Coordination Problems By CP-Nets. Appearing in *Mathematical Foundations of Computer Science*, J. Winkowski (ed.), Springer, Berlin, Vol. *64*, 1978, pp. 462–473.

[SCHU80] G.D. SCHULTZ, D.B. ROSE, C.H. WEST, J.P. GRAY. Executable Description and Validation of SNA. *IEEE Transactions on Communications COM-28*, 4 (April, 1980), 661–677.

[SCHWAB81A] D. SCHWABE. *Formal Techniques for the Specification and Verification of Protocols*, PhD dissertation (report CSD–810401), Computer Science Department, University of California, Los Angeles, 1981.

[SCHWAB81B] _____. Formal Specification and Verification of a Connection Establishment Protocol. Appearing in *Proceedings, Seventh Data Communications Symposium*, 1981, pp. 11–26.

[SCHWAR81] R.L. SCHWARTZ, P.M. MELLIAR-SMITH. Temporal Logic Specification of Distributed Systems. Appearing in *Proceedings, Second International Conference on Distributed Computing Systems*, IEEE Computer Society Press, 1981, pp. 446–454.

[SCHWAR82] _____. From State Machines to Temporal Logic: Specification Methods for Protocol Standards. *IEEE Transactions on Communications COM-30*, 12 (December, 1982), 2486-2496.

[SHAP83] V. SHAPIRO. *On Formal Verification of Systems Described By A Graph Model of Behavior*, MS thesis, Computer Science Department, University of California, Los Angeles, 1983.

[SIFA77] J. SIFAKIS. Petri Nets for Performance Evaluation. Appearing in *Proceedings, 3rd International Symposium on Measuring, Modelling and Evaluating Computer Systems*, H. Beilner, E. Gelenbe (ed.), North-Holland Publishing Company, 1977, pp. 5-93.

[SIMO82] G. SIMON, D. KAUFMAN. An Extended Finite State Machine Approach to Protocol Specification. Appearing in *Proceedings, Second International Workshop on Protocol Specification, Testing, and Verification*, C.A. Sunshine (ed.), North-Holland Publishing Company, 1982, pp. 113-134.

[SUNS78B] C.A. SUNSHINE, Y.K. DALAL. Connection Management in Transport Protocols. *Computer Networks 2*, 6 (November, 1978), 454-473.

[SUNS79] C.A. SUNSHINE. Formal Techniques for Protocol Specification and Verification. *IEEE Computer 12*, 9 (September, 1979), 20-27.

[SUNS82A] C.A. SUNSHINE, D.H. THOMPSON, R.W. ERICKSON, S.L. GERHARD, D. SCHWABE. Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models. *IEEE Transactions on Software Engineering SE-8*, 5 (September, 1982), 460-489.

[SYMO80A] F.J.W. SYMONS. Introduction to Numerical Petri Nets, a General Graphical Model of Concurrent Processing Systems. Appearing in *Communication Protocol Modeling*, C.A. Sunshine (ed.), Artech House, 1981.

[SYMO80B] _____. The Verification of Communication Protocols Using Numerical Petri Nets. Appearing in *Communication Protocol Modeling*, C.A. Sunshine (ed.), Artech House, 1981.

[TCP] Transmission Control Protocol. Request for Comments 793. Appearing in *Internet Protocol Transition Workbook*, Network Information Center, SRI International, 1981.

[TENG78] A.Y. TENG, M.T. LIU. A Formal Approach to the Design and Implementation of Network Communication Protocol. Appearing in *Proceedings, COMPSAC*, 1980, pp. 722–727.

[UMBA82A] L.D. UMBAUGH, M.T. LIU. A Comparison of Communication Protocol Validation Techniques. Appearing in *Proceedings, ICC 1982*, 1982, pp. 4c.4.1–4c.4.7.

[UMBA82B] L.D. UMBAUGH, M.T. LIU, C.J. GRAFF. Specification and Validation of the Transmission Control Protocol using Transmission Grammar. Appearing in *Proceedings, COMPSAC 1982*, 1982.

[VERN82] M.K. VERNON. *Performance-Oriented Design of Distributed Systems*, PhD dissertation, Computer Science Department, University of California, Los Angeles, 1982.

[VOSS80] K. VOSS. Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems. *IEEE Transactions on Software Engineering SE-6*, 6 (November, 1980), 539–544.

[WEST78] C.H. WEST, P. ZAFIROPULO. Automated Validation of a Communications Protocol: The X.21 Recommendation. *IBM Journal of Research and Development 22*, 1 (January, 1978), 60–71.

[X.21] *Recommendation X.21 (revised)*, AP VI-Number 55-E, International Telegraph and Telephone Consultative Committee, 1976.

[YAU83] S.S. YAU, M.U. CAGLAYAN. Distributed Software System Design Representation Using Modified Petri Nets. *IEEE Transactions on Software Engineering SE-9*, 6 (November, 1983), 733–745.

[ZAFI80] P. ZAFIROPULO, C.H. WEST, H. RUDIN, D.D. COWAN, D. BRAND. Towards Analyzing and Synthesizing Protocols. *IEEE Transactions on Communications COM-28*, 4 (April, 1980), 651–660.

[ZUBE80] W. ZUBEREK. Timed Petri Nets and Preliminary Performance Evaluation. Appearing in *Proceedings, 7th Annual Symposium on Computer Architecture*, 1980, pp. 8–96.