Modelling the logical structure of flexible manufacturing systems with Petri-nets

P David Stotts*, Robert W Newcomb† and Z Ning Cai† review the use of Petri-nets in FMSs and discuss the robotic lattice structure

Flexible Manufacturing Systems are computer-integrated systems which have many concurrent components, very complicated logical relations, and a distributed computer system structure. They have been increasingly and rapidly adopted for use in industry. Basic Petri-net definitions, both classical and extended for timing analysis, are reviewed in the paper. In addition, recent research using Petri-net theory as applied to the design and analysis of FMSs is reviewed. In particular, one of the most flexible of manufacturing line structures is discussed, the Robot Lattice Structure, *which is analysed using a new form of timed Petri-nets, termed Binary Timed Petri Nets. A graphical modelling language for BTPNs is also briefly discussed.*

Keywords: Flexible Manufacturing Systems, Petri-nets, modelling, Binary Timed Petri Nets

Petri-net theory was originated by Petri in 1962 with his doctoral dissertation *Communication with Automata*[1]. In the 1970s, Petri-net theory was developed much further by two projects, the 'Information System Theory Project' and the 'Project MAC' at the US Massachusetts Institute of Technology[2]. In the 1980s, Petri-net theory has been used for advanced computer integrated system design, such as for Artificial Intelligence (AI) in network systems[3-5], and for Flexible Manufacturing Systems (FMS)[6,7]. With technology developing so fast, most high technology systems must use computers as elements integrated into the system. Such systems are usually very complex and very large, with many interactive components, concurrency requirements, and a hierarchical controlling structure. The circuitry of a computer itself is vastly complex in terms of possible and desirable analyses. How can Petri-nets be used to model such systems precisely? How can a model be analysed to efficiently obtain a satisfying design? Considerable research has been undertaken in recent years to answer these questions. As mentioned above, Petri-nets offer one approach for solving these problems.

Research in Petri-net theory is very active, and is growing in visibility. Petri and his group in West Germany concentrate on basic theory, which they call General Net Theory (GNT). Petri himself has given the following synopsis of GNT[8]:

> For the treatment of very large, very small, very fast or highly complex structures under the aspect of information processing, general net theory offers a set of new conceptual and mathematical tools. GNT is a theory of structured signal flow in spatially extended systems; it is accordingly founded on axioms which express or respect the combinatorial properties of real-world signals as viewed by physics (upper bound for velocity, for distinctness, for packing density, for precision of timing).

*Institute for Advanced Computer Studies, Computer Science Department, †Microsystems Laboratory, Electrical Engineering Department, University of Maryland, College Park, MD 20742, USA

0140-3664/89/040193-11 $03.00

Classical Petri-net theory has been widely applied throughout computer systems research, including areas in computer software modelling[9], industrial flexible manufacturing[7, 10], parallel computation modelling[11], and Very Large Scale Integration (VLSI) design[12]. Recent papers show that the research interests in other subjects have increased, among these being: applying Petri-net theory to modelling and analysing software systems with timing constraints[13]; modelling techniques for production scheduling and control based on extensions to Petri-nets[5, 14]; software techniques for generating actual programs[15]; simulations and animations from Petri-nets[16]; and logical languages and intelligent decision theory for Petri-nets[3, 17, 18].

A number of projects are underway in Europe specifically applying various forms of Petri-nets to automated manufacturing. For example, Petri-nets with coloured tokens have been used by French and Spanish researchers to create a FMS model[19]: they present techniques for verifying logical properties of such systems with their model. (A recent survey by Valette[20] presents a good overview of other current uses for Petri-nets in industrial applications.) As pointed out in this survey, the logical structure of FMSs is very complex. Generally, a system can be decomposed into several levels:

- local machine control level: lowest control level for specific automachine or devices;
- workstation level: control and coordinate a group of automachines and robots;
- coordination of flexible cells: a set of workstations in the FMS;
- and the global control level: coordinate the whole system's workings, making production planning and intelligent decisions.

The activities at each level can be represented by various forms of nets, perhaps augmented with data structure. In particular, Valette mentions that the French automobile manufacturers Renault and Peugeot have applied Petri-net concepts to Programmable Logic Controllers (PLC) for manufacturing. PSI and Siemens of Germany are also using Petri-nets for the design of real-time process controllers.

BASIC FMS MODELLING WITH PETRI-NETS

Petri-net theory is a tool for modelling and designing a system in which there are interacting concurrent components and complex logical relationships. Petri-nets provide a mathematically precise representation to directly describe the control flow and control structure of a system, and to indirectly describe the data flow and data structure derived from a real system. The evaluation of these mathematical net models is only practical with computer software. Analysis results can tell designers important information about a system's structure, and can suggest improvements to the design.

Informally, a Petri-net is an automaton composed of an interpreted directed graph structure and an execution rule. For example, Figure 1 shows a simple Petri-net in which the circular nodes are termed *places* and the bar nodes are termed *transitions*. The dot in place *s1* is called a *token*; its residence in place *s1* represents the existence of some condition modelled by that place. The vector

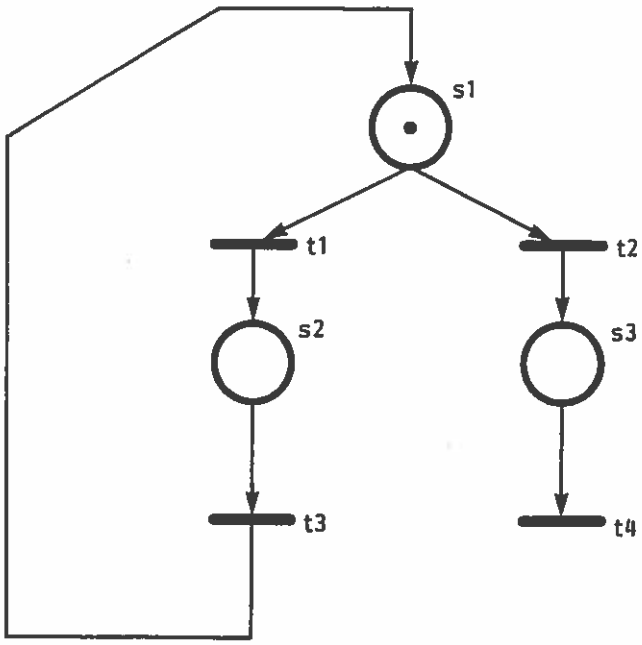


Figure 1. Marked Petri-net example

showing the number of tokens in each place in the net is termed the *state*. A state change occurs when a transition *fires*, causing a token to be removed from each of the transition's input places, and causing a token to be deposited in each of its output places. Execution continues until no transition can be fired.

Because FMSs work in real industrial environments, their models have to be based on realistic requirements, such as the system should have: the capability to deal with both linear and non-linear systems; continuous processes and discrete time processes; numerical approaches and logical approaches; and automatic machine action control and system supervisor control. All of these represent challenges to traditional modelling concepts and techniques. It is very important to find the appropriate techniques for modelling, design and analysis of such complex systems. So far, research has shown that Petri-net theory is one of the most promising techniques with which to approach these FMS modelling problems. However, many factors such as parallelism and flexibility are used in these complex systems, leading to very complicated Petri-net models. To overcome this difficulty, Petri-net models have been developed rapidly in recent years, and separated widely among scientific and engineering fields. Research usually extends the classical Petri-nets to general conditions (i.e. the extensions embed the model into a timed environment) by associating a time either to transitions or to places of the Petri-nets. The most general extension allows the use of stochastic times and probabilities, which leads to a classification called General Stochastic Petri Nets (GSPN).

Petri-net theory comprises two related areas: pure Petri-net theory, and applied Petri-net theory. Pure Petri-net theory focuses on theoretical principles and basic concepts, whereas applied Petri-net theory is the application of Petri-nets to the modelling and design of computer systems and computer integrated systems. Actually, applied Petri-net theory yields construction techniques for significant Petri-net models of a physical system, gives a means to analyse such models, and converts these

models back to real systems after modifying the original designs. Use of applied Petri-net theory is currently widespread in computer operating systems, computer concurrency, distributed operated systems, asynchronous systems, hierarchical controller design, and AI. In recent years, many research projects have focused on the modelling of FMSs, among many other new application fields.

For the application of Petri-nets to FMS modelling and design, the following procedures are customarily used:

1 Specify a physical system's design or performance requirements. One may need to understand the system on different levels. At a low level, very detailed information is required: at a high level, computer scientists and manufacturing engineers can give the requirements of the model controllability, reliability, and complexity.
2 Construct Petri-net models to accurately reflect the interesting behavior of the specified system. The inherently hierarchical structure of Petri-nets can be effectively applied to limit the complexity of such models. Different levels of Petri-net design involve different details of the real system. Hierarchical modelling allows Petri-nets to represent a complex system, and enables the use of computer software for analysis.
3 Apply Petri-net theory to analyse the model, and then to modify the model to more accurately realize the system requirements.
4 Convert the Petri-net to a real system design so that the production tasks can be accomplished.

The above procedures can be repeated until the design satisfies all requirements. By these procedures, a system's dynamic behavior and structure can be fully analysed (such as concurrent characteristics and system control structures). Steps two and three involve analysis techniques from several areas of mathematics, such as formal language theory, automata theory, graph theory, abstract algebra, and linear algebra.

APPLICATIONS TO FMSs

A FMS may consist of many workstations. An individual workstation may be composed of several robots for which there is a local controller. Each local controller is associated with the FMS central controller, which is global in scope. A local controller mainly performs functions for managing and controlling the work of the machines in each workstation. The global controller performs functions to control the entire FMS's information flow and decision making.

Petri-net theory is a powerful tool for FMS modelling and analysis. First, Petri-nets can express precisely in a mathematical form the logical relations among interacting system components. Because a FMS has not only many concurrently executing components but a distributed computer structure as well, these logical relations quickly become very complicated. The Petri-net model can be used for the analysis of system-wide concurrent computation, dynamic task scheduling, and internal communication protocols.

Second, a Petri-net model can be constructed with a natural hierarchical organization. For manufacturing system flexibility there must be numerous different functions in the robots or automatic tools of each workstation. Because the global controller and the local controllers have different emphases, and because the system has too many parameters to be computed at one time, it is necessary to use a hierarchical structure in system modelling. Without hierarchical decomposition, a system model cannot usefully represent most industrial FMSs in practice. Therefore, this is one of the most important issues in applying Petri-nets to FMS modelling.

Third, Petri-nets can deal with the system data structure and data flow indirectly. Because Petri-net models can be constructed to an almost arbitrary degree of detail, computer programs can be automatically generated from them. This feature provides a way in which a computer can automatically produce source code for the simulation or implementation of a complex software system. The ability to construct extended Petri-net models that represent both control flow and data flow in a system allows their use for programming controller actions.

Control structure design

Petri-net models can accurately represent an FMS control structure. As mentioned above, a Petri-net structure includes places, transitions, directed arcs showing inputs and outputs, and (as discussed below) it may include related timings. One can use places to represent the system processing conditions, and transitions to represent the system actions or the events. The tokens in the model are ideal for representing available resources, control signals, or other predefined system parameters. For example, in Japan, Hitachi Inc. has been doing research on FMS controller design with Petri-nets[7, 21]. The local controller is part of Hitachi's factory manufacturing system. The controller's task is to coordinate and direct the 68 000-based sequencing operations of manufacturing workstations. In their work, Petri-nets model the interaction of the local controller with the sensors and actuators of the physical system under control. Tokens represent a system control signal. A token arriving at a place means that a control signal is triggering an action. While a token resides at a place it means that the workstation is in a busy condition. When a corresponding input signal is received that indicates the current work has been completed, the token moves out of the place, representing the end of the action. The input signal can be one of several to indicate conditional completion for enabling and firing transitions. In a physical system, this signal acts as a gate signal to be implemented in the logical system. A transition cannot be fired until the gate signals are activated. A Petri-net can also be used in a programmable logic controller to activate and respond to the discrete Input/Output (I/O) of a manufacturing system[6, 7].

On-line visible control

In real-time industrial processes many production procedures should be tracked and monitored by the system control operator. Visual graphs and real-time execution tracing techniques can greatly aid the operators in keeping appraised of the current process states, and in adjusting control variables for optimal system functioning.

Being directed graphs, Petri-net models are well suited

to visual presentation and manipulation. Crockett and Desrochers[6] show a Petri-net control description entered into a FMS using an interactive graphics terminal. The visual form is used to generate a tabular representation of the actions and conditions associated with places and transitions. By using a tabular representation, places are contents in the table. Structure pointers in the table are used to specify I/O events, which means that the place table contains the transition list. Similarly, each transition table has the related place list and corresponding data structure pointer. Therefore, the FMS control flow can be clearly illustrated on the graphical terminals for operators. Because coloured tokens can recognize similar but different actions, some FMS controllers adopt coloured tokens at a particular place by using a coloured Petri-net model. The colour of a token might indicate which portion of a part is proceeding through the controlled manufacturing system.

Automatic controller design

An FMS controller may be a very complicated software system. For increasing the reliability and optimum ability of system performance, techniques for the automated generation of controller software have been developed. These methods are based on well-known engineering techniques in Computer-Aided Design (CAD), and Petri-net theory again plays an important role.

The University of Montreal has one research group working in this area of FMS modelling with Petri-nets (see Chocron and Cerny[22] for some of their research results). This sequencer directly implements an interpreted safe Petri-net by a high level language, and it generates prototype code and tables for Intel 8085-based controllers. The interpretation of a net is that output signals are associated with places, and input signal functions are associated with transitions. As stated by Crockett and Desrochers[6]:

> An interesting aspect of Chocron's paper is the description of the development flow during the design of applications. The design stages are as follows: (1) a written description of the control function; (2) interactive graphics input of the system and controlled system models; (3) conversion from graphics description to Petri-net language description; (4) compilation of the language description to produce code and tables; (5) the use of compiler output by the on-line controller or by simulation package.

This topic has also been studied by the research group at the Robotics Institute of Carnegie-Mellon University: Ekberg and Krogh[15] have applied Petri-net modelling techniques to the automated generation of prototype software for manufacturing control systems. For a complex concurrent system, their techniques can save significant software development time, as well as make the resulting software more reliable than that of traditionally programmed systems.

HIERARCHICAL GRAPHS FOR HIERARCHICAL MODELLING

The Hierarchical Graph (HG) model of concurrent systems[13] uses a combination of timed Petri-nets for control flow description, and HGs[23] for data modelling. This model has been successfully applied to the analysis of combined concurrent software/hardware system architectures[24], and is also serving as the formal basis for a concurrent software development and analysis environment[11]. An HG procedure model has three components: the data model; the static program model; and the control flow model. The data model describes the composition and organization of the data to be manipulated; the static program model describes the form and relationships among the instructions that directly manipulate the data; and the control flow model describes the possible execution sequences of groups of these instructions. A mathematical basis in set theory and formal grammars provides the framework in which programs are analysed.

Systems modelled in HG are composed hierarchically. One three-part HG model represents a callable procedure. In the static program portion of an HG procedure model, other procedures are called, and the HG model of an entire system is then composed of the collection of individual procedure models. To analyse one HG procedure, the results of analysis from lower-level procedures are used. Timing results are developed from the bottom-up, based on the assumptions that timing durations are available for low-level hardware operations, and that recursion is absent.

In the next subsection, the formal definitions of the timed Petri-nets employed in HG semantics are presented, along with a short discussion of the execution rules associated with these nets. A graphical notation called PFG (Parallel Flow Graphs), used to construct HG models, is then described — in essence, a modelling language.

Formal definitions

The basic formal definitions of Petri-nets that deal with the control aspects of a computation are introduced below, i.e. describing FMS organization, controller functions, and software system implementation, showing the various possible sequences of parallel code block executions that an algorithm will allow. An extended form of a Petri-net is used to do this. Parts of the following definitions and notation are adapted from classical Petri-net theory (a good synopsis appears in Reference 25). Though the standard net structure is employed, modifications have been made to the surrounding net theory to allow time bounds to be placed on the components of a modelled software system. The definition of a marking has been extended to facilitate the timing. In the following definitions, $\mathbf{N}$ denotes the set $\{0,1,2,\ldots\}$ of natural numbers, and $\mathbf{Z}$ denotes the set $\{\cdots,-2,-1,0,1,\cdots\}$ of integers.

Definition 1: Net.

A general *net structure* is a triple $N = <S,T,F>$ in which

- $S = \{s_1,\ldots,s_n\}$ is a finite nonempty set of *places*;
- $T = \{t_1,\ldots,t_m\}$ is a finite nonempty set of *transitions*, with $S \cap T = O$;
- $F \subseteq (S \times T) \cup (T \times S)$ is the binary *flow relation* for N.

The net structure can be thought of as a bipartite graph in which $S \cup T$ is the set of nodes. The flow relation F defines the directed arcs connecting places to transitions and transitions to places.

Definition 2: Marking.
A *marking* M for a net N is a function
$M : S \rightarrow \mathbf{N} \times \mathbf{Z}$
mapping each place in the net into an ordered pair of numbers, the first a non-negative integer and the second an integer.

A marking can also be described as an ordered n-tuple of pairs, $M = <M_1, \ldots, M_n>$ in which each $M_i \in \mathbf{N} \times \mathbf{Z}$ and $n = |S|$. The two representations are related by $M(s_i) = M_i$, where $s_i \in S$.

Each ordered pair in a marking provides information about a particular place in the net. For a particular ordered pair M_i, the notation $<M_i^Y, M_i^F>$ is used to distinguish the individual components. The first component M_i^Y is the number of tokens residing in place s_i: the second component M_i^F represents the age of the 'active' token at place s_i, i.e. the token of the current resident group that was first to arrive. A positive value is placed in M_i^F when a token first arrives, and it is then decremented by one with each state change. A value of 0 indicates a fully aged token. A negative value in M_i^F indicates that the active token is fully aged, but is waiting, i.e. has not been consumed by a transition firing, due to some synchronization constraint.

Modelling real-time software requires analysis techniques for determining the time bounds on execution of portions of the system. Towards that end, timing figures are included in the Petri-nets used for programs. As other researchers have done[26], we associate a number of time units with each place in the net; transitions are still considered to fire instantaneously, as in classical Petri-net theory.

Definition 3: Timing.
A *timing* τ for a net N is a function
$\tau: S \rightarrow \mathbf{N} \setminus \{0\}$
mapping each place in the net into one of the positive natural numbers.

The timing can also be described as an ordered n-vector, $\tau = [\tau_1, \ldots, \tau_n]$ in which $n = |S|$ and each $\tau_i \in \{1,2,\cdots\}$. The two representations are related by $\tau(s_i) = \tau_i$, where $s_i \in S$.

Definition 4: Marked timed Petri-net.
A *marked timed Petri-net* is a triple $TP = <N, M_0, \tau>$, in which:
N is a general net structure;
M_0 is a marking for N, termed the *initial marking*;
and τ is a timing for N.

In a marked timed Petri-net, execution begins in the state defined by the initial marking M_0. When a token arrives at a place s_i, it must reside there for τ_i units of time before it can enable, or participate in enabling, the transitions that follow s_i, defined as the set $\{t_j \mid (s_i, t_j) \in F\}$.

Thus, a token residing in s_i could represent the condition that a corresponding instruction or group of instructions in the modelled program is executing, and the timing figure would then represent the relative amount of time required for it to complete. On completion (i.e. after τ_i time units), the transitions following s_i are partially enabled by the resident token, and may fire when fully enabled. Firing a transition τ_i causes a token to be placed in each of its output places, defined as the set $\{s_j \mid (t_i, s_j) \subset F\}$. For example, consider again the Petri-net in Figure 1, and let its timing vector be $\tau = [2,4,3]$. With initial marking $M_0 = <1{:}2, 0{:}0, 0{:}0>$, the state of the net after three state changes could be either $<0{:}0, 1{:}3, 0{:}0>$ or $<0{:}0, 0{:}0, 1{:}2>$, depending on whether t_1 or t_2 fires during the second state change.

For modelling concurrent software and its logical relations on data, several aspects of a marked timed Petri-net require special interpretation. If a token arrives at a place s_i in which a token is already resident, the new token does not begin to age until the active token has resided for the full duration τ_i, and has been consumed by a subsequent transition firing. Thus, when interpreting a Petri-net in terms of a concurrent program and its execution, a place represents not only a block of instructions, but also a single processor resource to execute those instructions. This processor is assumed to satisfy only one request at a time.

A concurrent transition firing rule is also used, in which the classical Petri-net execution paradigm of firing one transition per state change is relaxed. The get-the-next-state-from-the-current-one, a maximal subset of the enabled transitions, is fired, and their token movements collectively determine the new marking. The reachability set of such a concurrently fired Petri-net is then a subset of the state space under classical execution; a state obtained by collectively firing a set of transitions can also be obtained by a sequence of single transition firings. For timing purposes, though, the length of the sequence is important. A concurrently fired transition set creates a single state change, thereby using a single time unit. This reflects concurrent activity more appropriately than a sequence of single firings.

PFG language

The PFG[27] graphical notation is currently being developed, which has a direct interpretation in the HG formalism. PFG is a language for the expression of concurrent, time-dependent computations. Its syntax is graphical and hierarchical to allow the construction and viewing of realistically sized computations. Figure 2 shows an example of a parallel flow graph in PFG. In this figure, a rectangle icon (bearing the notation *bb1, bb2, . . .*) represents a basic block, containing a sequence of procedure calls. A base-down triangular icon (bearing the notation *sel1, sel2, . . .*) represents a concurrent branch point. The selector in the icon designates a location in the data state to be examined when a thread of control reaches the branch. All arcs bearing the value of that location as a label are followed in parallel. In Figure 2, when control arrives at the topmost parallel branch, the selector *sel1* is evaluated, and if the value is α then two threads of control go on to *bb2* and *bb3* in parallel. If no out-arc bears the value of *sel1*, then that thread expires. A base-up triangular icon (bearing notation Υ) is a synchronization node. Execution waits at the icon until an active thread has arrived on each in-arc, and then a *single* thread goes on from there. The semicircular icon is similar in execution to the parallel branch, except that it does not create parallel threads. After the selector is evaluated, a non-deterministic choice is made among the arcs bearing the selector's value as a label, and a single thread of control continues down the chosen arc. With these few

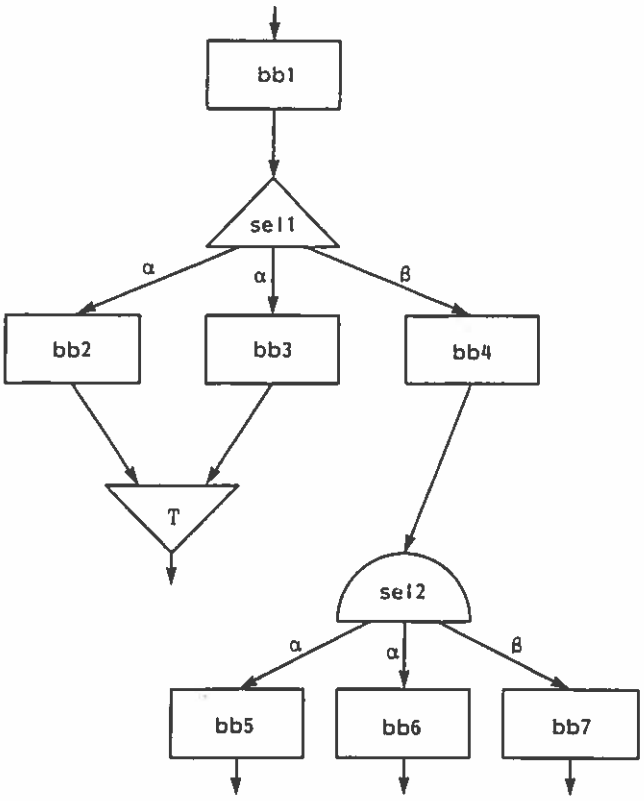


Figure 2. Parallel flow graph

structures, PFG is rich enough to express many of the common concurrent control structures found in parallel languages, as well as some less common ones.

The execution semantics of PFG programs are defined by the HG concurrent system model. Each syntactic structure in PFG has a direct translation into a portion of a place-timed Petri-net model. Figure 3 illustrates this translation process for the PFG shown in Figure 2. The net created by legally combining PFG structures is guaranteed to be well-formed, in the sense that each Petri-net is in a restricted subclass of general nets, and each has a clear interpretation in terms of the hardware and software in a robot lattice structure (considered below). It is important to note that PFG techniques can analyse the concurrency properties of a robot lattice such as deadlock freedom and proper mutual exclusion on shared data.

BINARY TIMED PETRI-NETS FOR ROBOT LATTICE MODELLING

This example discusses the most flexible of manufacturing line structures, the *Robot Lattice Structure*. The lattice configuration for a production line of robots is one of the most flexible for industrial manufacturing[10]. As can be seen in Figure 4, a high degree of flexibility can be obtained with a lattice configuration, because alternate paths exist between workstations, even in the presence of a single down station[28]. In fact, numerous faults can be accommodated as long as both workstations in a vertical pair are not simultaneously out. Among the available tools for analysis of such structures, Petri-net theory appears to be the most promising. The robot lattice structure is highly reliable and highly concurrent, with multi-channel communication protocols used for communication among

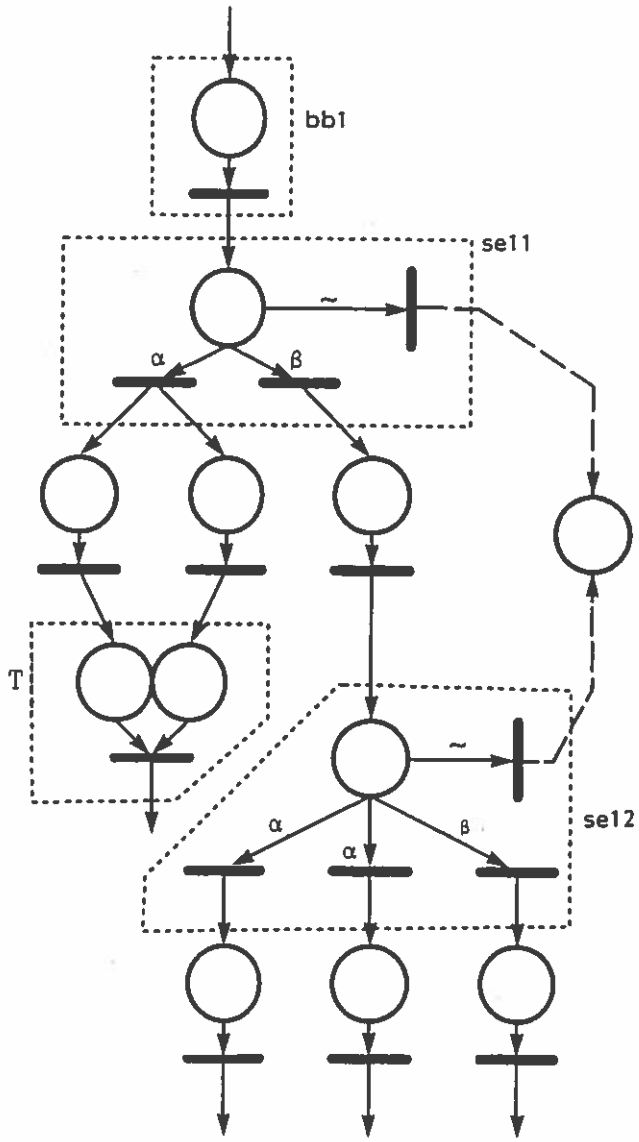


Figure 3. Petri-net components for PFG shown in Figure 2

multi-robot workstations, represented by nodes of the lattice.

To analyse the behavior of a robot lattice a form of Petri-nets called *Binary Timed Petri Nets* (BTPN) are used. These are an extension to an earlier model called Binary Petri Nets by Alayan and Newcomb[29], which have been shown to be useful for describing and analysing complex FMSs[3]. BTPNs are formed by combining the binary Petri-

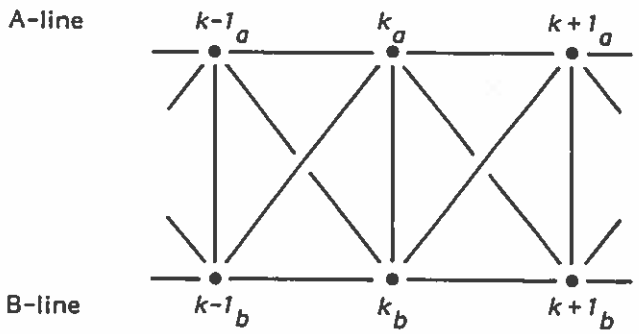


Figure 4. Graph nodes representation of on-line workstations

net model with the determinate timing aspects from previous Petri-net applications. For BTPNs, an integer time is associated with the place. Analysis of the reachability graph of a BTPN provides deadlock detection and calculation of minimum/maximum execution times for each station, using techniques employed for earlier timed Petri-net models[24].

In summary, the advantages of these techniques are listed here. BTPNs provide a means of analysing both the temporal and concurrency behavior of robot lattice structures for automated manufacturing. This flexible structure has several characteristics that are accurately reflected in BTPN models:

- there is consistency throughout the entire lattice;
- no workstation or robot in a workstation repeats the task already completed by another workstation. This is accomplished by allowing only one Central Processing Unit (CPU) to direct all firing functions;
- the implementation consistency is maintained by setting the communication protocols in the central CPU and each of the related local controllers;
- and these are simple logical relations and simple mathematical representations.

With the BTPN models, the behavior of robot lattices can be analysed in terms of both concurrency aspects and execution time aspects. In particular, we can determine:

- minimum and maximum execution times for robot workstations performing particular tasks;
- improper simultaneous access by two or more concurrently operating robot controllers to shared data structures;
- and the possibility o. deadlock of the robot workstation network during manufacturing.

General behaviour equations for a BTPN

The formal timed Petri-net definitions given earlier remain in effect for BTPNs, but a different firing rule is employed due to the logical differences between software behavior and physical communication line behavior. For BTPNs, this execution rule prohibits having more than one token in a place, and this is enforced by normalization after every state transition.

In FMSs, in the processors, when an output signal is present, the signal will get transmitted to all parts connected to it, and enable it to receive signals. In terms of Petri-nets, these properties mean that at most one token will appear on all places fed by the transition. Thus, the transition state equations for general timed Petri-nets are developed here with the restriction that the number of tokens in a given place is a binary number, and with the execution extension that a token in a place can cause all transitions connected to that place to fire. The development is similar to the earlier work of Alayan and Newcomb[29]; a mechanism for aging tokens has been worked into the behavior equations.

To designate a positive, integral firing time λ is used, and tokens are moved through fired transitions to create a state change from time λ to $\lambda + 1$. For binary nets, a place may contain either one token, or none. If it contains one token, then that token is propagated by firing *all* the transitions connected to the place containing it. Tokens can enter the net from external inputs X and leave via external outputs Y by the firing of output transitions. As in classical Petri-net theory, a transition is only enabled to fire if all the places incident upon it are marked. In BTPNs, however, a place is marked only if it contains a *fully aged* token.

Given a BTPN of n places and m transitions, we let $\Omega(\lambda)$ be the n-vector containing the number of tokens in each place at time λ; it is composed of M_i^Y for $i = 1,2,\cdots,n$ from our earlier definition of marking. In addition, the age vector $A(\lambda)$ indicates the age of each token in the net; it is composed of the M_i^τ components of a net marking. $\Phi(\lambda)$ is the m-vector of transitions to fire at time λ. The external inputs are denoted by the n-vector $X(\lambda)$, and the external outputs are denoted by the n-vector $Y(\lambda)$; all of these vectors contain integers, and all but $A(\lambda)$ are restricted to being 1 or 0 in a proper BTPN marking. The flow relation F of the BTPN is represented by a $m \times n$ connection matrix $\mathbf{F}$, where:

$$\mathbf{F} = \mathbf{F}^+ - \mathbf{F}^- \tag{1}$$

in which, for a transition t_i and a place s_j, the element f_{ij}^+ is 1 if $(t_i, s_j) \in F$, and the element f_{ij}^- is 1 if $(s_j, t_i) \in F$; all other entries are 0.

As Alayan and Newcomb[29], we work with both binary and normal integers in the same equations. Consequently, several vector operators are required to normalize results. The *dot equality* $\doteq$ transforms its right side by converting each positive value into a 1, and every other value into a zero; it is used to normalize the marking at each place s_i after concurrent transition firing. Conversely, the *double dot equality* $\ddot{=}$ transforms its right side by converting any value not 1 into a 0; it is used to normalize the token aging vector. The unary operator **dec** takes an age vector and leaves all 0 and 1 entries unchanged, but subtracts 1 from all others. The $\square$ operation is defined for a row n-vector a, having 0 and 1 entries, and a column n-vector b, having integer entries, to yield a scalar value given by:

$$a \square b = 1 \tag{2}$$

if at least one of the pairs a_i and b_i has $a_i = 1$ and > 0, and no pair has $a_i = 1$ and $b_i < 0$, for $i = 1,2,3,\cdots,n$. Otherwise:

$$a \square b = 0 \tag{3}$$

defines the result. The $\square$ operation is used to normalize the firing vector at time λ.

Using these operations, the following behaviour equations for BTPNs are obtained:

$$\Omega_1(\lambda) \doteq \Omega(\lambda) + X(\lambda) \tag{4}$$

$$A_1(\lambda) = \mathbf{pmax}\{A(\lambda), [X(\lambda)^*\tau]\} \tag{5}$$

in which τ is the n-vector interpretation of the net timing function, **pmax** is a pairwise comparison of vector elements, producing a new vector, and * is a pairwise multiplication of vector elements, producing a new vector:

$$A_2(\lambda) \ddot{=} \mathbf{dec}[A_1(\lambda)] \tag{6}$$

$$\Omega_2(\lambda) = \Omega_1(\lambda) * A_2(\lambda) \tag{7}$$

$$\Phi(\lambda + 1) = \mathbf{F}^- \square \Omega_2(\lambda) \tag{8}$$

$$\Theta^{-}(\lambda+1) \doteq \mathbf{F}^{-T}\,\Phi(\lambda+1) \tag{9a}$$

$$\Theta^{+}(\lambda+1) \doteq \mathbf{F}^{-T}\,\Phi(\lambda+1) \tag{9b}$$

$$\Omega(\lambda+1) = \Omega_2(\lambda) + \Theta^{+}(\lambda+1) - \Theta^{-}(\lambda+1) + \Omega_1(\lambda) * \overline{A_2(\lambda)} \tag{9c}$$

in which the bar indicates Boolean complementation of matrix elements:

$$A(\lambda+1) = \mathbf{pmax}\{[\mathbf{dec}[A_1(\lambda)] - \Theta^{-}(\lambda+1)],\ \Theta^{+}(\lambda+1) * \tau\} \tag{10}$$

$$Y(\lambda) = D\,\Phi(\lambda) \tag{11}$$

in which D is a matrix describing transition connectivity to external outputs.

The effect of equation (4) is to add external inputs into the current marking. Then, equation (5) adds the timings for these external tokens to the age vector. Equation (6) decrements by one the age of each place having a token, and then produces a normalized mask indicating which places are fully aged. Equation (7) uses this mask to produce a new intermediate marking composed of those places containing fully aged tokens, which may participate in enabling transitions. Equation (8) then computes the transitions to fire as before. Equations (9a) and (9b) compute normalized versions of the contribution each connectivity matrix component makes to the new marking generated by firing: equation (9c) then computes the new marking using these normalized components. Equation (10) generates the new age vector after firing, and equation (11) gives the output at each step.

Execution of BTPNs, as defined by equations (4 to 11), proceeds synchronously, and no decisions are made when places are in classical conflict over enabled transitions. A token that participates in enabling two or more transitions is replicated as required so that all may fire in the next state change. If firing a transition deposits a token into a place in which another token is present, but not fully aged, then the age for that place is reset to the timing value τ for the place. This approach is useful in modelling hardware and communication structure, e.g. as in Figure 4. To illustrate this firing rule, consider again the marked Petri-net shown in Figure 1, with timing vector $\tau = [2,4,3]$. After two state changes (to age tokens), both transitions t_1 and t_2 are enabled, and *both* fire to create state three, with marking <0:0, 1:4, 1:3>.

Example using BTPNs

The following example shows how the matrix equations describe the behavior of a system modelled with BTPNs. This development uses the same example which can be found in Alayan and Newcomb's paper on (untimed) Binary Petri Nets[29], illustrated in Figure 5. Let the net timing vector, the initial marking vector, and the external input vector be:

$$\tau = [\tau_{s1}, \tau_{s2}, \tau_{s3}, \tau_{s4}, \tau_{s5}]^T = [32010]^T$$

$$\Omega(0) = [11010]^T$$

$$X(\lambda) = \begin{cases} [10000]^T, \text{ if } \lambda = 0,2,4,\cdots \\ [00000]^T, \text{ otherwise} \end{cases}$$

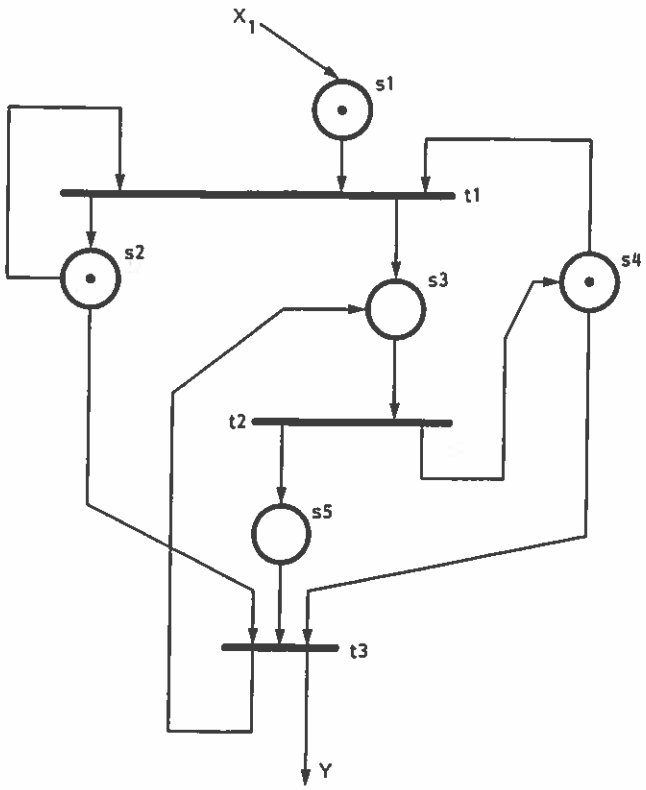


Figure 5. Petri-net for BTPN example. Place timings: $\tau = [3,2,0,1,0]$

respectively. According to our BTPN algorithm:

$$\Omega_1(0) \doteq \Omega(0) + X(0) = [11010]^T + [10000]^T$$

is the expression for the intermediate marking. The normalized result is then:

$$\Omega_1(0) = [11010]^T$$

The initial age vector $A(0)$ giving the age for each place has a value of:

$$A(0) = [32010]^T$$

corresponding to the timing applied to the initial marking $\Omega(0)$. Adding the input vector $X(0)$, we get the new age vector:

$$A_1(0) = \mathbf{pmax}\{A(0), [X(0) * \tau]\} = \mathbf{pmax}\{[32010]^T, [30000]^T\} = [32010]^T$$

and the age of an input place can be considered to be reset when there is a new token inserted into it from external inputs. After the **dec** operation, we have:

$$\mathbf{dec}[A_1(0)] = [21010]^T$$

According to equation (6) the normalized age vector is given by:

$$A_2(0) = [01010]^T$$

Because only the fully aged places can be used for BTPN transition firing, the new marking $\Omega_2(0)$ identifies all

places that have fully aged tokens. With $\lambda = 0$ in equation (7), we get:

$$\Omega_2(0) = [11010]^T * [01010]^T = [01010]^T$$

as the value of this marking. To get the next iteration firing vector $\Phi(\lambda + 1)$ in which $\lambda = 0$, equation (8) is used:

$$\Phi(1) = \mathbf{F}^- \square \Omega_2(0) = \begin{bmatrix} 1&1&0&1&0 \\ 0&0&1&0&0 \\ 0&1&0&1&1 \end{bmatrix} \square \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This means that no transitions fire, since none are connected to places all of which contain fully aged tokens. By equation (9c):

$$\Omega(1) = \Omega_2(0) + \Theta^+(1) - \Theta^-(1) + \Omega_1(0) * \overline{A_2(0)}$$

is then the starting marking for time $\lambda = 1$. Its normalized value is:

$$\Omega(1) = [11010]^T$$

From equation (10):

$$A(1) = \mathbf{pmax}\{[\mathbf{dec}[A_1(0)] - \Theta^-(1)], \Theta^+(1) * \tau\} = [21010]^T$$

is the initial age vector for the $\lambda = 1$ iteration. Let the external inputs (defined by the modeller) at time $\lambda = 1$ be:

$$X(1) = [00000]^T$$

From equation (4), then, the new intermediate marking is:

$$\Omega_1(1) \doteq (11010]^T + [00000]^T = [11010]^T$$

Similarly, from equation (5) we get:

$$A_1(1) = \mathbf{pmax}\{[21010]^T, [00000]^T\} = [21010]^T$$

The normalized timing is then given by:

$$A_2(1) \doteqdot \mathbf{dec}[A_1(1)] = [11010]^T$$

and the new fully aged marking is:

$$\Omega_2(1) = \Omega_1(1) * A_2(1) = [11010]^T$$

Therefore, we obain the new firing vector from equation (8):

$$\Phi(2) = \mathbf{F}^- \square \Omega_2(1) = \begin{bmatrix} 1&1&0&1&0 \\ 0&0&1&0&0 \\ 0&1&0&1&1 \end{bmatrix} \square \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The third set of calculations, for $\lambda = 2$, are presented here without further explanation:

$$\Omega(2) = [01100]^T$$

$$A(2) = [02000]^T$$

$$X(2) = [10000]^T$$

$$\Omega_1(2) \doteq [01100]^T + [10000]^T = [11100]^T$$

$$A_1(2) = \mathbf{pmax}\{[02000]^T, [30000]^T\} = [32000]^T$$

$$A_2(2) = [01000]^T$$

Applying equations (7) and (8), we find:

$$\Phi(3) = [000]^T$$

which shows that there are no enabled transitions to be fired.

CONCLUSION

Petri-net theory and its applications are among the most active current research topics. Research on Petri-nets focuses both on basic theoretical problems, such as concurrency modelling and state-space analysis, and on applications, such as in robotic system modelling, FMS, and software algorithm verifications. Recent projects demonsrate that Petri-net models can be used for automatically generating controller source codes for an on-line computer in a FMS. This means that difficulties in going from design to implementation can be overcome by computer-aided techniques. Researchers have introduced the concepts of Coloured Petri Nets, General Stochastic Petri Nets, Place Timed Petri Nets, Transition Timed Petri Nets, and Discrete Time Petri Nets into parallel computation systems, distributed computation systems, and FMSs. These investigations are still under development, and use high level languages and CAD techniques[3, 6, 17, 24, 30, 31]. To date, Petri-nets have been studied from topological aspects, functional analysis aspects, computer software engineering aspects, microelectronics aspects, and manufacturing and quality control aspects. It is, therefore, clear that many new concepts and new application areas will become a part of Petri-net theory and applications.

This paper contributes to the body of Petri-net theory and applications in several ways. First, BTPNs have been presented, a new place-timed formalism of Petri-nets that allows modelling of both the software structure of a FMS, and its communication lattice. The behavior equations for BPTNs are presented, and an example of their use is given. Finally, a graphical language for easily constructing these BTPN models for individual robot lattices has been mentioned. Though the analysis techniques described in this paper are appropriate for a large class of BTPNs, not all such nets represent appropriate structures for robot lines. The PFG language restricts the BTPN structures to those that are interpretable in terms of manufacturing lines. The syntax is more easily employed than constructing Petri-nets directly, and it incorporates both a data state and a control state.

REFERENCES

1 **Petri, C A** *Kommunikation mit Automaten* PhD dissertation, University of Bonn, FRG (1962)

2 **Holt, A and Commoner, F** 'Events and conditions' *Record of Project MAC Conf. on Concurrent Syst. & Parallel Computation* New York, USA (1970) pp 1–52

3 **Cai, Z N, Farnham, A, Gomez, P, Hermida, M, Newcomb, R W, Rodellar, V and Tian, J** 'Binary Petri-nets and Prolog for intelligent robots' *IEEE Int. Workshop on Lang. for Automation* Vienna, Austria (August 26–27, 1987) pp 57–60

4 **Courvoisier, M, Valette, R, Bigou, J M and Esteban, P** 'A programmable logic controller based on a high level specification tool' *Proc. IECON Conf. on Ind. Electr.* San Francisco, USA (1983) pp 174–179

5 **Silva, M and Velilla, S** 'Programmable logic controllers and Petri nets: a comparative study' *IFAC Conf. on Software for Comput. Control* (1982) pp 83–88

6 **Crockett, D H and Desrochers, A A** *Manufacturing Workstation Control Using Petri Nets* Technical Report 83, Robotics and Automation Laboratory, Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, USA (August 1986)

7 **Murata, T, Komoda, N and Matsumoto, K** 'A Petri-net based controller for flexible and maintainable sequence control and its applications in factory automation' *IEEE Trans. Ind. Electr.* Vol 33 No 1 (February 1986) pp 1–8

8 **Petri, C A** 'Tools of General Net Theory' *Proc. Int. Workshop on Petri Nets & Perf. Models* Madison, Wisconsin, USA (August 1987) p 2

9 **Peterson, J L** *Petri Net Theory and the Modeling of Systems* Prentice-Hall, USA (1981)

10 **Cai, Z N, Farnham, A, Ghalwash, A Z, Gomez, P and Rodellar, V** 'Petri nets for robot lattice' *Proc. IEEE Int. Conf. on Robotics & Automation* Raleigh, USA (March 31–April 3, 1987) pp 999–1004

11 **Stotts, P D** 'The PFG environment: parallel programming with Petri net semantics' *Proc. 21st Hawaii Int. Conf. on Sci. & Syst.* Hawaii, USA (January 1988) pp 630–638

12 **Gopalakrishnan, G C** *A Compositional Model For Synchronous VLSI Systems* Technical Report UUCS-87-024, Department of Computer Science, University of Utah, USA (September 1987)

13 **Stotts, P D (Jr)** *A Hierarchical Graph Model of Concurrent Real-Time Software Systems* PhD dissertation (TR-86-12), Department of Computer Science, University of Virginia, USA (August 1985)

14 **Martinez, J, Alla, H and Silva, M** 'Modelling and design of flexible manufacturing systems' in **Kusiak, A (ed)** *Manufacturing Research and Technology 3* Elsevier, Netherlands (1986) pp 389–406

15 **Ekberg, G and Krogh, B H** *Prototype Software for Automatic Generation of Computer Programs for Discrete Manufacturing Processes* Technical Report CMU-RI-TR-87-3, Robotics Institute, Carnegie-Mellon University, USA (February 1987)

16 **Lin, C and Marinescu, D C** 'On stochastic high-level Petri nets' *Proc. Int. Workshop on Petri Nets & Perf. Models* Madison, Wisconsin, USA (August 1987) pp 34–43

17 **Ajmone Marsan, M, Balbo, G, Chiola, G and Conte, G** 'Generalized stochastic Petri nets revisited: random switches and priorities' *Proc. Int. Workshop on Petri Nets & Perf. Models* Madison, Wisconsin, USA (August 1987) pp 44–53

18 **Murata, T and Zhang, D** *A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs* Technical Report UIC-EEC-87-1, University of Illinois at Chicago, USA (March 1987)

19 **Alla, H, Ladet, P, Martinez, J and Silva-Suarez, M** 'Modelling and validation of complex systems by coloured Petri nets: application to a flexible manufacturing system' *Lecture Notes in Computer Science # 188: Advances in Petri Nets 1984* Springer Verlag, FRG (1984) pp 15–31

20 **Valette, R** 'Nets in production systems' *Lecture Notes in Computer Science (255)* Springer-Verlag, FRG (1987) pp 191–217

21 **Komoda, N, Kera, K and Kubo, T** 'An autonomous, decentralized control system for factory automation' *IEEE Comput.* Vol 17 No 12 (December 1984) pp 73–83

22 **Chocron, D and Cerny, E** 'A Petri net based industrial sequencer' *Proc. IEEE Int. Conf. & Exhib. on Ind. Control & Instrumentation* (1980) pp 18–22

23 **Pratt, T W** 'Formal specification of software using H-graph semantics' in **Ehrig, H, Nagl, M and Rozenberg, G (eds)** *Lecture Notes in Computer Science # 153: Graph Grammars and Their Application to Computer Science* Springer-Verlag, FRG (1983) pp 314–332

24 **Stotts, P D (Jr) and Pratt, T W** 'Hierarchical modeling of software systems with timed Petri nets' *Proc. Int. Workshop on Timed Petri Nets* Turin, Italy (July 1985) pp 32–39

25 **Reisig, W** *Petri Nets: An Introduction* Springer-Verlag, FRG (1985)

26 **Coolahan, J E (Jr) and Roussopoulos, N** 'Timing requirements for time-driven systems using augmented Petri nets' *IEEE Trans. Software Eng.* Vol 9 No 5 (September 1983) pp 603–616

27 **Stotts, P D** The PFG language: visual programming for concurrent computation' *Proc. 1988 Int. Conf. on Parallel Processing* St Charles, IL, USA (August 1988) pp 72–79

28 **Alayan, H, Newcomb, R W and Niznik, C A** 'Reliability of basic robot automated manufacturing networks' *Proc. IEEE Southeast Conf. 84* Louisville, USA (April 1984) pp 291–294

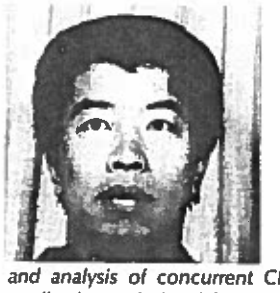
29 **Alayan, H and Newcomb, R W** 'Binary Petri-net relationships' *IEEE Trans. Circuits & Syst.* Vol 34 No 5 (May 1987) pp 565–568

30 **Haas, P J and Schedler, G S** 'Stochastic Petri-nets with simultaneous transition firings' *Proc. Int. Workshop on Petri Nets & Perf. Models* Madison, Wisconsin, USA (August 1987) pp 24–32

31 **Jensen, K** 'Coloured Petri-nets and the invariant-method' *Theor. Comput. Sci.* Vol 14 No 3 (June 1981) pp 317–336

David Stotts *received his BSc in mathematics and physics from the University of Richmond, USA in 1979, and his MSc and PhD degrees in computer science from the University of Virginia, USA in 1981 and 1985 respectively. In 1985 he joined the Department of Computer Science at the University of Maryland Institute for Advanced Computer Studies. His research interests are in the areas of concurrent computation models and languages, visual programming, and Petri-net-based hypertext.*

Robert W Newcomb *received his BSEE from Purdue University, USA in 1955, his MS from Stanford University, USA in 1957, and his PhD from the University of California, Berkeley, USA in 1960, all in electrical engineering. Dr Newcomb was a research intern at the Stanford Research Institute from 1955–57, and was on the faculties of UC Berkeley from 1957–60, and Stanford from 1960–70. He then joined the Electrical Engineering Department at the University of Maryland, where he now directs the Microsystems Laboratory.*



Z Ning Cai *received his BSEE from Jiaotong University, China in 1983, and his MSEE from the University of Maryland, USA in 1986. He is currently a PhD student in the Electrical Engineering Department at the University of Maryland, having received the University of Maryland Scholarship and a Dupont Fellowship to support his studies. His research interests include modelling and analysis of concurrent CIM systems, Petri-net theory and its application to industrial process control system design, and discrete time dynamic event systems.*