The International Journal of Flexible Manufacturing Systems, 1 (1989): 255–280

Modeling FMS with Decision Petri Nets

S. WADHWA and JIM BROWNE
CIM Research Unit, University College, Galway, Republic of Ireland

Abstract. Decision point extended timed Petri nets or decision Petri nets (DPN) are introduced as an extended modeling framework for FMS performance evaluation. The decision point extension allows the explicit modeling of the control of the flow of tokens in timed Petri nets and hence represents the control of the flow of material, resources, and information in FMS. Further, the concept of a bounded transition is proposed to conveniently model the blocking logic in an FMS with limited buffer capacities. The motivation to present these conventions is to develop a user-friendly graphic model to represent FMS designs for analysis by discrete event simulation. DPN affords concise models that can be conveniently developed and easily transformed into discrete event simulation models. With the help of a simple FMS example, which includes a number of part types, loading rules, dispatching rules, and probabilistic branching (at an inspection station), we illustrate the DPN model development. As an illustration of the ease with which it can be transformed into a simulation model, we have developed a generalized simulator called ROBSIM and outline here its methodological basis. The proposed concepts should be of interest to users of discrete event simulation in FMS design or elsewhere to tap the potential of basic Petri net concepts for graphic representation and specification purposes. In particular, our work should encourage other researchers to develop extensions relevant to their own areas of interest.[1]

1. Introduction

The analysis of flexible manufacturing systems (FMS) for design and control is complex. Typically, investments in FMS are very large, and the design and operation of such systems must be planned in detail. A number of analytical and simulation techniques to address various levels of FMS design complexity are available. This article explores the use of Petri-net-based simulation as a design technique. The Petri nets serve as a graphic representation and specification framework, and simulation is the analysis tool for this framework.

Petri nets are used as a formal graph method of modeling the flow of information and control in systems, especially those that exhibit asynchronous and concurrent properties (Agerwala and Flynn 1975; Peterson 1981). Flexible manufacturing systems exhibit such characteristics during the manufacture of part types with different processing sequences (Dubois and Stecke 1983).

The dynamic element in Petri nets is the transition, which can be viewed as an event. A place represents the definition of a state. The tokens in a place represent the quantification of this state. Thus, the fundamental feature that a Petri net portrays is the state-event or state-transition relations of a system. The transition firing represents an evolution of the states based on logical relations in the system modeled as a Petri net.

Timed Petri nets involve the notion of time between the firing of various transitions. The fundamental aspect of Petri nets that is useful for discrete event simulation is the representation of the state-transition relations of a system over time. The literature on timed Petri nets is growing (see Ramchandani 1974; Sifakis 1979; Ramamoorthy and Ho 1980; Dubois and Stecke 1983; Alanche et al. 1984; Martin and Alla 1987; Germain and Descotes-Genon

1987; Wadhwa and Browne 1988). Dubois and Stecke (1983) outline the usefulness of timed Petri nets for planning and control problems in FMS. They present a brief discussion and comparison of the Petri net approach and other approaches: queueing networks (Solberg 1979); perturbation analysis (Ho, Chi, and Cao 1983); and simulation (Stecke and Solberg 1981). Performance evaluation using timed Petri net modeling has been attempted by analytical techniques (Sifakis 1979; Ramamoorthy and Ho 1980; Dubois and Stecke 1983).

Dubois and Stecke (1983) provide examples of the use of timed Petri nets in manufacturing systems. They argue that under some structural assumptions, the timed Petri net models translate into linear equations in a (max, +) algebra. Efficient algorithms can solve these equations for the purposes of performance evaluation and real-time control. The structural assumption calls for a deterministic, decision-free, safe, and live Petri net class. In conclusion, Dubois and Stecke point out that there appears not to be much of a limit to the modeling capabilities of Petri nets. However, at present, the decision-free requirement is necessary for using analytical techniques to evaluate the performance of a system with timed Petri nets. To analyze non-decision-free Petri nets, the modeling capabilities of simulation are usually needed. Since FMS is characterized by flexibility and real-time control, the decision-based aspect is crucial for detailed design in this area. Thus, analytical techniques at present are limited in analyzing FMSs in detail. Simulation is now widely accepted for detailed FMS design. Dubois and Stecke suggest the need for developing new modeling conventions with respect to the use of timed Petri nets in new applications. We propose some extensions to timed Petri nets to conveniently model flexibility and real-time control aspects of an FMS.

2. Motivation for our work

Several researchers who have used Petri nets to model systems have found them too simple and limited to easily model real systems (Peterson, 1981). Thus, there has been a marked tendency to extend the Petri net model. Peterson (1981) discusses some of these extensions, one involving an exclusive-OR transition and the other a priority transition. In the extension involving an exclusive-OR transition, the transition firing rule is to fire a transition if and only if exactly one of its input places has tokens and all the other input places have zero tokens. When the transition fires, it removes a token only from the input place with tokens. Similarly, in the priority transition extension, the concept of associating priorities with the transitions is used: if two or more transitions are simultaneously enabled, the highest-priority transition is chosen to fire first.

These extensions to Petri nets were created for solving specific problems that the researchers encountered in their attempts to model real systems. With regard to bounded Petri nets, Peterson (1981) shows that the modeling power of Petri nets is vast and that the above extensions are only for convenience. It can be argued that extensions to bounded Petri nets have been suggested for the following reasons:

- To achieve convenience in the translation of the system into the extended Petri net model and its subsequent interpretation. The extensions are based on identifying a set of pertinent state-transition rules in the system domain. New graphic notions and new transition firing rules are then defined to represent these systems easily.

- To achieve convenience in transforming the extended Petri net model into an analysis technique. For instance, Dubois and Stecke (1983) chose to translate a manufacturing system and its control problems into decision-free and safe timed Petri nets to ease the transformation into the analytical technique based on linear equations in a (max, +) algebra. For modeling the blocking logic, they define two transitions, an instantaneous and a noninstantaneous transition. The decision-free and safe timed Petri net is not an extension but a subclass. However, such a subclass is chosen to facilitate the analysis.

It can be concluded that the simplicity (and hence generality) of the transition firing rule in a conventional Petri net allows us to model a large spectrum of systems but involves a lot of modeling effort and results in models that are not concise. The extensions proposed in the past and the ones proposed in this article are primarily motivated by the need to develop a concise and convenient graphic model to represent a specific domain. *While retaining the fundamental Petri net concept of representing states, transitions, and their relations, one should explore what type of transition firing rules will represent the pertinent features of a particular domain more conveniently.* The use of such extensions is only justified if the means for analyzing the resulting models can also be formulated.

We present some simple yet effective extensions as four types of transition firing rules to timed Petri nets to model the flexibility and real-time control in an FMS when discrete event simulation is the chosen analysis tool. Based on these, we propose a decision point extended timed Petri net framework, referred to as decision Petri nets (DPN). Each extension involves a new transition firing rule and is associated with a graphic notation as the rule identifier. We have simply added these to the normal transition firing rule of conventional timed Petri nets. Thus, the DPN framework contains five types of state-transition rules. Further, we develop a methodological basis for a data-driven DPN simulator. The features of Petri net execution logic are coded in the SLAM II simulation language, and suitable data input formats are interfaced. Such formats allow the user to express the Petri net model unambiguously to the simulator. This simulator is *generalized*, since any class of Petri nets can be simulated. It is *data-driven*, because the user needs no programming or simulation expertise: it is only required to input the data defining the DPN model. We call this DPN-based simulator ROBSIM. It is implemented on a VAX-11/780.

There are already simulators based on Petri nets (Alanche et al. 1984; Martin and Alla 1987; Germain and Descotes-Genon 1987). These are based on colored Petri nets (CPN). Some of the concepts in DPN-based ROBSIM are similar to that of CPN, i.e., providing attributes to the tokens. However, we believe that the flexibility and decision-based aspect of FMS are more convenient to model by DPN.

If we need to use simulation, why should we build models with timed Petri nets initially and then simulate them? FMS involves complex materials flow. In our experience, the use of discrete event simulation for the detailed modeling of FMS (flexibility, real-time control, nondeterminism, and multiple part flow) is inevitable irrespective of whether we use Petri nets. The problem with the use of discrete event simulation is that the available simulation languages do not provide a graphic means to specify the simulation model. Seen from another perspective, this suggests that the operational features of an FMS design cannot

be portrayed graphically by a designer in a manner that could be understood unambiguously by the simulation modeler, and vice versa. The DPN modeling framework proposed by us is essentially a step towards meeting this requirement.

Bel and Dubois (1985) point out that a model should be a communication medium between people and also a tool for computer-based experiments. In our opinion, the graphic nature of Petri net models allows them to be used as a design specification and representation scheme. Further, the use of Petri nets helps in the development of a well-planned simulation model (Dubois and Stecke 1983). Thus, a Petri-net-based framework has the potential of becoming an expedient mode of communication between the designers and the simulation modelers.

3. Requirements of a graphic model

The logical requirements of a user-friendly graphic model for supporting the design specification of an FMS to be analyzed by discrete event simulation may be listed as follows:

1. The model should portray the most significant elements of an FMS as explicitly as possible. Flexibility, real-time control, a number of part types with different operation sequences, and limited buffer capacities are some of these elements.
2. It should provide a process-oriented view of the FMS operation. Such a view is easier to understand and interpret. Further, the graphic model development is simplified, because this is the natural way in which we view manufacturing systems, as a set of interacting processes competing for resources. The design problem is the synchronization of these processes.
3. The graphic model should be concise, and the notions used in the graphic model should be simple to understand and unambiguous to model.
4. The graphic model should allow an easy transformation into a simulation language based on discrete event simulation.
5. If possible, such a model should be an extension of an existing and popular model, for greater user familiarity and acceptance.
6. If possible, there should be a one-to-one mapping of the notions in the graphic model with every operation logic in the modeled FMS. This means that there should be a minimum of auxiliary modeling effort involving the combination of basic graphic notions to represent a simple element in the real system. In many graphic models, we fear, such auxiliary modeling efforts tend to make the models look much more complicated than the system itself! Thus, the graphic model loses its user friendliness both in its development and its interpretation.

Petri nets have the fundamental property of providing a process-oriented representation of the logic of the flow of tokens through transitions. A transition is similar to an event in discrete event simulation. If we can develop a one-to-one mapping between the FMS elements and the Petri net elements with suitable extensions, the model development in a process-oriented fashion is facilitated. Similarly, if we can outline a one-to-one mapping

of the transition firings into an event scheduling logic, we can achieve simplicity in simulation model development. The model then can act as a communication medium between the designer and the simulation modeler.

Following the above guidelines, we have evolved the concept of a DPN model. The extensions in DPN are convenient and may not be necessary from the Petri net model development point of view. Since in DPN we define the places with bounds to map the constrained buffers in FMS, the key advantage in using DPN is modeling convenience and not the modeling power.

4. Modeling an operation with timed Petri-net-based representations

All manufacturing systems involve the production or assembly of components. These in turn consist of a series of operations that must be carried out. The operation is therefore the basic step in manufacturing. Two timed Petri-net-based representations for an operation are presented in Figure 1. In Figure 1a, the start and end of an operation are modeled as transition $t1$ and transition $t2$, respectively. A token in place $p1$ models an idle resource, and a token in place $p2$ implies a job waiting for an operation to begin on the resource. Places $p1$ and $p2$ are input places to transition $t1$, which is enabled to fire if both $p1$ and $p2$ have a token. After it fires, it consumes the enabling tokens in $p1$ and $p2$, and then it sends a timed token instantaneously to place $p3$, which is its output place. This token remains in an indisposable state until the operation activity time associated with it expires. For descriptive purposes, we say that transition $t1$ schedules transition $t2$ to fire after the activity time expires. Further, we call transition $t2$ the *scheduled transition* when this time expires. Transition $t2$ is enabled to fire when this token reaches the disposable state. Note that the transition firing is an instantaneous event. When end transition $t2$ fires, it frees the resource by sending a token to place $p1$ and releases the job by sending a token to place $p4$.

Figure 1a illustrates timed Petri-net-based representation where time is explicitly associated with a timed token in a place. This form of representation is conventionally used to conform to the theory of Petri nets. We present an alternative representation of time in Figure 1b, which in our opinion is more user-friendly, although it deviates from the Petri net theory. Here we do not associate time with a token in a place. Instead, we simply mark an activity $a1$ underneath the directed arc joining transitions $t1$ and $t2$. While we suggest using this convention, the DPN framework and the simulator ROBSIM are not restricted to this convention.

5. Decision Petri nets (DPN)

In this section, we outline some extensions to timed Petri net graphs and corresponding extensions to the execution logic of the nets to cater for the modeling of FMS. We refer to this extension as a *decision point extended timed Petri nets* and in short as DPN (decision Petri nets). These extensions result from superimposing the concepts of a decision point framework (Wadhwa, Maguire and Browne 1986) onto Petri net concepts. At each point in the Petri net where a priority based decision is to be made about the flow of a

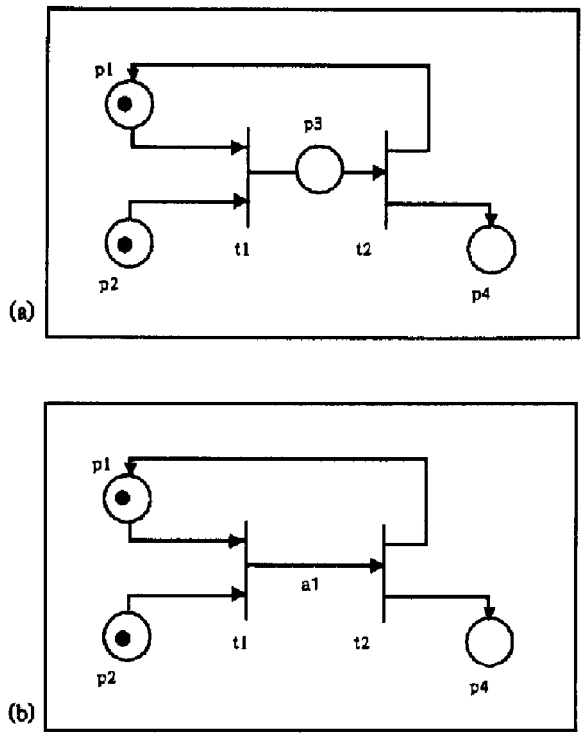


Figure 1. Illustration of two timed Petri-net-based representations to model an operation.

token, a hollow box is printed. Associated with each hollow box is one of the three decision point types: token priority decision point (DP1); transition priority decision point (DP2); place priority decision point (DP3). Figures 2–4 illustrate these decision points (DP).

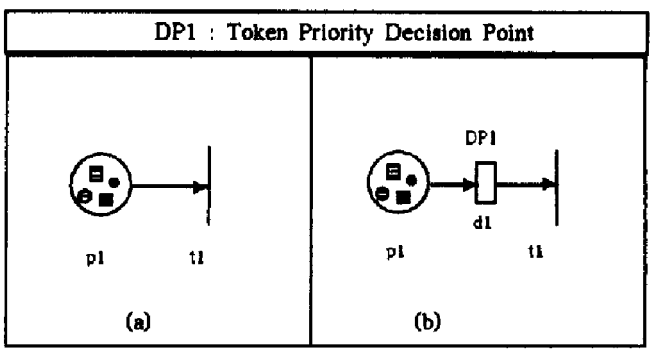


Figure 2. Illustration of the token priority decision point.

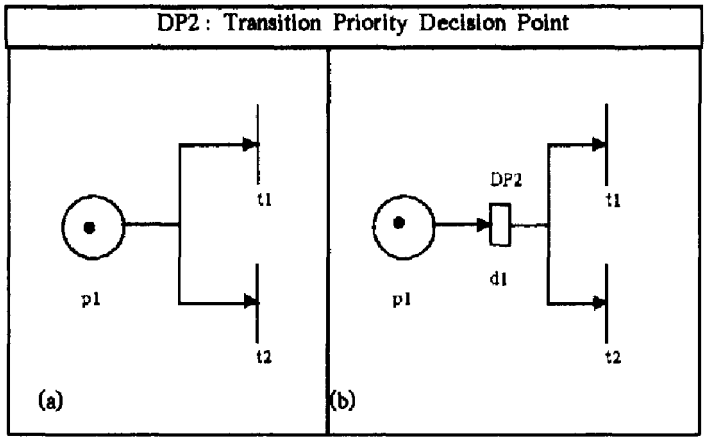


Figure 3. Illustration of transition priority decision point (DP2).

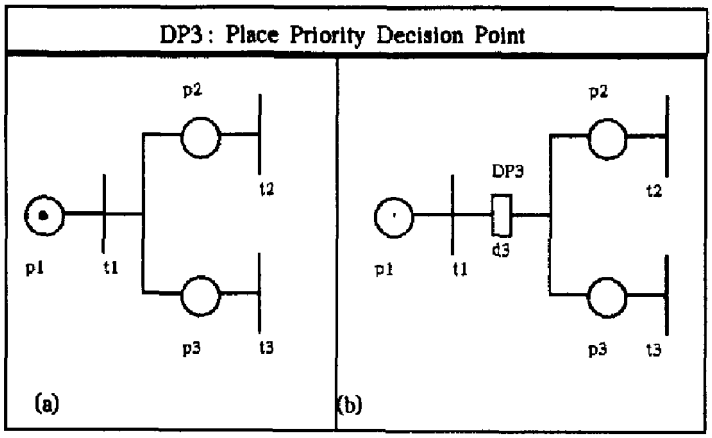


Figure 4. Illustration of the place priority decision point (DP3).

Figure 2a illustrates a marked Petri net with four token types in place $p1$. Transition $t1$ is enabled by any one of the four tokens. Figure 2b shows a corresponding decision point extended timed Petri net (DPN) for the net in Figure 2a with DP1 as the token priority decision point. DP1 selects which token should be consumed by transition $t1$ when it fires. The user identifies this decision point type with identifiers $d1$, $d2$, etc. Further, the ROBSIM interface prompts the user to input a priority control option on it. Figure 3a illustrates a marked Petri net with one token in place $p1$. Transitions $t1$ and $t2$ are simultaneously enabled by this token. Figure 3b shows a decision point extended timed Petri net for the same, with DP2 as the transition priority decision point. It selects which transition should fire and consume the enabling token. The other transition is disabled automatically. Figure 4a illustrates a marked Petri net with one token in place $p1$. Transition $t1$ is enabled by

this token. After firing of transition $t1$, the token can either go to output place $p2$ (say, a probability of .2) or to place $p3$ (say, a probability of .8). Figure 4b illustrates a decision point extended timed Petri net (DPN) for the same, with DP3 as the place priority decision point. It probabilistically decides the output place where the token should be deposited. The choice of the next place may also be defined as a priority.

We also introduce the concept of a *bounded transition* to model the blocking logic of an FMS operation explicitly. A bounded transition is represented by a double bar, as shown in Figure 5. Conventional Petri nets do not have such representation. However Dubois and Stecke (1983) showed how to model the blocking situation in FMS using conventional Petri nets. We propose to extend the modeling conventions of Petri nets to include this double bar representation, as it is simpler to use. We adopt the Petri net with bounded places to have a one-to-one mapping with a constrained buffer in FMS. We define a new rule for the firing of a bounded transition: a bounded transition is enabled to fire only if its input places have enabling tokens and its output places have capacity to hold the released tokens. Referring to Figure 5, suppose that place $p2$ is bounded to a maximum of three tokens. As shown in the figure, place $p1$ has one token and place $p2$ has three tokens in its current marking. Transition $t1$, though enabled by its input token at place $p1$, cannot fire because its output place $p2$ has no capacity to hold the released token. We refer to this as the blocking of the bounded transition $t1$. As soon as place $p2$ has some capacity in any future marking, this blocking is relieved and transition $t1$ will be set to fire. Such a case will arise when a transition such as $t2$ is scheduled to fire and as it fires it consumes a token from $p2$.

We now summarize the transition firing rules for each of our proposed extensions to the Petri net modeling conventions.

Normal transition execution

A normal transition is one that is not associated with any decisions and whose firing is not influenced by the status of the bounded output places. This is a transition in conventional Petri nets. Its firing rule is the same as in conventional timed Petri nets. It fires when its scheduled time has expired and it has enabling tokens in its input places. After firing, the tokens are deposited to its output places.

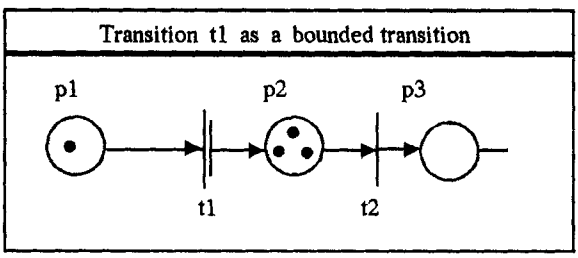


Figure 5. Transition $t1$ is represented as a bounded transition.

DP1 (token priority) execution

DP1 is executed when its associated transition is enabled. The execution of DP1 results in a selection of a token type from all enabling tokens in its associated place. In Figure 2b, the associated transition and place for DP1 are $t1$ and $p1$, respectively. The selection criteria can be based on priorities assigned to tokens or some other attribute of the tokens.

DP2 (transition priority) execution

DP2 is executed when its associated transitions are enabled simultaneously. We can refer to this set of transitions as *flexible transitions.* In Figure 3b, transitions $t1$ and $t2$ are flexible transitions. The execution of DP2 results in a selection of one transition from this set of flexible transitions. Such a selection will be based on predefined priorities of the transitions. Once a transition is selected to fire, all other transitions in its set are disabled immediately.

DP3 (place priority) execution

DP3 is executed when its associated transition fires. In Figure 4b, the associated transition is $t1$. The execution of DP3 results in a selection of the output place to which the token should be deposited. The set of possible places to which the token can be probabilistically deposited can be referred to as *probabilistic places* or *priority places.*

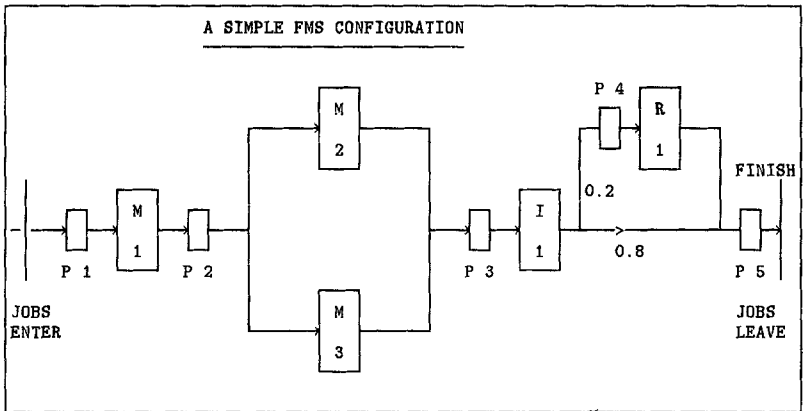
Bounded transition execution

A bounded transition is enabled to fire only if its scheduled time has expired and it has enabling tokens in its input places and its output places have capacity to hold the released tokens.

In the following sections we illustrate and discuss DPN model development for a simple FMS example.

6. An example of the use of DPN

Figure 6 gives a schematic of a simple FMS configuration and a description of the resources, work-in-process (WIP) buffer locations, allowable buffer levels, part flow control points (the decision points), the nature of decision points (loading control, dispatching control, probabilistic routing point), and the activity times. Though ROBSIM provides the facility to specify times as samples from a number of standard distributions, we have chosen deterministic times to keep the description simple. The objective of the simulation is to estimate the throughput in a production shift of 480 minutes. We assume no breakdowns during this duration.



THE FMS SPECIFICATION

An FMS consists of 3 machines (M1,M2,M3), an inspection station (I1) and a repair station (R1). All the machines and stations are flexible and can process any part type. There are 3 Part types to be processed in this system. The routing for each part type is given below :

PART | 1 : ENTER - M1 - M2/M3 - I1 - R1/FINISH
PART - 2 : ENTER - M2/M3 - I1 - R1/FINISH
PART | 3 : ENTER - M1 - M2/M3 - FINISH

(note : M2/M3 indicates M2 or M3 and R1/FINISH indicates R1 or FINISH)

After inspection either the parts go for FINISH or are sent for Repair R1. On average 80% parts are passed as finished and 20% need Repair R1.

The part types enter in a cyclic sequence with time between arrival of 3 minutes. The parts are assigned static priorities at the time of entry into the system. Part # 3 has highest priority, followed by Part # 2.

Loading control is exercised at all station inputs i.e. P1,P2,P3 and P4. The control rules used for part type selection are :

	P1	P2	P3	P4
RULE :	FCFS	JOB PRIORITY	RANDOM	FCFS

Figure 6. An FMS configuration.

6.1. DPN model development

The following outline is presented to accomplish the DPN model development task. First, the elements of a DPN model are transitions and their types, places and place types, tokens and token types, activities, input and output functions, decision points (DP), DP types, and their location. In the following sections, we describe how to derive these elements from the FMS specification.

Dispatching control is exercised only at P2. If both machines, M2 and M3 are simultaneously free and a part is waiting at P2 to be processed then part is dispatched to the priority machine M2.

The maximum allowable buffer levels are limited and are shown as under.

PLACE	BUFFER CAPACITY
P1	3
P2	3
P3	3
P4	2
P5	5

If a buffer is full, then the preceding operation can not unload its finished part. In this case the operation is blocked and the station is unavailable to take in a new part. The situation exists until some available capacity is created in the full buffer due to part movement to the next station.

The operation times for the part types are given in minutes as under :

PART #	M1	M2	M3	I1	R1
1	2.	4.	6.	3.	10.
2	-	2.	5.	4.	11.
3	6.	3.	7.	-	-

Assume that the transport required in moving the parts from one station to next entails no time. Also assume that there are no breakdowns.

Figure 6. An FMS configuration (continued).

6.1.1. Transitions. A transition can be viewed as an instantaneous event at which the status of the system changes. The status of the system changes every time a resource starts or ends an operation. Figure 7 shows the transitions with identifiers and definitions for the FMS example. There are four types of transitions in DPN. The start transition signifies the entry of jobs into the FMS. The finish transition refers to the departure of finished jobs from the FMS. The flexible transitions refer to a set of transitions that can all be enabled by a part token in a place if their resource places have a token also. The concept of flexible transitions is used to represent a situation in FMS where a job has the option of being dispatched to one of the many machines available to process it. In the present example, M2 and M3 are such machines. Their corresponding transitions $t4$ and $t6$ are thus flexible transitions.

TRANSITIONS

IDENTIFIER	DEFINITION
t1 :	A job enters the system.
t2 :	A job starts on M1.
t3 :	A job finishes on M1
t4 :	A job starts on M2.
t5 :	A job finishes on M2.
t6 :	A job starts on M3.
t7 :	A job finishes on M3.
t8 :	A job starts on I1.
t9 :	A job finishes on I1.
t10:	A job starts on R1.
t11:	A job finishes on R1.
t12:	A Finished job leaves the system.

Figure 7. Transitions with identifiers and definitions for the FMS example.

A bounded transition is a transition that is blocked if its part output place cannot hold the released token. A bounded transition is denoted as a double bar on the DPN model.

6.1.2. Places. A place can be viewed as a representation for a static state of a system with regard to the condition of a resource, a WIP buffer, or an activity.

The complete status of the system is a set of places defining the state of the system for each resource, WIP buffer, and activity. We identify a place by a place identifier ($p1$, $p2$, $p3\ldots$). Figure 8 shows the identification and definitions for the places. There are two types of places in DPN: the job type (or part type) places and the resource type places. The job type places model the status of the jobs, whereas resource type places only store the availability or nonavailability of a resource.

6.1.3. Tokens. Tokens may be viewed as quantifying the magnitude of the state of each place in the system. For instance, two tokens present in place $p2$ implies that two jobs are waiting for M2/M3. The Petri net with tokens is called a *marked Petri net.* A marking defines the distribution of tokens in various places at any instant. Thus, a marking holds the information of the complete state of the system. The tokens may be of different types. For instance, we have a resource type token and a job type token. ROBSIM implicitly controls the movements of token types into corresponding place types. The job type token carries various attributes about the status of a token, thereby uniquely identifying it. These attributes include the part number, priority, last transition, and next transition.

6.1.4. Activities. The operation time on machine M1 is represented by the time between the firing of transition $t2$ and transition $t3$. We identify activities on DPN graphs by symbols $a1$, $a2$, $a3, \ldots$. These are marked under the output functions from one transition to the other and are associated with the activity places.

PLACES

IDENTIFIER	DEFINITION
p1 :	A Job waits in queue for M1.
p2 :	A Job waits in queue for M2/M3.
p3 :	A Job waits in queue for I1.
p4 :	A Job waits in queue for R1.
P5 :	A Job waits in queue to leave system.
p6 :	M1 is idle.
p7 :	M2 is idle.
p8 :	M3 is idle.
p9 :	I1 is idle.
p10:	R1 is idle.
p11:	Operation on a job on M1 in progress
p12:	Operation on a job on M2 in progress
p13:	Operation on a job on M3 in progress
p14:	Operation on a job on I1 in progress
p15:	Operation on a job on R1 in progress

Figure 8. Places with identifiers and definitions for the FMS example.

6.1.5. Decision points. The decision points (DP) indicate the points in the FMS where decisions regarding the *control* on the flow of parts have to be made. The decisions involve the selection of part flows, resource flows, and where the part is routed. The part flow is a physical flow, whereas the resource flow may be a flow from one state to another. For instance, if the state of M1 changes from idle to not idle, we say the resource M1 flows. The flow is represented in the DPN by the movement of tokens. The tokens move only when a transition fires. Thus, all the decision points are associated with one or the other transition on the DPN graph.

Lenz (1983) describes six types of controls in the operation of FMS. We have covered three of these in our current example, those dealing with part flow control. The other three controls involve transport control. This simply implies the use of DP1, DP2, and DP3 for transporter type tokens if such an FMS is being modeled. Figure 9 illustrates the decision point information for the example.

6.1.6. Input and output functions. The input/output functions to a transition can be viewed as the representation of input/output conditions, activity relations and decision points to the start and end of each FMS operation. The input and output functions for transitions are linked to each other. The token movement takes place along the linking of these functions. The operation sequence and the flexibility in the FMS operation determines these links.

6.1.7. Integration of DPN elements. The DPN model is an integration of transitions, places, activities, and decision points linked by input and output functions in logical relations derived from the logic of an FMS operation. Tokens represent the flowing entities (parts, resources, and information) of the FMS.

DP1 identifiers	Definition
d1	Loading Control at P1 (input queue to M1.)
d2	Loading Control at P2 (input queue to M2/M3.)
d3	Loading Control at P3 (input queue to I1.)
d4	Loading Control at P4 (input queue to R1.)

DP2 identifiers	Definition
d1	Dispatching control at P2 (selection between M2/M3)

DP3 identifiers	Definition
d1	Probabilistic routing on parts after inspection.

Figure 9. Decision points with identifiers and definitions for the FMS example.

To obtain a process-oriented view we need to concentrate on the flow of each part type. The operation sequence of each part type will be different depending on the resources needed for each operation. Thus, when there are n part types, we have n different operation sequences and hence n different logical relations. How do we represent n sets of logical relations graphically in two dimensions such that the resulting model remains concise and conveniently interpreted?

We realize this objective by assigning attributes to tokens and *graphically portraying each of the n logical relations as n disjointed DPN graphs.* Each graph represents the operation sequence logic for each part type. We generate these graphs using the common set of identifiers and definitions for transitions and places. There may be common places and transitions on two or more nets. The complete marking of the DPN model is the union of the markings of the n disjointed nets. The individual token types will flow on a particular net but under the constraints of the complete marking. However, the logical sequence followed by each token will depend on its own net.

The physical interpretation of this concept is simple. Since all the transitions and places have been identified from all the resources in the FMS, the different nets simply portray the different operation sequences being followed by the part types under the status of resources no matter what part type is involved. If a resource type token is consumed from its place by the transition fired on one net, this information is available on each net through the same place identifier. Any other token on its net requiring a resource token in this place will have to wait in the input buffer place for this resource's transition. Since the input buffer places are also identified from a common set, this information will be available on every net having this place. Thus, we may have different token types with their attributes deposited in the same input buffer place and every net "knowing" this via the union of the marking.

6.1.8. Token entry control. The disjointed DPN model requires a mechanism for *net assignment* when the tokens enter the nets. In FMS we have part entry control, which governs the part entry into the system. In ROBSIM we provide a menu for options on entering the tokens into the nets. When a token enters, we invoke the start transition of the net (operation sequence) assigned to it. The token carries the attributes that store its identity uniquely, i.e., token type, assigned net number, priority and time of entering the net, and so on. ROBSIM automatically moves this token on the assigned net under the constraints of the union of the disjointed net markings as the simulation progresses.

Following the above steps, we construct the DPN disjointed nets for our current example. These are shown in Figures 10, 11, and 12. Note that the places $p11$, $p12$, $p13$, $p14$, and $p15$, which represent the busy states of the machines M1, M2, M3, I1, and R1, respectively, are shown simply as activities $a2$, $a5$, $a7$, ... for the sake of clarity.

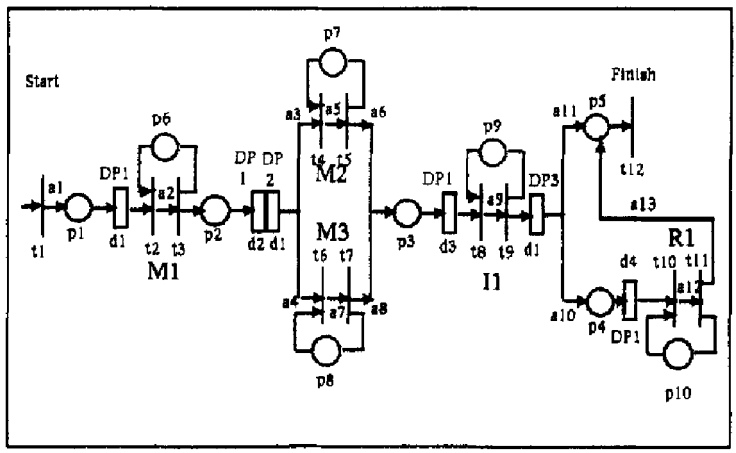


Figure 10. DPN-based disjointed net to model the logical relations for the operation sequence followed by part type 1.

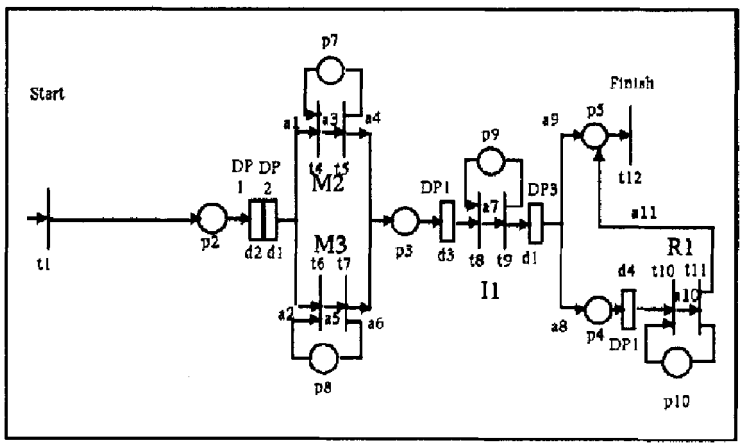


Figure 11. DPN-based disjointed net to model the logical relations for the operation sequence followed by part type 2.

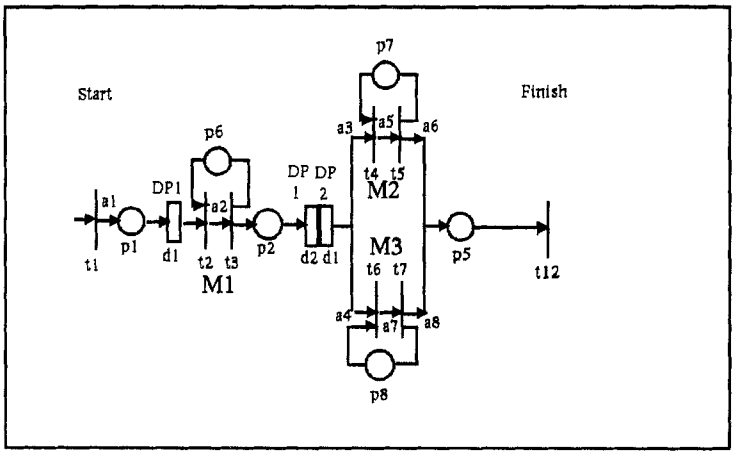


Figure 12. DPN-based disjointed net to model the logical relations for the operation sequence followed by part type 3.

6.2. Discussion

This concept of disjointed Petri nets makes DPN model development very easy and facilitates its easy interpretation. The disjointed nets allow us to model the FMS design in a process-oriented fashion, that is, a number of part types flowing through the system and competing for some common resources. Transitions representing the events at individual resources, and tokens representing the part types, allow us to view the part types flowing through the resources. The decision points on the nets capture the control points and their locations with respect to the resources in the FMS. One can envision the flow of part types being controlled at these decision points. Further, the use of decision points does not pose any limitation in developing the model as disjointed nets. The resulting models are concise because the number of transitions and places is independent of the number of part types. Further extra modeling effort to model blocking has been avoided by the use of a bounded transition. There is a one-to-one mapping of the FMS elements and the DPN elements. Thus, we meet the requirements for a user-friendly graphic model.

In the following section, we outline the ease of transforming a DPN graphic model into a discrete event simulator called ROBSIM.

7. Overview of ROBSIM

ROBSIM has been developed as a *data-driven DPN simulator.* It can simulate both conventional Petri nets and decision point extended timed Petri nets (DPN), is based on the discrete event framework of the simulation language SLAM II (Pritsker 1987), and is implemented on a VAX-11/780 under VMS. It provides the user with Petri-net-oriented data input formats. The output results are provided as SLAM summary reports and application-oriented output screens. A trace report can be generated at the user's request for verification purposes.

7.1. The methodological basis of ROBSIM

The development of ROBSIM was based on the following ideas:

1. The identification of a unified framework of concepts between FMS, decision point extended timed Petri nets (DPN), and discrete event simulation.
2. The development of a nomenclature for describing a DPN as input data.
3. The identification of *static* and *dynamic* elements of a DPN. The static element represents the system design and the relations between the various components of a system. The DPN graph portrays this information. The dynamic element is the execution of a DPN and represents the system operation in a time frame. In the context of ROBSIM, the static element is data input for a DPN to be simulated and is *system-dependent.* The dynamic element is a set of fixed execution rules, which are *system-independent.* Thus, ROBSIM is basically a collection of these execution rules, which operate on the input data.
4. The identification of a generalized set of execution rules in DPN. These rules were presented in the previous section. An FMS operates through control on the flow of parts through a set of resources, the basic dynamic element being an operation. A DPN model is executed by *control* of the *flow* of tokens (representing parts and resources), the basic dynamic element being a transition.
5. Coding of these execution rules into the SLAM II simulation language using the defined nomenclature.

7.2. DPN transformation to ROBSIM

We have developed a nomenclature in which the DPN can be unambiguously input as data to ROBSIM. Here we illustrate only one input format, which is the heart of DPN model — the logic table. For each net we have a different logic table. Figure 13 shows the logic table for net number 1. The first column, named the *source transition,* covers all the transitions in the net. A source transition implies a firing transition to ROBSIM. The transition $t1$ is the *start transition* and is invoked by the token entry control mechanism in ROBSIM. This transition has no input places and thus it fires as soon as it is invoked. After firing, it releases a part token with its attributes (assigned in token entry control) and deposits the token in its output place $p1$. It then schedules transition $t2$ to be invoked by activity $a1$ corresponding to the time for the part to be present for machine M1 after it enters the system. The transition $t2$ is invoked by $a1$ when the simulation current time equals the scheduled time of $a1$. ROBSIM identifies $t2$ as the source transition and checks the status of its input places to see if it can fire. If $t2$ is specified as a bounded transition, the status of output places is also checked. If the input places $p1$ and $p6$ for source transition $t2$ have a token each (ROBSIM assumes one input/output function for each place and hence only one token is needed from each input place), the transition can fire. If it cannot fire, ROBSIM stores this transition on a wait file as a request from the token waiting in $p1$. Every time a token moves into a part place, a request is sent to the transition to fire. Thus, the transition wait file holds the request from a token to fire. In ROBSIM each transition has a corresponding wait file. In each of these files, the token requests are stored. If source

NET # : 1

Source Transition	Input Places	Output Places	Scheduled Transitions	Activity identifiers
t1	- - -	p1 - -	t2 - -	a1 - -
t2	p1 p6 -	p11 - -	t3 - -	a2 - -
t3	- - -	p2 p6 -	t4 t6 -	a3 a4 -
t4	p2 p7 -	p12 - -	t5 - -	a5 - -
t5	- - -	p3 p7 -	t8 - -	a6 - -
t6	p2 p8 -	p13 - -	t7 - -	a7 - -

Figure 13. ROBSIM sample data inputs for the logic data table.

transition $t2$ can fire, it consumes one token from each of the places $p1$ and $p6$. ROBSIM checks the decision points corresponding to firing of $t2$ (DP menus are described later). Corresponding to $t2$, it finds DP1 with identifier $d1$. ROBSIM then selects a token based on a criterion input for $d1$ in net 1. The attributes of this token are saved to be passed over to the output token place. Then the output places for source transition $t2$ are checked. Transition $t2$ has $p11$ as output place, so the selected token is deposited as a timed token with its saved attributes and transition $t3$ is scheduled with activity $a2$. Again, $t3$ is invoked at the scheduled time and the above logic is repeated for source transition $t3$.

7.3. ROBSIM implementation

Each transition in a DPN is an event. The start transition is treated as a token arrival event—subroutine ARVL. In this routine, we assign initial attributes such as part number, time of arrival, and net number to the arriving token. All other transitions are modeled by the event ENDSV.

Figure 14 shows the logical sequence of subroutines in ROBSIM. The routine USERD reads the ROBSIM input data for the DPN nets. The user interface for ROBSIM (not shown) saves all these data in ROBSIM input formats. After reading the data, the subroutine SLAM is called to bring in a discrete event simulation environment. The routines INTLC, ARVL, ENDSV, OTPUT are written in this environment. The routine INTLC is used to initialize the marking of the DPN model and to initiate the token entry control. ARVL is coded as a SLAM event routine to exercise token entry control. The event routine ENDSV is used to analyze the scheduled transitions. Finally, SLAM standard routine OTPUT is used for statistics presentation over multiple runs.

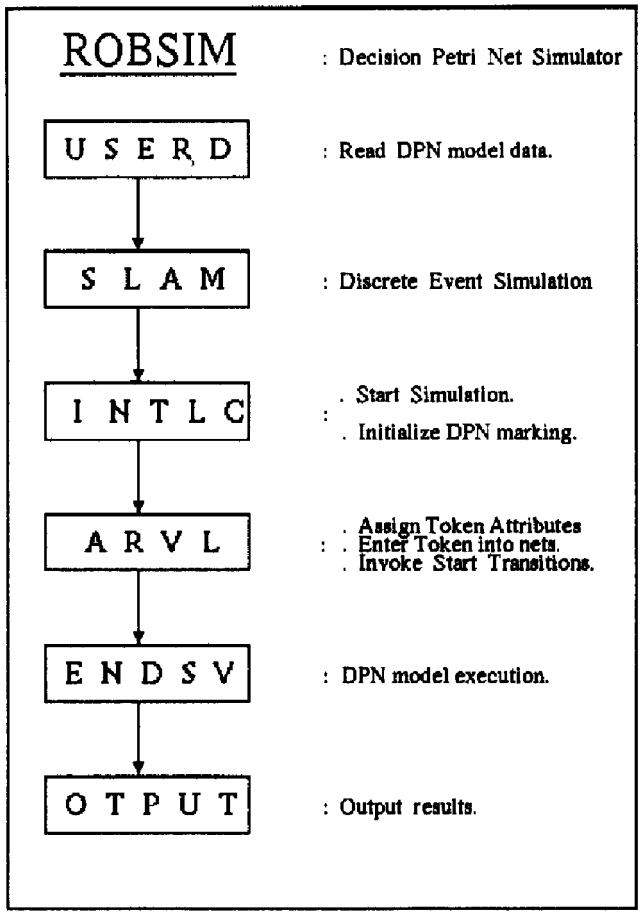


Figure 14. Subroutines used in the ROBSIM software.

Figure 15 illustrates the flow diagram for event subroutine ENDSV. This routine is invoked when the current simulation time equals the scheduled time for any transition. The invoking transition is identified from its attributes for transition number and net number. Then routine FIRE-CHECK is called, which checks if this transition can fire. If the transition cannot fire, it is saved on the transition wait file (file number identified by transition number) with its attributes. If it can fire, subroutine FIRE is called, which takes all necessary actions on the input and output places as defined by the logic table. Then, time-persistent and observation statistics are taken. Finally, a SEARCH routine is set to see if the new created marking

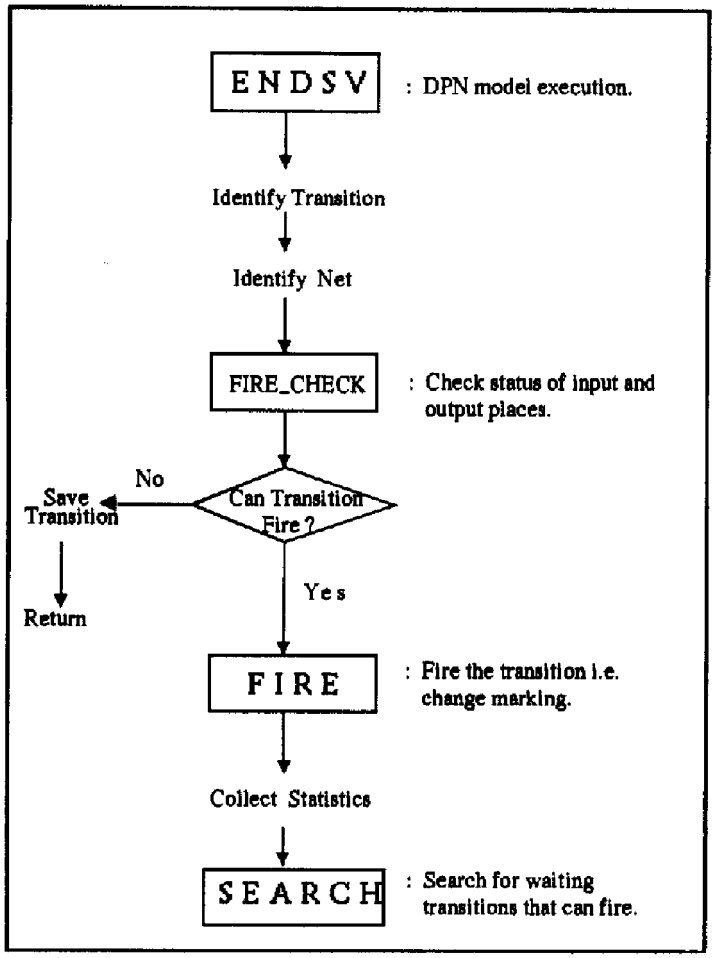


Figure 15. Flow diagram for the subroutine ENDSV.

can fire any of the waiting transitions. Figure 16 illustrates the FIRE routine. This routine takes all the actions with regard to token and attribute flow on the input and output side of the calling transition. First, a check is made to see whether the identified transition is a flexible transition. If it is, a special routine collects all firable transitions of its set invoked by the same part and exercises DECISION TYPE 2 to select the one to be fired. All other transitions of this set are removed, both from the calendar and the wait files. If it is not a flexible transition, the actions of dispatching the released token to a priority place are taken in DECISION TYPE 3. The token attributes of the transition are deposited in the part place (since tokens trigger the transitions, the transitions always carry their attributes). The next action is to consume the selected input token. This is done by DECISION

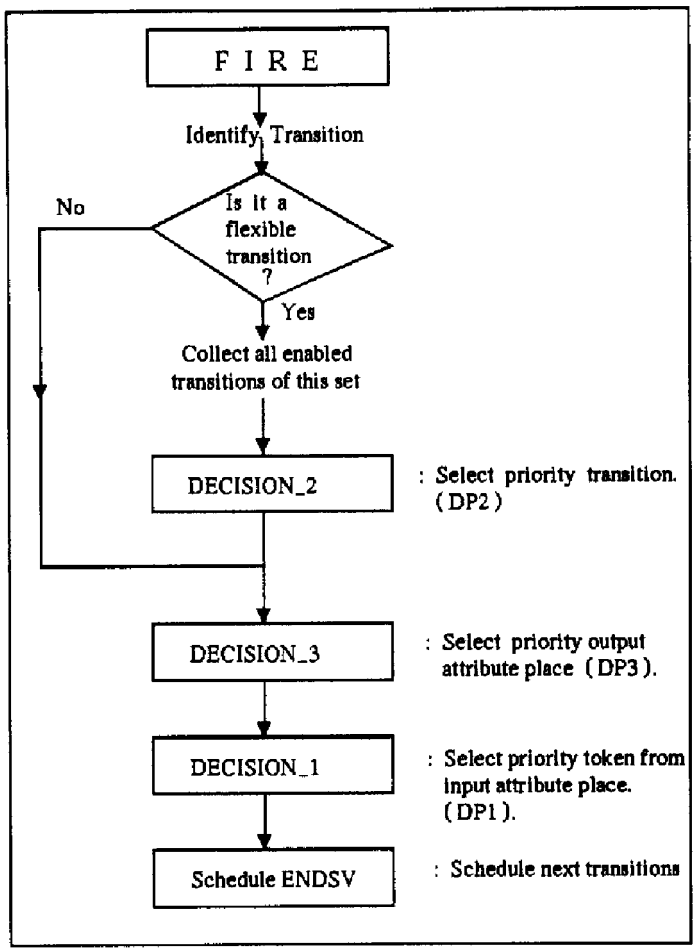


Figure 16. Flow diagram for the subroutine FIRE.

TYPE 1. Then the next transitions are scheduled through activities. Since a new token has been selected for the transition, its net number may be different than the net number of the firing transition. Finally, we schedule the activities corresponding to the logic table for this new net number.

Thus, we see that the selected token drives ROBSIM to move from one net to the other. This is a mirror of what happens in an FMS. When one part finishes on a machine, a new part starts and its process requirements are followed.

8. Comparison of DPN with other modeling frameworks

As mentioned earlier, the Petri nets graphically represent the state-transition relations in a system viewed in a discrete event perspective. The transition firing rules basically represent

the state-transition logic by which the dynamics of the system are modeled.

In DPN we have simply added four state-transition rules (transition firing rules) to timed Petri nets. These rules represent the part priority state-transition, resource priority state-transition, queue priority state-transition, and bounded operation state-transition. The first three rules essentially map a type of flexibility and its corresponding control element, and the fourth one models a "lookahead" operation constraint element in an FMS. The addition of a DP associated with a firing transition simply exercises a priority-based selection mechanism. The interface in ROBSIM allows the user to specify the selection criterion for each DP on the Petri net.

The motivation for adding these rules is to model the flexibility and the corresponding priority-based control. The idea is that if a system can be expressed completely in terms of (entity) states and transitions, there are only four types of flexibilities that can arise at any point in time:

- A number of alternative input states may invoke a transition.
- One input state may invoke a number of possible alternative transitions.
- An invoked transition may change one of the possible output states.
- A number of transitions may change one output state.

The last situation is not possible in our discrete event view: more than one transition cannot fire simultaneously. Thus, only three possible types of state-transitions can take place at any one time.

The priority control rule in each case is simply a selection criterion as to which states or transitions take part in the state-transition logic. Flexibility in any system is completely determined by one of these three types. Thus, if we provide a mechanism to define this and set a priority rule for each type of flexibility, we can model any FMS with real-time control.

Bel and Dubois (1985) developed a unified setting for various models of automated material flow and production systems. They concluded that various types of models of production systems can be obtained, according to the way the state-transition logic is described. A discrete event based unified setting is used to compare various models such as stochastic, event-graph, Petri-net-based, and discrete event simulation models. They point out that the limitation in using Petri nets for systems such as FMS are the risks of combinatorial explosion of the size of the Petri nets and the difficulty of expressing elaborated state-dependent control rules. To overcome the first limitation, the use of colored Petri nets (CPN) and the use of processes can be contemplated. Using a simple FMS example, we have shown how we can achieve the same result using a DPN model. The concept of token attributes and disjointed nets is similar to those of CPN models.

With bounds specified with the places, each new transition firing rule may be modeled as a process with the conventional Petri net firing rule. For instance, Peterson (1981) shows how to model a priority transition extension. Similarly, the blocking logic as modeled by the bounded transition in DPN can be modeled by safe and decision-free Petri nets, as shown by Dubois and Stecke (1983). Thus, extensions in DPN can be viewed as higher-level functions describing these processes. It is the identification of these functions that can be regarded as the core effort in developing a DPN modeling framework.

There are other proposed nets to express the control rules, for instance, interpreted Petri nets (Peterson 1981). In these Petri nets, each transition is associated with a logical predicate, which must be true when firing occurs, otherwise firing is delayed. These predicates express control rules that may depend upon external events, external criteria, the state of the system, or its history (for instance, past resource utilizations). In our opinion, the DPN framework provides us with a basis to view the flexibility and control in FMS. Being domain-specific, it is less general than the predicate approach but more user-friendly in its domain.

Bel and Dubois (1985) further point out that the gain in expressive power is balanced by the increased difficulty of a priori analysis of structures or dynamic behavior. Available techniques for Petri nets cannot apply to their extensions; hence the structural validity cannot be determined. Such models are then only developed for simulation purposes. Thus, the DPN model is only useful for simulation purposes.

One of the ways to compare various models is by their state-transition rules. Figure 17 presents various models classified by their state-transition logic, with a suggested place for the DPN model indicated. DPN models essentially provide a capability to conveniently represent the priority-based control rules for various flexibility types. Thus, with respect to the classification of the models, the position of DPN lies just short of the discrete event simulation model. The present DPN model appears to be useful for comparatively simple flexible systems with priority rules based on the status of immediately interacting states and transitions (i.e., it cannot conveniently express complex algorithms).

Static-Transition logic	Model
. Transition Probabilities between states with FIFO queue discipline.	Markov chains, queuing networks.
. Linear order of trajectories for all entities.	Event-graphs, Cyclic event-graphs,
. Partially defined trajectories for all entities.	Timed Petri nets.
. Predicates	Interpreted Petri nets.
. PRIORITY RULES	DECISION PETRI NETS (DPN)
. Priority rules to complex algorithms.	Discrete event simulation model.

Figure 17. A classification of models based on state-transition logic. Adapted from Bel and Dubois (1985).

9. A note on ROBSIM

The ROBSIM simulator is developed to show the ease of transforming a DPN into a discrete event framework in SLAM II (Pritsker 1987). At present, its use is for academic purposes only. It is implemented on VAX-11/780. There are approximately 1,000 lines of executable source code. The program is written in VAX FORTRAN, and it calls the necessary SLAM routines. The size of the net, i.e., the number of places and transitions, is limited to 100. The CPU time for the simulation of the present FMS example is nearly 2 seconds. The execution of the model is event-driven. There seems to be a clear possibility of improving the CPU time by optimizing the SEARCH routine. At present, whenever a transition fires, a search for the new firable waiting transitions involves a check on all transition files. This can be easily modified to search for only those transitions that are associated with the input and output places of the currently fired transition. In general, the CPU time should logically depend on the maximum number of places and transitions in any net and the simulation runtime. ROBSIM has the capability of simulating both Petri net models and DPN models.

10. Summary, conclusions, and future research

10.1. Summary

Several researchers who have used Petri nets to model systems have found them too simple and limited to easily model real systems. In this article, we present some extensions to timed Petri nets in order to develop a user-friendly graphic modeling framework, which enables us to conveniently and concisely model different types of flexibilities and real-time priority controls in FMS. The resulting modeling framework is called decision Petri nets (DPN). We propose that while retaining the fundamental Petri net concept of representing states, transitions, and their relations, one should explore what type of transition firing rules can represent the pertinent features of a particular domain more conveniently. Based on this approach, we introduced three types of decision points that can be associated with the firing of any transition: token priority, transition priority, and place priority decision points. Through an example, we show how these can explicitly and simply model the real-time flexibilities in an FMS involving priority part selection, priority resource selection, and priority queue selection, respectively. The priority selection encompasses both criterion-based selection and probabilistic selection. For conveniently modeling the blocking logic in FMS, the notion of a bounded transition is defined. Such a transition involves a firing rule that does not allow the transition to fire until its output place has capacity to hold the released token. To model multiple part types, the DPN framework allows tokens to be defined with attributes, and to simplify the model development and its interpretation, the DPN framework is based on the disjointed net approach. One disjointed net is developed to represent the processing logic of a corresponding part type.

While the DPN framework is suggested primarily as a convenient and concise representation for detailed modeling of FMS, for analysis purposes DPN can be easily supported by discrete event simulation. The methodological basis for the development of a data-driven generalized simulator (ROBSIM) to execute DPN models is outlined. We have developed and tested this simulator for industrial case studies.

10.2. Conclusions

Decision point extended timed Petri nets (DPN) can be used as a convenient graphic model for performance evaluation of FMS using discrete event simulation. The use of disjointed DPN model allows us to view the FMS operation in a process-oriented framework and hence simplifies the model development and its interpretation. The purpose of the DPN model is limited to simulation, as it lacks techniques for validating the specification of the system. It is possible to build a generalized data-driven simulator for analyzing DPN models using discrete event simulation.

10.3. Future research

With respect to its modeling potential, the disjointed DPN approach can model a wide spectrum of FMS. Research is in progress to model more complicated systems, for instance, the multilevel assembly in flexible assembly systems, and for systems requiring transport control decisions. Either of these applications involve multiple attribute places to the transitions. In the present ROBSIM version, there is only one attribute place per transition. Thus, the flow can be controlled on only type of entity (at present, part types).

The expert system language OPS5 has an execution strategy similar to a Petri net model. Based on this, ESPNET (Duggan and Browne 1988) has been developed as an expert-system-based simulator. Our future research will involve the extension of this software for the DPN models. Further, there is potential for the DPN approach in interactive model development using expert systems. This potential stems from the nature of DPN graphs, which graphically represent a discrete event simulation model for decision-based systems.

Acknowledgments

The work described in this article is research being carried out in the Republic of Ireland as part of the European Strategic Programme for Research and Development in Information Technology (ESPRIT), project no. 623, led by IPK in West Berlin, Federal Republic of Germany.

We acknowledge with thanks some useful suggestions made by the referees, which resulted in a revised version of this article.

Note

1. While retaining the fundamental Petri net concepts, we have presented four new transition firing rules as extensions to the fundamental transition firing rule of conventional Petri nets. Our primary motivation in developing decision Petri nets (DPN) is to facilitate a convenient and concise graphic representation for detailed simulation modeling of FMS. We leave it to the reader to decide whether decision Petri nets are an extension to Petri nets or a modeling framework loosely based on Petri nets, used to structure a problem for input to a computer simulation.

References

Agerwala, T., and Flynn, M.J., "On the Completeness of Representational Schemes for Concurrent Systems," presented at the Conference on Petri Nets and Related Methods, M.I.T. Cambridge, MA (July 1975).

Alanche, P.; Benzakour, K.; Dolle, F.; Gillet, P.; Rodrigues, P.; and Vallete, R., "PSI: A Petri Net Based Simulator for Flexible Manufacturing Systems," *Lecture Notes in Computer Science 188: Advances in Petri Nets*, Springer Verlag-Berlin (1984).

Alla, H.; Ladet, P.; Martinez, J.; and Silva, 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*, Springer Verlag-Berlin, (1984).

Bel, G., and Dubois, Didier, "A Unified Setting for Various Models of Automated Material Flow and Production Systems," *Computers in Industry*, Vol. 6, pp. 227–235, North-Holland (1985).

Dubois, Didier, and Stecke, Kathryn E., "Using Petri Nets to Represent Production Processes," *Proceedings of the 22nd IEEE Conference on Decision and Control*, San Antonio, TX (December 1983).

Duggan, J., and Browne, Jim, "ESPNET: Expert System Based Simulator of Petri Nets," *IEE Proceedings-D on Control Theory and Applications*, Vol. 135, No. 4 (July 1988).

Germain, R., and Descotes-Genon, B., "A Tool for Production Management: Computer Aided Simulation," in *Modern Production Management Systems*, A. Kusiak (Ed.), Elsevier Science Publishers B.V. (North-Holland), pp. 161–172 (1987).

Ho, Yu Chi, and Cao, Xiren, "Perturbation Analysis and Optimization of Queueing Networks," *Journal of Optimization Theory and Applications* (1983).

Lenz, John E., "MAST: A Simulation Tool for Designing Computerized Metalworking Factories," *Simulation*, Vol. 40, No. 2 (February 1983).

Martin, F., and Alla, H., "Discrete Event Simulation by Timed Coloured Petri Nets," in *Modern Production Management Systems*, A. Kusiak (Ed.), Elsevier Science Publishers B.V. (North-Holland), pp. 173–186 (1987).

Peterson, J.L., *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ (1981).

Pritsker, A. Alan B., *Introduction to Simulation and SLAM II*, 2d edition, John Wiley and Sons, Inc. New York, NY (1987).

Ramchandani, Chander, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Ph.D. dissertation, Department of Electrical Engineering, M.I.T., Cambridge, MA (1974).

Ramamoorthy, C.V., and Ho, Gary, S., "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5 (September 1980).

Sifakis, J., "Use of Petri Nets for Performance Evaluation," *Acta Cybernetica*, Vol. 4, No. 2, pp. 185–202 (1979).

Solberg, James J., "Analytical Performance Evaluation of Flexible Manufacturing Systems," *Proceedings of the 18th IEEE Conference on Decision and Control*, San Diego CA, pp. 640–644 (December 1979).

Stecke, Kathryn E., and Solberg, James J., "Loading and Control Policies for a Flexible Manufacturing System," *International Journal of Production Research*, Vol. 19, No. 5, pp. 481–490 (September-October 1981).

Wadhwa, S.; Maguire, J.; and Browne, Jim, "A Design Evaluation Procedure for Robot Based Flexible Assembly Systems," *Proceedings of 8th International Conference on Industrial Robot Technology*, H. Van Brussels, (Ed.), IFS Publications, Brussels, Belgium, pp. 539–548 (September-October 1986).

Wadhwa, S., and Browne, Jim "Analysis of Collision Avoidance in Multirobot Cells Using Petri Nets," *Robotersysteme* (Springer-Verlag), Vol. 4, pp. 107–115 (1988).