Introduction

New technologies in processors and networks allow system designers to conceive and build advanced and sophisticated parallel and distributed architectures, which need to integrate non-functional real time and stochastic constraints with functional distributed processing and communication.

The global behavior of such systems depends first on the local activities and data, but also on the messages sent and received by the various interconnected sub-systems. As a matter of fact, understanding, expressing, specifying and validating such global behaviors proves to be a problem of very high complexity, leading to many design and implementation difficulties and bugs. For example, when considering *n* connected processors, they can run, at a given instant in time, using 2 x 2, 3 x 3 communications, etc., or a full communication, in which all *n* processors interact. The sum of the resulting combinations, of the order of 2^n, shows the complexity of the resulting conceptual problems, and explains in particular the increasing difficulty obtained when passing from an interconnection of a few processors to an interconnection of a large number of processors: when the number of processors varies from 2 to 10, the difficulty coefficient goes from 4 to about 1,000.

It should then be clearly understood that designing such distributed architectures leads to a very complex conceptual task, which has to be based on a well-defined methodology to be able to manage all system requirements and behaviors.

Design and specification

The design process starts by giving the different functions and agents which are required, and the way they are structured; second, the designers define the behaviors of the various processes and entities, and the way they communicate; then, if they want to analyze the correctness of the design as soon as possible, an adequate approach is needed to represent, in an explicit way, the (full) system global behavior, in particular to be able to check potential unanticipated sub-behaviors.

To check the design correctness, it is essential to use a precise *model* of all critical mechanisms, functions, sub-systems, etc., and then, whenever possible, to use a *formal model*, to define a mathematical representation of the system. Checking the *correctness* validation of the design at this step is then conducted by checking the behavior of this system model.

Note that, after a given adequate sequence of more or less formal validation steps based on models, the system will be defined as 'fully designed' and will be implemented using adequate tools and languages.

Formal approaches have been used for many years for the verification of communication protocols. Two principal approaches have been used., i.e. basic formal models, such as automata, Petri nets, process algebras, etc., and formal description techniques for protocols, such as Estelle, LOTOS, SDL, etc.

This volume proposes and develops a design and validation methodology that relies on the use of a family of basic formal models that are rather easy to understand, and able to:

– describe the semantics of all basic building mechanisms;

– clearly specify the interconnection and communication semantics;

– unambiguously describe the resulting behaviors;

– validate the system during the first phases of its design by using support tools.

In general, basic non-language oriented graphical models, that do not include language-specific operators and statements, lead to the simplest solutions for representing basic mechanisms in a very abstract and integrated way.

For this reason too, this volume selected a basic, language-independent set of models to represent and manipulate the fundamental concepts of communicating architectures.

Selecting a *model*

Several models exist, and each model has particular characteristics, more or less relevant for a specific design. Consequently, the choice of the right model depends on the designed system and on the properties to be analyzed, as the model must be able to describe the design, and also to allow the designer to check the validity of the required properties.

In general, the designer must have a good understanding of the fundamental semantics of the system, i.e. of its basic building mechanisms. Thus, for simple architectures, modeling will be able to represent in a faithful way all details of the system. However, for complex systems, it will generally be impossible, for economic reasons, to represent the details of all existing functions, and it will become necessary to select and validate certain building blocks, i.e those most likely to lead to erroneous behaviors.

Of all existing models, *Petri nets* (PN) and their extensions are of undeniable fundamental interest, because they:

– provided the first modeling approaches for the semantics of concurrent systems, and were used to model the behaviors of the first parallel and distributed basic mechanisms;

– define easy graphic support for the representation and the understanding of these basic mechanisms and behaviors;

– prove to be, starting from state machines, an easy extension of previous approaches and handle, at the same time, the creation and the analysis of models;

– express very simply the main basic concepts in communication, including waiting and synchronization, and furthermore take into account their temporal and stochastic parameters;

– ensure, being unrelated to a particular implementation language, the independence of the specification with respect to its implementation.

Furthermore, many validation methods have been developed, using a great number of theoretical results and support tools, able to manipulate functional, temporal, and stochastic behaviors. Finally, models based on PNs will help us to understand, define and analyze the behavior of these systems, in the preliminary and first steps of their design.

For all these reasons, a set of Petri net models was selected in this book to represent and manipulate the fundamental concepts of communicating architectures.

Petri nets

PNs were introduced by A.C. Petri in 1962 to synchronize communicating automata, and were then extended to define a large set of models, with increasing complexity and capabilities.

As will be seen, this family of PNs, starting from the simple traditional state machines, now allows system designers to handle in an integrated way the functional (qualitative) and the non-functional (e.g. quantitative) temporal and stochastic capabilities of systems.

Extensions of PNs were proposed according to two important axes:

a) for *qualitative properties* and behaviors, to use simpler and more compact models, by high level PNs, for handling generic behaviors (e.g. individual) and data, predicates and functions;

b) complementing this first axis, for *quantitative properties* and behaviors, to extend the previous models by integrating quantitative constructs and parameters related to temporal and stochastic requirements.

It is significant to note that all first and conceptual studies in these quantitative fields were carried out using PN-based models.

Functional qualitative properties

The first PN model, called the Condition–Event PN, was based on the use of Boolean values: true or false. It was generalized by Place–Transition PNs, now simply called PNs, which can use integers. This volume will begin with their presentation and validation.

Non-functional quantitative properties

The fundamental contributions of the second axis considers:

– *time PNs*, or TPNs, used for systems whose behaviors depend explicitly on temporal values;

– *stochastic PNs*, or SPNs, for which distributions are attached to the model, in particular for performance evaluation and reliability.

Families of PNs

When applied to the modeling of systems, it rapidly becomes apparent that these models do not have the same application power, in terms of:

– definition and description of the concepts for parallelism, distribution, and synchronization;

– understanding and using the temporal and stochastic semantics;

– analyzing the possibly different mechanisms and behaviors, in very different contexts and applications.

Figure 1 represents some of the principal models of this family. In this figure, an arrow means that the model at the end of the arrow was proposed after the model at the beginning of the arrow, and so gives the steps followed by the research to propose and develop these principal PN-based models.

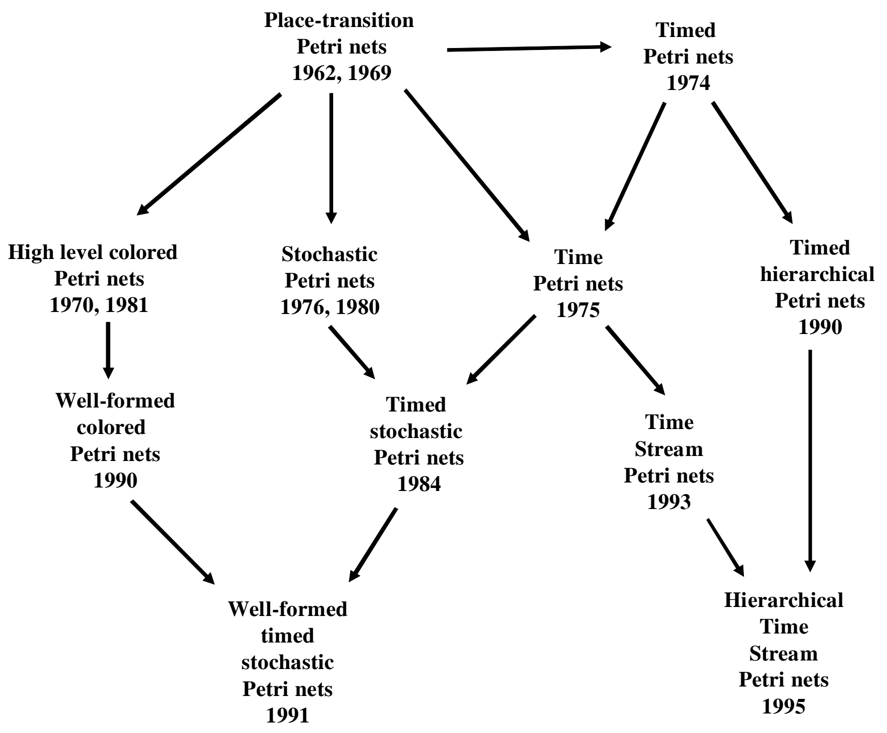
Place-transition Petri nets 1962, 1969
Timed Petri nets 1974
High level colored Petri nets 1970, 1981
Stochastic Petri nets 1976, 1980
Time Petri nets 1975
Timed hierarchical Petri nets 1990
Well-formed colored Petri nets 1990
Timed stochastic Petri nets 1984
Time Stream Petri nets 1993
Well-formed timed stochastic Petri nets 1991
Hierarchical Time Stream Petri nets 1995

Figure 1. *The main Petri net models*

Figure 2 gives a more conceptual view of these models, by clarifying their syntactic and semantic relationships. In this figure, three fields are respectively defined by:

– a discrete state semantics, for non-temporal and non-stochastic nets, behaviors being represented by a finite graph of all model states;

– a semantics on continuous time, for extended behaviors based on dense time models;

– and stochastic semantics, for behaviors including distributions.

Let us emphasize that these models have three models of reference, respectively PN, TPN, and SPN. Moreover, each model is a pure extension of a previous one, as it can by simplified to become a basic PN model.

As seen in the figure, the models derived from the reference models:

– *lead to more compact models, i.e. are* ***abbreviations***, that do not increase the expressiveness of the model, but simplify the model and the system specification;

– *or are more powerful in terms of expressive power*, i.e. are able to describe mechanisms which could not be described by the unextended models (e.g. introducing time parameters, stochastic distributions, etc., for real-time or dependable systems).

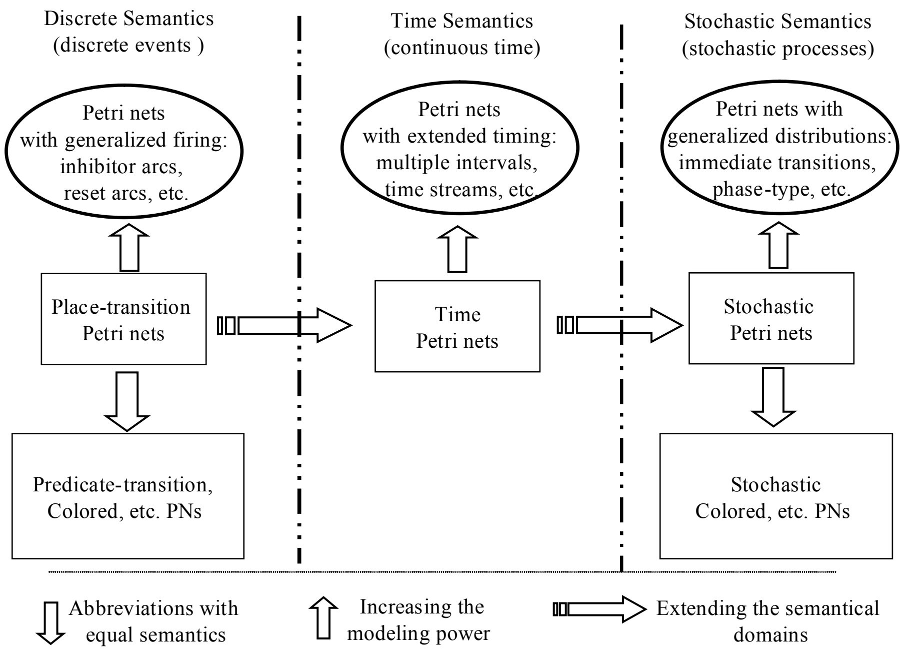
Discrete Semantics (discrete events)
Time Semantics (continuous time)
Stochastic Semantics (stochastic processes)
Petri nets with generalized firing: inhibitor arcs, reset arcs, etc.
Petri nets with extended timing: multiple intervals, time streams, etc.
Petri nets with generalized distributions: immediate transitions, phase-type, etc.
Place-transition Petri nets
Time Petri nets
Stochastic Petri nets
Predicate-transition, Colored, etc. PNs
Stochastic Colored, etc. PNs
Abbreviations with equal semantics
Increasing the modeling power
Extending the semantical domains

Figure 2. *Semantics domains of the Petri net-based models*

For example:

– PN led to PN with inhibitor arcs (to test the presence of zero token in one place), PN with reset arcs, etc.;

– TPN led to TPN with streams to compose and synchronize independent behaviors with independent temporal constraints, etc.;

– SPN led to SPN with immediate transitions in order to manage the case where transition cannot be delayed, etc.

Consequently, many different models exist, of different power and for different fields of application, but they follow the same semantics basis, and will allow the designers to carry out coherent complementary analyses to validate the correct operation of the modeled (and designed) systems.

The semantics of these models and their properties were used to select, define, and study the most important members of the PN family in the two parts of this volume.

Table of contents of the volume

Part 1 is dedicated to fundamental models and contains 11 chapters. Part 2 addresses verification and applications, and contains the last 7 chapters.

Part 1

Chapter 1 introduces Place–Transition PNs, more simply called Petri Nets (PNs). It gives their fundamental definitions, presents some basic models and clarifies their interest.

Chapter 2 illustrates an application in a very important area: communication protocols; simple PN examples show at the same time the power of the model and the interest of the formal analysis.

Chapter 3 first introduces the general properties that can be checked using PNs (blocking, reachability or accessibility, etc.) and the verification approach that uses the graph of the reachable (or accessible) states. The set of reachable (or accessible) states is the set of states that are reachable or accessible from a given initial state. Two optimization methods of analysis are then presented, one based on linear algebra techniques, and the other one exploiting the topological structure of the PNs.

Chapter 4 deals with the decidability and complexity problems related to checking these general properties.

Chapters 5 and 6 consider models and behaviors based on explicit values of time, and show how to model temporal mechanisms.

Chapter 5 presents the general model, Time PN or TPN, which associates a given interval (minimum, maximum) to each transition; this gives the first semantics for handling time and verifying temporal behaviors.

Chapter 6 presents a general model for composing temporal behaviors and systems. It gives the semantics of temporal composition by a new model, Time Stream PN, for composing autonomous (temporal) flows. It emphasizes their interests and applications for systems having independent temporal constraints, which sometimes interact.

Chapters 7 and 8 again consider PNs, i.e. non-temporal PN models, but define an abbreviation of a PN by a general model, which becomes able to represent, in a very compact way, a given set of similar parallel behaviors. The problems associated with this abbreviation are, on the one hand, to define a compact formalism and, on the other hand, to propose new validation techniques to handle this model, i.e. to avoid the obvious solution that consists of unfolding it into a very large PN.

Chapter 7 presents the main PN abbreviations, while concentrating on Colored PNs, which is the most frequently used model.

Chapter 8 gives one well-defined version of this formalism, Well-formed Colored PNs, which allows the development of efficient analysis techniques.

Chapters 9, 10 and 11 introduce distributions, which take into account probabilistic properties of systems. They introduce stochastic PNs, or SPNs, and define their semantics in terms of stochastic processes, and, for some classes of models, their relationships with Markov chains. The principal methods of analyzing SPNs are then presented. Chapter 9 introduces stochastic PNs.

Chapter 10 introduces well-formed SPNs by combining the formalisms presented in Chapter 7 (Well-formed Colored PNs) and Chapter 9. Modeling a multiprocessor architecture illustrates the expressivity of this formalism and its interest for performance evaluation. Chapter 11 develops a tensorial composition of classical and well-formed SPNs, showing that such a compositional approach reduces the complexity of the corresponding validation.

Part 2

The second part of this volume presents important advanced analysis techniques and finally gives some significant and illustrative case studies.

Chapters 12 and 13 address checking and verifying non-temporal behaviors. They present the main approaches that are based on building and manipulating the

Chapter 1

Basic Semantics

1.1. Automata or state machines

1.1.1. *Automata and state machine models*

The first models for numerical systems led to the definition of automata, or state machines. Automata or state machines are based on three fundamental assumptions, often implicitly given.

The first two assumptions are as follows:

– There exist, for the considered systems, a concept of global state, a set of these global states, and an explicit representation of these states (i.e. they can be precisely defined).

– There exists a global initial state, and the behavior (operational behavior) of the system starts from this global initial state:

- the behavior moves from the initial state by a set of transitions and goes to other global states, one per transition, called "next states",

- this behavior can be fully described by all the transitions that go from each global state to the next, one per transition, also called next states.

Such a transition between two states will take place when a given enabling event occurs during the evolution of the system.

Chapter written by Michel DIAZ.

The description of the transition (of the corresponding system behavior) will be represented in the model by an arc starting from a "before the event" state and going to a next "following the arrival of this event" state.

As a consequence, the full behavior of the model will be described by a global graph that represents the way the system operates, and defines all possible global states (represented by circles) and all possible transitions (represented by arcs) that exist between these states.

Note that this (behavioral) model is built step by step, starting from a state called "the initial state", a well-defined and specific state from which the behavior starts.

The two preceding assumptions are complemented by a very important one needed to construct the graph, the indivisibility of the transition between two states:

– when one event occurs in a given state and triggers a transition between two states, the transition has to be completed before another trigerring event can occur.

This means there is no state between the present state and the next state, as the system leaves the present state and reaches the next state indivisibly (the state reached when this transition occurs).

In general, automata and sequential machines are related to their environment by inputs and outputs: the evolution of the model depends on the values of the inputs. In particular, the transitions between two states depend on the values of the inputs. In each state, a value of the inputs can trigger or enable the execution of one transition (each input being able to trigger or enable one transition). The outputs are produced either in a state or during a transition (i.e. a pair state/input or a pair arc/output).

As soon as one input in a given state enables a transition, the transition is executed: its execution starts from the *present state* and leads to the *next state*, and produces new values for the outputs.

In this model, the assumption of indivisibility implies:

– first, that the transition is executed when the significative input of the automata enables the transition; and

– second, that the next state has to be reached before a new input enables a transition in the next state (of the automata or state machine).

Indivisibility with inputs and outputs means that a transition and its outputs (the actions coming from this transition) must be completed before reaching the next state, i.e. before the arrival of an event that can enable one of the transitions starting from the next state (the state newly reached).

Thus, a global behavior of the model can be defined by considering, one after the other, the set of the inputs, transitions, and outputs that define the system execution.

Furthermore, note that the assumption of indivisibility implies that when complex actions (e.g. related to many outputs or to computations) are associated with a transition, these actions must be completed before reaching, and thus defining, the next state.

As a consequence, whatever the actions are, only two global states exist:

– one before the transition, i.e. the starting global state — having for values all values existing at the instant before the transition is executed;

– one after the execution of the transition, the next global state; it has for values the new values of the automata or state machine, i.e. the values either left unchanged by the transition, or modified by the complete execution of all actions associated with the transition.

Of course, for these models to represent behaviors correctly, the real behavior of the modeled system must satisfy the assumption of indivisibility, i.e. the behavior of the system must fulfill the indivisibility assumption, to be coherent with the behavior of the model.

Thus, modeling must represent the real indivisibility that exists in systems. Conversely, if some sub-behaviors are not indivisible, they cannot be represented by only one transition, and must be represented by a set of transitions, each of which represents the various indivisible sub-behaviors.

1.1.2. *Tasks and processes*

Of course, a program or a process can be represented by a state machine:

– the *initial state* is given by the value of the program counter and of the program variables immediately after their initialization;

– the *execution* of a transition is defined by the set of actions that is the execution of the program instruction or intructions associated with this transition;

– the execution leads to a next state that includes the new values of the program counter and of all program variables that have been modified by this transition.

The assumption of indivisibility can make modeling difficult, and the model must be built carefully: modeled transitions must indeed be indivisible in the real system for the model to represent the real behavior.

Again, any divisible behavior of the system must be broken up into indivisible sub-behaviors, and each of these indivisible sub-behaviors can be represented by a transition. For example, some instructions can be suspended, and their model may have to account for them, depending on the level of modeling, by breaking them up into indivisible subinstructions.

1.1.3. *Some models*

Let us consider the partial and full state machines given in Figure 1.1a, composed of circles for the states and of arcs for the transitions between the states.

Each transition has only one starting state, and only one following state.

Let us suppose that to mark the initial state at the initial instant a ***token*** is drawn in the state (note that sometimes the initial state is marked by an arrow entering it and coming from no other state).

When a transition is executed, after being enabled by an event, the fact of going from the present state to the next state will be called ***executing*** or ***firing*** a transition, and this firing can be represented graphically by passing the token from the present (starting) state to the next (following) state. Note that,for a transition to be firable the token must be in the place which is at the "input of the transition".

A token always exists in the graph, and indicates the present state of the behavior (of the automata or of the state machine), and each state represents a global state of the model (and of the modeled behavior), i.e. a global sequential activity.

In this figure, the notation "A; B" means that "A" is an input and that "B" is an ouput.

From what has been said before, this means that when the token is in the input state of the transition, the arrival of the input event "A" causes the execution of the transition from the present state to the next state, and the firing of this transition produces the output action "B".

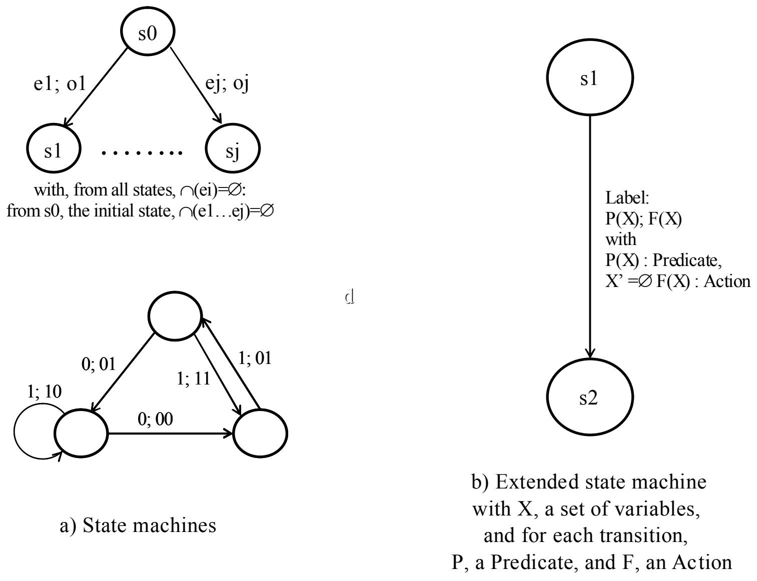
s0
e1; o1
ej; oj
s1
sj
with, from all states, ∩(ei)=∅:
from s0, the initial state, ∩(e1…ej)=∅
0; 01
1; 10
1; 11
1; 01
0; 00
a) State machines
s1
Label:
P(X); F(X)
with
P(X) : Predicate,
X' =∅ F(X) : Action
s2
b) Extended state machine
with X, a set of variables,
and for each transition,
P, a Predicate, and F, an Action

Figure 1.1. *Simple and extended state machines. a) State machines with: ∩ (ei) = ∅, and s0 being the initial state; b) extended state machines with variables X (with indivisible semantics), Predicate: P(X); Action: X' ← F(X)*

Figure 1.1a represents first the general case of a state having a set of next states, each one being enabled by an input ei, and the firing of the transition producing the output oi; second, it gives a very simple automaton, having three states, with one input (0 or 1), and two outputs, also binary. Note that in one state the transition enabled by (the input value) 0 is not given, i.e. it is not specified; the next state is the same state.

Figure 1.1b models a complex transition of an extended state machine with a logical condition, e.g. a step of a program, having the following behavior:

– If, when s1 is marked, predicate P(X) is true, then the transition is enabled, it can fire, and, when it fires, the program progresses towards its next state, s2. Firing produces the output action, here executing the procedure F(X), producing X', the new set of program variables (the new data values). Let us note that P represents the enabling condition of evolution (of firing) and F gives the action of firing the transition for the program variables (new values).

1.2. State machines and Petri nets (PN)

1.2.1. *Composing state machines*

Let us consider an extension of the concept of *transition*, given in Figure 1.2. Such a transition, t_i, will be denoted by a bar or by a rectangle *and can have several input (ingoing) arcs and several output (outgoing) arcs at the same time*: such a transition represents the basic transition of a PN.

Note that the circles do not represent global states, but local state: they are called the ***places*** *of the PN.* The tokens in the places also represent local information.

We will see that such a simple extension proves to be particularly powerful for expressing and analyzing parallel and distributed behaviors.

The two main consequences of such a transition are:

– several tokens can exist in the model at the same time (for instance, one per automaton in the simple case of composing basic automata);

– there are no more explicit global states, and the global state of the system is now the set of all places and tokens (partial states): the set of all these places (and in particular the ones with tokens) constitutes the global state of the PN.

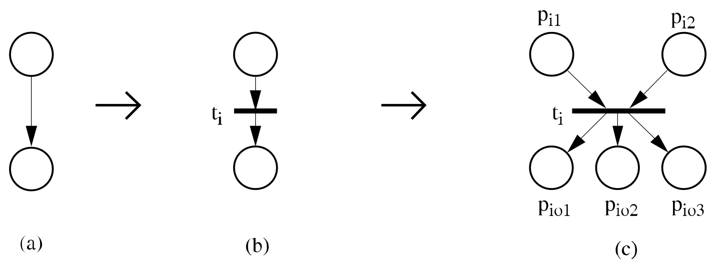

Figure 1.2. *From state machines to PN: a) arc of a state machine; b) arc of the simplest Petri net, a state machine; c) transition in a Petri net (as seen later, with weight 1)*

As a consequence, PNs will be graphically represented by a graph having two types of nodes — transitions and places — these nodes being connected between them by *arcs* from places to transitions and from transitions to places. Note that the arcs never connect two similar nodes.

Places having arcs which connect them to a transition T, i.e having arcs connecting these places to T, will often simply be called the input places of T; similarly, the places connected to a transition by arcs going from T to these places will be called output places of the transition.

Input transitions and output transitions of a place are defined by the same method.

The global state of a PN can be defined by the set of its places, some of them being marked, i.e. having one or, more generally, several tokens, and some others not being marked, i.e. having zero tokens. Each of these places becomes a component of the global state, and is a substate or a partial state of the global state of the system.

The corresponding token distribution in all places is called the ***marking*** *of a PN.*

In other words, a PN is a model defined by:

– a set of places, denoted graphically by circles;

– a set of transitions, denoted graphically by bars or rectangles;

– a set of arcs, denoted by arrows, joining places to transitions, and transitions to places; and

– by a distribution of tokens in the places.

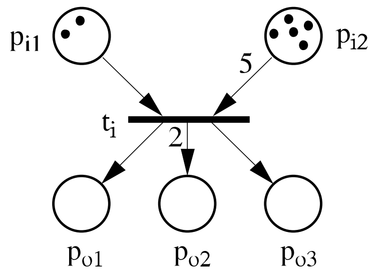

Figure 1.3. *PN having weights different from 1 on the arcs; firing is indivisible*

Places are denoted by pi or p_i or p and transitions by ti or t_i or T, according to the context.

Furthermore, weights defined by integers can be associated with the arcs.

By convention, when nothing is attached to the arc, an arc will have a weight of value 1; more generally, the weight of an arc can be an integer higher than 1, and its value will be explicitly indicated on the corresponding arc.

Figure 1.3 represents such an extended basic PN, consisting of a transition t_i and of five places: two input places, and three output places.

In Figure 1.11a the condition that specifies waiting, for the two tasks, is the existence of a token in the ExclMut place. This token, when present in the place, means that the resource is free, allows one task to fire its Alloc-S* transition, and thus to enter the critical section: the marking of the places SC* (SCa or SCb) indicates when a task is in the critical section.

The ExclMut resource is released and made available again when the task leaves the critical section by the firing of la-SC or lb-SC. Note that exclusion follows from the presence of one and only one token in place ExclMut.

The general mechanism of mutual exclusion can be represented in a very elegant way by the PN of Figure 1.11b, with three main places, ExclMut, ATT and SC, and three transitions, Ri-SC (request for entering the critical section by task Ti), Alloci-SC (for Ti to enter the critical section) and li-SC (the critical section is released by Ti).

Each task fires Ri-SC to request access to the critical section, and thus as many tokens as the number of waiting processes are in ATT (this number represents the number of processes that need the critical section and were not allowed to enter it).

Moreover, these tokens are identical, which means that all of them will be treated in an equivalent way: we do not know specifically which process will be selected, or when Alloci-SC will fire, to enter the critical section: the choice is non-deterministic. As the tokens are equivalent, one of them will be chosen and removed from place ATT when firing Alloci-SC.

We will later consider a model to characterize each of the tokens (by coloring the tokens), for instance to identify each of the processes.

1.5.4. *The reader and writer mechanisms*

Readers and writers need to be synchronized for updating their common data. The corresponding fundamental mechanism is defined by the three following conditions:

– the writing processes must operate in mutual exclusion, because their procedures can modify part of the data;

– the reader and writer processes are also in mutual exclusion, because the readers must read a coherent set of data (in general, data cannot be modified in the middle of a reading);

– the readers can read in parallel, because the procedures of reading do not modify the data and thus can run freely (in parallel, without exclusion).

Figure 1.12 gives a very subtle, integrated, and clever model of the reader and writer synchronization mechanism.

In the figure, the marking of place RES with the integer value *N*, defines the number of readers that can read in parallel. Note that, by definition, the maximum number of possible parallel readers has to be given, but this value can be as large as necessary (but finite), which ensures the generality of the specification.

A reader is authorized to read when Alloc-lec is fired, and this action removes a token from RES and adds it to LEC: Alloc-lec can be fired N times, which means that N readers can read in parallel.

Then, when at least one reader reads, at least one token has been removed from RES, and the marking of RES is strictly lower than N: the transition authorizing the writings, Alloc-ecr, cannot be fired (because its firing requires N tokens in RES, the arc joining this place to the transition has a weight of N), and this ensures the exclusion between the readers and the writers (when one or more readers are reading).

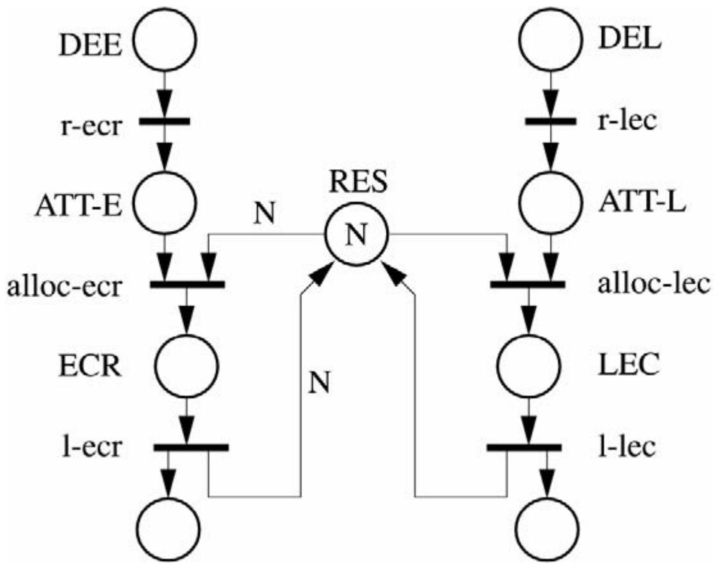

Figure 1.12. *Readers and writers*

Now, if there is no reading in progress, then Alloc-ecr can be fired and the *N* tokens are removed from RES, and RES = 0: transition Alloc-lec becomes unfirable, preventing the readers from reading and enforcing mutual exclusion between readers and writers; in the same way transition Alloc-ecr can no longer be fired as RES = 0, which ensures the mutual exclusion of the writers between themselves.

Let us note that readers and writers make the requests when places DEL and DEE are marked, by the requests r-lec and r-ecr, then waiting for the authorizations in ATT-L and ATT-E.

Finally, note that when (exclusive) writing is completed, firing l-ecr gives back N tokens in RES, which re-initialize the reader and writer mechanisms.

Let us emphasize the compactness and the power of this model, which gives a very high level and very easily understood specification.

As we saw earlier for mutual exclusion, note that there is no priority given to the readers or to the writers (to places ATT-L and ATT-E). For example, if several processes of the two classes are waiting for authorization to enter the critical exclusion section at the same time, one of them will be selected in a non-deterministic way (we will see in Chapter 7 that it is possible to express token identity and conditions using colored PNs).

1.5.5. *Bounded buffers*

A model of the bounded buffer is given in Figure 1.13. In its initial state, the producer produces its information (e.g. a message), when firing transition prod, and then can store the information in the buffer by firing transition deposit. This pair of actions can be repeated up to a maximum of N times, i.e. as long as there are tokens in place FREE: the marking of place FREE is then equal to the maximum length of the buffer, or the number of possible writings (N in the initial state). Then, consumer Tb has the authorization to read one piece of information (e.g. a message) as long as there is at least one token in OCC, i.e. as long as there is at least one piece of information stored in the buffer: consumption is allowed as long as place OCC has at least one token. When all information has been consumed, or at the initial state when there is no information stored, OCC = 0 and the action “consume” is impossible.

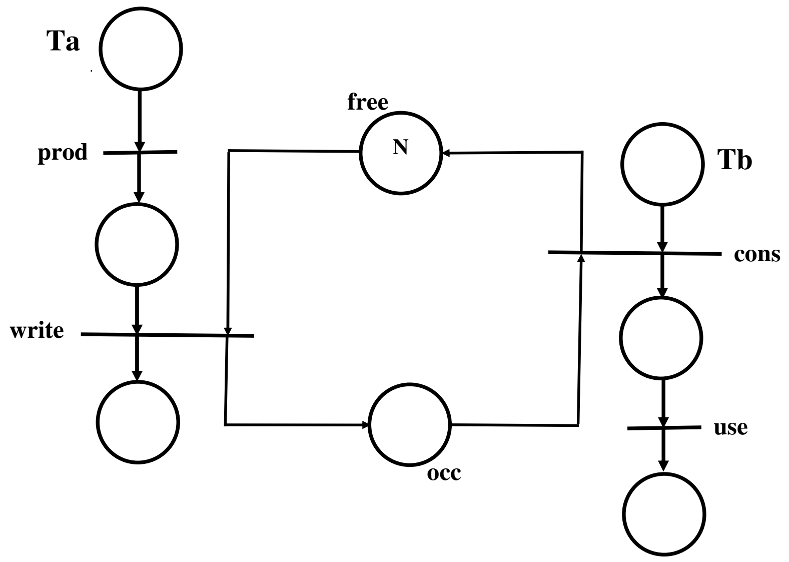

Figure 1.13. *A bounded buffer*

1.6. Conclusion

This chapter introduced place-transition PNs, the basic PN model. Fundamental and very elegant examples of how to model the specification of important synchronization and communication mechanisms were also given.

Let us emphasize that proposing a specification model is extremely easy, but defining a “good” model, i.e. a simple, clear and easily understood model, able to specify in a very concise way the considered mechanism, often proves to be extremely difficult.

Moreover, after having developed many models, it will be understood that the ones described in this chapter came after many proposals, and they show a powerful conceptualization of the corresponding mechanisms.

In the next chapter, before formally studying the properties of PNs, we will consider some more illustrative examples in communications protocols, an important field of application.

Chapter 2

Application of Petri Nets to Communication Protocols

2.1. Basic models

Communications protocols define the set of rules of various degrees of complexity that are needed to exchange messages between two or several communicating entities. Because of their importance in distributed systems, this chapter presents two basic reference examples for modeling and analyzing protocols.

Let us first consider a simple exchange of messages between two entities. In this case, the global communication model must define, in a precise way:

– the behaviors of the two entities (e.g. programs) that exchange messages;

– the *exchange semantics* between the sending and the reception of each message.

Figure 2.1, where "!" and "?" respectively denote the sending and the reception of a message, considers two processes, P1 and P2, that need to send and receive a message (or more simply a signal) E. This figure presents the two actions in the processes P1 and P2, and the three simplest solutions that can be used to model the exchange between the two entities, while passing information or not, i.e.:

– (a) *transition merging*, i.e. merging the sending and receiving transitions, this model being symmetric;

Chapter written by Michel DIAZ.

– (b) using a *shared place*, now an asymmetric model; and

– (c) *acknowlegdement* communication, also asymmetric.

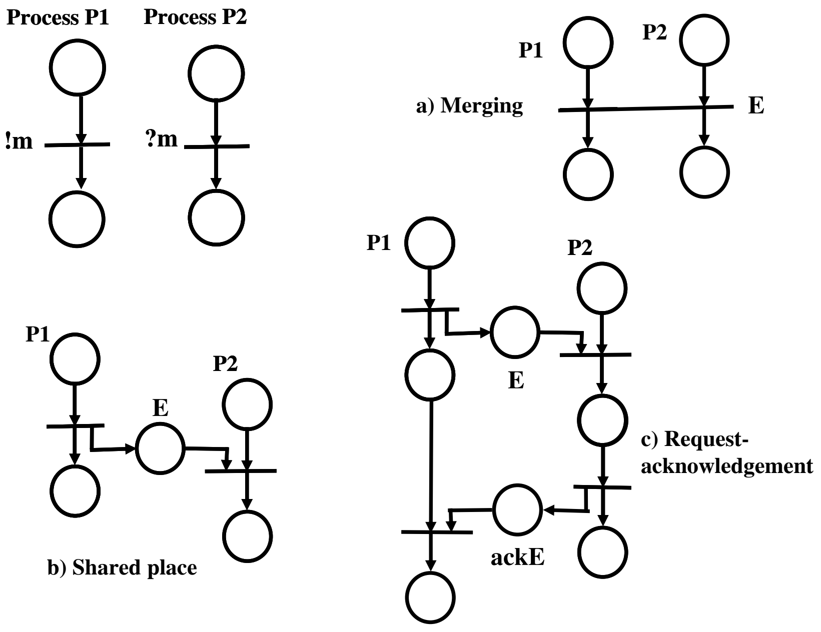

Figure 2.1. *Interconnections of automata and PN, with process P1 and process P2: a) merging; b) shared place; c) request acknowlegdement*

Transition merging (Figure 2.1a) is a model that defines a strong synchronization mechanism between two entities, because the behaviors of each of these entities must be in the same (precise) state, i.e. waiting, before exchanging the message. Note that at the instant of this synchronization, a passage of values can take place, but it is not represented in the model given in Figure 2.1a; if we want to represent it, its model must be added explicitly (using the model given in Chapter 7).

The communication model using a shared place (Figure 2.1b), explicitly shows two actions: the sending of a message by an entity and the reception of this message by the other. When the shared place E is marked, this means that the transmitted message has been sent, but has has not yet been received, i.e. is in transit in the medium or in the communication network. Also, note that the message can be without data content as in the given model, and the passage of values has to be added if needed.

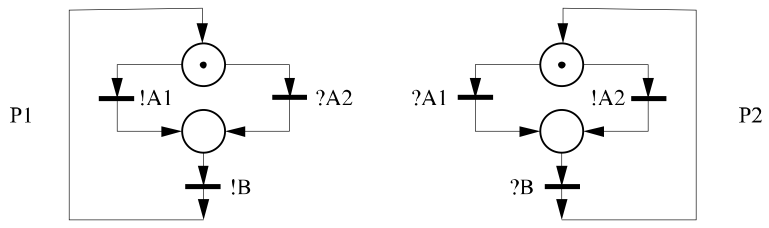

Figure 2.2. *A simple example of communication connection establishment*

Finally, request–acknowlegdement (Figure 2.1c) explicitely expresses that the transmitter, before continuing its processing, has to wait for an acknowlegdement (sent by the receiver) that acks the message reception. By extension, sending a signal or information has to be followed by the reception of an acknowledgment (again, the representation of data is not given).

2.2. A simple establishment of a connection

In order to illustrate the interest of these models and of their properties, let us consider Figure 2.2, which represents a simple protocol for *connection management*: process P1 on the left can open a connection by sending message A1, and process P2 can also open the connection by sending message A2; nevertheless, only P1 has the right to close the connection, by sending message B.

2.2.1. *Different global semantics*

In order to analyze the behavior of these two simple architectures, we must connect P1 and P2 and define their interactions. This means that we must specify the semantics of the exchanges, i.e. the semantics which precisely and explicitly express the way a sending transition is connected to its corresponding receiving transition.

For simplification, let us assume in this example that the communication or *interaction semantics* are the same for all the send–receive transition couples (note that this assumption is not true in the most general case, in which each couple can have a particular semantics, but it is quite often used in real systems).

Now let us consider, for Figure 2.2, the three possibilities given in Figure 2.1, i.e. transition merging, shared place and request–acknowlegdement. Then, Figures 2.3a, 2.3b and 2.3c present the corresponding three global models, i.e. the models obtained by replacing the three send–receive pairs in Figure 2.2 by the three models in Figure 2.1. It clearly appears that the corresponding global PNs are different, so they may have different behaviors.

Figure 2.4a gives the reachability graph *G* of Figure 2.3a, and Figures 2.4b and 2.4c give a subset of the *G* graph in Figures 2.3b and 2.3c. It is interesting to note that:

– *G* in Figure 2.4a behaves correctly;

– *G* in Figure 2.4b shows that places A1 and B can receive an arbitrarily large number of tokens, which are generated by the firable sequence, which can be infinite: !A1; !B; !A1; !B; !A1; !B; !A1; …. As all real implementations are finite, this behavior is likely to go beyond the allocated real resources, and it is said that it is not *bounded*, so incorrect;

– *G* in Figure 2.4c contains a behavior that, by firing (in any order) !A1 and !A2, leads to the marking in which the places AT1, A1, AT2 and A2 are marked. In this (global) marking, no transition is firable; in fact, the PN has now no firable transition, and thus the model (and the modeled system) is fully blocked, and said to be in *deadlock*. Of course, such a behavior is incorrect and has to be checked and avoided.

2.2.2. *Conclusion*

Two quite important conclusions can be derived from this example.

1. For such a very simple protocol, according to the communication models, the resulting global analysis leads to three different graphs (and behaviors), that show very different properties, defined later as:

(a) having a limited (*bounded*) number of tokens in each place (later called boundedness) and where all transitions can always be fired (later called liveness); when the maximum number of tokens in a place is equal to 1, the PN will be said to be *safe*;

(b) being not bounded (able to have an unlimited number of tokens) and *live*;

(c) bounded and with some transitions no longer firable (called not live), and furthermore in this example transition can no longer be fired (called in deadlock).

Let us emphasize that, as the last two cases can lead to potential and real implementation problems, the knowledge resulting from defining and analyzing the full behavior shows why there is interest in global models.

Chapter 6

Temporal Composition and Time Stream Petri Nets

6.1. Time, synchronization and autonomous behaviors

Petri nets (PNs) are used to specify basic functional behaviors, in particular synchronization, waiting and sequence, and transitions and places are fully or partially ordered, but without using explicit temporal values.

Furthermore, time Petri nets (TPNs) have been defined for systems whose behavior depends on real and explicit values of time. For example, for expressing the duration of actions, or some relationships between some occurrences of actions, temporal parameters are needed. It has also been shown that TPNs allow the designer to specify a great number of these problems, in particular for the durations of actions and for time-outs.

Nevertheless, it appears that TPNs are not fully adapted to the specification of some systems, in particular the ones related to *multimedia systems*, because they contain some complex events that have inherently sophisticated temporal behaviors.

More precisely, these advanced systems possess complex objects, which are defined by a well-defined *autonomous temporal behavior*. For example, voice, music, and video are such objects. Their behavior is made up of simple presentation

Chapter written by Michel DIAZ and Patrick SÉNAC.

actions, i.e. sending a voice, music, or video sample (an image, or frame, in this latter case) to some appropriate peripheral devices. To be correct, these presentations must follow a very well-defined temporal schedule, e.g. the presentation of 25 frames per second for a video flow. In complex multimedia systems, the behaviors of the different flows may also need to be co-ordinated, for example by synchronizing voice and video for lip-synchronization in movie-like presentations.

These new objects will be called “streams”, in order to emphasize their temporal behaviors, which causes them to behave in a way similar to a stream.

6.2. Limitation of time PNs

In TPNs, firing a transition is defined by fulfilling the three following conditions:

(i) the considered transition must be enabled;

(ii) counting the time starts at the instant at which the transition is enabled;

(iii) firing takes place in the interval [$q_{\min}$, $q_{\max}$], when the transition is continuously enabled during the interval (between these two values).

Let us now take the example of a transition having two input places. By analyzing its behavior in depth, it appears that the firing condition of the TPN implies a subtle constraint, with has to be well understood: *the behavior of the two input places are not temporally autonomous and neither of them has independent temporal behavior.*

This observation comes from the formal definition of firings in TPNs:

– firing starts by waiting for the transition to be enabled, that is waits for a non-temporal synchronization of the token in the two input places (as in a PN);

– and then, and only then, when the transition is enabled, the semantics start the timer, and as this counting is related to the temporal interval associated with the transition, the timer applies to all input places: it follows that the behaviors of the tokens are not independent.

6.3. Temporal composition

As there is no temporally independent synchronization in TPNs, firing can be decomposed as follows:

– firstly, enabling is a logical synchronization, a logical composition defined by T, without any reference to time;

– and then, secondly, the temporal behavior starts, i.e. waiting during a certain amount of time, but this is done in an identical way for all input places of T, and the tokens will be deleted from all input places by the firing, and at the same time.

As a consequence:

– the tokens do not have an autonomous behavior and they cannot model temporally independent behaviors;

– moreover, these behaviors cannot of course be composed while keeping their particular temporal characteristics.

This modeling and composition problem was solved by introducing time stream PNs in [DIA 93, DIA 94, SEN 95, SEN 96].

Let us show how TPNs can lead to temporal composition.

6.4. Temporal composition and temporal synchronization

6.4.1. *The semantics of "waiting"*

The unit of time is global in TPNs, but counting time is local to a transition when it is enabled, because the implicit (waiting) timer related to the firing is started when the transition is enabled.

In particular, this implies that the waiting time is not related to the behaviors of each token (tasks or processes) but to a waiting of all of them occurring *after* their logical synchronization (Figure 6.1a).

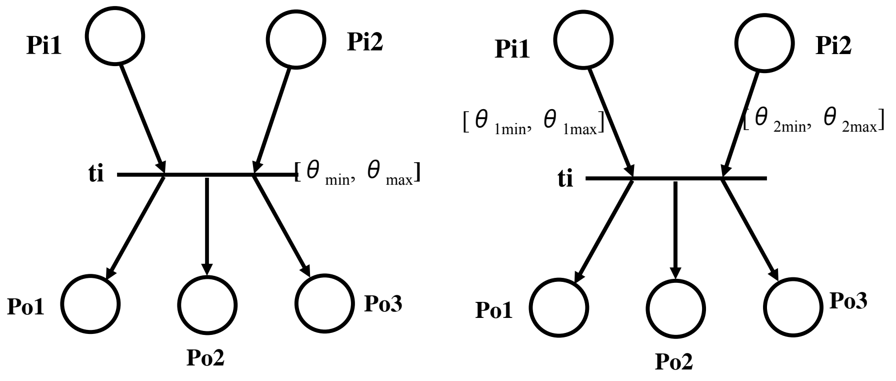

Figure 6.1. *Intervals in TPNs and STPNs*

Chapter 7

High Level Petri Nets

7.1. Introduction

Place/transition nets define a simple and powerful theoretical framework for studying concurrency problems. Moreover, they are supported by a large number of available tools for specifying and analyzing concurrent systems in large application domains. However, this formalism emphasizes control while overlooking data structures: it is easy to model and analyze control problems but difficult to deal with large numbers of objects belonging to many classes and to integrate definition and manipulation in the specification. Moreover, for large nets it is difficult to take advantage of the symmetries of objects inside each class to synthesize the analysis of their behaviors. To cope with this, without modifying the semantics of the Petri net, several abbreviations have been proposed. These high level formalisms attach new information to nodes, arcs and tokens, to obtain new dense behavioral semantics using these parameters, but a key point is to remain able to "unfold" these abbreviations to obtain large classical Petri nets showing the same behaviors. Indeed better efficiency is obtained for high level formalisms which allow direct analysis (avoiding unfolding the net) and even a parametrizable one. Moreover, they enable object symmetries to be well exploited.

These formalisms differ by a greater or lesser degree of natural syntax and by parametrization modes. In particular, we may cite colored nets [JEN 91], predicate/transition nets [GEN 81], algebraic nets [REI 87, REI 91], well-formed nets [CHI 91, CHI 93], and object-oriented nets [LAK 02]. Well-formed nets have led to symmetric nets, which are now a standard (IS/IEC 15909 [HIL 06]). Many tools

Chapter written by Claude GIRAULT and Jean-François PRADAT-PEYRE.

are now available for them, such as CPN-AMI [KOR 99], Design/CPN [KRI 98], GreatSPN [CHI 95] or Prod [VAR 97].

In this chapter, we focus on colored nets and well-formed nets (WN), which both have the same power of expression:

– Colored nets are characterized by a simple functional semantics: colored information is associated with tokens and firings, while the values of arcs are functions of colors. These functions specify the number and the colors of the tokens that are consumed and produced by each particular transition firing, with the convenient choice of its color parameters. Since there is no syntactical constraint on these functions, modeling is simplified but direct analysis techniques are difficult or impossible to construct.

– Well-formed nets are a functional and parametrized extension of Petri nets where the functions of colors are restricted to compositions of a few elementary functions (*identity, successor* and *diffusion*). Moreover, the color values must be tuples of values from basic sets called classes. The benefits of this limited syntax are the parametrized extension of classical analysis techniques (such as invariant calculus and structural reductions) and also the development of more general approaches (such as the symbolic reachability graph).

After an informal introduction (section 7.2) presenting several examples of high level nets, we define and illustrate colored nets (section 7.3), then well-formed nets (section 7.4), for which analysis techniques will be presented in Chapter 8. We finally look at two higher level formalisms: interpreted Petri nets and algebraic nets (section 7.5).

7.2. Informal introduction to high level nets

For a simple client-server system, its place/transition model roughly describes the exchanges between processes but fails to study their individual behavior. This drawback has motivated the definition of colored nets, which attach extra information to the tokens, nodes and arcs. With the same underlying graph of the net, colored models of the system are able to distinguish the behaviors of clients and servers. Furthermore, it is possible to systematically expand a colored net to reconstruct an equivalent (unabbreviated) place/transition net. A model of the alternate bit protocol shows the ability of colored nets to concisely describe the management of data structures.

7.2.1. *A client-server model*

The following place/transition net motivates colored nets by showing some difficulties encountered by designers in describing the behaviors of multiple system entities. Net R1 (Figure 7.1) models two clients connected to one server, but it may be parametrized to more clients and also generalized to more servers by adding tokens.

Chapter 8

Analysis of High Level Petri Nets

8.1. Introduction

Of the formalisms derived from Petri nets, colored nets and well-formed nets are most used for modeling complex systems. Although they enjoy greater modeling power, they are defined as abbreviations of Petri nets, therefore it would be possible to reduce their analysis to the analysis of the place/transition nets obtained by unfolding them.

However, this method comes up against the problems of the size of the unfolded net, the combinatorial explosion of its reachability graph, and the difficulty of interpreting the results obtained. This greatly complicates the analysis of the obtained results and, moreover, deprives the analysis of the information introduced by the designer concerning the symmetries occurring between the behaviors of the problem objects.

Much research [GIR 01, DES 04] has been done on direct analysis techniques for high level nets. In this chapter we study three of them for colored nets and well-formed nets:

– The first one, presented in section 8.2, is the construction of the symbolic reachability graph. It specifically applies to well-formed nets and fully exploits the symmetries of the model to construct a very compact type of reachability graph: each of its nodes represents a set of ordinary markings involved in an equivalence relation based on these symmetries.

Chapter written by Claude GIRAULT and Jean-François PRADAT-PEYRE.

– Section 8.3 presents the computation of generating families of invariants for colored and well-formed nets. The main difficulty in comparison with the place/transition nets comes from the management of systems of equations whose coefficients are linear applications instead of simple integers. We give a general algorithm for obtaining generating families, non-parametrized for colored nets and parametrized for regular well-formed nets. We also give some insight into the computation of generating families of flows with positive coefficients.

– Section 8.4 ends this chapter with a presentation of structural reduction techniques for colored and well-formed nets. Indeed, it is very valuable to apply these reductions before other analysis techniques because they simplify the model while preserving the set of basic properties (boundedness, home states, liveness, etc.). In the chapter on verification of temporal logic formulae it will be shown that under simple conditions these reductions also preserve LTL formulae.

Most of these techniques have been implemented in analysis tools, including CPN-AMI [KOR 99] and GreatSPN [CHI 95].

8.2. The symbolic reachability graph

The symbolic reachability graph aims to reduce the size of the classical reachability graph by taking advantage of the symmetries among modeled system objects by gathering some "equivalent" markings into *symbolic markings* and by using a symbolic firing rule compatible with the classical firing rule. Therefore, this symbolic reachability graph (SRG) is a dense but equivalent representation of the classical graph allowing direct and efficient checking of the model properties, even on very large specifications.

This representation relies on a data structure and a method [CHI 93]:

– A *symbolic marking* represents a set of equivalent markings w.r.t. some color permutations preserving the model symmetries;

– the *symbolic firing rule* defines a firing relation in the SRG adequately representing a set of classical firings.

We illustrate the building of the SRG for a simple well-formed net (see Figure 8.1) of a peer to peer network. This net is derived from the client–server model already presented in Chapter 7 (see Figure 7.5), but now any site can send requests to any other, which creates a deadlock risk as we will verify by building the SRG.

Each of the n sites may play both the role of a client sending message requests (transition $csend$) to a chosen server and of a server receiving a request (transition $srec$) from a client before sending its answer (transition $ssend$) to the calling client which receives it (transition $crec$). To avoid a site sending a request to itself, a guard $[xy]$ is added to the transition $csend$. This guard is a condition which restricts the

Chapter 18

Performance Evaluation in Manufacturing Systems

18.1. Introduction

The increasing complexity of production systems and their control, as well as the increasingly high costs of their components (machines with numerical control, automated guided vehicles, automated stores, robots, etc.) oblige companies to optimize their systems during their design (installing a new system or modifying existing structures), but also during their exploitation (minimizing work-in-progress, determination of dynamic rules, etc.). To create or modify a production system involves high costs for a company. If the manufacturing system is undersized, the company will not be able to achieve its goals. This system will therefore have to be modified, which will result in significant over-costs. If the manufacturing system is oversized, the company would have spent its capital unnecessarily. Thus to create or modify a manufacturing system, it is important to initially define the various performance indicators which have to be reached. Next, based on the experience of industrial experts, a model of the system is established by achieving as well as possible the performance indicators previously defined. Then the performance of this model will be evaluated before analyzing the obtained results. This phase of analysis makes it possible to modify or adjust the model so that it can fulfill as far as possible the desired performance requirements. To find the optimal manufacturing system (i.e. the system best meeting, with lowest costs, the desired performance indicators) thus involves loops of the phases of modeling and performance evaluation.

Chapter written by Isabel DEMONGODIN, Nathalie SAUER and Laurent TRUFFET.

The first part of this chapter is devoted to modeling of manufacturing systems using the formalism of Petri nets (PN), which are powerful tools enabling the specification, modeling, evaluation and management of dynamic systems. Indeed, this formalism has many analytical properties that often allow a simple evaluation of the qualitative and quantitative properties of the considered manufacturing system during the design period but also in the exploitation phase [DES 94, PRO 94, SIL 97, VAL 97]. Based on transition-timed PNs, the modeling and analysis of cyclic production systems (i.e. repetitive behavior), where operational times are deterministic or stochastic, are dealt with in this chapter. After having defined the different characteristics of manufacturing systems with cyclic behavior, the various stages of modeling of flexible manufacturing systems (job-shops) and of just-in-time systems (or kanban systems), are described. Then, high throughput (i.e. high speed) production systems are studied, which generally require tools from the field of hybrid dynamic systems. Next, some temporal properties of these systems are presented, in term of cycle time, permanent state, etc. Complementing Chapter 5, the available analysis techniques are dedicated to timed marked graphs (T-timed marked graphs), for which a deterministic or stochastic firing delay is associated with each transition (see Chapter 5). These techniques are based on the analysis of the graph structure, and not on the evolution of the model. Three classes of marked graphs are studied: discrete-deterministic, discrete-stochastic and continuous-deterministic. Finally, the optimization of discrete models (deterministic or stochastic) will be discussed. More precisely, the problem of marking optimization for structures such as marked graphs is investigated. For a manufacturing system, this problem consists of maximizing productivity using the smallest possible number of transport resources (carriages, pallets, stocks, etc.).

18.2. Modeling of manufacturing systems

The various machines (or resources) of a manufacturing system, through which the product moves, constitute the manufacturing process and carry out operations that can be classified in four categories:

– Transformation operations, which modify the physical status of the product. They operate on only one product and finish by releasing the transformed product (example: filling of a bottle in a conditioning line).

– Operations of modification of the structure, which combine some products to form a new product (example: creation of a pack composed by several basic products).

– Information operations which control, for example, the quality of the product.

– Transfer operations, which modify the position of the product in space without changing its physical status or structure. These operations can be performed by