GRAFCET and Petri Nets in Manufacturing

Alessandro Giua,

Dip. di Ingegneria Elettrica ed Elettronica, Università di Cagliari,

Piazza d'Armi, 09123 Cagliari, Italy

Phone: +39-070-675-5892 – Fax: +39-070-675-5900 – Email: giua@diee.unica.it

Frank DiCesare

Dept. of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute

Troy NY 12180-3590, USA – Email: dicesare@ecse.rpi.edu

Abstract

Petri nets and GRAFCET are two complementary models for discrete event systems. Petri net models are suitable to represent the states of a system and its evolution. GRAFCET has been especially designed to specify the desired input/output behaviour of software control systems. In this paper we compare them in regard to the modeling and control of manufacturing systems.

Published as:

A. Giua, F. DiCesare, "Grafcet and Petri Nets in Manufacturing", in *Intelligent Manfacturing: Programming Environments for CIM*, W.A. Gruver and J.C. Boudreaux (Eds.), pp. 153–76, Springer-Verlag, 1993.

1 Introduction

The application of control theory into the domains of manufacturing, robotics, computer and communication networks, and so on, has led to the creation of a new discipline, known as *discrete event systems* (DES) [1, 2].

Manufacturing systems show several characteristics that do not pertain to the systems considered by traditional control theory:

- *Discrete state space.* The states, that are often in finite number, assume logical or symbolical, rather than numerical, values.
- *Event-driven*, i.e., their evolution can be represented listing a sequence of discrete events which may also be described in nonnumerical terms.
- *Asynchronous mode of operation*, i.e., it is necessary to describe the order in which the events may occur, not the actual time at which they occur.
- *Concurrency*, i.e., several operations may occur simultaneously.
- *Nondeterminism*, i.e., from a given state several different evolutions may be possible.

New mathematical formalisms, other than differential or difference equations used in traditional control theory, are necessary to model these systems. These models are called *discrete event models* (DEM).

This chapter discusses and compares Petri nets (PN) and GRAFCET, two models that have been extensively applied in the domain of manufacturing. In the remaining part of this section the characteristics of discrete event systems and their models are discussed. In Section 2 and Section 3 Petri nets and GRAFCET are introduced and their power as discrete event models is evaluated. In Section 4 we compare the Petri net and GRAFCET formalism. In Section 5 we present a manufacturing example and compare the Petri net and GRAFCET model that has been derived for the same manufacturing process.

1.1 Discrete Event Systems and Models

As defined by Ramadge and Wonham [1], "A DES is a dynamic system with a discrete state space and piecewise constant state trajectories; the time instant at which state transitions occur, as well as the actual transitions, will in general be unpredictable".

Following [1], the state transitions of a DES are called *events* and may be labeled with the elements of some alphabet Σ. These labels usually indicate the physical phenomenon that caused the change in state. For example, in a manufacturing environment typical event labels are *"machine A starts working on part one"*, *"machine B breaks down"*, etc.

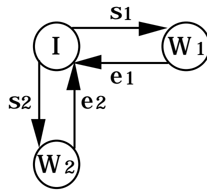
Consider the finite state machine in Figure 1a. It represents a machine capable of working on two different part types. The states are: I (idle), W_1 (working on part type one), W_2 (working on part type two). The events that cause the change of state are: s_1 (start working on part type one), s_2 (start working on part type two), e_1 (end working on part type one), e_2 (end working on part type two). An example of a state space trajectory is represented in Figure 2a.

The purpose of modeling such a system is the design of a control structure to drive its state space trajectories. This problem could be approached in a different way. Given a physical device that we will call the *operating unit*, i.e., the machine in this example, we could model the *control unit* that effectively drives the operating unit according to the orders received from the *supervisor*, i.e., from the external agent that chooses which sequences of operations are to be executed next. This control unit can be specified in terms of inputs and outputs. A black box model for the previous example is given in Figure 1b. Here we have two different sets of inputs: the inputs coming from the supervisor that specify which operations are to be performed; and the inputs coming from the operating unit that specify the last operations performed. In our example the inputs coming from the supervisor are the requests to start machining (s_1 and s_2), while the inputs triggered by the machine signal the completion of the machining operations (e_1 and e_2). In a similar way we have two different sets of outputs: control commands directed to the operating unit (W_1 and W_2), and messages to inform the supervisor of the state of the system (I). A possible evolution of the control unit for this example is represented in the timed diagram in Figure 2b.

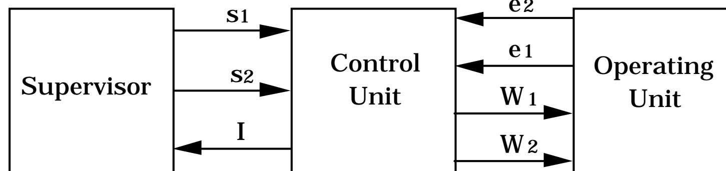
These two approaches to modeling are complementary. In the first case the focus is on the state of the system and its evolution. In the latter, we are trying to describe the input/output behaviour of the control unit. Petri net models are suitable to represent the state evolution, while GRAFCET has been especially designed as a specification language for the desired behaviour of the control system.

1.1.1 Model Classification

The many areas in which DES arise and the different aspects of behaviour relevant in each area have led to the development of a variety of DEM. Ramadge and Wonham have given the following classification [1]:



(a)



(b)

Figure 1: Two different views of a discrete event models.

1. *Logical DEM*, in which a common simplifying assumption is to ignore the times of occurrence of the events and consider only the order in which they occur. This simplification is justified when the model is to be used to study properties of the event dynamics that are independent of specific timing assumptions.

2. *Timed* or *performance DEM*, which are intended for the study of properties explicitly dependent on inter-event timing. These models can be further classified as:

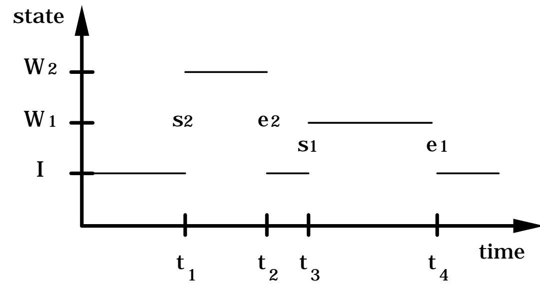
   a) *nonstochastic*: if the timing is known a priori;

   b) *stochastic*: if the timing is not known a priori due to random delays or random occurrences of events.

1.1.2 Evaluation of Discrete Event Models

A great number of logical DEM have been proposed. In order to evaluate and compare their suitability for control applications, as suggested in [2], the following aspects have to be considered.

1. *Descriptive power*
   The model must give a means to specify the set of admissible event trajectories.



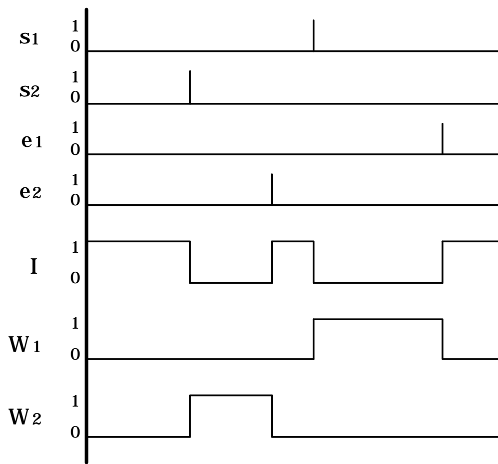


Figure 2: Evolution of the two models in the previous figure.

This may be done using some form of *state transition structure*, e.g., finite state machines, Petri nets, Büchi automata, by means of a *set of equations*, e.g., boolean models, communicating sequential processes, finitely recursive processes, or by a *logical calculus*, e.g., temporal logic. Some models, e.g., GRAFCET, may use a mixture of state transition structure and equations.

The behaviour of a logical model, i.e., the set of admissible event trajectories, may be represented by a language L. In general different formalisms generate different classes of languages, i.e., each model has its own *language power*. For example, regular languages (generated by finite state machines) are a subset of Petri net languages, that are a subset of recursively enumerable languages (generated by Turing machines).

We also note that complex systems can be regarded as being built out of interacting subsystems. For example, two subsystems may be made to interact with each other with a rendez-vous synchronization or a semaphore. As stated by Inan and Varaya, "A modeling formalism will be useful if it contains operators that combine one or more models in ways that reflect the ways in which systems are connected" [2]. The class of operators related to each model defines its *algebraic power*.

2. *Validation and Analysis*

   Once a model of the system has been constructed, the model should be used to verify that the system has the desired properties or that it is free of abnormal behaviour.

   This can be done in two steps. First "translate" the desirable properties of the system into properties of the model. Then "derive effective algorithmic, analytical, or simulation methods to verify that the model possesses the desired properties" [2].

   In this last step, the issue of *computational complexity* is a key concern. It is often the case that the greater the modeling power of a formalism, the less amenable the model is to analysis. Thus one tries to mitigate the complexity by the use of aggregation or modularity, or by exploiting hierarchical or other special structures.

3. *Control Implementation*

   The modeling and analysis of systems is only the first step in the study of DES. The final goal is to modify, by control action, the set of admissible trajectories so that each event trajectory has the desired properties. A model should then provide a guide to constructing a controller. At best this is done by an automatic compilation of the model into control code; at worst "ad hoc" solutions must be studied.

2 Petri Nets

Petri nets have been developed from the original model presented in 1962 by Carl Adam Petri in his doctoral dissertation: "Kommunikation mit Automaten" (Communication with Automata). The theory of Petri nets is now well established and many different Petri net models have been defined.

The literature on Petri nets for modeling, analysis, and simulation of manufacturing systems is vast, and a comprehensive discussion of it goes beyond the scope of this work. Silva and Valette [3] provide a very good review of Petri nets for manufacturing. Other references are [4, 5].

In the following we will describe the basic place/transition Petri net model.

2.1 Basic Definitions

A *place/transition net* (P/T net) [6, 7, 8] is a structure $N = (P, T, I, O)$ where:

- P is a set of *places* represented by circles.
- T is a set of *transitions* represented by bars.
- $I : P \times T \to \mathbb{N}$ is the *input function*. The value of $I(p, t)$ is the number the arcs from p to t.
- $O : P \times T \to \mathbb{N}$ is the *output function*. The value of $O(p, t)$ is the number the arcs from t to p.

A *marking* M is a function that assigns to each place of a P/T net a non negative integer number of tokens, represented by black dots. A *marked net* $\langle N, M_0 \rangle$ is a net N with an initial marking M_0.

A transition $t \in T$ is *enabled* by a marking M if $(\forall p \in P)\ [M(p) \geq I(p, t)]$. The firing of transition t generates a new marking M' with: $M'(p) = M(p) + O(p, t) - I(p, t)$. The set of markings reachable on a net N from a marking M is called the *reachability set* of N and M.

In the manufacturing domain, the places of a net may represent a resource status or an operation process [9]. A token in a resource status place indicates that the resource is available, no tokens indicates that it is not available. A token in a operation process place shows that the operation is being performed, no tokens shows that no operation is being

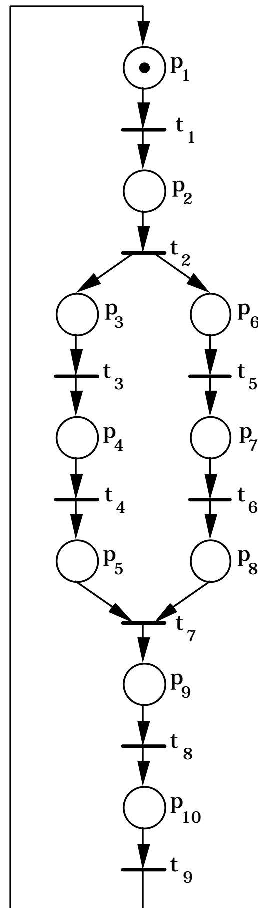


Figure 3: The Petri net for Example 2.1.

performed. The number of tokens in a place may also be greater than one, to indicate how many units of a resource are available or how many units are being processed. A transition represents either a start or a completion of an operation process or the event associated to the change of status of a resource.

Example 2.1. The Petri net in Figure 3 describes a systems where composite parts arrive, one at a time. A composite part is disassembled into two subparts A and B that are machined separately. Finally the part is reassembled and removed. Here the resource places are: p_1 (part available), p_5 (machined part A available), p_8 (machined part B available). The operation process places are: p_2 (disassembling), p_3 (loading part A), p_4 (machining part A), p_6 (loading part B), p_7 (machining part B), p_9 (assembling part), p_{10} (removing part). Transition, say, t_2 represents the completion of the disassembly process and the starting of the load operation for partA and partB.

The initial marking of the net in the previous example is shown in Figure 3, where only place p_1 is marked. The marking will change as the transitions fire. Note that if two or

more transition are enabled by a given marking, only one of them may fire at a time. Thus if two transitions, say t and t', are enabled by a given marking we may fire only one of them, say t, and update the marking. It is possible that t' is no longer enabled by the updated marking and thus cannot fire.

2.2 Petri Nets as Discrete Event Models

The Petri net formalism satisfies all the requirements a good DEM should have:

1. *Descriptive power*

   Petri nets have been designed specifically to model systems with interacting components and as such are able to capture many characteristics of an event driven system, namely concurrency, asynchronous operations, rendez-vous and semaphore synchronization, deadlocks, conflicts, etc.

   The basic place/transition net cannot model a condition of the form: "Fire transition t if place p is empty". Hence a place/transition Petri net cannot simulate a register machine which in turn is equivalent to a Turing machine [10]. It is possible to extend the formalism with the introduction of *inhibitory arcs*, i.e., arcs from places to transitions that prevent the firing of a transition if the input place is marked. However we note that the descriptive power of place/transition nets is large enough for most common applications, and often even more restricted models, such as conservative PN, are considered.

   Conservative PN are essentially equivalent to state machines, since the number of reachable markings, i.e., the number of "states" of the model, is finite. However, since the states of a PN are represented by the possible markings and not by the places, they allow a compact description, i.e., the structure of the net may be maintained small in size even if the number of the markings grows. See also the Example 2.2 in the following.

   Furthermore, the PN formalism permits description of logical models (Place/Transitions PN, Colored PN), nonstochastic performance models (Timed PN) and stochastic performance models (Stochastic PN, Generalized Stochastic PN).

2. *Validation and Analysis*

   The desired properties of a system map fairly well into properties of the corresponding Petri net model. Many algorithms, with a well developed mathematical and practical foundation, have been developed to study these properties.

   The analysis techniques for Petri nets may be divided into the following groups [11].

- *Analysis by enumeration*, that requires the construction of the *reachability tree* representing the set of reachable markings and transition firings. If this set is not finite, a finite *coverability tree* may be constructed.
- *Analysis by transformation.* A net N_1 is transformed, according to certain rules, into a net N_2 while maintaining the properties of interest. The analysis of the net N_2 is assumed to be simpler that the analysis of the net N_1. Examples of this analysis technique are *reduction methods*, that permit the simplification of the structure of a net.
- *Structural analysis*, that allow the demonstration of several properties almost independently of the initial marking. Structural analysis may be based on the study of the state equation of the net or on the study of the net graph.
- *Simulation analysis*, that is useful for timed nets and to study the behaviour of nets that interact with an external environment.

Petri nets also represent a hierarchical modeling tool and allow reduction of the computational complexity of analysis exploiting *modular synthesis*. By modular synthesis, complex systems may be constructed by aggregation of simpler modules while preserving the properties of interest.

3. *Control Implementation*
Petri net based controllers may be implemented in both hardware and software. Programmable Logic Controllers [13, 12] and Petri-net like languages [14] have been used in different applications. In [15, 16] a Petri net interpreter is implemented using a mixture of application dependent and independent code. However, it is necessary to point out that there exists no general technique for compiling a Petri net description into a control system.

It may be worth comparing PN with other models for logical discrete event systems [17]. It is possible to partition the current approaches into models in which the basic notion is that of *state*, such as state machines, and models in which the basic notion is that of *action*, such as interleaving models that describe the sequences of actions rather than the states. Petri nets show more flexibility in this respect, since states and actions are treated on equal footing and both step and interleaving semantics can be defined on Petri nets.

In the next example we will see that Petri net gives a very compact description of systems which, due to concurrent behaviour, have a large state space.
Example 2.2. Suppose we have a cyclic process where five jobs may be in four different stages. In Figure 4 we have a Petri net where each token represents a job and each place represents a different stage. Although the number of reachable markings is 56, the PN

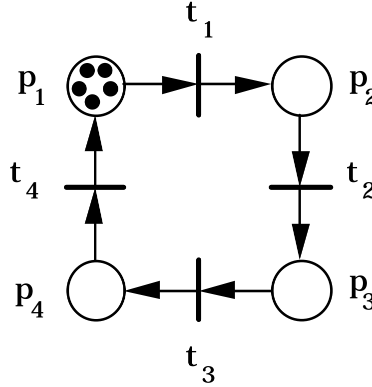


Figure 4: Petri net in Example 2.2.

model is very simple compared to a state machine model where we need to explicitly represent all states.

Models based on actions do not suffer from this state space explosion, since they do not require the explicit enumeration of all the states. As an example, an interleaving model of the system considered in this example may be described as follows. Let P_1 be the behaviour of the cyclic process with a single job, i.e., a single token. Then possible evolutions of the system are: $\lambda, t_1, t_1t_2, t_1t_2t_3, \ldots$, i.e., $P_1 = \overline{(t_1\ t_2\ t_3\ t_4)^*}$; here the bar stands for the prefix operator and $*$ is the Kleene star operator. Then the behaviour of the process when five jobs are been processed concurrently is given by:

$$P_5 = P_1 \parallel P_1 \parallel P_1 \parallel P_1 \parallel P_1,$$

where $\parallel$ is the concurrent composition operator.

The shortcoming of models based on actions is given by the cumbersome way in which specifications involving counters and semaphores are modeled.

Example 2.3. Consider the Petri net in Figure 5. Here we have two sequences (t_2t_1) and (t_3, t_4) that can be run up to a total of n times, n being the number of tokens in p_3. Using an interleaving model to represent this process is awkward: we have to explicitly specify all concurrent firing sequences. In fact, for $n = 1$ the behaviour is

$$P_1 = \overline{t_2t_1 + t_3t_4}.$$

For $n = 2$ we have:

$$P_2 = \overline{(t_2t_1)^2 + (t_2t_1) \parallel (t_3t_4) + (t_3t_4)^2}.$$

For a generic n:

$$P_n = \overline{(t_2t_1)^n + (t_2t_1)^{n-1} \parallel (t_3t_4) + \ldots (t_2t_1) \parallel (t_3t_4)^{n-1} + (t_3t_4)^n}.$$

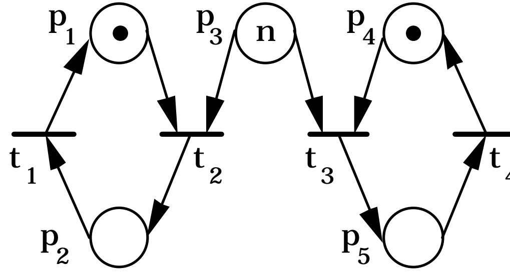


Figure 5: Petri net in Example 2.3.

3 GRAFCET

GRAFCET is an acronym for Graphe de Commmande Etape/Transition (Step/Transition Control Chart). GRAFCET has been proposed in 1977 by the Association Française pour la Cybernétique Economique et Technique (AFCET) as a standard to represent specifications for software control systems. Promoted by the Agence Nationale pour le Developpement de la Production Automatisé (ADEPA), it was accepted in 1982 as French standard by the Association Française de Normalization (AFNOR) [18]. In 1987 it was accepted as international standard by the International Electrotechnical Commission (IEC) [19].

In the following we will use the word "GRAFCET" to denote the standard, while a particular GRAFCET program will be called a "grafcet".

3.1 GRAFCET Syntax

A GRAFCET program [18, 20] is composed of: *steps*, *transitions*, and *links*.

The links are directed arcs joining steps to transitions and transitions to steps. They express the flow of control. The direction of the links is assumed to be downward unless an arrow specifies otherwise.

Steps are represented by square boxes. A step may be *active* (as denoted by the presence of a token in the step) or *inactive*. The boolean variable X_i is TRUE when step i is active. The *initial steps*, i.e., the steps that are active when the program is initialized, are represented by double boxes. *Actions* may be associated to a step. An action may change the value of a boolean variable, increase a counter or simply represent an order to the operating unit. *External actions* are those that represent orders or modify variables

that are supposed to be read by the external environment; these actions may be regarded as "outputs" of the grafcet. *Internal actions* are those that modify internal variables that will be used by the grafcet itself. The *state* of the grafcet is represented by the set of active steps and by the value of the internal variables.

Transitions are represented by horizontal bars. A *receptivity* is associated to each transition. A receptivity is an *and/or* expression of boolean variables;" +" represents the logical *or* and "." represents the logical *and*. A receptivity that is always TRUE is denoted as" = 1". The variables that appear in a receptivity may be "inputs" of the system, i.e., variables that will be modified by the external environment, or internal variables that depend on the current state of the grafcet. A transition is *enabled* when all immediately preceding steps are active. A transition may be *cleared* (or *fired*) if it is enabled and its receptivity is TRUE. The clearing of a transition deactivates all steps immediately preceding and activates all steps immediately following.

Example 3.1. The grafcet in Figure 6 describes the system discussed in Example 2.1. In the grafcet, the action associated with a step is shown in a box on the right of the step. Some steps, such as step 1, 5, and 8, do not have any associated action: they represent a waiting state. To the right of each transition is shown its receptivity; to the left is shown the transition label. Note that a double line follows transition $< 2 >$ and precedes transition $< 7 >$. The double line is introduced to emphasize that more that one step follows transition $< 2 >$ and more than one step precedes transition $< 7 >$. In this particular example all actions and all receptivity variables are external, i.e., the evolution of the system is controlled by the value of the inputs variables *part_arrived*, etc.

Algorithm 3.1. The execution algorithm of the grafcet is the following:

1. Activate the initial steps.
2. Determine the set T of transitions that are enabled and whose receptivity is TRUE. If this set is empty go to 4.
3. Clear all transitions in T. Go to 2.
4. When a new external event occurs go to 2.

A possible evolution for the grafcet in Figure 6 is shown in Figure 7. Given the shown trajectories for the inputs *part_arrived*, etc., the active steps at any given instant have been determined using the previous algorithm. Note that step 8 in this evolution is active only for an instant, since as soon as it is activated transition $< 7 >$ is cleared. We say that the state $\{X_5 X_8\}$ is an unstable state.

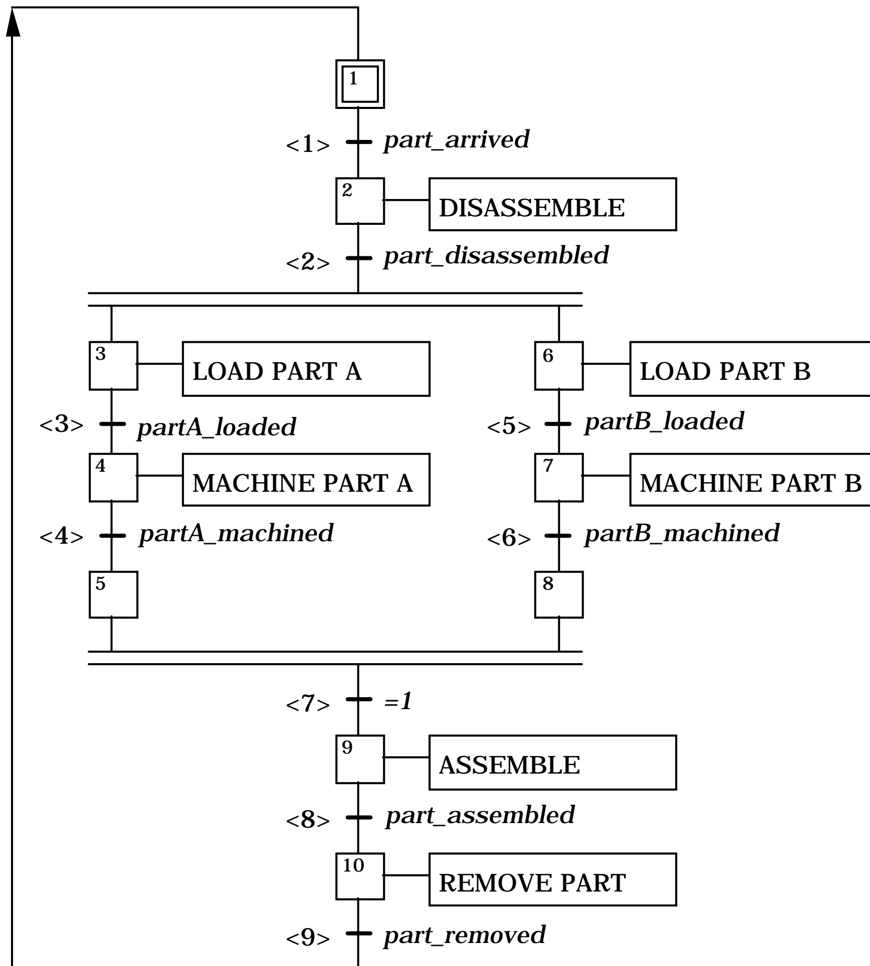


Figure 6: The grafcet for Example 3.1.

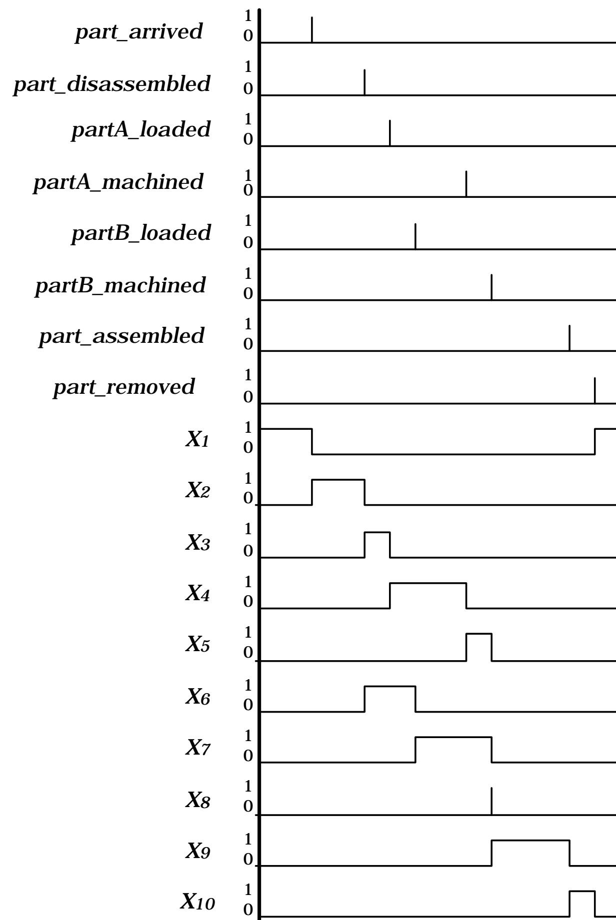


Figure 7: A possible evolution of the grafcet in Example 3.1.

3.2 GRAFCET as Discrete Event Model

1. *Descriptive power*

   Any algorithm that specifies how inputs and outputs of the control unit are related may be implemented in GRAFCET. Thus GRAFCET has the same language power of a Turing machine.

   It is possible to explicitly represent concurrency by activating more than one step at a time, hierarchic structures by the introduction of macro-steps that represent GRAFCET sub-programs, synchronization by transitions with multiple fan-in and fan-out. Thus a complex system may be represented by the interaction of GRAFCET programs each describing a system module.

   It is possible to include timing information in a grafcet. The standard defines a variable "$t/i/\Delta$" that is true if a time interval greater or equal to Δ has elapsed since the last activation of step i. However we cannot model random delays in GRAFCET. In fact random delays occur only in the operating unit (that is not modeled in GRAFCET) while any randomness should be avoided in a control unit.

2. *Validation and Analysis*

   The main drawback of GRAFCET is the total lack of analysis techniques to validate the model. This is because the flow of control is not completely captured in the graphical structure of a grafcet. In fact any action may modify the receptivity of a transition by changing the value of internal variables. This problem will be discussed in Section 4.3, where GRAFCET and Petri nets are compared. Thus the only possible form of validation is by simulation.

3. *Control Implementation*

   GRAFCET shows how the controller operates, without burdening the description with implementation details. This naturally leads to the creation of portable control code.

   When the outputs of a GRAFCET program are boolean variables, i.e., simple signals to the operating unit rather than commands, there are standard techniques for both hardware and software implementations of a grafcet [20].

   Commercial tools based on GRAFCET are available. FLEXIS, a system developed by Savoir, uses a GRAFCET-like visual language [21, 22]. In FLEXIS the actions associated to the steps are expressed in C language and a grafcet may automatically be compiled into executable code.

In [23] GRAFCET is compared with other languages for software control systems. Current approaches may be divided in: *visual* (or *graphic*), such as GRAFCET, Ladder Relay

Logic (LRL), and *textual*, such as procedural or object-oriented programming languages (C, C++).

Visual languages have the great advantage of representing abstract concepts more clearly and concisely than textual languages, thus increasing the understandability of programs. The drawback consists of the fact that a specification given in a visual language is "further removed from machine language" [23] and generally requires a more complex translation into code.

When compared with Ladder Relay Logic, GRAFCET shows many positive features. In general GRAFCET programs are more compact than a corresponding LRL program. Also they allow a better specification of the timing of events and actions.

4 Comparison between Petri Nets and GRAFCET

The basic difference between Petri nets and GRAFCET stems from the different ways in which they describe a system.

GRAFCET has been built as a specification formalism to describe software control systems. In GRAFCET a system is an entity that interacts with the environment: the inputs are the receptivities of the transitions and the outputs are the actions performed in the steps. A GRAFCET program unambiguously specifies the relationship between inputs and outputs. The behaviour is completely deterministic: given a GRAFCET program and a set of trajectories for the input variables there is only one possible set of corresponding trajectories for the output variables. This is possible because there is an "interpretation", i.e., a predefined algorithm to control the execution of the program.

A Petri net model, on the contrary, aims to describe a system that may well be nondeterministic. Here the focus is on representing the behaviour of the system, given as a set of sequences of events, i.e., firing sequences of transitions. It is often the case that more that one transition is enabled by a given marking and there is the "choice" of firing one or another transition, thus reaching different new markings.

In the following we will compare the two formalisms by means of some examples. The three most significant differences are based on the different notions of parallelism, i.e., concurrency, on the difference between places and steps, and on the fact that a grafcet transition has an associated receptivity that must be true for the transition to be cleared. References for this section are [3, 20, 24, 25].

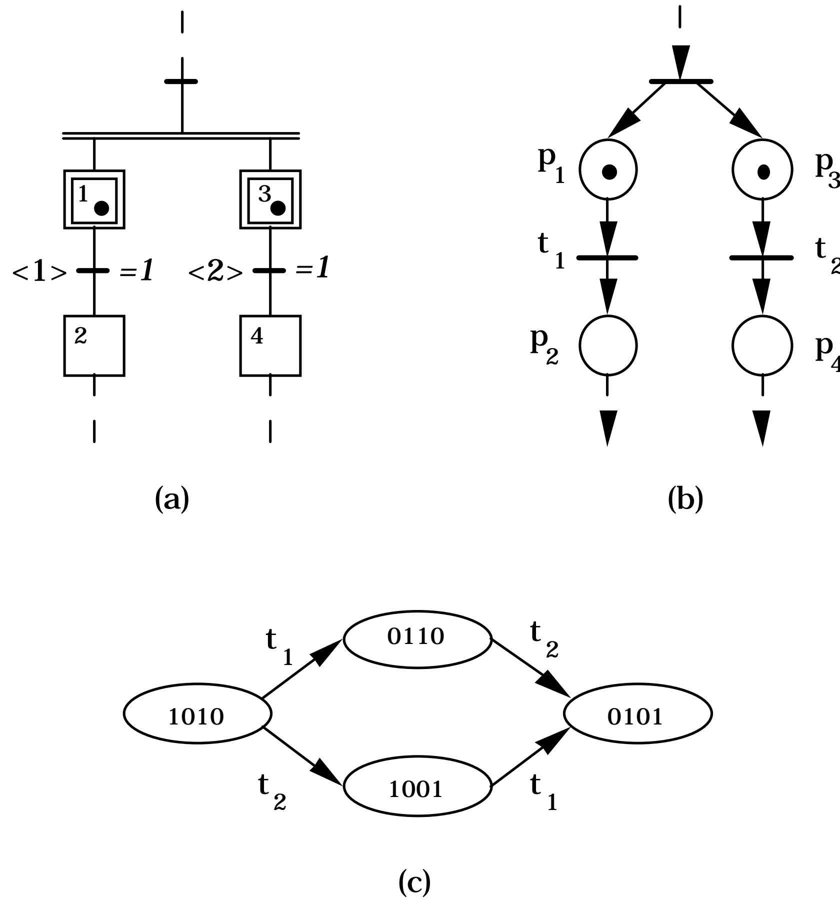


Figure 8: Two transitions in structural parallel mode in Example 4.1.

4.1 Parallelism

4.1.1 Structural Parallelism

Example 4.1. Consider the grafcet in Figure 8a, representing two concurrent sequences. In the state shown in the figure both transitions $< 1 >$ and $< 2 >$ are enabled and since their receptivities are true they will fire simultaneously. Thus the system passes from the state $\{X_1, X_3\}$ to the state $\{X_2, X_4\}$. The states $\{X_1, X_4\}$ and $\{X_2, X_3\}$ will never be reached.

In the Petri net in Figure 8b, instead, we may fire the sequences t_1, t_2, $t_1 t_2$, $t_2 t_1$. The reachable markings and the possible firing sequences are shown in Figure 8c.

The concurrent sequences $< 1 >$ and $< 2 >$ in the previous example are simultaneously

firable and they are said to be in a "structural parallel mode". Note that in some Petri net classes there is an interpretation similar to the GRAFCET interpretation. Examples of these models are: the *synchronized Petri nets* [26], and the *controlled Petri nets* [10], where the property that all enabled transitions may fire is named "decision free firing".

We may say that the GRAFCET interpretation defines an "a priori" scheduling policy. The Petri net model, on the contrary, does not consider any scheduling: the decision on which transition should fire will be resolved in the implementation phase.

4.1.2 Interpreted Parallelism

A different kind of concurrent behaviour, that has no counterpart on a Petri net, may be present in a grafcet. This behaviour may occur when two transitions have outputs from the same step. In manufacturing applications, this structure is characteristic of shared resources.

Example 4.2. Consider the grafcet in Figure 9a. It represents an available resource such as a robot. The robot may be used for two different operations: when the receptivity a of transition $< 1 >$ is TRUE the robot will perform the first operation (step 2 active in Figure 9b); when b is TRUE, transition $< 2 >$ will fire and the robot will perform the second operation (step 3 active in Figure 9c). However, should both a and b be TRUE, transition $< 1 >$ and $< 2 >$ will fire simultaneously and the grafcet will reach the state in Figure 9d that is physically meaningless, since the robot may be performing only one operation at a time

The same step precedes transitions $< 1 >$ and $< 2 >$ in this example: this structure is called a "selection". When transitions $< 1 >$ and $< 2 >$ fire concurrently they are said to be in "interpreted parallel mode". In fact the parallelism is only due to the GRAFCET interpretation, that requires simultaneous clearing of all transitions that may be cleared.

The same structure may be present in a Petri net and it is called "choice". However, there is no concurrency: only one transition may fire. This is shown in Figure 10: the marking reached after firing t_1 is in Figure 10b and the marking reached after firing t_2 is in Figure 10c.

It is possible to avoid this form of parallelism in a grafcet. We simply have to ensure that the receptivities of two (or more) transitions exiting from the same step(s) are mutually exclusive. A solution is shown in Figure 11. Here we have enforced a priority schedule: if a and b are both TRUE only transition $< 1 >$ will fire.

Again we note that to resolve this parallelism in GRAFCET we have to impose an "a priori" schedule. In the Petri net model the scheduling problem need not be addressed in

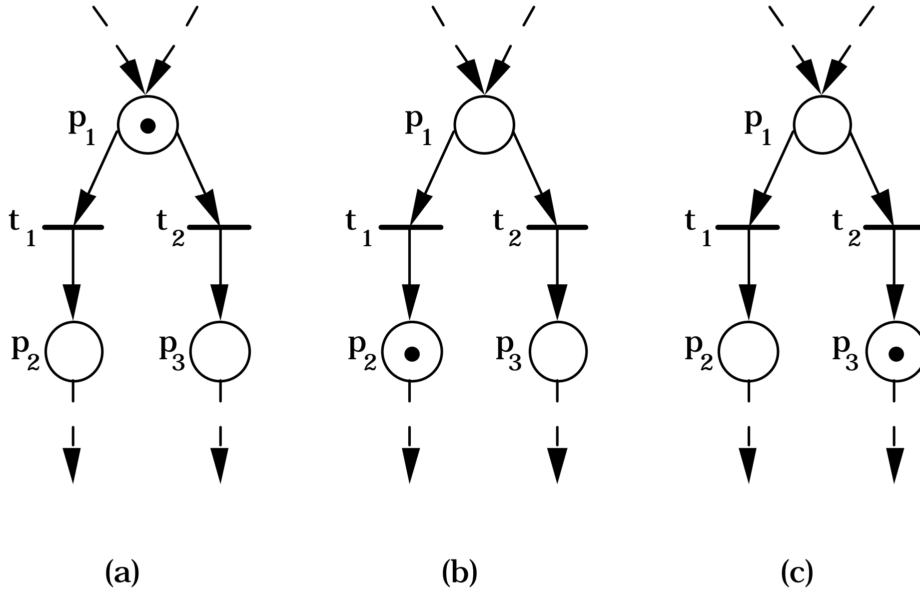


Figure 9: Two transitions in interpreted parallel mode in Example 4.2.

the modeling phase.

4.2 Steps and Places

Another difference between GRAFCET and Petri nets is that while GRAFCET steps are "flip-flops", i.e., may only be active or inactive, Petri net places are "counters", i.e., may have an integer number of tokens.

Example 4.3. In Figure 12 we have a simple model of a discrete flow process. We assume that jobs will be entering the process on the firing of the source transition t_1 for the Petri net model. After the source transition has fired once, we reach the state in Figure 12a. Assume that t_1 fires again before the firing of transition t_2; then we reach the state in Figure 12b, where place p_1 contains two tokens to indicate the presence of two jobs.

In the GRAFCET model, after the firing of the source transition $< 1 >$ we reach the state in Figure 12c. However, should $< 1 >$ fire again before $< 2 >$ is cleared, the state of the grafcet does not change, as shown in Figure12d. Here the information that two jobs have entered the process is lost.

This problem is called "step reinitialization". When a step reinitialization may occur, the designer should verify whether this behaviour is acceptable, (i.e., it is not really important to keep track of how many time an input transition of the reinitialized step has fired) or it

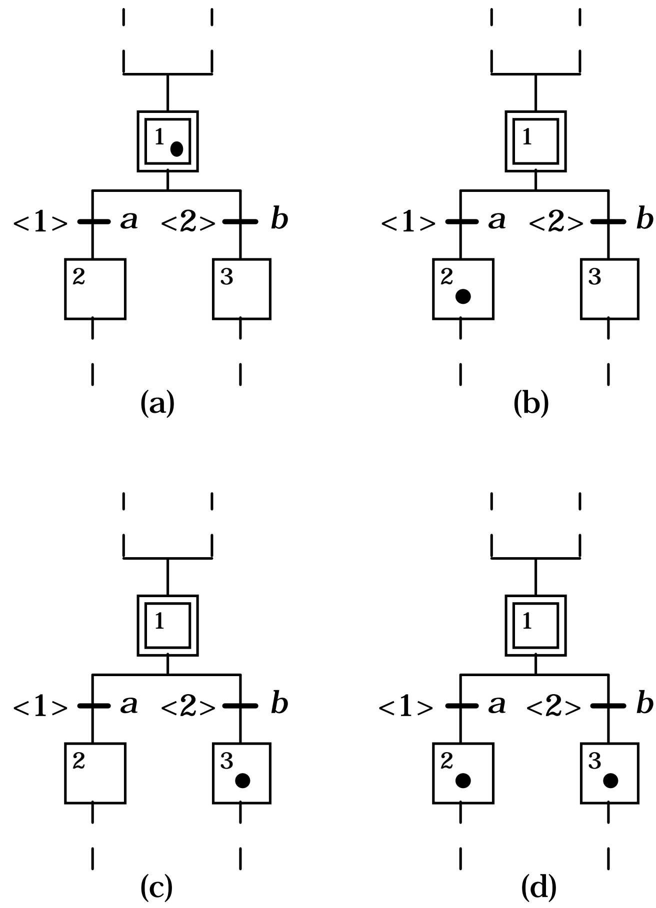


Figure 10: A choice structure in a Petri net.

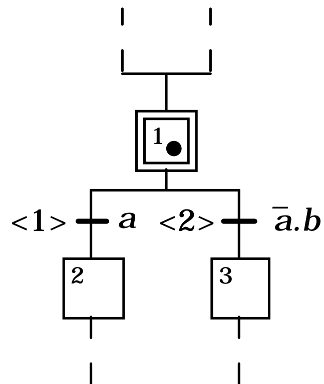


Figure 11: A priority schedule to avoid interpreted parallelism.

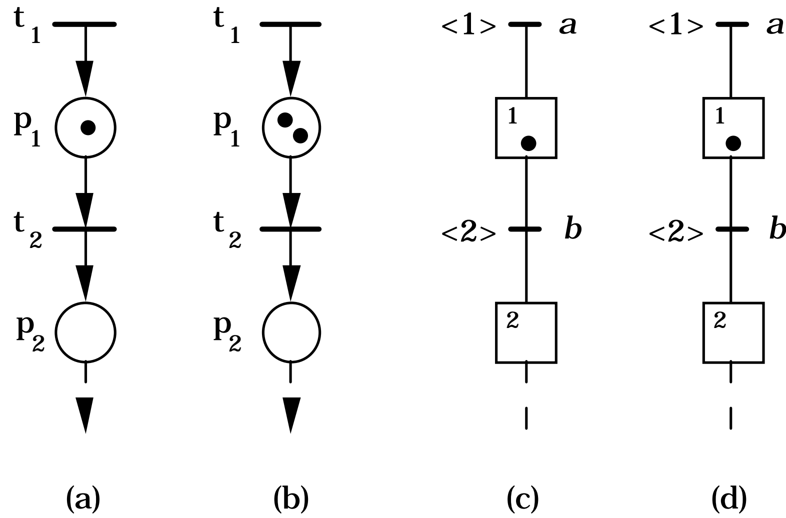


Figure 12: A continuous process in Example 4.3.

is due to a specification error. There are some GRAFCET models, called *safe GRAFCET*, where we can be sure that step reinitializations never occur [20]. These models are similar to *safe Petri nets*, i.e., Petri nets where the number of tokens in any place never exceed one.

4.3 Receptivities and Transition Firing

In GRAFCET the clearing of a transition depends on the value of its receptivity. The value of a receptivity may change because of the occurrence of an external event or due to the execution of the action associated with some step. Thus the clearing of a transition may depend on the state of steps that are not directly linked to the transition itself.

Example 4.4. Consider a manufacturing system composed of two machines and a buffer of capacity n. In Figure 13 is a Petri net model for this system. Here places p_1, p_2 belong to the first machine, p_3, p_4 belong to the second machine, and places p_5, p_6 belong to the buffer. The initial marking assigns n tokens to place p_5 to represent n available buffer slots.

In a GRAFCET model we need to introduce a variable c to act as a counter. Each time a part is put into the buffer, c is incremented; each time a part is taken from the buffer, c is decremented. In Figure 14 is the grafcet for this example. The steps associated with the buffer are: step 7, the initial step; step 8, active when the buffer is empty; step 9, active when the buffer contains some parts but is not full; step 10, active when the buffer

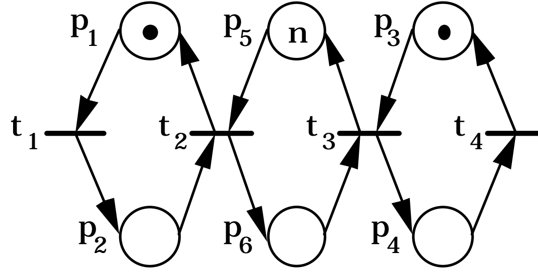


Figure 13: Petri net model of two machines and a buffer.

is full. The variable X_i is TRUE when step i is active. The three programs (two machines and a buffer) are not explicitly synchronized by common transitions but are synchronized through a global database where the values of the variables c, X_8, X_9, X_{10} are kept and updated.

The possibility of changing a receptivity value while executing any action, gives to GRAFCET a great modeling power but partially destroys the advantage of a graphical representation, since the "flow of control" is not completely represented by the underlying graph. There exist Petri net models, such as the *interpreted Petri nets*, where the firing of a transition may depend on the the markings of places not directly connected to it. In [20] it is proved that safe grafcets and a particular class of interpreted Petri nets (*command interpreted Petri nets*) are equivalent.

5 A Manufacturing Example

Given a Petri net, the initial marking and the structure of the net completely define the behaviour of the model. In a GRAFCET, on the contrary, we may encode part of the constraints in the actions associated to the step. Thus we gain a greater modeling power but reduce the possibility of analysis. Is has been suggested [24] that we use GRAFCET to first describe the specifications for a control system and that subsequently we translate the grafcet into a Petri net model for validation. However, this translation is only possible if we do not make use of the many features (such as interpreted parallelism, step reinitialization, non explicit control flow) that increase GRAFCET modeling power with respect to Petri nets.

In this section we discuss a manufacturing system for which a GRAFCET model has been constructed. The GRAFCET model is converted into an equivalent Petri net model which can be validated. The grafcet, however, may be compiled into control code with greater

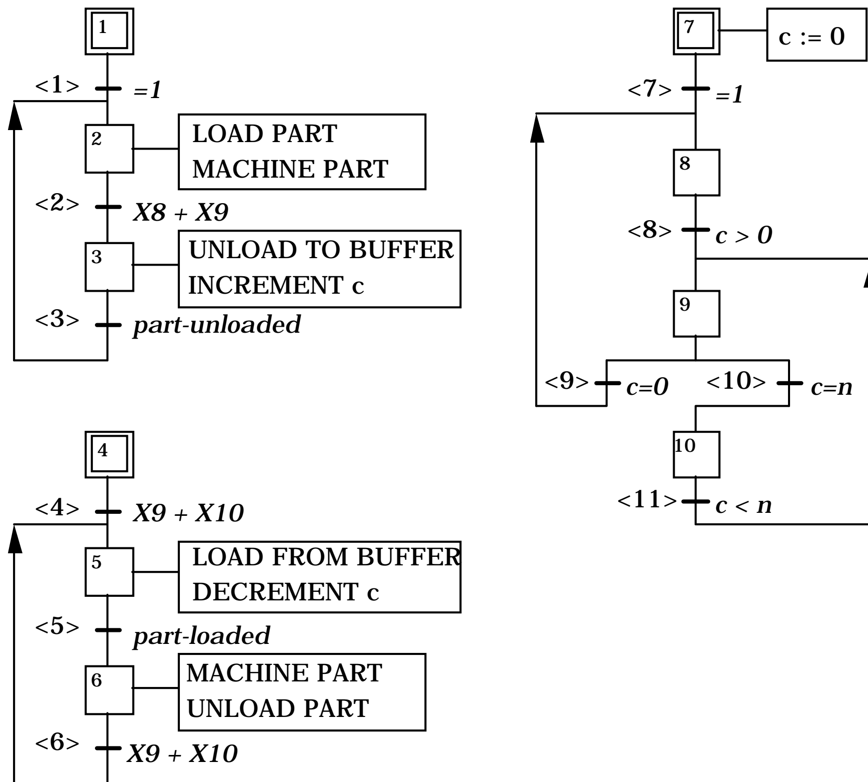


Figure 14: GRAFCET of two machines and a buffer.

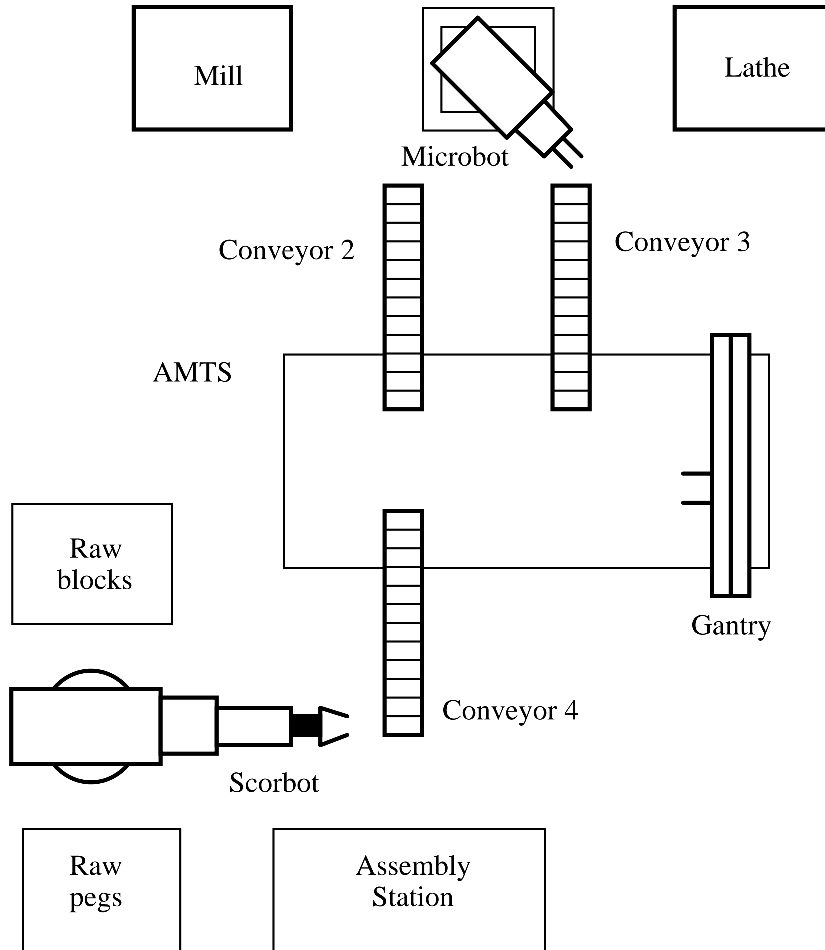


Figure 15: A manufacturing system.

ease than the Petri net.

The system considered consists of: two machines, a Mill and a Lathe; two robots, a Scorbot and a Microbot; an Automated Material Transfer System (AMTS) with three conveyers, C2, C3, C4, and a Gantry robot. Raw blocks are sent to the Mill and raw pegs are sent to the Lathe to be machined. The machined parts are assembled and removed. The layout of the manufacturing system is shown in Figure 15. This system has also been discussed in [15].

A GRAFCET controller for the system has been designed using FLEXIS. Here we have an actor for each device, i.e., machines, robots and material transfer devices. An actor is a simple GRAFCET program that controls a single subsystem. Actors communicate by exchanging messages trough a Manufacturing Application Protocol (MAP) network

[22, 27]. As an example, the actor associated with the Scorbot receives a request from the supervisor to start one of the four operations: 1) move raw block to C4; 2) move raw peg to C4; 3) move machined block to assembly; 4) move machined peg to assembly, assemble and remove. When the operation is completed, a message is sent back to the supervisor. If more than one request is received by an actor, the requests are served according to some priority scheme, e.g., FIFO queue, that needs not be discussed here. The actors for the devices are not shown, since their structure is quite simple.

A supervisor for the system can be implemented as a separate actor that exchanges messages with all device actors. Figure 16 shows the grafcet of the supervisor. Here the actions associated with the steps are requests to the device actors. For example, the action associated with step 6 is a request to the Microbot to pick up a raw block from conveyor C2 and load it on the Mill. The recipients of the requests are: S (Scorbot), M (Microbot), Mill, Lathe, G (Gantry), C2 (conveyor 2), C3 (conveyor 3), C4 (conveyor 4). The receptivity of a transition will become TRUE when a message from some device actor is received. For example, the receptivity *block_loaded* of the transition $< 6 >$ will be TRUE when a message from the Microbot confirms that a block has been loaded on the Mill. We also assume that as soon as a transition is cleared its receptivity is set to FALSE by one of the newly activated steps.

It is possible to recognize two parallel sequences in the grafcet of the supervisor, one for the processing of blocks and one for the processing of pegs. Only one block and one peg is processed at a time. The two parallel sequences use shared resources, such as the Scorbot, the AMTS, and the Microbot. Thus they must be synchronized and we have used both steps (steps 14 and 15) and transitions (transition $< 5 >$) to control the execution of the two sequences.

Now the validation of the model is quite difficult: we have to consider the joint execution of the supervisor and of the device actors and there are no general analysis techniques, with the exception of a series of simulations. In this example, however, since we have carefully avoided the use of any GRAFCET feature that embeds the flow of control in the actions associated to the steps, it is possible to construct an equivalent Petri net model. The Petri net may be validated with any of the analysis techniques proper for this formalism. The equivalent Petri net for this example is shown in Figure 17.

The Petri net in Figure 17 models the joint behaviour of the supervisor and of the devices. Note that, in order to keep the graph of the net simple, we have drawn several places associated with the same resource. Thus all the places labeled, say, p_M are in effect only one place, with one initial token to represent the initial availability of the Microbot.

It may be worth discussing how the Petri net model for this example is related to the

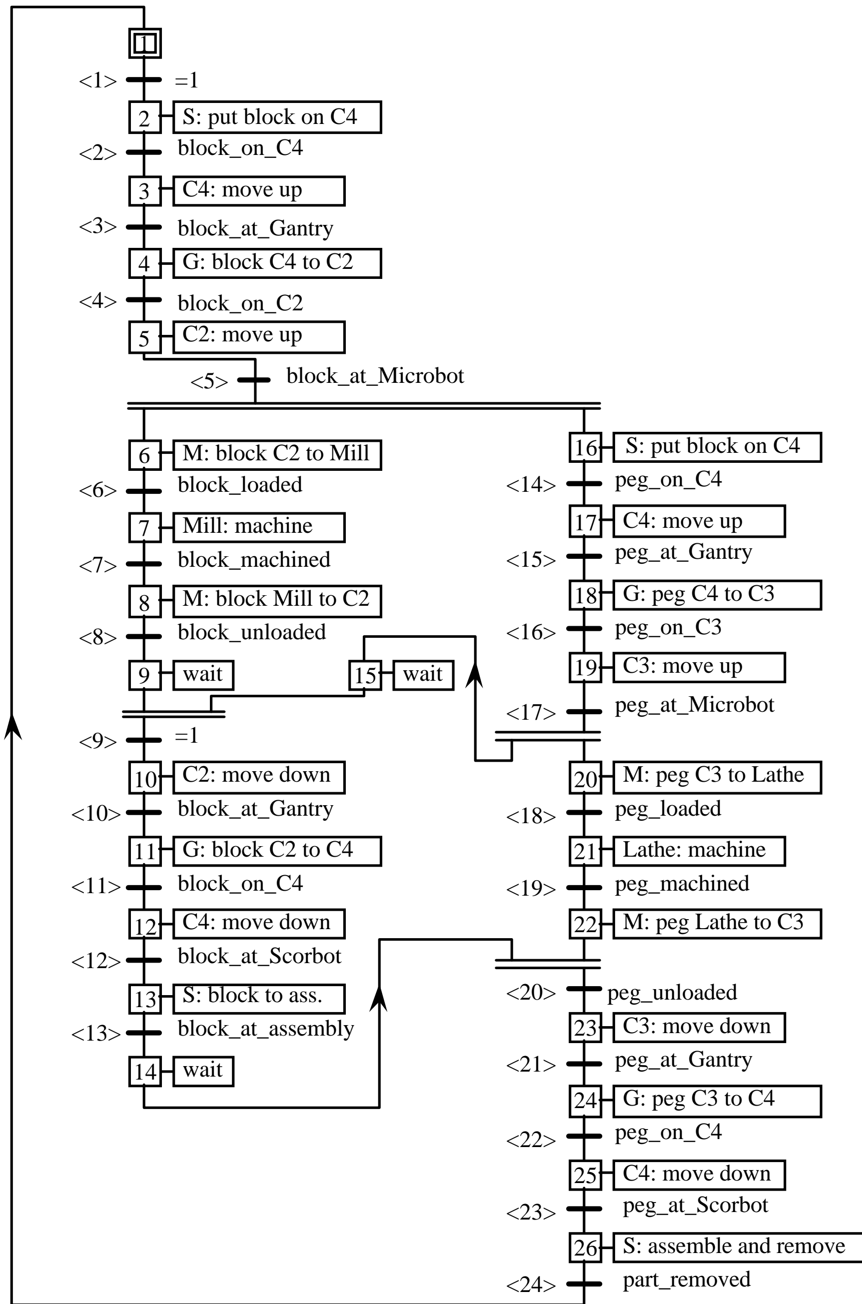


Figure 16: GRAFCET model of the system in Figure 15.

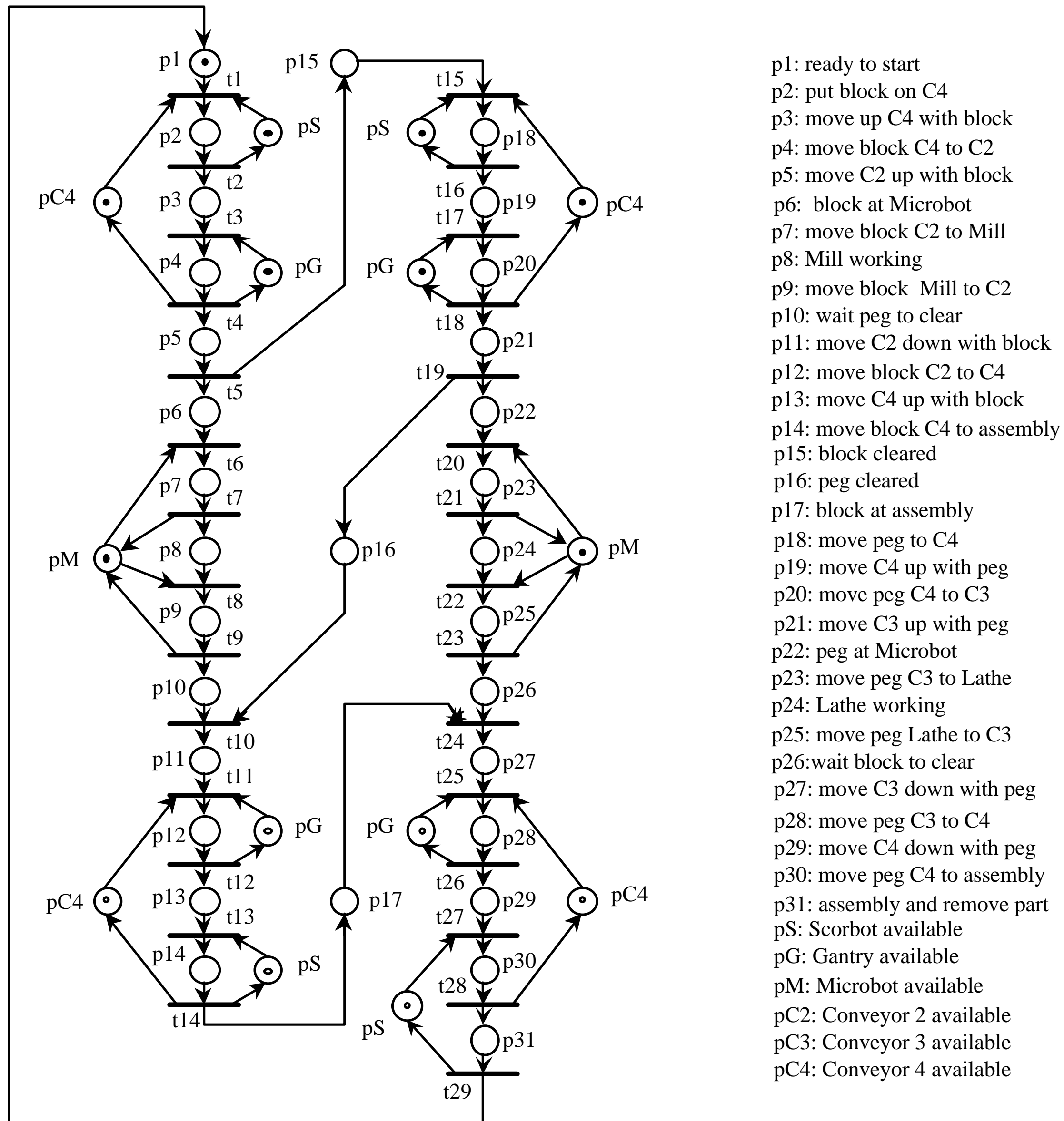


Figure 17: Petri net model of the system in Figure 15.

grafcet. In the grafcet the joint evolution of the supervisor and of the device actors is asynchronous: when step 2 in the supervisor is active, the Scorbot may be either idle or loading a block on conveyor 4. In the Petri net model, instead, there is a stronger synchronization between actors and supervisor: when place p_2 is marked the Scorbot is always loading the a block. Also in the Petri net we have often modeled the end of an action and the start of the next one with the same transition. For example, transition t_2 represents the end of the "loading a block on conveyor 4" and the start of "moving conveyor 4 up with block". When it was necessary to distinguish between the end of an action and the start of another one we have introduced additional places, such as place p_6 that separates the event "block arrives at Microbot" (t_5) from the event "Microbot loads block on Mill" (t_6).

The Petri net model may be validated by efficient techniques, such as the synthesis methodology presented in [9]. For example, it is possible to prove, without an exhaustive search of the state space, that the model is deadlock free and reversible.

Finally, the GRAFCET model may be directly compiled into control code by FLEXIS.

A final remark on this simple example regards the ease of modeling with Petri nets and GRAFCET. The design of the model was straightforward and the simple constraints enforced on the system sequence of actions have been easily implemented with semaphore synchronization.

6 Conclusion

This chapter has discussed two discrete event models, Petri nets and GRAFCET. The two models have been compared, by means of several examples, with regard to their application in the manufacturing domain.

Petri net models are useful to system designers for formally describing the behaviour of a manufacturing system. The great variety of analysis techniques proper of this formalism permits the validation of the model for properties of interest. GRAFCET is a good tool for describing the control specifications and a grafcet may be easily compiled into executable code by a programmer.

The two model may be used together, as in the manufacturing example we have discussed in section 5. Whenever this is possible both the analysis power of Petri nets and the ease of implementation of GRAFCET may be fully exploited.

References

[1] P.J. Ramadge, W.M. Wonham, “The Control of Discrete Event Systems,” *Proceedings IEEE*, Vol. PROC-77, No. 1, pp. 81–98, January, 1989.

[2] K. Inan, P. Varaiya, “Finitely Recursive Process Models for Discrete Event Systems,” *IEEE Trans. on Automatic Control*, Vol. AC-33, No. 7, pp. 626–639, July, 1988.

[3] M. Silva, R. Valette, “Petri Nets and Flexible Manufacturing,” *Advances in Petri Nets, 1989*, G. Rozenberg (ed.), Lecture Notes in Computer Science, No. 424, pp. 374–417, Springer-Verlag, 1990.

[4] *Proc. Int. Conf. on Computer Integrated Manufacturing* (Troy, New York), 1988, 1990, 1992.

[5] *IEEE Trans. on Robotics and Automation*, Special Issue on Manufacturing, R. Akella (Ed.), Vol. RA-6, No. 6, December, 1990.

[6] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[7] W. Reisig, *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer-Verlag, 1885.

[8] T. Murata, “ Petri Nets: Properties, Analysis and Applications,” *Proceedings IEEE*, Vol. PROC-77, No. 4, pp. 541–580, April, 1989.

[9] M.C. Zhou, F. DiCesare, “Parallel and Sequential Mutual Exclusions for Petri Net Modeling of Manufacturing Systems with Shared Resources,” *IEEE Trans. on Robotics and Automation*, Vol. RA-7, No. 4, pp. 515–527, August, 1991.

[10] A. Ichikawa, K. Hiraishi, “Analysis and Control of Discrete Event Systems Represented by Petri Nets,” *Discrete Events Systems: Models and Applications*, P. Varaiya and A.B. Kurzhanski (eds.), Lecture Notes in Control and Information Sciences, Vol. 103, pp. 115–134, Springer-Verlag, 1988.

[11] M. Silva, *Las redes de Petri en la Automatica y la Informatica*, Ed. AC, Madrid, Spain, 1985.

[12] M. Silva, S. Velilla, “Programmable Logic Controllers and Petri nets: a comparative study,” *IFAC Conference on Software for Computer Control*, E.A. Puente and Ferrate (Eds.), pp. 83–88, Pergamon Press, 1983.

[13] M. Silva, “Logic Controllers,” *Proc. IFAC Int. Symp. on Low Cost Automation* (Milan, Italy), pp. 157–165 bis, November, 1989.

[14] T. Murata, N. Komoda, K. Matsumoto, K. Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Application in Factory Automation," *IEEE Trans. on Industrial Electronics*, Vol. IE-33, No. 1, pp. 1–8, February, 1986.

[15] D. Crockett, A. Desrochers, F. DiCesare, T. Ward, "Implementation of a Petri Net Controller for a Machining Workstation," *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, North Carolina), pp. 1861–1867, April, 1987.

[16] E. Kasturia, F. DiCesare, A. Desrochers, "Real Time Control of Multilevel Manufacturing Systems Using Colored Petri Nets," *Proc. IEEE Int. Conf. on Robotics and Automation* (Philadelphia, Pennsylvania), pp. 1114–1119, April, 1988.

[17] E. Best, "Design Methods Based on Nets: Esprit Basic Research Action DEMON," *Advances in Petri nets, 1990*, G. Rozenberg (ed.), pp. 487–506, Springer-Verlag, 1991.

[18] Union Technique de l'Electricite, "Function Chart "GRAFCET" for the Description of Logic Control Systems," *French Standard NF C 03-190*, June, 1982.

[19] International Electrotechnical Commission, "Preparation of Function Charts for Control Systems," Publication No. 848, 1988.

[20] R. David, H. Alla, *Du Grafcet aux réseaux de Petri*, Hermes, Paris, 1989.

[21] B.H. Thomas, C. McLean, "Using Grafcet to Design Generic Controllers," *Proc. Int. Conf. on Computer Integrated Manufacturing* (Troy, New York), pp. 110–119, May, 1988.

[22] Savoir, *FLEXIS Toolkit User Manual*, Savoir Publications, 1989.

[23] A.D. Baker, T.L. Johnson, D.I. Kerpelman, H.A. Sutherland, "GRAFCET and SFC as Factory Automation Standards," *Proc. 1987 American Control Conference* (Minneapolis, MN), pp. 1725–1730, June, 1987.

[24] R. Valette, "Etude comparative de deux outils de représentation: Grafcet et réseaux de Petri," *Le Nouvel Automatisme*, No. 3, pp. 377–382, December, 1978.

[25] M. Blanchard, "Automatisme logiques: Grafcet ou réseaux de Petri?" *Le Nouvel Automatisme*, pp. 45–52, May, 1979.

[26] M. Moalla, J. Pulou, J. Sifakis, "Réseaux de Petri synchronisés," *Revue RAIRO Auromatique*, Vol. 12, No. 2, pp. 103–130, 1978.

[27] D. Wilczynski, “A Common Device Control Architecture — The Savoir Actor,” *Proc. AUTOFACT ‘88* (Chicago, Illinois), pp. 10.31–10.40, November, 1988.