ELSEVIER

The Journal of Systems and Software 43 (1998) 133–160

The Journal of Systems and Software

Comparative study and categorization of high-level petri nets

Vasilis C. Gerogiannis [a,b], Achilles D. Kameas [b], Panayotis E. Pintelas [a,b,*]

[a] Sector of Computational Mathematics & Informatics, Department of Mathematics, University of Patras, P.O. Box 1399, 26500 Patras, Greece
[b] Educational Software Development Laboratory, Department of Mathematics, University of Patras, Patras, Greece

Received 10 August 1996; received in revised form 27 November 1996; accepted 1 May 1997

Abstract

The graphical formalism of Petri Nets (PNs) is established on a strong mathematical foundation that can be applied in systems specification, analysis and verification. However, classical (low-level) models suffer from the state explosion problem as resulting PNs become larger. Thus, their ability to represent and analyze realistic large scale systems is reduced. High-level PNs have been introduced in order to extend the modeling power of low-level models. This paper presents an assessment of high-level PNs from an engineering perspective. A set of categories is proposed for classifying several extensions presented in the literature. Models which belong to the same category are compared by discussing the formalism, the descriptive power and the inherent limitations of each. All categories are compared using a set of general criteria including compactness, ease of analysis, degree of supporting refinement/abstraction and specifying communication. The modeling power of representative models of each category is discussed by presenting illustrative application examples.

Keywords: System modeling using petri nets; High-level petri nets; Individual tokens; Hierarchical modeling; Specification methods; Descriptive power of petri nets

1. Introduction

Petri Nets (PNs), introduced by Petri (1962) [1], are a powerful graphical modeling tool with firmly incorporated mathematical foundations that represent a system as a set of interacting active and passive entities. Active entities are called transitions; passive entities are called places (Reisig, 1992). PNs embody the characteristics of both finite-state machines and bipartite directed graphs and thus, they can express state transitions caused by events, as well as activities that proceed in parallel. A PN is graphically represented as a bipartite, weighted, directed graph with two types of nodes, which stand for transitions (rectangles) and places (circles), and arcs from a node of one type to a node of the other type. An arc does not represent a system component, but symbolizes a relationship between components. Items used to represent a piece of information or control that flows between the places and through the transitions are called tokens. Tokens are depicted as black dots within the places.

The PN model has been proven as a popular tool for describing and analyzing systems that are characterized as asynchronous, distributed, parallel, nondeterministic, and/or stochastic (Murata, 1989). Typical application areas are real-time systems, discrete event dynamic systems, flexible manufacturing control, robotics, avionics, automotive systems, multiprocessor environments, communication aznd synchronization protocols, distributed systems, interactive systems, expert systems and logic programming, neural nets and fuzzy controllers. The basic reasons for this wide popularity of PNs are their ability to provide visualisation, which makes them easily understood and learned by system designers, as well as their sound analysis techniques. Other reasons contributing to the popularity of PNs can be summarized as follows.

1. PNs allow the representation of nondeterminism and thus, a PN, like the system it models, is considered as a sequence of discrete events, the order of occurrence of which is one out of many possible ones allowed by the basic structure. The asynchronous nature inherent in PNs implies that there is no measure of time or of the

* Corresponding author.Tel.: +30-61-997313; fax: +30-61-992965; e-mail: pintelas@math.upatras.gr, vasilis@math.upatras.gr.

[1] In fact there are no nets in Petri's dissertation. It contains the fundamental ideas which later led to PNs.

0164-1212/98/$ – see front matter
PII: S0164-1212(98)10028-6

flow of time. In real life, events require different amounts of time, and the PN model reflects this variability by not depending on the notion of time to control the sequence of events. Events which are independent of each other are not projected on a linear time axis; instead a non-interleaving partial-order relation of concurrency has been introduced. Therefore, there is no explicit indication of the events' order because this order in time occurs by chance.

2. PNs have the ability to model a system hierarchically: an entire net may be substituted by a single place or transition, or places and transitions may be substituted by subnets. Consequently, designers can model a system in different levels of abstraction, without changing the modeling formalism.

3. PNs provide explicit representation of causal system dependencies and independencies, and support representation of both the system and its properties by using the same paradigm (Levi and Agrawala, 1990).

The major disadvantage of PNs is that they are characterized of low manageability and legibility, even for the description of systems of average complexity. Constraints arise because PNs model only the flow of control or data and cannot explicitly model the flow of resources, parts and information. In summary, the weaknesses of PNs are the following (Valavanis, 1990):

1. Only one place type is used to represent system states or events, and thus actual system processes are insufficiently represented.
2. Only one token type is used to represent either a piece of information or flow of control. For this reason there is no simultaneous representation of the flow of parts, resources, information and control through the system.
3. Since there are no special inscriptions (such as predicates and linear functions) associated with net places, transitions, and/or arcs to constrain the flow of tokens evolving through the net, large schemata are usually needed to represent aspects, such as individual system entities, complex precedence constraints and conditions among system processes.

However, research in PN theory and software engineering has introduced a number of new integrated approaches in order to reduce the size and, at the same time, increase the modeling power of PNs, two objectives that are very crucial in practical applications. More specifically, a number of modifications and extensions to the PN model, known as high-level PNs or PN-based models, have been proposed. High-level PNs allow for a formal treatment of individual (distinguishable) system entities. In addition, they provide manageable and concise system representation, as well as explicit modeling of specific system conditions and actions.

It is not possible to address all high-level PN (HPN) models in a single paper. Also, due to the rich body of knowledge regarding the various timed versions [2] of PNs, they have not been included in this study, which aims at classifying and comparing several HPNs proposed in the literature. Although a number of other surveys on PNs have been proposed (interested readers are referred for example to Peterson, 1981; Billington, 1988; Murata, 1989; Jensen, 1990; Lakos and Christensen, 1994), no one among them classifies the various presented models from an engineering perspective or covers the practical characteristics of HPN approaches. In this paper, HPNs are classified into categories based on the type of extension on the basic PN model that they propose. The key issues of each category are presented and the characteristics of representative models in each category are critically discussed. The advantages and limitations of representative models are graphically demonstrated (due to space limitations, we do not include all discussed models in the graphical presentation). Comparison is based on classical synchronization examples which are presented in an increasing order of complexity, in order to indicate that all models are not suitable enough to equally represent complex situations. In particular, the producers–consumers and the readers–writers synchronization problems will serve as the basis for the comparative study. Figs. 1–3 and 5, have been reproduced from Reisig (1992), Fig. 4 from Murata (1989), Figs. 6–8 from Christensen and Hansen (1994), Figs. 9 and 10 from Jensen (1995), Figs. 11 and 12 from Genrich and Lautenbach (1981), Figs. 13 and 14 from Christensen and Hansen (1993), and Figs. 15–17 from Kameas (1995), respectively.

The rest of the paper is organized as follows. In Section 2, the fundamental PN concepts are presented. In Section 3, low-level models are assessed and analysis methods are critically discussed. In Section 4, a framework is defined for classifying several extensions presented in the literature from an engineering point of view. Advantages and disadvantages of high-level PNs which belong to the same class are critically discussed. In Section 5, representative models in each category are compared based upon illustrative synchronization "benchmark" problems. The paper concludes with comparing the presented classes according to a set of general criteria.

[2] The concept of time has been introduced in several extensions on the PN model to model systems with strict timing constraints. Timed versions of PNs have been applied especially to real-time systems modeling and include Timed PNs (Ramchandani, 1974), Time PNs (Berthomieu and Diaz, 1991), Temporal PNs (Sagoo and Holding, 1991) and Stochastic (generalized or not) PNs (Hatono et al., 1991; Marsan et al., 1984). The introduction of time is mainly related to the location of time delays (at places, transitions and/or arcs) and the type of delays (fixed, intervals or stochastic).

2. PNs: Basic concepts

There are many closely related formal definitions of PNs (see for example Peterson, 1981; Reisig, 1985; Murata, 1989). The following definition has been adopted from Murata (1989) and closely follows to that in Reisig (1985).

Let N be the set of natural numbers, including 0. The most classical PN model, the place-transition net (PT-net), is defined as a 5-tuple structure $\mathrm{PN} = (S, T, F, W, m_0)$, in which

1. S is a finite set of places;
2. T is a finite set of transitions;
3. $S \cap T = \varnothing$ and $S \cup T \neq \varnothing$ (the sets of places and transitions are disjoint and nonempty);
4. $F \subseteq (S \times T) \cup (T \times S)$ is a set of arcs (the flow relation);
5. $W: F \to (N \setminus \{0\})$ is the weight function on the net arcs and
6. $m_0: S \to N$ is the initial marking, giving the initial distribution of tokens to places.

Arcs are not allowed between the same type of nodes (places or transitions); they define pre- and post- conditions for transitions and can be labelled with weights which are nonnegative integers, while an arc weight equal to one can be omitted. For the sake of simplicity the value of the weight function W on a net arc $\langle x, y\rangle$, where x is place or transition and y is a transition or place, is denoted as $W(x, y)$. The domain of the weight function W is usually extended to $(S \times T) \cup (T \times S)$ by setting $W(x, y) = 0$ iff $\langle x, y\rangle \notin F$, and then W is defined as a function from $(S \times T) \cup (T \times S)$ to N. For a place $s \in S$ and a transition $t \in T$, s is called an input place of t iff $\langle s, t\rangle \in F$ (there is an arc from s to t), and s is called an output place of t iff $\langle t, s\rangle \in F$ (there is an arc from t to s). The set of input (output) places of a transition t is named the preset (postset) of t, and is denoted by $\bullet t (t\bullet)$. Input and output transitions of a place s (that is, its preset ($\bullet s$) and postset ($s\bullet$)) are defined similarly. A transition (place) without any input place (transition) is called a source transition (place), whereas one without any output place (transition) is called a sink transition (place). [3]

Any function $m : S \to N$, giving the distribution of tokens over the places (drawn as black dots or integer values), is called a net *marking*. A net marking m is simply an assignment of a nonnegative integer to each net place, and therefore, it is often convenient to be denoted in a n-vector form Murata (1989), where n is the total number of places in S and the sth component, denoted by $m(s)$, is the number of tokens in place s. The notion of net marking is used to represent the dynamic state of a system. A PN model becomes dynamic as tokens that mark the net "travel" between places and through transitions. This travelling takes place as a result of a "transition firing", which is the transformation that changes a net marking.

A transition t of a PT-net $N = (S, T, F, W, m_0)$ is said to be firable (enabled) at a given marking m iff there are enough tokens on the input places (this is called the weak transition rule)

$$\forall s:\ s \in \bullet t, \quad m(s) \geqslant W(s, t).$$

A firable transition t at a marking m may, but need not fire. If it does, it yields another marking m' which is obtained by removing $W(s, t)$ tokens from each place in $\bullet t$ and adding $W(t, s)$ tokens to each place in $t\bullet$ (i.e., the firing of a transition removes tokens from its input places and deposits tokens on its output places, as many as the respective arc weights require)

$$\forall s:\ s \in S, \quad m'(s) = m(s) + W(t, s) - W(s, t).$$

Transition firing is voluntary, instantaneous and complete: a transition that is firable does not fire obligatory, yet it may fire only if it is enabled; firing takes zero time; and if a transition fires, the case of partial firing in which some token is not removed or not added, does not occur.

For all the above mentioned, it is assumed that each place can accommodate an unlimited number of tokens. Such a PT-net is called infinite-capacity net. In case of finite-capacity nets, a capacity function K is defined which specifies the maximum number of tokens that each place s can hold at any time, that is $\mathrm{K} : S \to N$ and any net marking m satisfies $\forall s:\ s \in S, m(s) \leqslant K(s)$. In a finite-capacity net, for a transition t to be enabled, there is an additional condition: after t fires, the number of tokens in each place s in $t\bullet$ cannot exceed its capacity $K(s)$ (this is called the strict transition rule)

$$\forall s:\ s \in \bullet t, \quad m(s) + W(t, s) - W(s, t) \leqslant K(s).$$

3. Assessment of low-level models and analysis methods

The PT-net model, as described above, generalizes on the simple Condition-Event net (CE-net), in which all arcs' weights and places' capacities are equal to one (Reisig, 1985). Fig. 1 shows a CE-net modeling a simple instance of the producer-consumer problem (i.e., a system with one producer and one consumer). The producer/consumer problem in its general form involves a set of producer processes that supplies messages to a set of consumer processes. All processes share a common

[3] Likewise, Peterson (1981) proposed transition input and output functions which are mappings from transitions to bags (multi-sets) of places (a multi-set is a generalization of a set that allows multiple occurrences of an element) and correspond to preset and postset of a transition in the adopted definition. In addition, a k-weighted arc, in the adopted definition, can be alternatively interpreted as a set of k parallel arcs, all having a weight equal to one, from a place to a transition (or vice versa).

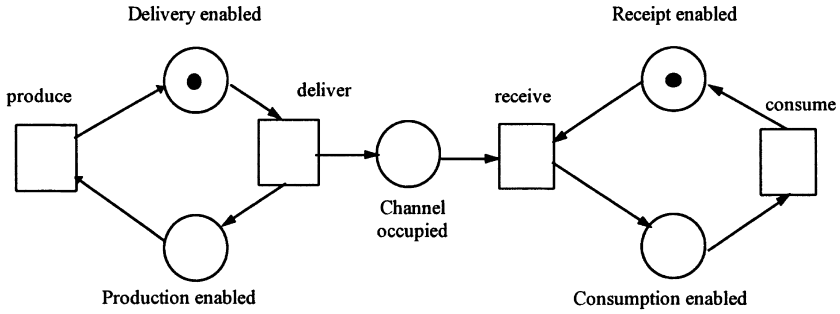

Fig. 1. A CE-net modeling a system with one producer and one consumer.

data area (communication channel) into which messages may be placed by the producers or removed by consumers. In the CE-net representation of Fig. 1, transitions and places are interpreted as events and (pre- or post-) conditions, respectively. For example, event "deliver" may occur if certain preconditions have been met: the producer is ready to deliver ("delivery enabled" is true) and the communication channel is not occupied. Event "receive" may occur if the consumer is enabled to receive ("receipt enabled" is true) and the channel is occupied. Only one token can be stored in the channel indicating its status (occupied or not). Obviously, design becomes more complex as the number of producing/consuming objects increases or additional producers and/or consumers are involved in the communication.

The PT-net depicted in Fig. 2 appears more suitable to represent a large number of producing/consuming objects. The communication channel corresponds to a single place with a capacity of 10, in order to accommodate up to 10 objects. The net represents one producer and two individual consumers, all depicted in three different dotted frames for modeling convenience. Instead of conditions and events, circles and boxes are more general and stand for passive (places) and active (transitions) entities respectively. If one is interested in the representation of the total number of consumers only without distinguishing among them, Fig. 3 appears more compact: two consumers have been placed into a single net component. However, the main limitation of Figs. 2 and 3 is that they do not distinguish among in-

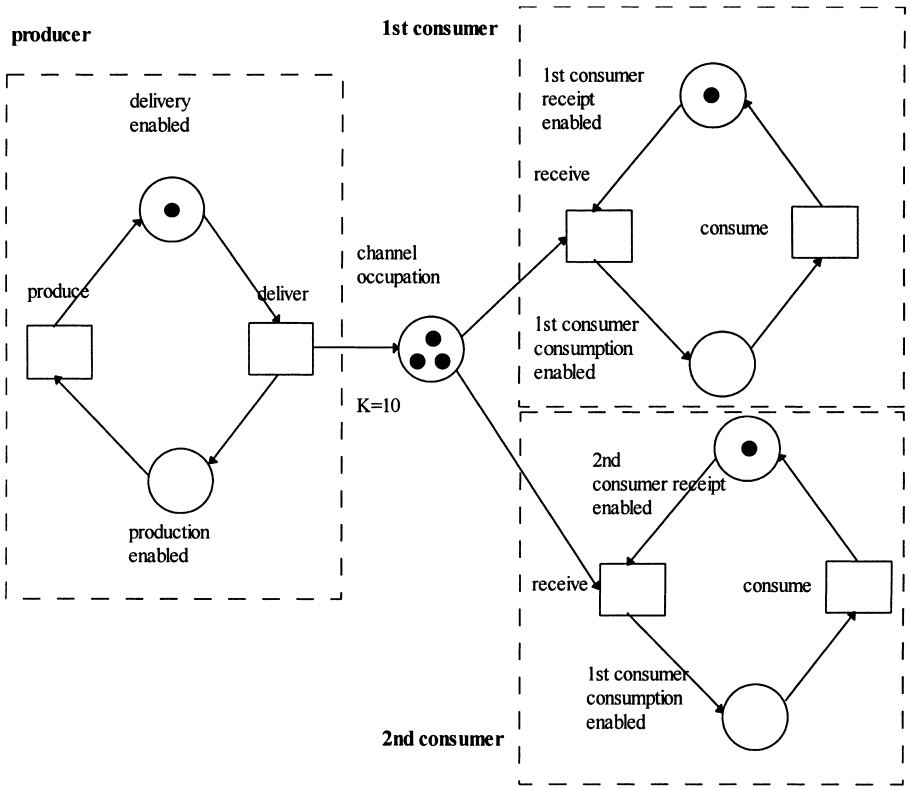

Fig. 2. A PT-net representing a system with one producer, two individual consumers and three objects in the channel.

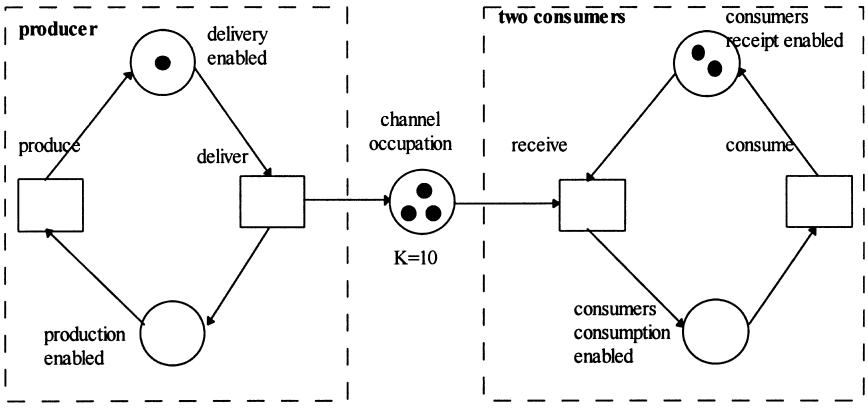

Fig. 3. A compact representation of the PT-net of Fig. 2 without distinguishing the two individual consumers.

dividual producers, consumers and communicating objects. Thus, if one is interested to indicate the individual producers, consumers and objects, a CE-net seems to be a more suitable, but the derived schemata will be too large and complex.

In addition, inhibitor arcs may be contained in a PT-net (Agerwala, 1974; Peterson, 1981). Such an arc can only be defined from a place to a transition. An inhibitor arc from a place s to a transition t modifies the firing rule as follows: the transition t is disabled (not firable) at a marking m if the place s has w or more tokens (i.e., $m(s) \geqslant w$), where w is the weight of the inhibitor arc, that is w is equal to $W(s,t)$. However, an inhibitor arc does not change the marking of a place when the associated transition fires. For example, the PT-net of Fig. 4 models a producers–consumers system where the first consumer has a higher priority over the second. In particular, the first consumer is able to consume as long

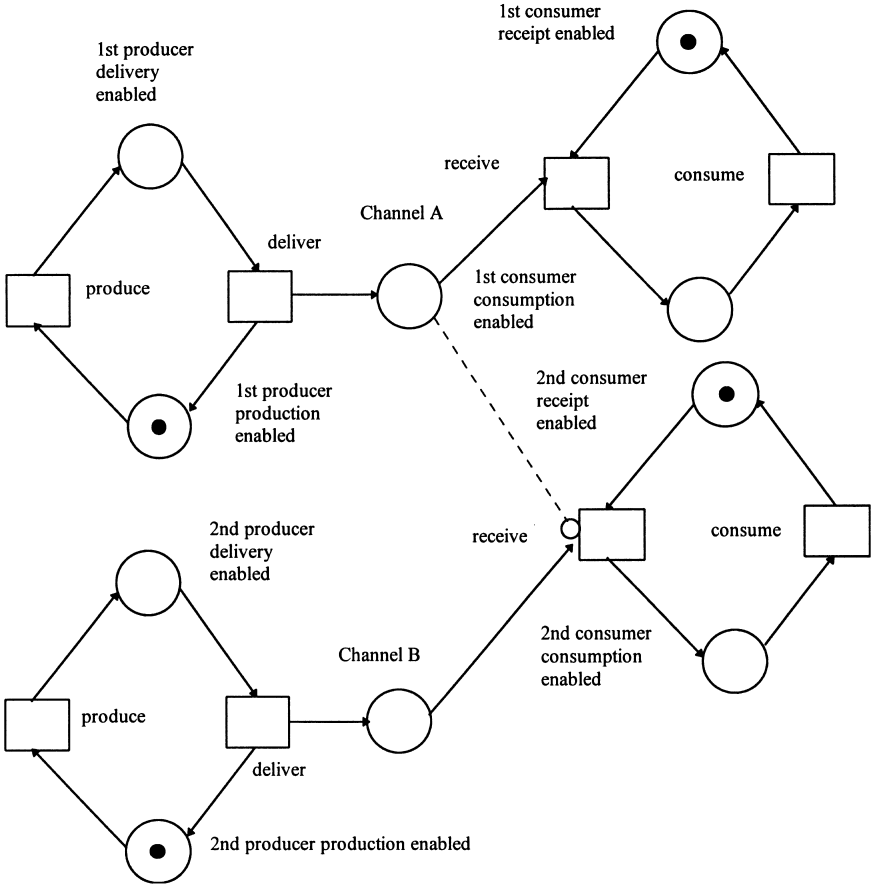

Fig. 4. A PT-net with inhibitor arcs modeling a producers-consumers system with priorities.

as Channel A has tokens, while the second consumer can consume only if Channel A is empty and Channel B has tokens (this is why an inhibitor-arc connects Channel A to the transition *receive* of the second consumer). It has been shown that if a PT-net is used, such a system cannot be modeled without inhibitor-arcs. In general, the embodiment of inhibitor arcs adds the ability of zero testing (i.e., testing for the absence of tokens in a place) and increases the expression (computational) power of PNs to the level of turing machines (Peterson, 1981).

From this point onwards, the term PN, unless explicitly stated, will refer to the PT-net model. Several subclasses of PNs have been defined by introducing certain structural restrictions on the general model (Best, 1987; Murata, 1989). Verifying if a net belongs to a particular subclass may be helpful because some properties specific to each subclass can be used to facilitate the system analysis. Therefore, if the main intention is analysis, then the choice of a low-level PN may be proved very applicable. However, in case of cumbersome realistic systems, there is always the possibility that such a choice will lead to complex nets. For this reason, several systematic synthesis approaches, which construct nets using components that belong to a specific and well-analyzed subclass, have been proposed in the literature (Valavanis, 1990; Koh and DiCesare, 1991). The main objective is to minimize as much as possible the complexity of the analysis process in the final constructed nets.

More specifically, in a state machine (SM) each transition t has exactly one input place and exactly one output place (i.e., $|\bullet t| = |t\bullet| = 1$ for all $t \in T$). Any of the three subnets in Fig. 2 (depicted in dotted frames) is an SM. A situation where a place has two (or more) output transitions, it is referred to as a conflict, decision or choice, depending on the application. Two transitions t_1 and t_2 are said to be in conflict if either one can fire but not both, and they are concurrent if both can fire in any order without conflicts. In Fig. 2, for example, a case where the place "channel occupation" accomodates only one token (object) will be a conflict for the two "receive" transitions. SMs allow the representation of conflicts, but not the synchronization of concurrent activities.

In a Marked Graph (MG) each place s has exactly one input transition and exactly one output transition (i.e., $|\bullet s| = |s\bullet| = 1$ for all $s \in S$). For example, PNs depicted in Figs. 1–3 are all examples of MGs. MGs allow representation of concurrency but they do not represent decisions (conflicts). Thus, they are suitable for modeling decision-free concurrent systems.

In a Free-Choice net (FC) every arc from a place is either a unique outgoing or a unique incoming arc to a transition (i.e., for all $s \in S, |s\bullet| \leqslant 1$ or $\bullet(s\bullet) = \{s\}$).

An Extended Free-Choice net (EFC) is a PN where

$$s_1\bullet \cap s_2\bullet \neq \varnothing \Rightarrow s_1\bullet = s_2\bullet \quad \text{for all } s_1, s_2 \in S.$$

Finally, an Asymmetric Choice net (AC) is a PN where:

$$s_1\bullet \cap s_2\bullet \neq \varnothing \Rightarrow s_1\bullet \subseteq s_2\bullet \vee s_2\bullet \subseteq s_1\bullet \quad \text{for all } s_1, s_2 \in S.$$

A case where both conflict and concurrency take place is called confusion. FCs do not allow confusion situations to appear. There are two types of confusion: symmetric and asymmetric. The former is a straight confusion situation, while the latter will lead to a conflict when a specific sequence of firings occurs. Concluding, SMs allow no synchronization of concurrent activities, MGs permit no conflicts, FCs disallow confusion and ACs may represent asymmetric confusion but disallow symmetric confusion.

Two types of system properties can be studied and verified via PNs: those which depend on the initial marking, and those which are independent of the initial marking (Murata, 1989). The former are referred to as marking-dependent or behavioural properties (e.g., reachability, boundedness, liveness, reversibility, persistency), whereas the latter are called marking-independent or structural properties (e.g., structural liveness, controllability, structural boundedness, consistency, repetitivity, fairness).

As far as analysis is concerned, the simplest technique for a PN model is simulation of its dynamic behaviour. Although simulation may be useful in discovering some (un)desirable system behaviour, it generally does not help in formally proving (behavioural or structural) system properties. Therefore, the following formal methods have been developed for analyzing and verifying both the static and the dynamic behaviour of PNs and consequently the system specification (Murata, 1989; Murata et al., 1989; Peterson, 1981; Koh and DiCesare, 1991):

- coverability (or reachability) tree method,
- matrix-equation approach,
- identification of the specific subclass at which a given PN may be classified,
- reduction and synthesis methods.

The first method involves the enumeration of markings that are reachable from the initial marking m_0 (a particular marking is called reachable from the initial one when it can be reached after a sequence of transition firings). This method results in a tree (called the coverability tree) representation of all reachable net markings (i.e., the reachability set) for the particular initial marking m_0. Nodes represent markings reachable from the initial marking m_0 (root) and directed arcs represent transition firings, which transform one marking to another. For a bounded PN (when, for all reachable markings, the number of tokens in each place is not greater than a finite number), the coverability tree is called the reachability tree, since it represents all possible reachable markings [Murata, 1989]. In addition, the corresponding graph-like representation of a coverabil-

ity (reachability) tree is called the coverability (reachability) graph. Concluding, the method is an enumerative technique leading to sequentialized views of the net behaviour and its application is usually restricted to small, bounded nets, since the resulted coverability tree usually grows exponentially with the size and complexity of the corresponding net (the state space explosion problem).

As far as the matrix-equation approach is concerned, every marking is represented as a vector, and consequently, a PT-net can be interpreted as a set of linear equations. The approach is suitable for analyzing structural net properties. The primary goal is the computation of place and transition invariants. A place invariant is a vector in which every place is associated to a weighted count of tokens, that remains unchanged during net execution; a transition invariant is a vector containing the number of times that each transition fires in a firing sequence which leads back to an initial marking. However, it is often difficult to compute the net invariants. In case of large nets there are many invariants and it is hard to find which of them are useful in analysis. Two are the main reasons for this limitation: the non-deterministic nature inherent in PNs and the constraint that the solutions must be non-negative integers.

Verification of system properties is often facilitated by a directed graph analysis which is mainly concerned with identification of the specific subclass to which a given PN can belong.

The major disadvantage of all the above methods is that they have been proven impractical in analyzing complex systems. Two approaches have been proposed to cope with this problem: reduction and synthesis. The former is concerned with reducing a large system model to a simpler one, while the properties of the initial model have to be preserved. The latter is concerned with constructing nets systematically in such a way that the desired properties are guaranteed without the need of analyzing the final nets. Two kinds of synthesis have been suggested: top-down and bottom-up. In a top-down synthesis, transitions and places are refined into more detailed nets. In a bottom-up synthesis, the final net is constructed by fusing common places or common transitions or common paths between subnets. The main problem with these approaches is that there exist no general and complete algorithms applicable to all PN classes. However, an interesting bottom-up synthesis approach which is based on fusing along common paths and minimizes the need to analyze the final nets can be found in Koh and DiCesare (1991).

4. High-level petri nets

The major restriction of the PN model described so far is that large nets are usually needed to describe systems of a medium complexity. Therefore, it is hard to compute net invariants and the coverability tree usually suffers from state explosion. In the PN model there is no formal treatment or clear identification of individuals (i.e., distinguishable system entities) and their properties and relations. Explicit structuring mechanisms (e.g., composition operators) are not included in the formalism, while conditions and operations are modeled only by the enabling and firing rule. In addition, there is a lack of a formal and general transformation procedure to a programming language.

For all these reasons, more general approaches have been proposed. These are known as high-level PNs (HPNs) or PN-based models (Gerogiannis et al., 1995; Jensen, 1995). Such models often provide more concise and manageable system representation and can be used to explicitly model data/control flow, specific firing conditions/actions, various system resources etc.

However, in case of systems with high complexity, even HPNs are not adequate enough, since they still describe a single view on the system under perspective. This motivated the development of structuring mechanisms and, in particular, the introduction of hierarchies (hierarchical high-level PNs–HHPNs) (Huber et al., 1990). HHPNs can be considered as a specific subclass of HPNs which concentrates on the substitution of net places and/or transitions by more detailed nets. HHPNs surely facilitate systematic specification, but analysis is often performed on the flat executable detailed net (i.e., the state space explosion problem of the reachability set still exists). For this purpose, structuring mechanisms have been proposed which support also "compositional" analysis by defining equivalently behaving substitutes of net places or transitions and thus, they significantly reduce the size of the state space (see for example Valmari, 1993). These approaches usually exploit concepts from other specification methods e.g., from the *communicating sequential processes* paradigm – CSP (Hoare, 1985). In the following, several extensions of PNs proposed in the literature will be reviewed in detail. The presentation will follow a classification scheme based on the type of extension that each HPN proposes. This scheme consists of the following categories:

1. extensions based on individual tokens (pure HPNs),
2. high-level nets with modified semantics,
3. extensions based on structuring mechanisms (HHPNs),
4. extensions which support representation of uncertain (fuzzy) information,
5. approaches based on integration with other specification methods.

For each category, representative approaches will be reviewed and their advantages and disadvantages will be discussed. Speaking in advance, before choosing to apply a HPN model, an important trade-off should be

considered: the more powerful the model, the less tractable computationally it becomes. Thus, if the main purpose is analysis, then a low level model (like PT-net) is most suitable, while, for large systems, the need for reducing the system complexity leads to choosing a more powerful HPN.

4.1. The extension of individual tokens

PT-nets and CE-nets seem impractical when the system under modeling is composed of a number of identical components (i.e., system entities presenting the same characteristics and behaviour). Distinguishing among various identical system components can be achieved only by constructing an identical subnet for each of them and thus, the final net suffers from complexity and redundancy. The employment of individual tokens to model explicitly and only once all identical system constituents was the key idea of the development of several PN extensions. Typical examples are Predicate/Transition nets (PrT-nets), Coloured PNs (CPNs) and Individual Token Nets (ITNs). All these approaches belong to a class which will be referred to as pure HPNs.

In PrT-nets (Genrich and Lautenbach, 1981) places play the role of predicates and can accommodate sets (tuples) of individuals (i.e., distinguishable system objects represented as tokens). In fact, tokens play the role of a predicate extension. Inscriptions (logical formulas) may be associated with some or all net transitions; arcs may be labelled with formal sums of tuples of variables which evaluate over tokens. The PrT-net model has added a new direction to the descriptive power of PNs: the formal (and more compact) treatment of all distinguishable system objects and their changing properties and relations. As far as analysis is concerned, a generalization of the linear algebraic technique (i.e., the matrix-equation approach) used to compute the net invariants has been proposed. The basic idea is to replace matrices of integers by matrices of formal sums over tuples of variables (i.e., arcs' inscriptions). The development of PrT-nets may be compared to the transition from the propositional logic to the first-order predicate logic (Genrich and Lautenbach, 1981; Peterka and Murata, 1989), and therefore, the model has been proved exceptionally suitable in modeling and analysis of logic programs.

CPNs extend PNs by associating colours with the tokens, the places, or perhaps the transitions of a net (Jensen, 1981, 1995). A colour is simply a type, a method of distinguishing classes of elements that belong to the same structural category. As long as the number of colours is finite, a CPN can be transformed into a complex PT-net (i.e., a PT-net can be considered as a special case of a CPN, where all the set of colours have only one element). On the contrary, an infinite number of colours gives CPNs the computational power of Turing machines, allowing any computable system to be modeled. CPNs and PrT-nets are equivalent in computational power, in the sense that, any concept, algorithm or theorem which applies to one model, applies to the other as well. However, compared to the latter, the former appear more convenient for systems description. The difference between the two models is the underlying formalism: CPNs are defined using types, variables, expressions and functions in a way similar to procedural programming languages (e.g., functions can be attached to a CPN arcs), while the formalism of PrT-nets is based on an algebraic notation (e.g., formal sums of tuples of variables are attached to a PrT-net arcs). [4] Moreover, an alternative linear algebraic method has been proposed for CPNs (Jensen, 1981) which appears more transparent compared to the one applied to PrT-nets. This is because in PrT-nets sums of tuples of variables are attached to the net arcs, while in CPNs linear functions between sets of (place and transition) colours can be used for this purpose. Therefore, invariants of a PrT-net may contain free variables, which have to be substituted according to the specific firing sequence that leads to the marking under consideration, while, in the case of CPNs, matrices of integers are replaced by matrices of linear functions over the sets of colours. The individual variables, occurring as coefficients in the invariants of PrT-nets, are eliminated, and consequently there is no need for additional substitutions.

The development of ITNs was based on the distinguishable tokens extension that influenced the introduction of both CPNs and PrT-nets (Reisig, 1985, 1992). The model formalism appears similar to the one of CPNs. Moreover, in ITNs transition enabling and firing rules have been slightly modified: additional explicit conditions and operations may be associated with each transition which influence the enabling and the firing rule of the transition, respectively. For this reason, ITNs may be also classified into the category of HPNs with modified semantics discussed in the next subsection.

4.2. HPNs with modified semantics

A number of approaches have been presented for extending either high-level nets (e.g., PrT-nets or CPNs) or low-level models (e.g., PT-nets) with arcs, places and transitions that have modified semantics. The many reasons why HPNs with modified semantics may be preferable are as follows:

- they provide a higher degree of encapsulation, since details can now be hidden in the net components (e.g., in arc and token types) and thus, they often re-

[4] This is why PrT-nets have motivated the introduction of algebraic HPN extensions which will be reviewed in a subsequent paragraph.

sult in more comprehensible schemata (that is, nets which do not have so much information hidden in net inscriptions);
- they can relate semantics of a specific programming language to terms of the PN formalism in order to facilitate the transition from design to implementation. Although in this case we lose power in abstraction, when someone is interested, for example, in automated code generation and rapid prototyping language-dependent semantics in a PN formalism may be very helpful (for example, interested readers can find in Hartung (1988) an approach combining HPNs with Concurrent Pascal);
- they support true concurrency (multiple transitions are concurrently enabled and occur together), as well as interleaving semantics (by considering the effect of each transition firing in isolation);
- they extend the descriptive power of the PN model in order to cover specific application areas.

It should be mentioned that modified semantics increase only the compactness and not the computational power of HPNs. A HPN with modified semantics can be converted to an equivalent HPN, but the result will be an unnecessarily complex net. Furthermore, the major limitation of most approaches is that they still lack directly applicable analysis techniques, and thus analysis is usually performed on the equivalent HPN.

A number of approaches have been defined for extending and modifying operational semantics of HPNs (place capacities and arc types, transition semantics and place types). As far as the extension on *place capacities* and *arc types* for HPNs is concerned, the most general model, which encompasses the semantics of various similar approaches, seems to be the one proposed by Lakos and Christensen (1994). Their approach was based on an earlier work by Christensen and Hansen (1993). Both proposals incorporate the concepts of place capacities, *test arcs* and *inhibitor arcs* into the CPN formalism. For each finite capacity place, a triple consisting of a *capacity colour set*, a *capacity projection function* and a *capacity multi-set* is specified. A capacity projection maps markings of a finite capacity place onto multi-sets, over the capacity colour set. This linear function is used to disregard some information concerning the marking of a finite capacity place. For example, a projection function can be used to associate a place only with some colours or colour combinations. A capacity multi-set is a multi-set over the capacity colour set used to bound the possible acceptable markings of a finite capacity place. In this way, the number of appearances of each colour are now limited.

Test arcs and inhibitor arcs are interesting extensions regarding the arc types of a CPN. Test arcs access the tokens in a place, but do not modify the marking of the place and they are used to model concurrent access (i.e., concurrent read) to shared data without changing these data. Inhibitor arcs are a generalization of the inhibitor arc concept found in PT-nets. An inhibitor arc expression is associated with each inhibitor arc. The binding of this expression (substitution of its variables) results in a threshold value. Thus, a transition is disabled if the associated threshold value is exceeded; otherwise, the transition may fire without changing the marking of the place connected to the inhibitor arc. Christensen and Hansen (1993) propose that inhibitor arcs must be restricted to be adjacent only at finite capacity places. A capacity place is then transformed into a place together with a *complementary* place which holds the capacity multi-set less the marking of the original place. Lakos and Christensen eliminate this restriction at all: inhibitors arcs do not need to be adjacent at finite capacity places due to the construction of a *generalized complementary* place (for more detailed information about this concept see Lakos and Christensen (1994)). Furthermore, they propose a wider set of arc types including simple arcs, compound arcs and arcs with projected inscriptions. Simple arcs form a fundamental set and are combined to give more complex (compound) arcs. Arcs with projected inscriptions allow arc inscriptions to be modified by a linear projection function. Finally, a finite capacity place in the model of Christensen and Hansen can be equivalently modeled using the Lakos and Christensen's approach as a place where every input arc is coupled with an inhibitor arc, with the capacity as arc inscription.

To summarize, both approaches are applied directly to HPNs (CPNs) and not to simple models; both can represent true concurrency (i.e., multiple transitions occur together) and interleaving semantics (i.e., multiple transitions can occur in any order) as well. However, Lakos and Christensen's work is more general since it eliminates the need to define inhibitor arcs incident at finite capacity places and proposes explicitly a wider set of arc types. Furthermore, they have proven that their model incorporates a number of other arc extensions presented in the literature (Billington, 1988; Heuser and Richter, 1992). For example, their formalism can be used to model *place summary functions*, a concept derived from the Object-Oriented language LOOPN and originally presented in Lakos and Keen (1991). A place summary function is used to determine if all tokens in a place have a given colour or a colour from a range of possible colours, as well as to find the number of tokens in a place which satisfy a given condition. In Lakos and Christensen's model a place summary function is equivalent to an "equal" arc with a projected inscription.

An alternative model for extending arc semantics is the one presented by Ciardo (1994). This approach overcomes the limitation of PT-nets in describing practical situations where transition firings remove (add) from (to) a place a number of tokens which depends on

the net marking. The proposed class of models extends PT-nets (and not HPNs) with marking-dependent arc weights (cardinalities). The following subclasses of PT-nets with marking-dependent arc cardinalities are defined:

- *Ordinary* PT-nets: arcs have constant cardinalities,
- *Reset* PT-nets: arcs empty the tokens from a place,
- *Post self-modifying* PT-nets (set nets): the cardinality of output arcs can be any nonhomogeneous linear combination of the marking (e.g., a duplication of the number tokens in a place),
- *Transfer* PT-nets: the firing of a transition can move (but not duplicate) all the tokens from one place to another,
- *Linear transfer* PT-nets (reset-set nets): both reset and duplication of tokens are allowed,
- *Self modifying* PT-nets: the cardinality of both input and output arcs can be any nonhomogeneous linear combination of the marking.

The class of PT-nets with marking-dependent arc cardinalities includes PT-nets with inhibitor arcs as a subclass, since an inhibitor arc is equivalent to an arc with cardinality $2m(s)$, where $m(s)$ is the marking of a place s. In contrast to Lakos and Christensen's approach (Lakos and Christensen, 1994), marking-dependent arcs extend low-level models (PT-nets) and not HPNs. In addition to that, arcs with marking-dependent behaviour appear more restricted compared to the several arcs types proposed by Lakos and Christensen. However, Ciardo's proposal is combined with a "direct" analysis method based on a generalization of PT-net invariants.

Another interesting modification on the operational semantics of a PT-net arcs is the *debit arc* extension (Stotts and Godfrey, 1992). PT-nets with debit arcs (*debit nets*) are used to specify events that may proceed even when all firing preconditions are not met. A debit arc is permitted only from a place s to a transition t. In that case, if s is not marked then t may fire and a *debt* (also called an *antitoken*) will be created in s. On the contrary, if s is marked then firing t may either consume a token or create a debt. Debit nets semantically extend the firing rule of PT-nets by defining special annihilation policies for tokens: a token and an antitoken residing in the same place can annihilate (destroy) each other. Two annihilation policies have been defined, namely *instantaneous* and *delayed* annihilation. The former specifies that whenever a token and an antitoken are co-resident in a place then they immediately annihilate each other, while the latter defines that annihilation will take place in subsequent states of net execution. The class of debit nets under the delayed annihilation has been proven to be equivalent to the class of PT-nets, while debit nets under the instantaneous annihilation are equivalent to Turing machines, since they have been proved to be a superset of the class of PT-nets with inhibitor arcs (Stotts and Godfrey, 1992).

Numerical PNs (NPNs) (Billington et al., 1988) compose a comprehensible realization of HPNs which can be directly executed and analyzed. NPNs include Predicate/Transitions nets (PrT-nets) as a subclass since they generalize tokens to tuples of variables, like PrT-nets do. In addition, the model presents the following extensions:

1. A set of data variables is associated with the net (a data variable is represented by a place, with appropriate input and output arcs, and a token carrying its present value). Common supported variable types are integer, modulo, Boolean, enumerated and string.
2. Two place capacities are supported. The first (K) sets a bound on the number of tokens of a particular value that can be resident in a place, the second (K^*) sets a bound on the total number of tokens allowed in a place.
3. Three inscriptions are associated with each arc: (a) an input inscription, defining the condition which may be satisfied by a collection of tokens in the associated input place, (b) the destroyed token inscription, defining a multi-set of tokens, which is removed from the associated input place, when the transition fires, and (c) the created tokens inscription, specifying the multi-set of tokens which is added to the associated output place, when the transition fires.
4. Two optional inscriptions are associated with each transition: (a) a transition condition, defining a condition on net data variables associated with tokens residing in the transition input places, and (b) a transition operation, defining an operation on the data variables.

The formalism of NPNs presents many similarities with other approaches. For example, Billington (1988) introduced three capacity concepts for CPNs: the *total* (or integer) capacity, the multi-set capacity and the capacity with *unbounded* colours. Total capacity and multi-set capacity are identical to K^* and K place capacities defined in NPN formalism, respectively. On the other hand, the multi-set capacity with unbounded colours, in Billington's proposal, allows a multi-set over the set of colours in a place to contain infinite elements. However, these concepts have been further generalized by Christensen and Hansen (1993) and Lakos and Christensen (1994). Furthermore, the modified transition semantics in a NPN (the two optional conditions which can be associated with each transition) are also found in the ITN formalism. Concluding, NPNs incorporate essential features found also in other models. The main motivation for the approach was to define simple extensions which allow for an easier and more powerful specification, while "direct" verification of net properties (by constructing the reachability graph) is possible.

The same idea has influenced the development of other realizations of HPNs, such as PNs with transition enabling functions (Pn_e), which modify only the *transi-*

tion semantics of PT-nets (Papelis and Casavant, 1992). In a manner similar to PrT-nets, Pn_e allow the association of a predicate with each transition. However, in PrT-nets several conditions (i.e., arc inscriptions) can be associated with a place, which results in nets that may not visually correspond to the original modeled system. Therefore, PN_e try to specify the system structure with the use of PT-nets and the "control strategy" of the system textually using transition enabling functions, which are produced from a specific BNF grammar.

Finally, Extended PNs (EPNs) (Valavanis, 1990) represent a typical attempt to extend the *place semantics* of PT-nets. The formalism of EPNs incorporates six different types of places to account for the different classes of conditions that may arise in the modeled system. In particular, EPNs are suitable for the explicit modeling of the flow of control, resources, parts and information through a typical flexible manufacturing system. As far as analysis is concerned, a hierarchical methodology has been suggested for synthesizing EPNs and thus, the approach could also be classified at the category discussed in the next subsection. The methodology uses MGs for constructing the final EPN model of the system. The properties of the final net components (MGs) are preserved, in order to ensure that the final model will be error-free, live and consistent. Furthermore, in MGs all tokens at a place are expected to move simultaneously and therefore, the composite EPN model will not suffer from stagnation of tokens. This limitation is often associated with pure HPNs, since some tokens are stagnated at a place, as the succeeding transition is unable to fire.

4.3. The extension of structuring mechanisms (hierarchical high-level PNs–HHPNs)

HPNs represent a single and flat view of the modeled system, since their formalism does not support any structuring (hierarchy) mechanism. The requirements for visualizing selected parts of a system at varying levels of abstraction, thus facilitating an incremental design process, and allowing for reusability of net components, have motivated the introduction of the class of hierarchical high-level PNs (HHPNs). It should also be emphasized that the embodiment of a structuring mechanism does not extend the computational power of HPNs, but facilitates well-structured refinement and abstraction operations, allows reusability and bottom-up or top-down design, supports encapsulation (detailed information can be hidden in a well structured manner), and finally provides a clear separation of all system components.

In general, a HHPN allows net components to be connected (and thus to communicate with each other) by merging of transitions, places or arcs. Composition by merging transitions preserves the net properties and consequently facilitates the analysis process. The main limitation of this approach is that it leads to tightly coupled net components, since each component net does not clearly correspond to an autonomous subsystem; the set of all possible transition firing sequences of a net component represents both the internal behaviour and the interaction with the other net components. Composition by merging places allows for communication between subnets by resource sharing. This approach conforms directly to the natural perspective of PN theory (that is, transitions are active entities which communicate via passive entities called places), but leads to additional complexity, because correct synchronization must be ensured. In addition, strong coupling possibility between net constituents still exists. Finally, composition by merging arcs supports the loosest coupling between net components. Following this technique, communication between subnets is realized by sending and receiving messages. A representative example of connecting nets by arcs is the Cooperative Nets model (Sibertin-Blank, 1993, 1994) (the reader is referred to the subsection about the object-oriented extensions of HPNs).

A typical example of HHPNs is the hierarchical CPN (HCPN) (Jensen, 1990, 1995; Huber et al., 1990). A HCPN consists of a number of hierarchically interrelated subnets, called *pages*, which represent a substitution of transitions. A hierarchical transition can be replaced by a page in order to give a more detailed description of its internal transition firing sequence. Thus, a model is described as a set of related subnets (drawn on separate pages).

In a HCPN, the hierarchy mechanism is mainly used for specification purposes. Composition operators are applied directly to the static net structure and the analysis of the system dynamic behaviour and properties can be performed only on the completely refined executable net. As a consequence, the complexity of finding the reachability set (i.e., all possible reachable markings) grows significantly. This is the main reason that alternative models have been defined, which deal with this problem by including structuring mechanisms and supporting not only well-defined substitution of places and transitions, but also analysis techniques. For example, Valmari (1993) presented a synchronous composition of PNs based on the equivalence notion and composition operators that originate from the theory of CSP (Hoare, 1985). The reachability set of the composed net is generated by composing equivalent reachability sets of the component subnets, while behavioural preserving reductions are performed between refinement steps. Three composition operators have been defined, namely the *parallel composition*, the *hiding* and the *renaming* operator. These operators are applied directly not to a HPN but to a labelled (inscribed) PT-net model, in which labels (inscriptions) from a finite alphabet are associated

with net transitions. The first operator defines synchronous communication. Subnets communicate with each other by executing transitions with a common label. Communication takes place only if every subnet is ready to communicate and affects all the communicating subnets. The hiding operator is used to hide some internal events (the labels of the corresponding transitions become invisible), and thus to maintain a black-box view of system semantics. Finally, the renaming operator replaces some transition labels by new labels in order to be reused in different contexts. To summarize, this approach relates semantics of the process algebra theory (CSP) to the PN theory, and thus it can be also classified into the category of integrated approaches based on process algebras. However, the possibility of the reachability set state explosion in each isolated subnet still exists and therefore the approach is rather limited to subnets with small reachability sets.

More recently Bucholz (1994) presented an alternative methodology for compositional reachability sets generation, where the hierarchy mechanism defines an asynchronous interaction between subnets. The proposed HHPN is a generalization of hierarchical CPNs and results in a tree structure, where each node is a HCPN. For each node two kinds of views are defined: a detailed view describing the local behaviour and an aggregated view on the nodes above and the nodes below. Consistency between the detailed and aggregated view has to be assured. This can be achieved by checking the behaviour of detailed against the aggregated views. In comparison to other HHPNs, Bucholz's model appears more complicated from a practical point of view. However each subnet is executable and analyzable, while unrestricted refinements are allowed. In contrast to the Valmari's approach, the asynchronous interaction between a subnet and its environment is specified using input and output ports (so called place and transition borders), while the synchronous communication is modeled in the parent net (subnet environment), which is responsible for synchronizing the communication. The whole reachability set can be composed of much smaller parts because reductions performed on lower-level subnets reduce the reachability sets of higher-level ones. However, Valmari (1994) proposed a model for representing asynchronous and not only synchronous communication. In this approach communication takes place via places (instead of using transitions), an idea which is more natural to the original PN theory point of view.

Another model which could also be classified into the category of HHPNs is *interactive multi flow graph* (IMFG), a HHPN which has been introduced for designing interactive applications (Kameas et al., 1994; Kameas, 1995). An interactive application is considered as a set of communicating IMFGs. An IMFG is composed of active components, called *actors*, and passive components, called *links*; tokens are used to represent flow of different kinds of information. Actors are of four types: *context* actors, which represent integral user goals and are hierarchically decomposed into simpler goals, *action* actors, which represent application functions that must be eventually executed, *library* actors, which represent the goal decomposition schemata, and *group* actors, which are used to represent the PN refinement property. Links are of four types: *event*, which accommodate the user- or system-generated events that trigger actor firing, context, which accommodate the goal-achieving strategy that users employ, *data*, which accommodate the data that flow in the application, and *condition*, which are used to accommodate the status of the application and the control flow. Different kinds of usage are permitted on tokens: normal, OK, debit and read-only.

The model supports the separation of user interface operations from application functions, and the reusability of integral dialogue parts, while it provides designers with the ability to evaluate multiple application perspectives. State is represented by using a list of ready-to-fire actors. State transition is modeled with an actor firing policy that adds or removes actors from the list based on their firing rules. The latter are functions over actor input and output links. IMFG is based on HPNs (mainly CPNs) to which it adds structure and semantics. Furthermore, IMFG adds memory to PN formalism in the form of actor-ready list and distinguishes tokens at semantics level. The model supports structured interaction by providing two goal decomposition policies: AND decomposition and OR decomposition. Refinement of context actors is context-based and event-driven, and represents the hierarchical refinement of user goals into the strategy that must be employed in order to achieve them. However, in order to formally analyze the application under design, an IMFG must be transformed into a basic PN by removing most semantical and phenomenological aspects. Then the model becomes more general and suitable for modeling systems that lie on the event-action paradigm (e.g., discrete event dynamic systems).

4.4. Extensions supporting the representation of uncertain (fuzzy) information

Fuzzy PNs (FPNs) is an application specific PN-based approach developed to represent uncertain operations and approximate conditions in areas such as robotics, flexible manufacturing systems and fuzzy controllers. Because of this limited application range, FPNs will not be contrasted with the other categories. However, in this subsection a comparative assessment of the various proposed FPNs will be given.

The model was introduced by Looney (1988) for the specification of rule-based reasoning using propositional

logic. Places are interpreted as conditions having fuzzy truth values (tokens), while transitions represent the fuzzy decision values of rules. Reasoning in a FPN can be performed by iteratively maxing (generalized OR) and mining (generalized AND) transitions and fuzzy truth values of tokens, respectively.

In Garg et al. (1991) a variation of FPNs was proposed in which *negative arcs* represent negation of a proposition (place), while input and output places of a transition are conjuncted and disjuncted, respectively. In addition, the proposal includes an algorithm to determine possible inconsistencies in a fuzzy knowledge base, which is based on a set of reduction rules. The algorithm initially substitutes each negative arc with an equivalent inverted normal (positive) arc and subsequently performs specific net reductions. Finally, the appearance of a "*Null Transition*" (with a truth value greater than 0.5) will indicate whether the fuzzy knowledge base is inconsistent. As far as theorem proving is concerned, the refutation approach is followed: the negation of the theorem to be proved with truth value 1.0 is added to the given set of fuzzy rules; the new set of rules is checked for inconsistencies and if it is inconsistent, then the given theorem is true.

The above approaches are useful in rule-based Expert Systems modeling in order to help in automating the decision making process. However, their basic limitation is that they use real numbers in [0, 1] (crisp values) to describe the truth values of conditions and rules. In order to overcome this limitation, Cao and Sanderson (1993) introduced a generalized definition of FPNs which uses not only crisp values but also fuzzy sets. More specifically, they proposed the following types of fuzzy variables:

- a *local fuzzy variable* associated with a place, denoting uncertainty of the local variable which is attached to the given place (i.e., such a variable represents information which locally affects an operation),
- a *fuzzy marking variable* associated with a place, denoting uncertainty that a token exists in the place (i.e., such a variable represents information regarding the system state),
- a *global fuzzy variable* associated with a token, representing a characteristic of a global operation of the modeled system (i.e., such a variable is used to represent the uncertainty in planning a sequence of fuzzy operations).

Three different possibilities have been defined for the enabling and firing rules in a FPN: a transition firing is performed in the same way with simple CE-nets and local fuzzy variables remain unchanged, the firing follows that of CE-nets and local fuzzy variables are changed, or the firing depends on the input variables and local fuzzy variables are changed.

In contrast to low-level PNs or HPNs, the model of Cao and Sanderson represents fuzzy information and reasoning. Therefore, the enabling condition for a transition depends on the local fuzzy variables or the global marking of the net. The effects of a transition firing depend both on the marking of its input places and on a reasoning mechanism inherent in the transition. Therefore, tokens in a place do not have to disappear. Although this approach seems to be the most general one in the class of FPNs, it does not yet take into account any interrelations among the different fuzzy variables, which is a topic for further research.

4.5. Integration with other specification methods

Approaches based on integration with other specification methods can be further classified into four subcategories as follows:

- extensions based on systems of communicating finite state machines,
- extensions based on process algebras,
- extensions based on abstract data types,
- extensions based on the object-oriented approach.

4.5.1. Extensions based on systems of communicating finite state machines

Systems of CFSMs (Finkel and Rosier, 1988) include inherent mechanisms for handling queues and therefore have been used to model distributed and/or parallel systems, where processes communicate by sending messages via first in first out (fifo) channels. Systems of CSFMs pose strict constraints on design, since they adopt the fifo channel as a built-in communication mechanism. *First In First Out Nets* (FIFO nets) (Finkel and Choquet, 1988; Finkel and Rosier, 1988) have been proposed as a generalization of both PNs and systems CSFMs. Places and tokens in a FIFO net represent fifo queues and letters, respectively. Transitions enqueue (or dequeue) words (i.e., sequences of letters) to (or from) a fifo queue. PT-nets, can be considered as FIFO nets the fifo alphabets of which have a size of one.

Although this generalization has been proven to increase the expressive power of PNs to the level of Turing machines, FIFO nets have so far gained only limited acceptance. This is because they assume a specific communication mechanism among processes (i.e., the fifo place). Perhaps the most important research direction in modeling with FIFO nets is to define subclasses where the various analysis problems (i.e., liveness, boundedness, reachability etc.) are decidable. These subclasses maintain the fifo nature of the communication mechanism, but restrict the acceptable sequence of messages that can pass through a fifo queue.

4.5.2. Extensions based on process algebras

Both PNs and process algebras (e.g., CCS (Milner, 1989) and CSP (Hoare, 1985)) deal with the specification and analysis of parallel and distributed systems. The

former focus on the graphical representation of a system, while the latter place emphasis on modular system design by using compositional operators. Research on exploiting the advantages of both formalisms leads to PN realizations which incorporate features met in process algebras. In particular, compositional operators developed for process algebras can be applied almost directly to PN transitions. It should be mentioned that such an application facilitates modular design, but does not conform entirely to the original idea of PN theory, which assumes that active components (transitions) communicate through passive components (places).

For example, CPNs have been extended with coloured communication *channels* (Christensen and Hansen, 1994). The concept of channels has been influenced by CSP and provides simplified representation for synchronous communication. Synchronized communication in CSP is achieved by *!* and *?* operators (Hoare, 1985). Let us consider for example a channel *c*, a message *m* and a data variable *r*. The notation *c!m* in CSP means "send the message *m* over the channel *c*", while *c?r* means "receive a message over the channel *c* and assign the message to *r*". Similarly, in CPNs extended with channels, a communication between two transitions is possible only if the one is a *!?*-transition and the other is a *?!*-transition. The direction of the communication is not specified and thus the approach uses *!?* and *?!* operators instead of *!* and *?* operators met in classical CSP. Although CPNs extended with channels are equivalent in behaviour with (and can be transformed) to CPNs, they decrease the number of the transitions and arcs required. In addition, communicating via channels in a CPN differs slightly from merging transitions, in two major points: first, in transition merging, all transitions share a common binding, while the bindings of transitions used in a channel are independent (besides the conditions specified in the communication expressions); second, transitions that communicate by using a specific channel may fire without involving any other transitions using the same channel. Therefore, a channel communication can be equivalently modeled by associating a transition fusion set (i.e., by further decomposing a transition) with each possible way of the communication. As far as analysis is concerned, place invariants have been defined for CPNs extended with channels.

Concluding, the approach is mainly influenced by the same motivations that led to the development of HHPNs (i.e., hierarchical representation and reusability). In comparison to the proposal of Valmari (1993) described before, both have originated from CSP to model synchronous communication via transitions. Moreover, both extend analysis methods of HPNs: CPNs with channels extend the concept of place invariants, while the approach of Valmari uses the reachability graph. However, the Valmari's approach seems less intelligible since it incorporates operators applicable to a labelled PT-net model rather than to a HPN.

Other interesting research directions in combining process algebras and PN theory are oriented towards the development of net semantics for process algebras and the representation of PNs in terms of CCS or CSP formalism. Such orientations are outside the scope of this paper since their main objective involves proving the correctness of specifications expressed in process algebra terms. Interested readers can find in Degano et al. (1988) an approach defining a CE-net from a CSP process and in Dietz and Schreiber (1994) an attempt to translate an arbitrary PT-net into terms of the CCS formalism.

4.5.3. Extensions based on abstract data types

PNs and Abstract data types (ADTs) (Ventouris and Pintelas, 1992) offer complementary aspects of a system specification. PNs emphasize the specification of parallel and communicating activities using a graphical notation, but they are weak in dealing with data types and formally defined modularity. Although the various proposed HPNs support process abstraction, they often have a limited ability in supporting data abstraction. On the contrary, the algebraic formalism of ADTs is suitable for the representation of system functional behaviour, but they are often weak in representing parallel and synchronized activities. In addition, the most popular formalisms for data abstraction, such as the algebraic ones, are weak in supporting process abstraction, a significant need in concurrent systems modeling. The integration of PN theory with the algebraic theory of ADTs has led to nets having the data tokens of an ADT.

A typical example of a combination of both approaches are Predicate-Event Nets (PrE-nets) (Schmidt, 1991). PrE-nets form the basic part for a formal specification and design language called *SEGRAS* (Kramer, 1989) which tries to unify algebraic specifications and HPNs. The algebraic axiom part of the language is used to describe and analyze the properties of the ADTs comprising the system (i.e., the static data structures on which a system operates), while the PN part (i.e., a PrE-net) is used to specify the system initial state and the accessibility of the other states from the initial one (i.e., the dynamic system behaviour). A PrE-net specifies the interaction mechanism between the system relevant states and state-changing operations. The interaction protocol provides an abstraction from all parallel system activities. Therefore, PrE-nets are used to represent parallelism among the operations on abstract objects.

The approach followed in *SEGRAS* language gives readers who are unfamiliar with formal algebraic methods the possibility to get a "clear" view of the system behaviour. In comparison to other HPN models, PrE-nets are a variation of PrT-nets, which allows algebraic specification of the data that is flowing in the

net. PrE-nets inherit from the algebraic specifications the clear separation between syntactic and semantic information. Such a model syntactically is a PN with inscriptions on an ADT signature, while semantically its behaviour is modeled with the flow of black tokens as in simple CE-nets. Therefore, certain operations used in algebraic specifications (e.g., extension, combination and renaming) can be used to define a modular composition of a large system model. The basic limitation of the approach is that it is not fully integrated since the algebraic and net parts of the *SEGRAS* language are, to a certain degree, separated.

Another integrated model which uses algebraic terms instead of set theory terms for specifying the individual tokens flowing into the net is *OBJSA* net (Battiston et al., 1988). A token consists of a *name part*, which represents token instances and is not modified by transition firing, and a *data part*, which models the data structure of a sequential component of the system to be modeled and can be modified by transition firing. Thus, individual tokens are instances of a parameterized object consisting of name and data parts. Like PrT-nets, predicates can be associated with each transition. These predicates introduce constraints on the tokens enabling the transition, and define functions, which show how transition firings modify the object data parts.

4.5.4. Extensions based on the object-oriented approach

It is well-known that in the object-oriented paradigm (Booch, 1986), an object is actually an ADT which communicates with other objects through *methods* by using *messages*. The object-oriented approach mainly extends the concept of data abstraction through inheritance, that is higher level ADTs (objects) transfer some of their functionality to those at lower levels.

A typical example of combining PNs with the object-oriented approach is PROcess-Translatable PNs (PROT-nets) (Baldassari and Bruno, 1988, 1991). Two are the main characteristics that distinguish them from other HPNs: firstly, they can be easily translated into actual programs; secondly, they have an object-oriented structure. However, the formalism being used is similar to other HPN models. Places stand for process states, tokens represent process instances and contain data, and transitions describe synchronization among processes. Actions as well as predicates that impose further conditions on firing, besides the firing rules, are associated with transitions. Each object is represented by an autonomous net exchanging messages (tokens) with other objects. Objects can be decomposed to form other objects in a hierarchical way. In a similar way with *SEGRAS* language, the control structure specified by a PROT-*net* is separated from the textual code which performs well-defined sequential activities. Finally PROT-nets, like IMFGs, can describe multiple perspectives of a system, by extending state transitions and data-flow diagrams.

Petri Net Objects (PNO) (Bastide and Palanque, 1990) are used to design event-driven user interfaces. Firstly, an interface is viewed as an object, the methods of which can be interactively triggered by the user. Secondly, the behaviour of each object (i.e., the spontaneous object activity, the effect on the availability of object operations and the effect of operations on the internal object state) can be described by a PNO. The model seems more restricted compared to the IMFG model, since PNO emphasize modeling of the user interface operations and not of the user-application interaction.

Up to this point, several HPNs have been discussed which try to deal with the specification of the interaction among the net components (i.e., how net components communicate each other and how they are structured to form higher or lower components). Nevertheless, composition is often performed by merging places or transitions, an approach that results in strong coupling among net components. Three integrated formalisms which adopt features of the object-oriented paradigm and achieve loose coupling are *Communicative Nets*, *Cooperative Nets* and *Cooperative Objects* (Sibertin-Blank, 1993, 1994). In these approaches the communication is realized by a message sending (i.e., an arc from a transition to a place) and receiving (i.e., an arc from a place to a transition) schema. The net components are linked by arcs or, equivalently, by fused places. Objects stand for entities which belong to object classes. An object type (class) defines the structure of its objects. A Communicative Object represents an instance of its object type, whereas a Communicative Net is a common structure containing all instances associated with an object type. Therefore, the actual state of an object corresponds to the marking of a Communicative Net of its type. Communicative Objects interact by message sending, while Communicative Nets include *accept-places*, where tokens may be put by any net. However, for synchronization purposes, no net may both put in and take tokens from the accept-places. Finally, the Cooperative Objects formalism supports design at higher levels of abstraction. For this purpose, the formalism adopts the client-server protocol to construct higher-level nets where clients and servers are dynamically created and have communication dynamically set up (Sibertin-Blank, 1993).

5. Comparative presentation

In Section 3, the producer-consumer synchronization problem has been used as a basis for comparing low-level PNs. In the current section, the same problem, in a broader sense, is considered for the comparative study of HPNs, from the communication point of view: representative HPNs will be used to specify communication

between producer and consumer processes which wish to cooperate by sending and receiving messages through a communication channel.

Since it is difficult to examine the functionality of a variety of HPNs based on the characteristics of a single example (even if versions of the producers-consumers problem are presented an increasing order of complexity), comparison will continue by considering aspects of another classical synchronization problem, the readers-writers system. In that problem, reader processes access shared data but do not alter it, while writer processes change the values of shared information; any number of readers should be allowed to proceed concurrently in the absence of a writer, but only one writer may execute at a time while readers are excluded.

Finally, in order to present the applicability of HPNs in specific application areas, an example of using an application oriented HHPN (IMFG) will be presented. Although IMFG does not directly address a specific synchronization paradigm, it has been selected in order to show how a complex and asynchronous process, such as a user-system interaction, can be modeled in a structured way.

5.1. The producers-consumers problem

Individual token nets (*ITNs*): In the modeling example of a producers-consumers system, a pure HPN, like an ITN, can be appropriate to show who transacts with whom (i.e., the individual producers-consumers and objects can be explicitly modeled) and to provide simplicity and compactness in the representation. Fig. 5 shows an ITN which is analogous to the CE-net of Fig. 1 and models the producers-consumers system. Now, producing/consuming objects are messages, transition *deliver* becomes *send* while transition *receive* remains *receive*. The ITN models the distinguishable producers and consumers as individual tokens (tokens *p* and *c* respectively), takes into account the type of messages (according to the condition of transition *send*, four-bit messages are transmitted) and uses variables (*x*, *y* and *z*) to label the net arcs. For example, transition *send* can fire with $x = p$ (producer *p* sends messages) and an arbitrary four-bit message for *y*. A further extension to Fig. 1 is that there is a place (*storage*) that stores all the received messages. The formalism of ITNs appears very similar to the one of CPNs. However, ITNs may modify the transition firing rule by attaching explicitly an operation (action) to a transition. This is the case of the transition *send* which performs a specific action when it fires (a logical AND between message *y* and the message string 0101).

Coloured petri nets (*CPNs*) *and CPNs extended with coloured communication channels:* The net in Fig. 6 demonstrates how transitions in a *CPN* can communicate *synchronously* through a coloured communication channel. Transitions *send* and *receive* use a coloured communication channel ch. The channel, as well as the sending/receiving messages are of the same colour set (type) called *DATA*. The communication is possible because *send* is a *!?*-transition and *receive* is a *?!*-transition and they both use the same channel ch. *PROD* and *CONS* colour sets represent the producers and the consumers respectively, while inscriptions (so called communication expressions) attached to *send* and *receive* transitions specify the channel ch and the message communicated through this.

Communication through channel ch is enabled if and only the following conditions hold:

- there are sufficient tokens of the correct colours in the input places with colour sets *PRODxDATA* and *CONS* respectively (producer *p* is ready to send a message *x* to consumer *c*),
- the communication expressions *x!? ch* and *y?! ch* yield the same value when they are evaluated in the bindings for which the two transitions fire.

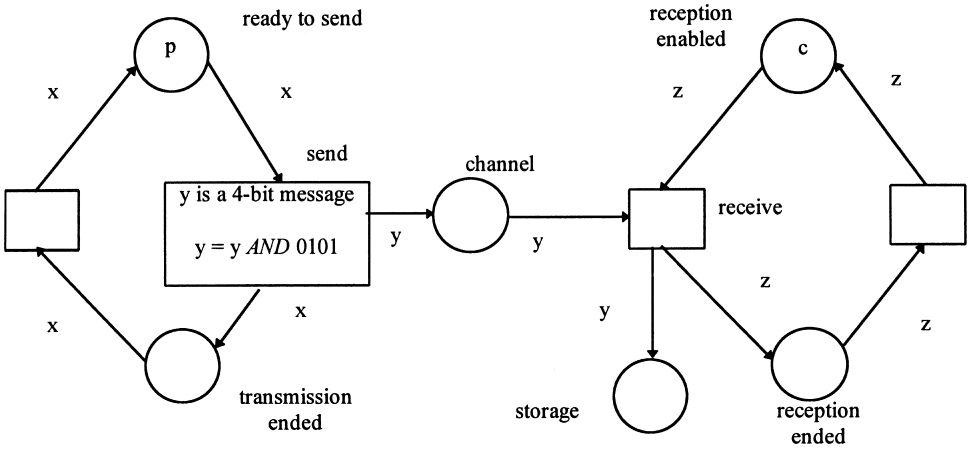

Fig. 5. An ITN modeling distinguishable processes which send and receive messages from each other.

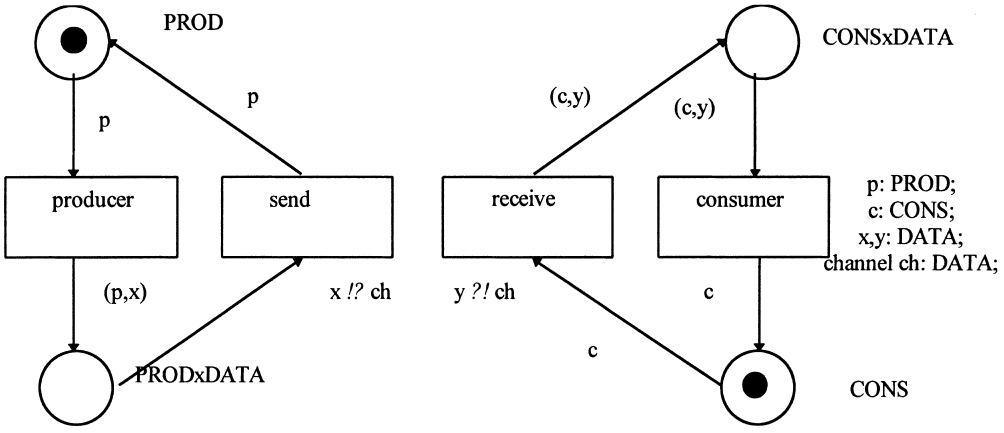

Fig. 6. A CPN extended with channels.

In the general case, communication between two communicating transitions is bi-directional. In Fig. 6, however, it should be noticed that x appears in the input arc expression of *send* and y appears in the output arc expression of *receive*. Therefore, the colour of the input token of *send* specifies the colour of the output token of *receive* and communication takes place from *send* to *receive*.

Fig. 7 shows the behaviourally equivalent CPN of the net presented in Fig. 6. The communicating transitions have been merged in a single transition called *communicate* with an inscription $[x=y]$ denoting that communication expressions must evaluate to the same value. If one is interested in the representation of a multiple producers-consumers system, each producer and consumer will be represented with an individual subnet (page in terms of Hierarchical CPNs–HCPNs) of Fig. 6. For example, the net in Fig. 8 demonstrates two producers and two consumers which communicate through the same channel ch. An equivalent CPN would suffer from complexity, since 4 (2 * 2) transitions and many crossing arcs would be required to represent the communication.

Hierarchical CPNs (HCPNs): A hierarchical HPN, such as the HCPN model, will be used to describe a more complex instance of the same problem. A sequence of messages is sent from one site (*Sender*) to another site (*Receiver*) via a *Network*, where messages can be delayed or lost. In addition, *Receiver* sends acknowledgements which *Network* transmits to the *Sender*. The first and more abstract *page* of the corresponding HCPN is depicted in Fig. 9, and consists of the *Sender*, the *Network* and the *Receiver* part (declarations of colour sets, variables, constants and functions are omitted for the sake of simplicity). A HCPN contains a number of interconnected subnets called (sub)pages, which result from substituting transitions. Places surrounding a substitution transition are called *socket* places (e.g., places *A*, *B*, *C* and *D*), while a each page contains a number of *port* places; a socket place has a corresponding port place with the same marking. Socket and port places can be considered as composition operators which facilitate

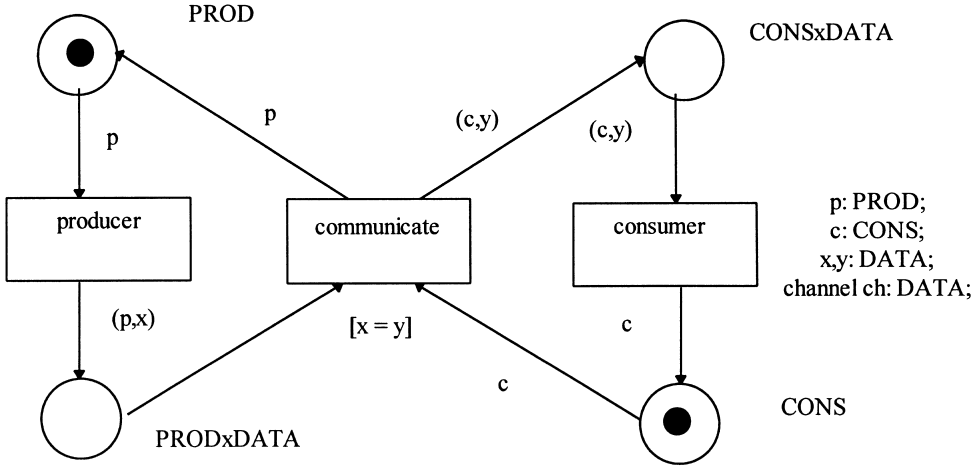

Fig. 7. A CPN equivalent to the net of Fig. 6.

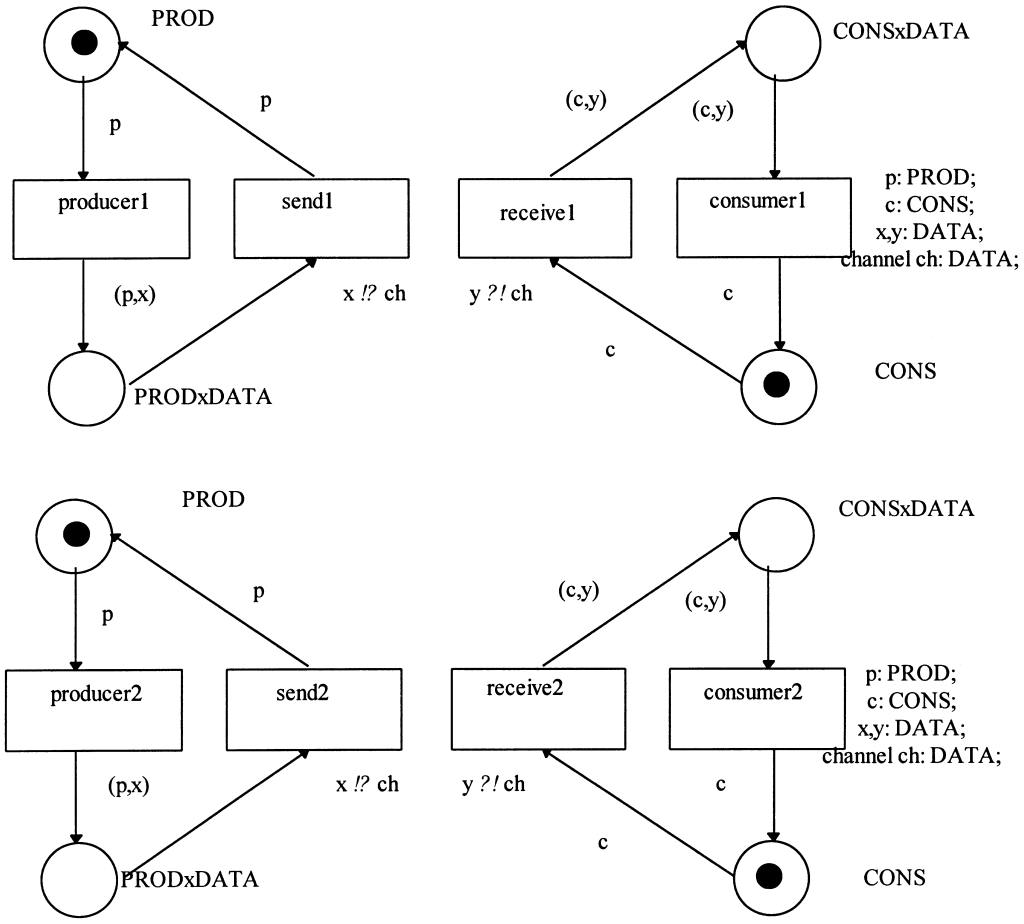

Fig. 8. A CPN extended with channels modeling two producers and two consumers.

systematic net synthesis: they provide modularity just like the binding of actual to formal parameters that takes place in a function call met in any procedural programming language. Fig. 10 presents the corresponding pages of the HCPN presented in Fig. 9.

The *Sender* page consists of:

two transitions, namely *Send Message* and *Receive Acknowledgement*, which are responsible to send messages (by creating a copy of the message into place *A*) and receive acknowledgements, respectively,

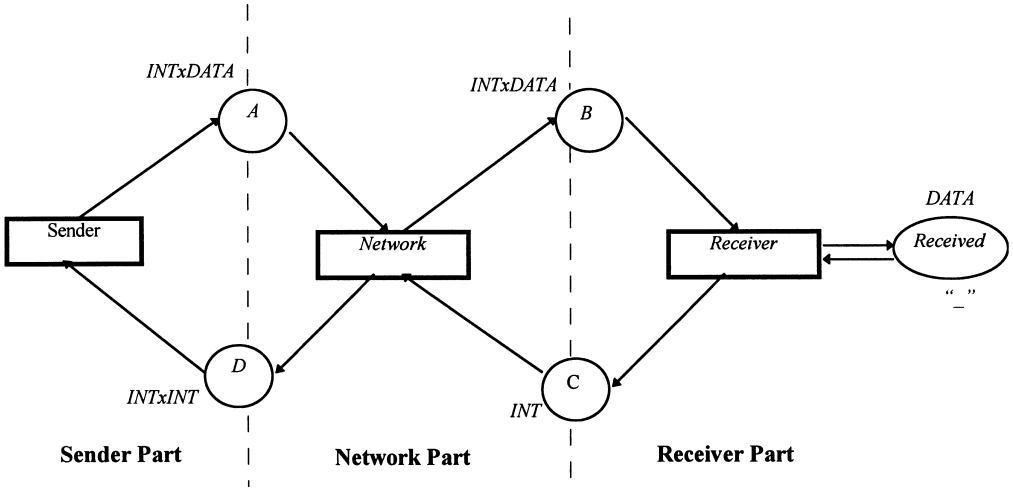

Fig. 9. The first page of a HCPN modeling network communication.

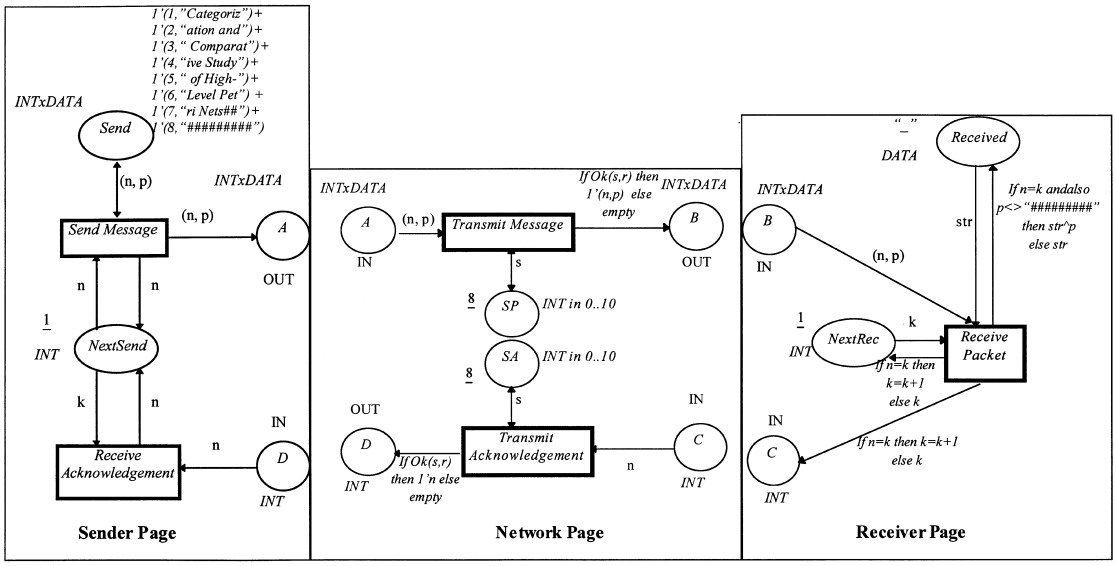

Fig. 10. Pages of the HCPN depicted in Fig. 9.

the place *Send* which stores the messages to be sent (each token in this place contains the message number and the data contents of the message);
the place *Next Send* which stores the number of the next message to be sent (at the initial marking this number is equal to 1 and it is increased each time an acknowledgement is received).

Firing of *Send Message* neither removes the message from *Send* (because of the bi-directional arc) [5] nor increases the message number at *Next Send*, in order to consider the case that a message is lost and need to be resent. The same message is transmitted until there is an acknowledgement that specifies that this message has been successfully received.

The *Receiver* page contains:

- the transition *Receive Message*, responsible to both receive messages and send acknowledgements;
- the place *Received*, which stores the content of the received message (a single token that represents a the concatenation of the text strings contained in the received messages – any duplicates and messages received out of order are ignored). At the initial marking this text string is empty ("_"), while at the final marking, the string "Categorization and Comparative Study of High-Level Petri Nets" is expected;
- the place *NextRec* containing the number of the next message to be received. At the initial marking this number is equal to 1 and it is increased each time a message is successfully received. *Receive Message* whenever receives a message, checks if the message number n is equal to the number k in *NextRec*. If this is the case, the number in *NextRec* is increased by 1 and the message text string is concatenated with the current text string in *Received* (unless it is equal to "##########" denoting the end of the message sequence). Otherwise, the message is ignored and the number in *NextRec* remains the same. In both cases an acknowledgement is sent with the number of the next message which should be transmitted.

The *Network* page consists of two transitions, namely *Transmit Message* and *Transmit Acknowledgement*, responsible to transmit messages and acknowledgements respectively. The Boolean expression *Ok* (s, r) at the output arc of *Transmit Message* is true when r is less than or equal to s and means that the message has been successfully transmitted. [6] Likewise, *Ok* (s, r) is used at the output arc of *Transmit Acknowledgement* to determine the probability that an acknowledgement is lost. Finally, there are four interface (port) places, places A and D between the *Sender* and the *Network* pages and places B and C between the *Network* and the *Receiver* pages.

The example demonstrated that the choice of a HHPN to model a complex instance of the producers-consumers problem facilitates systematic top-down design by substituting net transitions. Such a choice becomes more advantageous in more complex situations (e.g., in case of multiple receivers).

[5] Such an The bi-directional arc can be equivalently interpreted as a *reserve* arc (Lakos and Christensen, 1994), since firing of transition *Send Message* does not change the marking of place *Send*.

[6] r is supposed to be a random integer value between 1 and 10 and the probability of a successful transmission is specified by the token in the place *SP*. At the initial marking this token has a value of eight, and thus this probability is equal to 80%.

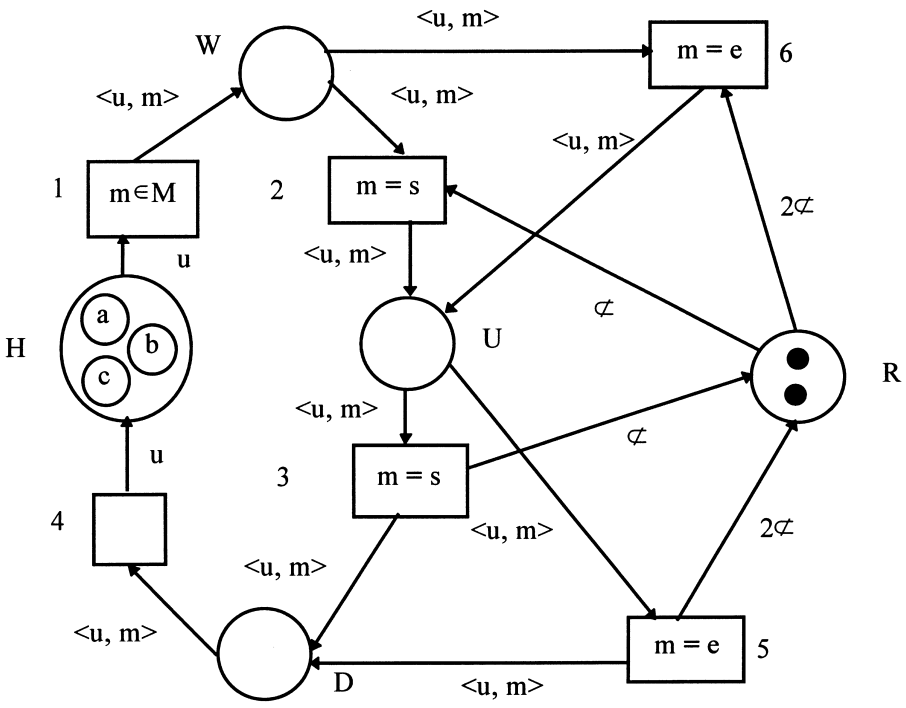

Fig. 11. A PrT-net representing the readers–writers system.

5.2. The readers–writers problem

Predicate-Transition Nets (*PrT-nets*): Fig. 11 demonstrates a PrT-net modeling an instance of the readers–writers problem. We assume that there are three individual users (readers or writers) $\{a, b, c\}$ of a single resource R, which may be used in either "exclusive" or "shared" mode. These two different modes of operation are denoted by the identifiers s (shared) and e (exclusive). Two individual variables, denoted as u and m, are used to represent an individual user and the mode of resource usage respectively.

The net consists of six transitions and five places H, W, U, D, R. Places correspond to predicates and model the following problem aspects:

- H (u) denotes that user u (reader or writer) has nothing to do with the resource.
- W (u, m) denotes that user u intends to use the resource in mode m.
- U (u, m) denotes that user u is using the resource in mode m.
- D (u, m) denotes that user u has finished using the resource in mode m.
- R ( ) represents the number of times the resource is still available for shared usage (i.e., the number of tokens accommodated in the resource).

In a PrT-net representation, places correspond to predicates and they are marked with their extensions (sets of tuples of individuals). For example, at the initial marking, predicate H is marked with the user set $\{a, b, c\}$ in order to indicate that initially no action takes place. In order to include ordinary places (like R), they are treated as zero-place predicates. Arcs are labelled by sums of tuples of the individual variables u and m, while the "zero-tuple" is denoted by $\not\subset$. All transitions, except the fourth one, are labelled with logical formulas. For instance, the second transition is labelled with $m = s$ to indicate that it may fire only under the "shared" operation mode. An instance of a transition is generated by replacing the arc variables by individual symbols (tokens a, b, c and s, m). For the sake of simplicity, Fig. 11 does not represent the "shared use" and the "exclusive use". This situation is explicitly depicted in Fig. 12. The presented subnet models the fact that place U cannot accommodate two pairs, one of which has an e at its second position, and thus if one user is using the resource in "exclusive" mode, no other user is using the resource at all. Net analysis by invariants (Genrich and Lautenbach, 1981) can show that transition of Fig. 12 will never be able to fire (i.e., it is a *dead* transition). Modeling with PrT-net retains the identity of the read-

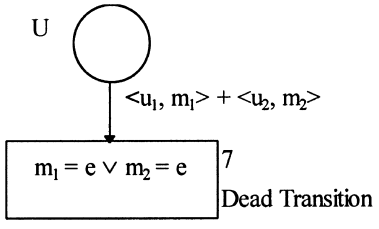

Fig. 12. An additional subnet representing the exclusive resource usage.

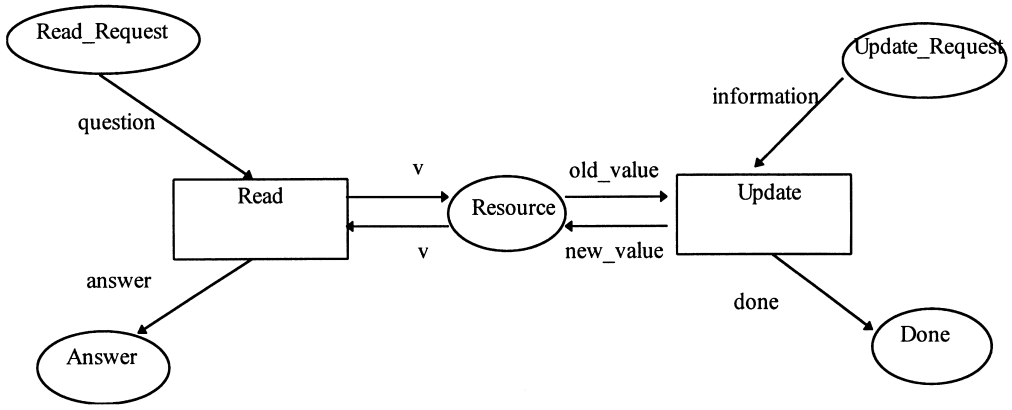

Fig. 13. A CPN describing the readers-writers system.

ers–writers, even in a highly condensed representation. If a PT-net has been used to represent the same problem, the individual users could not be identified; a PT-net would represent only the number of the involved users. However, in such a model it is possible to unfold the user part (i.e., to design a CE-net), in order to retain the users' identity. Unfortunately, the size of such a model would be very large.

CPNs: Fig. 13 demonstrates a CPN modeling the readers–writers problem in its general form. Any number of users (readers) are allowed to read data from a resource, but only one at a time is allowed to update the data. The shared data object is modeled as a token in the place *Resource*. Each firing of *Read* and *Update* transitions monopolizes the token in the place *Resource* (i.e., uses the resource), and thus simultaneous access is prevented. Although the detailed declarations are not presented for the sake of simplicity, one can see that functions (e.g., *question*, *answer*, *old_value*, *new_value*, *information*, *done*) instead of sums of tuples of variables are used to label the arcs. Therefore, the net of Fig. 13 is more intelligible than the corresponding PrT-net of Fig. 11. In addition, the CPN now represents the general (and more abstract) case in which the number of the involved users is not specified.

CPNs extended with test arcs: Although the CPN of Fig. 13 ensures data integrity, it does not represent (true) concurrency between simultaneous firings of the *Read* transition. In particular, firing of either transition *Read* or transition *Update* will consume the token of the place *Resource*. This situation is completely appropriate for the single-write case. However, the multiple-read case (i.e., multiple users are allowed to read data simultaneously) is not modeled. One can solve this problem by associating a limit with the number of simultaneous reads and by creating sufficient tokens for all of these in the place *Resource*. Each firing of the *Read* transition will access only a single token, while each firing of the *Update* transition will change all tokens. However, such an approach will lead to a complex net containing additional inscriptions and representing an exceptional situation (a limited number of simultaneous reads).

A more appropriate selection for this situation seems to be the embodiment of a *test arc* between the *Read* transition and the *Resource* place (Fig. 14). Such an approach sufficiently models concurrency (i.e., simultaneous firings of the *Read* transition) and prevents the net from additional complexity. Several instances of the *Read* transition may simultaneously occur, while no one

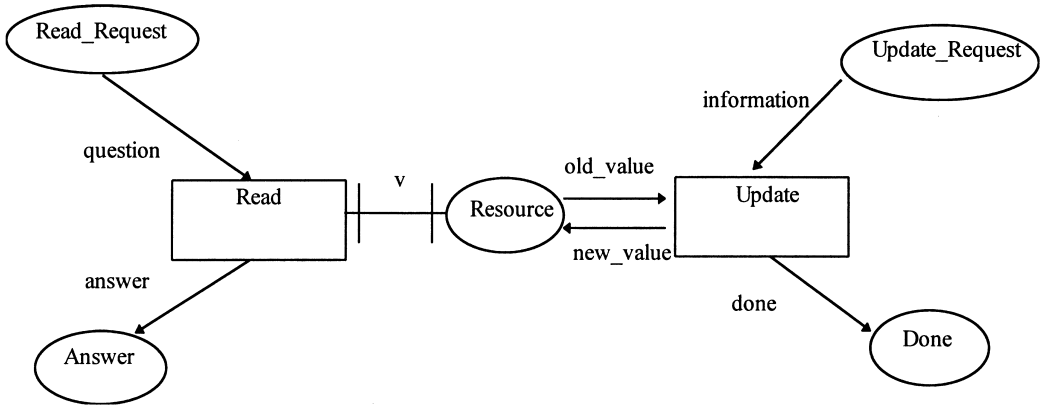

Fig. 14. A CPN with test arcs describing the readers-writers system.

of them is concurrent to an instance of the *Update* transition. This is because the test arc cannot change the marking of the *Resource* place, as well as it has lower priority than the two ordinary arcs between the *Resource* place and the *Update* transition (i.e., firing of *Update* still monopolizes the token of *Resource* making *Read* not able to fire).

5.3. Application specific example: modeling user-system interaction using IMFG

In the following, the IMFG model is used to demonstrate how hierarchy and composition mechanisms can be applied to modeling user system interaction via a HHPN. Fig. 15 shows an IMFG which models an aspect of the user-system interaction in an authoring environment, called *GENITOR* (Kameas and Pintelas, to appear), which supports the development of intelligent tutoring applications that use a learning cycle composed of pedagogical stages. This view is from the authors' goal-plan decomposition, where the authoring goal of stage definition is decomposed into a set of subgoals, at least one of which must be achieved (OR decomposition).

The overall goal is represented by context link *c*. First of all, *GENITOR* must be present, so one system action (event link *e*: load *GENITOR*-enable modification) is necessary. Then *c* is decomposed into two independent subgoals indicated by *A*1 (create new stage entry) and *A*2 (edit an existing stage entry). These subgoals can be achieved by causing events *ue*1 and *ue*2, which enable the firing of *A*1 and *A*2, respectively. Tokens are produced at the output event links *ue*1(OK) and *ue*2(OK) (which means that the user event has been processed successfully) and output context links *c;A*1(OK) and *c;A*2(OK) (corresponding subgoals have been successfully achieved).

In Fig. 16, the *A*1 context actor is refined into *VA*1, which represents the subgoal "fill entry components" and *A*1.3, which represents the subgoal "insert stage entry". Group actor *VA*1 is shown analytically in Fig. 17. Authors may fill a stage entry components either by using the system-provided user interface controls (context actor *A*1.1) or by typing these values directly into the provided area called CDA (action actor *A*1.2). Context actors *A*1.1 and *A*1.2 represent exactly these two capabilities, which may be indicated by using the controls menu or by typing inside CDA (actions that are represented by the user event links *ue*4 and *ue*5, respectively).

Tokens are being produced at *c;A*1;*A*1.1(OK) and *c;A*1;*A*1.2(OK) after the successful achievement of the two subgoals, and at *ue*4(OK) and *ue*5(OK) after the corresponding events have been successfully processed. Furthermore, independently of which of the two ways authors will select to fill the stage entry components, the context link *c;A*1;*VA*1(OK) will carry a token representing goal achievement. Authors must click on Insert button (user event *ue*3) in order to cause the firing of action actor *A*1.3, which will permanently record the new entry. Finally, the token produced at *c;A*1(OK) shows that the overall goal (create new stage entry) has been achieved correctly.

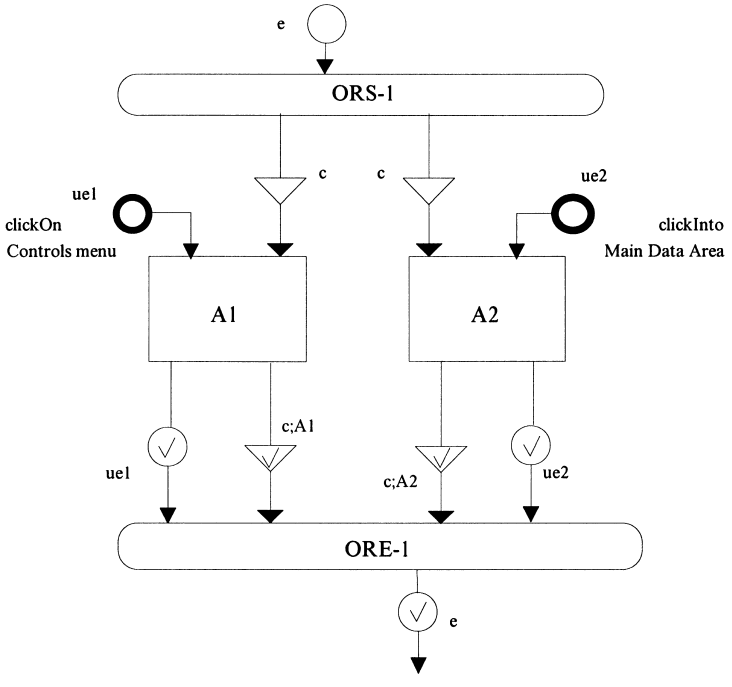

Fig. 15. IMFG 1 – Interacting with *GENITOR*.

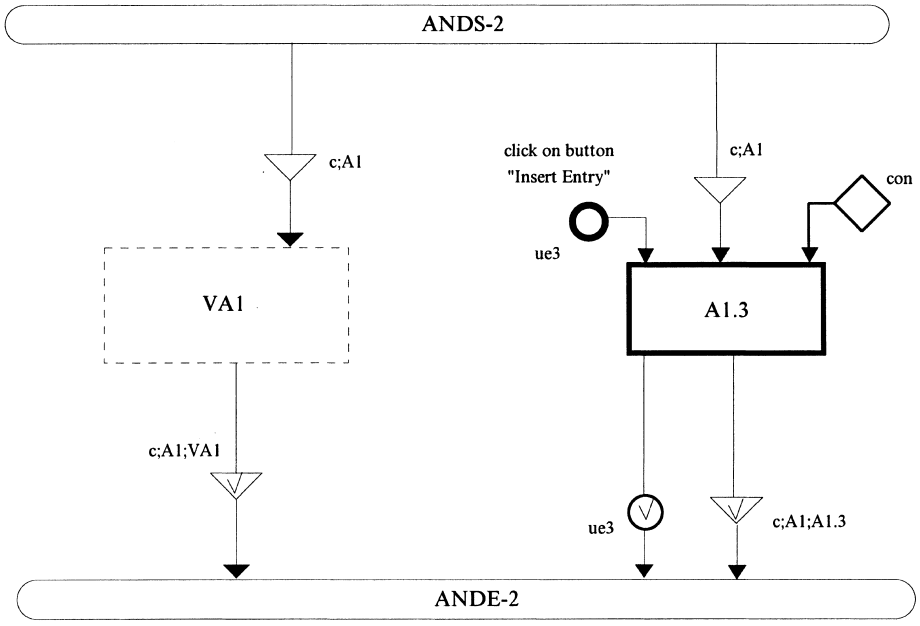

Fig. 16. IMFG 2 – Refinement of context actor *A1*.

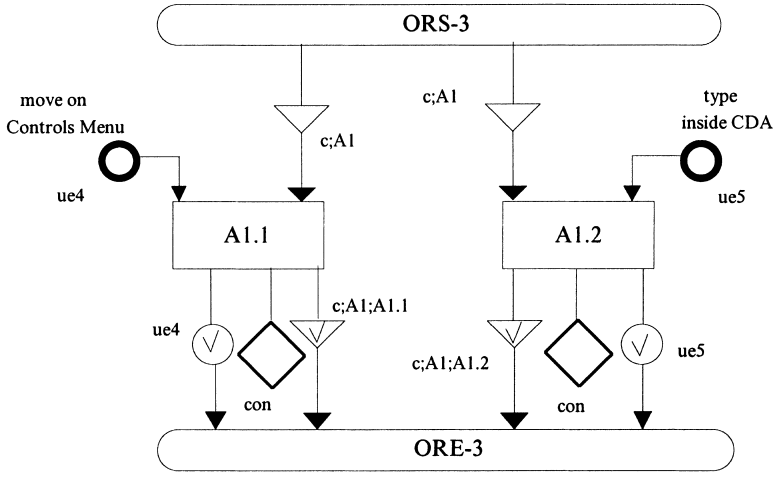

Fig. 17. IMFG 3 – Refinement of group actor *VA1*.

Note that goal "create new entry" has been decomposed into subgoals "fill entry components" *then* "insert entry". Subgoals are ordered with the use of condition con (entry defined), which gets a token after the entry components have been filled. Achievement of these subgoals requires an action plan of the form: [(use Controls Menu OR type inside Current Data Area) AND press insert entry button].

6. Comparison using general criteria

In Sections 4 and 5 we presented a set of categories for classifying several PN extensions and evaluated representative PNs expressed in these general approaches. In the current section we will compare and contrast the presented categories using a set of application criteria which will give reader a better overview of each category (Table 1). Some of the criteria are quite general and have been also applied to a survey regarding formal methods for Abstract Data Types (Ventouris and Pintelas, 1992); the rest were selected to evaluate additional issues concerning modeling using PNs. These criteria include: descriptive power, degree of difficulty in mastering a method, compactness, ease of analysis, degree of supporting encapsulation-abstraction-refinement, and degree of specifying communication. Besides these criteria discussed in detail below, interested readers may find in Gerogiannis et al. (1995) a classification based on categories such as interpretation, characteris-

Table 1
Comparison of the categories of PN extensions

Criterion	PNs	HPNs	HPNs with modified semantics	HHPNs	Extensions based on integration with other specification approaches
Descriptive power	Low	High	Very high	Very high	Very high
Degree of difficulty in Mastering the method	Low	Medium	High	Very high	Very high
Compactness	Low	High	High	Very high	Very high
Ease of analysis	Very high	Very high	Medium	Medium	Medium
Degree of supporting Encapsulation-abstraction-refinement	Low	Medium	High	Very high	Very high
Degree of specifying communication	Medium	Medium	High	Very high	Very high

tics, types and graphical representation of model elements, as well as application and usage attributes.

Descriptive power: All PN extensions are or can be computationally equivalent to Turing machines through the embodiment of inhibitor arcs. However, there is a different level of descriptive power that characterizes approaches in each category, which deals with the level of support that PN formalism provides designers with, so that they can describe efficiently their problem. Low-level PNs have low descriptive abilities since they include neither primitives to support the treatment of individuals (distinguishable system elements) nor any specific compositional operators. [7]Pure HPNs, unlike low-level models, do not suffer from the problem of specifying individuals, but their application to complex systems modeling usually produces nets which carry many implementation details. HPNs with modified semantics extend the functionality of arcs, places and transitions in order to deal efficiently with practical issues such as true concurrency, zero testing, marking dependent behaviour etc. HHPNs put emphasis on describing (synchronous and/or asynchronous) communication via specific compositional methodologies. Finally, the descriptive power of PNs is further increased by exploiting advantages from other specification methods (e.g., data abstraction, inheritance, compositional and communicating operators, synchronization mechanisms etc.).

Degree of difficulty in mastering the method: This factor influences the practical usefulness and the degree of adoption of PN-based specifications. Generally speaking, difficulty in mastering a specification method is related to the "heaviness" of its formalism. Therefore, a semantically rich PN-based formalism, even if it incorporates a graphical notation, will require extended training in order to be applied to software development. The degree of difficulty in mastering a PN-based approach varies from medium level for low-level PNs to high level for the integrated approaches. Nevertheless, the previous discussion has shown that rich formalisms allow for a higher descriptive power. Therefore, designers are faced with a choice among models that become more difficult to learn as they become more powerful.

Compactness: Compactness is mainly related to briefness and economy in design. This criterion is in conflict to the level of descriptive power that characterizes a specific approach, since primitives of a richer formalism retain complexity under acceptable limits and allow for simpler representations. For example, HHPNs allow for a structured design which prevents flat and cumbersome schemata, while the usage of special arc types provided by a HPN with modified arc semantics gives a compact representation of zero testing, set-reset a place, create a debit token etc. HPNs based on other specification methods (such as those using semantics from CSP) are exceptionally suitable in providing minimality in specifying communication. On the other hand, if one adopts design by HPNs or low-level models, without following a specific refinement or abstraction methodology, resulted nets will be very complicated.

Ease of analysis: Ease of analysis is one of the most important factors in software development, since it deals with the formal verification of system properties. Ease of analysis allows designers to use the specification as a proof of correctness tool in order to prove that a system provides all required functionality and has the desired (behavioural or structural) properties. As far as analysis of low-level PNs is concerned, there are well-known techniques established on algebraic methods and graph theory algorithms. This is because of the directed-graph nature and the direct executability of low-level PNs. In addition to that, useful results can be obtained from the simulation of the net dynamic behaviour. Analysis of HPNs is based on methods which generalize on corresponding methods of low-level PNs. Other categories (e.g., HPNs with modified semantics and HHPNs) often

[7] Although several studies have been made to define synthesis in PT-nets by fusing net places and/or transitions (see for example Kotov, 1979), the composition process in HHPNs, as previously described, is realized by means of more sophisticated notions such as hierarchies, specific composition operators and place/transition types, compositional analysis methods etc.

result in nets not directly executable, and consequently, analysis is performed on an equivalent HPN representation. However, as previously discussed, there are some exceptions which use direct analysis techniques; they construct the reachability graph (e.g., NPNs, Valmari and Bucholz's approaches) or extend the concept of place invariants of HPNs (e.g., CPN with channels). The problem with these exceptions is that they score a low level comprehensibility and require much more training in understanding than analysis methods applied to HPNs or low-level models.

Degree of supporting encapsulation-abstraction-refinement: Encapsulation and abstraction are closely related to the support a formalism provides for net construction in a hierarchical way that describes only the required functionality of a part of the modeled system. The main problem with low-level PNs is that they have no inherent mechanisms that force designers to specify abstract views of the modeled system. Although HPNs introduce the concept of colours they still describe a flat view on a complex system. HPNs with modified semantics score a high degree of encapsulation, since details can be hidden in net components (e.g., arc inscriptions, transition actions, token types etc.). HHPNs provide structured mechanisms (e.g., composition operators and pages) and allow modular substitution of places and transitions by more abstract views. Finally, integrated approaches offer abstract constructs (e.g., abstract data types, objects and channels etc.). In this way, these approaches not only define the behaviour of nets in terms of their functional abilities, but also provide designer with an abstract view of the modeled system. As far as the degree of supporting refinement is concerned, one may state analogous assertions, since refinement stands for the reverse process of abstraction.

Degree of specifying communication: Communication between nets is closely related to two important issues: how are they connected and what is the behaviour of the composite nets. Both HHPNs and integrated approaches provide a high level of specifying communication. They provide designers with explicit primitives to connect nets by merging transitions, places and/or arcs. Nevertheless, the proposed methodologies are not straightforward. As discussed above, if one is interested in specifying communication attention must be paid to the following issues:

- ease of analysis (i.e., the properties of each component net may be preserved or not),
- tight or loose coupling of the communicating component nets and,
- synchronization protocols that prevent undesirable phenomena and manage exclusive or shared usage of resources involved in the communication.

All other categories do not have any inherent compositional mechanisms, and thus they score rather worse in specifying communication.

7. Conclusions

Besides PNs, there is a number of other modeling and specification methods applicable to the same areas, including CCITT's Specification and Description Language (SDL), State Transition Techniques, Abstract Data Types, VDM/Meta-IV, Formal Grammars, Temporal Logics, Z Schemas, Process Algebras (CSP, CCS), STATECHARTS etc. Some fine review material related to such formal methods is available to the interested reader (e.g., Hall, 1990; Ostroff, 1992; Alur and Henzinger, 1992; Heitmeyer et al., 1995; Hinchey and Bowen, 1995; Saiedian, 1996). A detailed presentation and comparison of formal specification methods is beyond the scope of this paper, which has focused on comparatively presenting, evaluating and categorizing various models based on the formalism of PNs, one of the most popular graphical specification formalisms. A variety of advantages have been attributed to the application of formal specification methods. Among them are systematic system description and support of rigorous analysis (verification) techniques which provide software engineers with high confidence in software correctness. Until recently, the time needed to perform detailed specification and analysis has been a restrictive factor for adopting a formal method. Nowadays, there is a number of automated tools which can be used to significantly reduce the development effort. For example, concerning tools supporting PN based specification we can mention (among many others) *Design* CPN (Jensen, 1995), which supports specification and analysis of CPNs and HCPNs.

The major restriction of low-level PNs is that large nets are usually needed to describe systems of a medium complexity. The various higher level models try to maximize modeling convenience and to provide compact specifications. These can be achieved by associating tokens, places and transitions to multiple types, adding sets of data variables, supplying inscriptions to express the more complex enabling and firing rules, combine object-oriented practices, integrate data flows, control flows and state transition diagrams into a uniform and compact representation and support hierarchies.

If the main intention is analysis, then low-level PNs should be very applicable. For large and complex systems, the choice of a powerful high-level PN can be advantageous in order to reduce system's complexity. However, the fact that several semantics can be attached to places, arcs and transitions, results in nets, the entire structure of which is embedded in the inscriptions; analysis of these nets is often difficult to achieve. Therefore, the resulting net may not visually correspond to the modeled system in a straightforward manner.

The PN formalism used to represent concurrency and nondeterminism has been extended to deal with the strict timing aspects of a real-time system. In general, the

introduction of time is related to still open questions regarding the assignment of timing delays (at places, transitions and/or arcs), the kind of delays (fixed, intervals, stochastic) and the establishment of higher-level timed PNs which will provide a more compact representation. The answer to these questions mainly depends on the specific application area. Perhaps one of the most general and integrated approaches available for complex real-time systems design is the Interval Timed Coloured PN (Aalst, 1993), a HPN in which an interval is used to specify the timing characteristics by attaching a time stamp to every token. Assessment and comparative study of the large class of the various proposed timed versions of PNs are within the current research interests of the authors. Our research efforts are also devoted to exploiting the experience gained with IMFG and to embed timing aspects in an extension of the model, called Real-MFG (Gerogiannis et al., 1996 Gerogiannis et al., 1997), which are necessary in order to model real-time interactive systems and in general systems with time-critical requirements.

Acknowledgements

The authors would like to express their gratitude to the anonymous referee for his constructive comments and suggestions. We also thank Mr. Costas Diplas for helpful discussions.

References

Van der Aalst, W.M.P., 1993. Interval timed coloured petri nets and their analysis. In: Marsan, M.A. (Ed.), Proceedings of the 14th International Conference on Application and Theory of Petri Nets 1993, Chicago, IL. Lecture Notes in Computer Science, vol. 691, Springer, Berlin, pp. 453–472.

Agerwala, T., 1974. Complete model for representing the coordination of asynchronous processes. Comp. Res. Rep. vol. 32, John Hopkins University, Baltimore.

Alur, R., Henzinger, T.A., 1992. Logics and models of real-time: A survey. In: De Bakker, J.W., et al. (Eds.), Real-Time: Theory in Practice. Lecture Notes in Computer Science, vol. 600, Springer, Berlin, pp. 74–106.

Baldassari, M., Bruno, G., 1998. An environment for object-oriented conceptual programming based on PROT nets. In: G. Rozenberg (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 340, Springer, Berlin, pp. 1–19.

Baldassari, M., Bruno, G., 1991. PROTOB: An object-oriented methodology for developing discrete event dynamic systems. Comp. Lang. 16, 39–63.

Bastide, R., Palanque, P., 1990. Petri net objects for the design, validation and prototyping of user-driven interfaces. CHI-INTERACT'90, pp. 625–631.

Battiston, E., De Cindio, F., Mauri, G., 1988. OBJSA Nets: A class of high-level nets having objects as domains. In: Rozenberg, G. (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 340, Springer, Berlin, pp. 20–43.

Berthomieu, B., Diaz, M., 1991. Modeling and verification of time dependent systems using time petri nets. IEEE Trans. Software Engrg. SE –17, 259–273.

Best, E., 1987. Structure theory of petri nets: The free choice hiatus. In: Brauer et al., (Eds.), Petri Nets: Central Models and their Properties. Lecture Notes in Computer Science, vol. 254, Springer, Berlin, pp. 168–206.

Billington, J., 1988. Extending Coloured Petri Nets. Ph.D. Thesis, Technical Paper, vol. 148, University of Cambridge, Cambridge.

Billington, J., Wheeler, G.R., Wilbur-Ham, M.C., 1988. PROTEAN: A high-level petri net tool for the specification and verification of communication protocols. IEEE Trans. Software Engrg. SE –14, 301–316.

Booch, G., 1986. Object oriented development. IEEE Trans. Software Engrg. SE –12, 211–221.

Bucholz, P., 1994. Hierarchical high level petri nets for complex systems analysis. In: Valette, R., (Ed.), Proceedings of the 15th International Conference on Application and Theory of Petri Nets 1994, Zaragosa. Lecture Notes in Computer Science, vol. 815, Springer, Berlin, pp. 119–198.

Cao, T., Sanderson, A.C., 1993. Variable reasoning and analysis about uncertainty with fuzzy petri nets. In: Marsan, M.A. (Ed.), Proceedings of the 14th International Conference, Application and Theory of Petri Nets 1993. Chicago, IL. Lecture Notes in Computer Science, vol. 691, Springer, Berlin, pp. 126–145.

Christensen, S.A., Hansen, N.D., 1993. Coloured petri nets extended with place capacities, test arcs and inhibitor arcs. In: Marsan, M.A. (Ed.), Proceedings of the 14th International Conference, Application and Theory of Petri Nets 1993. Chicago, IL. Lecture Notes in Computer Science, vol. 691, Springer, Berlin, pp. 186–205.

Christensen, S.A., Hansen, N.D., 1994. Coloured petri nets extended with channels for synchronous communication. In: Valette, R. (Ed.), Proceedings of the 15th International Conference on Application and Theory of Petri Nets 1994, Zaragosa. Lecture Notes in Computer Science, vol. 815, Springer, Berlin, pp. 159–177.

Ciardo, G., 1994. Petri nets with marking-dependent arc cardinality: Properties and analysis, In: Valette, R. (Ed.), Proceedings of the 15th International Conference on Application and Theory of Petri Nets 1994, Zaragosa. Lecture Notes in Computer Science, vol. 815, Springer, Berlin, pp. 179–198.

Comparin, C., Lanzarone, C.A., Lautenbach, K., Panzerri, A., Torgano, A., 1985. Guidelines on using net analysis techniques with large specifications. In: Rozenberg, G. (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 222, Springer, Berlin.

Degano, P., Gorrieri, R., Marchetti, S., 1988. An exercise in concurrency: A CSP process as a condition/event system. In: G. Rozenberg, (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 340, Springer, Berlin, pp. 85–105.

Dietz, C., Schreiber, G., 1994. A term representation of P/T systems. In: Proceedings of the 15th International Conference, Valette, R., (Ed.), Application and Theory of Petri Nets 1994. Zaragosa. Lecture Notes in Computer Science, vol. 815, Springer, Berlin, pp. 239–257.

Finkel, A., Choquet, A., 1988. Fifo nets without order deadlock. Acta Inform. 25, 15–36.

Finkel, A., Rosier, L., 1988. A survey on the decidability questions for classes of fifo nets. In: Rozenberg, G., (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 340, Springer, Berlin, pp. 106–132.

Garg, M.L., Ahson, S.I., Gupta, P.V., 1991. A fuzzy petri net for knowledge representation and reasoning. Inform. Process. Lett. 39, 165–171.

Genrich, H.J., Lautenbach, K., 1981. System modelling with high-level petri nets. Theoret. Comp. Sci. 13, 109–136.

Gerogiannis, V.C., Kameas, A.D., Diplas, C., Pintelas, P.E., 1995. Petri Nets and High-Level Petri Nets: Formalism, Properties,

Analysis and Applications, Technical Paper, TR-95-01, Department of Mathematics, Sector of Computational Mathematics and Informatics, University of Patras, Patras, Greece.

Gerogiannis, V.C., Kameas, A.D., Pintelas, P., 1996. On the integration of high-level timed petri nets with schedulability analysis methods. In: Proceedings of the Third Hellenic European Research Conference on Mathematics and Informatics (HERMIS'96), Athens, Greece, pp. 688–697.

Gerogiannis, V.C., Kameas, A.D., Pintelas, P., Real-MFG: A high-level timed petri net model focusing on the integration of schedulability and fault-tolerance. In: Proceedings of the Third International Conference on Reliability, Quality and Safety of Software-Intensive Systems (ENCRESS'97), Athens, Greece (to appear).

Hall, A., 1990. Seven myths of formal methods. IEEE Software 7 (5), 11–19.

Hartung, G., 1988. Programming a closely coupled multiprocessor system with high level petri nets. In: G. Rozenberg, (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 340, Springer, Berlin, pp. 154–174.

Hatono, I., Yamagata, K., Tamura, H., 1991. Modeling and on-line scheduling of flexible manufacturing systems using stochastic petri nets. IEEE Trans. Software Engrg. SE –17, 126–132.

Heitmeyer, C., Jeffords, R., Labaw, B., 1995. Comparing different approaches for specifying and verifying real-time systems. In: Proceedings of the Tenth IEEE Workshop on Real-Time Operating Systems and Software, International Conference on Reliability.

Heuser, C.A., Richter, G., 1992. Constructs for modeling information systems with petri nets. In: Jensen, K., (Ed.), Proceedings of the 13th International Conference on Application and Theory of Petri Nets 1992, Sheffield. Lecture Notes in Computer Science, vol. 616, Springer, Berlin.

Hinchey, M.G., Bowen, J.P. (Eds.), 1995. Applications of Formal Methods, Prentice-Hall International Series in Computer Science, Hemel Hempstead, UK.

Hoare, C.A.R., 1985. Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ.

Huber, P., Jensen, K., Shapiro, R.M., 1990. Hierarchies in coloured petri nets. In: Rozenberg, G., (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 483, Springer, Berlin, pp. 313–341.

Jensen, K., 1981. Coloured petri nets and the invariant method. Theoret. Comp. Sci. 14, 317–336.

Jensen, K., 1990. Coloured petri nets: A high level language for system design and analysis. In: G. Rozenberg, (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science, vol. 483, Springer, Berlin, pp. 342–416.

Jensen K., 1995. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Brauer, W., et al. (Eds.), vols. 1 and 2, Springer, Berlin.

Kameas, A.D., Gerogiannis, V.C., Diplas, C., Pintelas, P.E., 1994. Encapsulating multiple perspectives in interaction specification. In: Proceedings of the 20th EUROMICRO Conference, System Architecture and Integration. IEEE Computer Soc. Press, Silver Spring, MD, pp. 463–469.

Kameas, A.D., 1995. A Formal Model for the Specification of Interaction and the Design of Interactive Applications. Ph.D. Thesis, Department of Computer Engineering, University of Patras, Greece.

Kameas, A.D., Pintelas, P.E., The functional architecture and interaction model of a generator of intelligent tutoring applications. J. Systems and Software (to appear).

Koh, I., DiCesare, F., 1991. Modular transformation methods for generalized petri nets and their application to automated manufacturing systems. IEEE Trans. Systems Man Cybernet. 21, 1512–1522.

Kotov, V.E., 1979. An algebra for parallelism based on petri nets. In: Proceedings of the MFCS 79. Lecture Notes in Computer Science, vol. 64, Springer, Berlin, pp. 39–55.

Kramer, B., 1989. Syntax and Semantics of SEGRAS – A Specification Language for Distributed Systems. Oldenbourg, Munchen, Wien.

Lakos, C., Christensen, S.A., 1994. A general systematic approach to arc extensions for coloured petri nets. In: R. Valette (Ed.), 15th International Conference on Application and Theory of Petri Nets, Zaragosa. LNCS vol. 815, Springer, Berlin, pp. 338–357.

Lakos, C., Keen, C.D., 1991. Modeling Layered Protocols in LOOPN. In: Fourth International Workshop on Petri Nets and Performance Models. Melbourne.

Levi, S.T., Agrawala, A.K., 1990. Real-Time System Design, McGraw-Hill, New York.

Looney, C.G., 1988. Fuzzy petri nets for rule-based decision making. IEEE Trans. Systems Man and Cybernetics 18, 178–183.

Marsan, M.A., Balbo, G., Conte, G., 1984. A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems. ACM Trans. Computer Systems 2, 93-122.

Milner, R., 1989. Communication and concurrency. Prentice-Hall, Englewood Cliffs, NJ.

Murata, T., 1989. Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77, 541–580.

Murata, T., Shenker, B., Shatz, S.M., 1989. Detection of Ada static deadlocks using petri net invariants. IEEE Trans. Software Engrg. 15, 314-325.

Ostroff, J.S., 1992. Formal Methods for the Specification and Design of Real-Time Safety Critical Systems. J. Systems and Software 18, 33–60.

Papelis, Y.E., Casavant, T.L., 1992. Specification and Analysis of Parallel/Distributed Software and Systems by Petri Nets With Transition Enabling Functions. IEEE Trans. Software Eng. 18, 252–261.

Peterka, G., Murata, T., 1989. Proof procedure and answer extraction in petri net model of logic programs. IEEE Trans. Software Engrg. 15, 209–217.

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

Petri, C.A., 1962. Kommunikation mit Automaten. Ph.D. Thesis, Institut fur Instrumentelle Mathematik, Bonn.

Ramchandani, C., 1974. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. Technical Report No. 120, Massachusetts Institute of Technology, MA.

Reisig, W., 1985. Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science. Springer, New York.

Reisig, W., 1992. A Primer in Petri Net Design. Springer, New York.

Saiedian, H., 1996. An invitation to formal methods, a Round Table (special Issue). IEEE Computer 29 (4), 16-30.

Sagoo, J.S., Holding, D.J., 1991. A comparison of temporal petri nets techniques in the specification and design of hard real-time systems. Microprocessing and Microprogramming 32, 111–118.

Schmidt, H.W., 1991. Prototyping and Analysis of Non-Sequential Systems Using Predicate-Event Nets. J. Systems and Software 15, 43–62.

Sibertin-Blank, C., 1993. A client-Server Protocol for the Composition of Petri Nets, in: Marsan, M.A. (Ed.), 14th International Conference on Application and Theory of Petri Nets 1993, Chicago, IL. Lecture Notes in Computer Science, vol. 691, Springer, pp. 453–472.

Sibertin-Blank, C., 1994. Cooperative Nets, In: Valette, R. (Ed.), 15th International Conference on Application and Theory of Petri Nets, Zaragosa, Lecture Notes in Computer Science, vol. 815, Springer, pp. 377–396.

Stotts, P.D., Godfrey, P., 1992. Place/Transition Nets with Debit Arcs. Information Processing Letters 41, 25–33.

Valavanis, K.P., 1990. On the Hierarchical Modeling Analysis and Simulation of Flexible Manufacturing Systems with Extended Petri Nets. IEEE Trans. Systems, Man and Cybernetics 20, 94–110.

Valmari, A., 1993. Compositional State Space Generation, in: Rozenberg, G. (Ed.), Advances in Petri Nets. Lecture Notes in Computer Science vol. 674, Springer, pp. 427–457.

Valmari, A., 1994. Compositional Analysis with Place-Bordered Subnets, In: Valette, R. (Ed.), 15th International Conference on Application and Theory of Petri Nets, Zaragosa. Lecture Notes in Computer Science, vol. 815, Springer, pp. 531–547.

Ventouris, K.P., Pintelas, P.E., 1992. A Practical Assessment of Formal Specification Approaches for Data Abstractions. J. Systems and Software 17, 169–188.

Vasilis C. Gerogiannis received his Diploma in Computer Engineering from the Dept. of Computer Engineering & Informatics at the Univ. of Patras, Greece, in 1992. Since 1993, he is a postgraduate student with the Dept. of Mathematics, Univ. of Patras, pursuing a PhD degree in the area of formal design verification techniques for real-time systems. His current research interests include Petri Net based specification, real-time systems modeling and schedulability analysis. He has participated in the ESPRIT projects OMI/CLEAR, OMI/TOOLS and OMI/ANTICRASH. He is a member of the Technical Chamber of Greece (TEE) since 1993. He joined the Educational Software Development Laboratory (ESD*Lab) at the Dept. of Mathematics, Univ. Of Patras, in 1993.

Dr. Achilles D. Kameas received his Diploma in Computer Engineering and his PhD in Human-Computer Interaction from the Dept. of Computer Engineering & Informatics (CEID) at the Univ. of Patras, Greece, in 1989 and 1995, respectively. His research interests also include Human Cognition and User modeling, Virtual Reality and Artificial Intelligence, Authoring and Multimedia Systems, and Intelligent Tutoring Systems. In recent years, he has contributed to a number of publications in those fields, as a result of the ongoing research he participates in. His research experience includes participation in several national and EU research projects, such as GESEM (in the context of EU program COMETT II) and AXE-10 (in the context of national program SYN). His tasks include the analysis, design, integration, verification and validation of project outcomes. He was a teaching assistant with CEID (1990-1995) and a part-time professor with the Technological Educational Institute (TEI) of Patras and Mesolongi (1995). He is a Voting Member of ACM, SIGCHI and SIGCUE since 1992, a member of IEEE and IEEE Computer Society since 1993, a member of the Society for Machines and Mentality since 1993 and a member of the Technical Chamber of Greece (TEE) since 1990. He joined the ESD*Lab in 1992.

Dr. Panayotis E. Pintelas received his BSc in Mathematics from the Univ. of Athens, Greece, in 1971, and his MSc and PhD in Computer Science from the Univ. of Bradford, UK, in 1973 and 1976, respectively. He is currently a Professor of Computer Science with the Dept. of Mathematics, Sector of Applied Mathematics and Informatics, Univ. of Patras. From 1991 to 1996, he was an Associate Professor with the Dept. of Mathematics, Univ. of Patras. He was elected Head of the Division of Applied Mathematics and Informatics from 1991 to 1993. In the recent past, he was with the Dept. of Computer Engineering and Informatics, Univ. of Patras, as a Visiting Scientist (1981-1984), Lecturer (1984-1987) and Assistant Professor (1987-1991). His current research interests include Software Specification and Software Design, Project Management, Computer Assisted Training and Education (CAL/CAI/CBT/ITS) and Software Engineering; in these fields he has numerous publications in international scientific journals and conferences. He is also the author of two textbooks which are included in the curriculum of the Dept. of Computer Engineering and Informatics. He has participated in numerous National and EU research projects. He is a member of the British Computer Society and of the British Association of Chartered Engineers, and a member of the Association for the Development of Computer-Based Instructional Systems (ADCIS). He is directing the Educational Software Development Laboratory (ESD*Lab), the establishment of which he proposed in 1992.