Prototyping of Concurrent Control Systems with Application of Petri Nets and Comparability Graphs

Remigiusz Wiśniewski, Andrei Karatkevich, Marian Adamski *Member, IEEE,*
Anikó Costa *Member, IEEE,* Luís Gomes *Senior Member, IEEE*

***Abstract*—The paper shows a novel prototyping technique for concurrent control systems described by interpreted Petri nets. The technique is based on the decomposition of an interpreted Petri net into concurrent sequential automata. Generally, minimum decomposition requires run-time that is exponential in the number of Petri net places. We show that in many cases, including the real-life ones, the minimum decomposition problem can be solved in polynomial time. The proposed method allows to implement a concurrent control system using minimal number of sequential components, which requires polynomial time and can be applied to most of considered cases. The presented concept is illustrated by a real-life industrial example of a beverage production and distribution machine implemented in an FPGA.**

***Index Terms*—Concurrent control systems, Petri nets, comparability graphs, Field Programmable Gate Array (FPGA).**

I. Introduction

Different kinds of concurrent control systems surround us in everyday life. They can be met with in transportation [59], [62], medical care [1], [15], manufacturing [16], [34], [56], artificial intelligence and robotics [22], [48], security and safety [25], [47], or even banking [23], [57]. Petri nets are one of the popular forms of graphical representation and specification of such systems [7], [35]. They turn out to be especially effective in the area of analysis, design and verification of logic controllers [31], [54], [74].

One of the most important steps in designing a logic controller specified by a Petri net is the parallel decomposition of the system [20]–[22], [48], [64]. Such decomposition is not necessary when the reachability graph of a net is small enough to be implemented as a single state machine, but it turns to be important even for the relatively small nets with many parallel transitions, because of the state explosion. Unfortunately, the whole decomposition process in the general case has exponential worst-case time complexity [46], [68]. Moreover, most of the known methods extract all possible components of decomposition, thus a subsequent selection from the obtained components is required [58].

A new prototyping method for the concurrent control systems described by the Petri nets is presented in the paper. The proposed method includes decomposition of a system into concurrently executed sequential automata, called *state machine components (SMCs)* [38], [50]. Such an operation is performed with application of the perfect graph theory [28], [45]. In most cases the presented technique makes it possible to obtain a state machine decomposition in polynomial time.

The paper is organized as follows: Section II introduces the preliminaries regarding comparability graphs and Petri nets. Section III describes the role of decomposition in prototyping of the concurrent control systems. The main idea of the proposed method is shown in Section IV, while a supplementary example is presented in section V. Section VI presents the results of experiments, and Section VII concludes the paper.

R. Wiśniewski and A. Karatkevich are with the Institute of Electrical Engineering, University of Zielona Góra, Poland, M. Adamski is with the University of Zielona Góra, Poland, Anikó Costa and Luís Gomes are with Nova University of Lisboa, Portugal. (e-mail: r.wisniewski@iee.uz.zgora.pl).

II. Main Definitions

Let us introduce the notations and definitions necessary to explain the idea of the proposed method. The notion of a *comparability graph* and related notions will be presented. Unique features of the comparability graphs are a key element of the prototyping method shown in the paper. Furthermore, *Petri nets* and related notions are presented in this section.

A. Graphs, Comparability Graphs

Definition 1 (Graph): A *graph* or *undirected graph* is defined by a pair [24]:

$$G = (V, E), \tag{1}$$

where:
V=$\{v_1, \ldots, v_n\}$, is a non-empty set of vertices;
E=$\{E_1, \ldots, E_m\}$, is a set of unordered pairs of vertices, called edges.

An undirected edge incident with vertex u and vertex v is denoted by uv. Two vertices u and v are *adjacent* (*neighbors*) if they are connected by an edge. $Adj(v)$ denotes the set of all vertices adjacent to vertice v.

Definition 2 (Digraph): A *digraph* (*directed graph*) is defined by a pair [5]:

$$D = (V, F), \tag{2}$$

where:
V=$\{v_1, \ldots, v_n\}$, is a finite, non-empty set of vertices;
F=$\{F_1, \ldots, F_m\}$, is a finite set of ordered pairs of vertices.

A directed edge is denoted by $\overrightarrow{uv}$, where u is the first vertex, and v is the second vertex of an edge.

Definition 3 (Orientation): An orientation of an undirected graph is an assignment of a direction to each edge, turning the initial graph into a digraph. Formally, an orientation is a set of oriented edges.

Definition 4 (Transitive digraph): The digraph $D = (V, F)$ is *transitive* if, whenever $\overrightarrow{uv} \in F$ and $\overrightarrow{vz} \in F$, $\overrightarrow{uz} \in F$ [32].

Definition 5 (Graph coloring): A *graph coloring* is an assignment of a color to each vertex such that no two neighbor vertices share the same color. *Minimum coloring* is a graph coloring using the minimal possible number of colors [24].

Definition 6 (Comparability graph): An undirected graph $G = (V, E)$ is a *comparability graph* if there exists an orientation of G such that the resulting digraph $D = (V, F)$ is transitive. Such orientation is called a *transitive orientation* of G [28].

B. Petri Nets, Interpreted Petri Nets, Concurrency Relations

Definition 7: A *Petri net* is a 4-tuple [50]:

$$N = (P, T, B, M_0), \tag{3}$$

where:

- P is a finite, non-empty set of *places*,
- T is a finite, non-empty set of *transitions*,
- $B \subseteq (P \times T) \bigcup (T \times P)$, is a finite set of *arcs* (directed edges),
- M_0 is an initial marking.

A union $P \cup T$ is a set of *nodes* of a Petri net.

Sets of input and output places of a transition (according to the relation B) are defined respectively as follows: $\bullet t = \{p \in P : (p, t) \in B\}$, $t\bullet = \{p \in P : (t, p) \in B\}$. Analogously, the sets of input and output transitions of a place are denoted: $\bullet p = \{t \in T : (t, p) \in B\}$, $p\bullet = \{t \in T : (p, t) \in B\}$. A generalized denotation for the input and output elements of the sets is used: $\bullet S = \{\forall u : s \in S \wedge \exists\, u : (u, s) \in B\}$, $S\bullet = \{\forall u : s \in S \wedge \exists\, u : (s, u) \in B\}$.

Definition 8 (Marking, marked place): The state of a Petri net can be seen as a distribution of *tokens* in the net places and is called *marking*. A *marked place* is such a place that contains one or more tokens. A marking is changed by *the firing* of a transition (see Definition 10). Since only safe nets are considered in the paper (see Definition 19), a marking M is understood as a set of the marked places.

Definition 9 (Reachable marking): A marking M' is *reachable* from marking M, if M' can be reached from M by a sequence of transition firings. If M is not specified explicitly, a *reachable marking* of the net is understood as a marking that can be reached from M_0.

Definition 10 (Transition firing): If every input place of a transition t contains a token, then t can be *fired*. Transition firing adds a token to each of its output places and removes a token from each of its input places.

Definition 11 (Incidence matrix): For a Petri net N=(P, T, B, M_0) with n transitions and m places, the *incidence matrix* $A = [a_{ij}]$ is an $n \times m$ matrix of integers, entries of which are given by:

$$a_{ij} = \begin{cases} 1 & \text{if } (t_i, p_j) \in B \text{ and } (p_j, t_i) \notin B \\ -1 & \text{if } (p_j, t_i) \in B \text{ and } (t_i, p_j) \notin B \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

Definition 12 (S-invariant): An m-vector x is an *S-invariant* of Petri net N with incidence matrix A if $Ax = 0$. The set of places corresponding to nonzero entries in an S-invariant such that $x \geq 0$ is called *support* of the invariant.

Definition 13 (SM-net): A *State Machine net* (*SM-net*) is a Petri net such that each of its transitions has at most one input place and at most one output place, i.e. $\forall t \in T : (|\bullet t| \leq 1) \wedge (|t\bullet| \leq 1)$.

If a state machine net has only one token in its initial marking, then it has one token in every reachable marking and can be seen as a Petri net without concurrency.

Definition 14 (Strongly connected net): A Petri net S=(P, T, B, M_0) is *strongly connected*, if its underlying graph (i.e. digraf $D = (P \cup T, B)$) is strongly connected.

Definition 15 (SM-component): A Petri net S=(P', T', B', M_0') is a *State Machine component (SM-component, SMC)* of a Petri net $N = (P, T, B, M_0)$, if and only if:

1) S is an SM-net;
2) $P' \subseteq P$;
3) T'=$\{x | x \in T, \exists p \in P' : ((p, x) \in B \vee (x, p) \in B)\}$;
4) $\forall x \in (P' \cup T'), \forall y \in (P' \cup T') : ((x, y) \in B \Leftrightarrow (x, y) \in B')$;
5) S is strongly connected;
6) $|M_0'| = 1$; $M_0' \subseteq M_0$.

An SM-component is a strictly sequential subnet of a given net. If a net can be covered by a set of SM-components it means that it can be represented as a system of locally synchronized sequential processes.

Definition 16 (Concurrency relation): Two places are *concurrent* if they are both marked on some reachable marking. The *behavioral concurrency relation* (or just *concurrency relation*) $||$ on the set of places P of a Petri net N=(P, T, B, M_0) is the relation such that $(p, p') \in ||$ if there is a reachable marking M such that $\{p, p'\} \subseteq M$. We use $p||p'$ as a shortened notation that means $(p, p') \in ||$.

Definition 17 (Structural concurrency relation): The *structural concurrency relation* $||^A$ on the set of places P of a Petri net N=(P, T, B, M_0) is the smallest relation such that [40]:

1) $\forall p, p' \in P : (p \in M_0 \wedge p' \in M_0 \wedge p \neq p') \Rightarrow (p, p') \in ||^A$;
2) $\forall t \in T\ \forall p \in t\bullet\ \forall p' \in t\bullet : p \neq p' \Rightarrow (p, p') \in ||^A$;
3) $\forall p \in P\ \forall t \in T : ((\forall p' \in \bullet t : (p, p') \in ||^A) \Rightarrow$
   $\Rightarrow (\forall p'' \in t\bullet : (p, p'') \in ||^A))$.

Structural concurrency between two places $\{p_i, p_j\} \in P$ is denoted as $p_i||^A p_j$.

The structural concurrency relation is a superset of behavioral concurrency relation which, unlike the latter, can be computed in polynomial time in the general case [39]. The intuition behind it is as follows: 1) all the places marked in the initial marking are mutually concurrent; 2) every two places being output for the same transition are concurrent, if the transition can ever fire; 3) if a place is concurrent with every input place of a transition, it is structurally concurrent with all of its output places. It is shown in [39] that every two places which are behaviorally concurrent are also structurally concurrent (but not always vice versa).

According to Lemma 3.3 from [39], if $||^A$ is not irreflexive, the Petri net cannot be covered by SM-components.

Definition 18 (Concurrency graph): A concurrency graph (structural concurrency graph) of a Petri net $N = (P, T, B, M_0)$ is a graph G_C=(V, E) such that $V = P$ and $E = ||$ ($E = ||^A$, correspondingly).

Definition 19 (Well-formed net):

A Petri net N is *live* if for every reachable marking M and transition t of N, t can be fired in M or in a marking reachable from M. A net is *safe* if there is no marking M such that a place $p \in P$ contains more than one token.

A Petri net is *well-formed* if it is live and safe.

Definition 20 (Interpreted Petri net): An *interpreted Petri net* is a well-formed Petri net, defined as a 6-tuple [2], [53]:

$$N = (P, T, B, M_0, X, Y), \tag{5}$$

where:

- P is a finite, non-empty set of places,
- T is a finite, non-empty set of transitions,
- $B \subseteq (P \times T) \bigcup (T \times P)$ is a finite set of arcs,
- M_0 is an initial marking,
- X=$\{x_1, x_2, \ldots, x_m\}$ is a finite set of logic inputs,
- Y=$\{y_1, y_2, \ldots, y_n\}$ is a finite set of logic outputs.

Logic inputs and logic outputs are connected to the net as follows. A Boolean function of the inputs is associated with every transition of the net, and value 1 of the function is a necessary condition of the transition firing. A subset of outputs (maybe empty) is associated with every place, and an output has value 1, if and only if at least one of the corresponding places is marked in the current marking.

Note that as far as the prototyping process presented in this paper is based on the interpreted Petri nets, we suppose that the considered nets are well-formed.

III. Decomposition in design of concurrent control systems

The design process of a parallel control system specified by an interpreted Petri net usually includes its decomposition into sequential components [4], [30]. It may be required for several reasons. One of them is the situation in which a concurrent algorithm has to be executed by several microcontrollers or processors, where each of them executes a sequential subprocess [21], [66]. This is not the case when the controller is implemented in FPGA, since a parallel logical control algorithm can usually be implemented in a single FPGA device. However, configuration of an FPGA is usually specified using a hardware description language, such as VHDL or Verilog. In such a language, every Petri net place can be described by a separate parallel module (process), which leads to a complex and difficult to manage description. Alternatively, each sequential component can be described by single process, which provides a more understandable description, and which is relatively easy for analysis and implementation [31]. Apart from the above reasons, decomposition is used in some methods of state encoding [14].

Most of Petri net decomposition methods are based on the preservation of properties and focus on net analysis. For example in [18] a decomposition method is presented based on shared places or shared transitions, demonstrating that place invariants of the global model are the same as of the decomposed model. However, the resulting subnets cannot be executed in a distributed way.

Aybar [3] proposes a decomposition method based on net structure analysis identifying overlapped sections. Overlapped places and transitions are repeated in all resulting subnets.

Other decomposition methods use input and output places for communication between the decomposed models [13], [51], [72]. In [21] a functional decomposition method is proposed, based on the definition of a cutting set. The main concern of this method is to maintain the same behavior before and after decomposition. The interface between the decomposed components is always composed by transitions, which communicate through output and input events. Common firing rules are also preserved.

An SM-decomposition of a Petri net $N = (P, T, B, M_0)$ can be obtained by means of a calculation of its binary S-invariants and by finding a cover of set P by their supports. The invariants can be obtained directly by finding the $(0, 1)$ solutions of the system of linear equations $Ax = 0$, where A is the incidence matrix of N [50], or more efficiently, for example, using the Martinez-Silva algorithm [46]. Both approaches are easy to implement, but since the number of S-invariants can be exponential in the number of Petri net places, generation of all SM-components requires exponential time and space in the worst case.

However, it is not necessary to generate all SM-components to obtain an SM-decomposition. Indeed, it is enough to get a set of the modified SM-components such that each of them covers some places of the given Petri net and may also contain the so-called *non-operational* places (NOPs) [31], [43], [69]. Such decomposition can be obtained by coloring the concurrency graph of the Petri net [4]. Then an SM-component can be constructed from the places which obtain the same colour, the transitions incident to them and, when necessary, the non-operational places (a way of adding NOPs is explained by Algorithm 4 presented below).

Given that minimum coloring of a graph is an NP-hard problem, no known polynomial-time algorithm can calculate the colouring of a graph using the smallest possible number of colours in the general case. Hence it is reasonable to apply the approximation colouring algorithms [24]. However, as shown below, in many practical cases minimum decomposition of a concurrent control system and the whole prototyping process can be performed in polynomial time.

IV. Idea of The Proposed Method

The proposed method of prototyping of the concurrent control systems consists of the following steps:

1) Specification of the concurrent control system:
   - Specification by an interpreted Petri net N.
   - (Optional: formal verification of the system).
   - (Optional: reduction of the net N).
2) Decomposition of the Petri net N:
   - Computation of the structural concurrency graph G_C of N.
   - Obtaining a transitive orientation (TRO) of G_C if it exists.
   - Minimum (optimal) colouring of G_C if a TRO exists (otherwise, another approach has to be used, such as colouring using an approximation algorithm).

- Formation of the set $\mathcal{S}$ of decomposed components.

3) Modelling, validation, logic synthesis and final implementation of the system in a programmable device.

We shall explain each of the above stages in more detail.

A. Specification of the system by an interpreted Petri net

Initially, the concurrent control system is specified by an interpreted Petri net N. Such a specification allows further decomposition and implementation of the system in programmable devices.

At this stage the Petri net model of concurrent controller can be additionally formally verified with the use of a model checking technique [19], [26], [29]. Such a verification ensures that the final system meets all the user-defined requirements. Different aspects of formal verification are widely described in the literature [9], [11], [17], [33], [44], [52], [55], [60].

Reduction of the Petri net [8], [35], [50] can be used as a step simplifying its SM-decomposition [4]. Some details on this topic will be presented in section V.

B. Decomposition of the interpreted Petri net

The decomposition process allows splitting the initial net into the concurrent modules (SM-components). Each of the obtained components is further modelled independently (see Section IV-C). Note that at this stage of prototyping of a concurrent control system, input and output signals (sets X and Y) are not taken into account.

The proposed decomposition method consists of five steps. Let us describe each of them.

1) Computation of the structural concurrency graph: Based on the initial Petri net N, a structural concurrency graph G_C is formed. Such a structure is obtained by the consecutive application of three rules presented in Definition 17.

The method for obtaining the structural concurrency graph G_C for a Petri net N is shown in Algorithm 1. The idea is taken from [39]. According to [39], computation of the structural concurrency relation $||^A$ of Petri net $N = (P, T, B, M_0)$ is bounded by $O((|P| + |T|)^5)$.

Algorithm 1 Computation of the structural concurrency graph

Input: Petri net $N = (P, T, B, M_0)$
Output: Structural concurrency graph G_C=(P, E) of N

$||^A \leftarrow \{(p_i, p_j) : (p_i \in M_0 \wedge p_j \in M_0)\} \cup \bigcup_{t \in T} t\bullet \times t\bullet$
repeat
  $\mathcal{A} \leftarrow ||^A$
  $\mathcal{T} \leftarrow T$
  while $\mathcal{T} \neq \emptyset$ **do**
    choose $t \in \mathcal{T}$
    $\mathcal{P} \leftarrow \{p_i \in P : \forall p_j \in \bullet t\ (p_i, p_j) \in ||^A\}$
    $||^A \leftarrow ||^A \cup \{(p_i, p_j), (p_j, p_i) : p_j \in t\bullet, p_i \in \mathcal{P}\}$
    $\mathcal{T} \leftarrow \mathcal{T} \setminus \{t\}$
  end while
until $\mathcal{A} = ||^A$
return $G_C = (P, ||^A)$

Note that a structural concurrency relation does not always coincide with a behavioral concurrency relation. It is shown in [41] that for every Petri net, $|| \subseteq ||^A$, and that for the well-formed EFC-nets $|| = ||^A$.

2) Obtaining a transitive orientation of the structural concurrency graph: This step is one of the essential stages of the whole decomposition process. Existence of a transitive orientation in G_C means that such a graph can be optimally colored in polynomial time. Otherwise, the proposed method cannot guarantee obtaining optimal decomposition and a different decomposition method ought to be selected.

The best known recognition algorithm uses a depth-first search (DFS). The set of graph vertices is split into the subsets called *implication classes* by a recursive orientation of subsequent edges [27]. At the beginning, an initial edge is selected arbitrarily. Another idea is shown in [32], where lexicographic breadth-first search (BFS) is used. The algorithm also partitions the set of vertices into the implication classes.

Let us introduce an alternative version of recognition of a comparability graph. The method shown in *Algorithm 2* finds the transitive orientation F of an undirected graph G=(V, E). The algorithm explores further edges with the use of the BFS method. The set F is supplied by the subsequent transitive orientations, while an additional set X (a copy of E) holds unexplored edges. Let us point out that in contrast to the known methods, the proposed one does not split the set of graph vertices into implication classes, it just retrieves a transitive orientation of G_C.

Algorithm 2 Finding of a TRO in an undirected graph

Input: An undirected graph $G = (V, E)$
Output: A transitive orientation F of G if it exists, $\emptyset$ otherwise

$F \leftarrow \emptyset$
$X \leftarrow E$
Q.clear()
while $X \neq \emptyset$ **do**
  Pick (arbitrary) an edge $uv \in X$
  Q.push $(\overrightarrow{uv})$
  repeat
    $\overrightarrow{uv} \leftarrow Q$.pop()
    if $\overrightarrow{vu} \in F$ **then**
      write: "Not a comparability graph"
      return $\emptyset$
    end if
    $F \leftarrow F \cup \overrightarrow{uv}$
    $X \leftarrow X \setminus uv$
    for each z=$Adj(u)$ **do**
      if [$z \neq Adj(v)$ or $\overrightarrow{vz} \in F$] and $\overrightarrow{uz} \notin \{F \cup Q\}$ **then**
        Q.push $(\overrightarrow{uz})$
      end if
    end for
    for each z=$Adj(v)$ **do**
      if [$z \neq Adj(u)$ or $\overrightarrow{zu} \in F$] and $\overrightarrow{zv} \notin \{F \cup Q\}$ **then**
        Q.push $(\overrightarrow{zv})$
      end if
    end for
  until Q is empty
end while
return F

An estimation of the computational complexity of *Algorithm 2* is as follows. Clearly, both outer loops (*while..do* and *repeat..until*) are executed $|E|$ times for each edge of the

graph. Each of the inner *for each* loops is executed for all the neighbor vertices. Therefore, their complexity is at most $O(\delta(G))$, where $\delta(G)$ means the maximum degree of vertices in G. Summarizing, we can formulate the following statement:

Statement 1: Comparability graph recognition and finding a transitive orientation by Algorithm 2 *can be done in* $O(|E|\delta(G))$ *time.*

3) Colouring of the structural concurrency graph: Since the graph G_C can be transitively oriented, its minimum colouring can be obtained in linear time [27]. *Algorithm 3* uses the ideas of colouring of already oriented comparability graphs presented in [27], [45]. The algorithm recursively calls an additional function *Coluor*. Such a function simply orders all the graph vertices by their *height*. The height of vertices strictly refer to the transitive orientation of a graph. Thus, it can be directly used for the minimum colouring of a graph. At the bottom level (with height/coluor equal to 1) there are *sink* vertices (vertices that have no outgoing arcs in relation F). The largest value of height (colour) is assigned to the vertices that have no incoming arcs.

Algorithm 3 Coloring of a comparability graph

Input: Transitively oriented graph $G = (V, F)$
Output: Vector $colours$ of assigned vertices colours

1: **for each** $v \in V$ **do** $colours[v] \leftarrow 0$
2: **for each** $v \in V$ such that $colours[v] = 0$ **do** COLOUR(v)
3: **return** $colours[v]$
4: **function** COLOUR(v)
5: **for each** $u \in V$ such that $[\overrightarrow{vu} \in F]$ **do** COLOUR(u)
6: **if** v is sink **then** $colours[v] \leftarrow 1$
7: **else** $colours[v] \leftarrow max(colour[w] : [\overrightarrow{vw} \in F]) + 1$
8: **end function**

Clearly, the number of the recursive calls of function *Colour* is equal to $|V|$, and every arc is checked twice. Therefore, the computational complexity of the presented colouring method is bounded by $O(|V| + |E|)$.

If the structural concurrency graph is not a comparability graph, then one of the approximation coloring algorithms should be used [42].

4) Formation of the set of the decomposed components: Based on the coloured graph, the set of components is formed. Each set of places which obtain the same colour during coloring of G_C refers to an SM-net $S \in \mathcal{S}$, where $\mathcal{S}$ is a set of components covering all the places of the Petri net. Such set of places $P' \subseteq P$ of net $N = (P, T, B, M_0)$ generates an SM-net $S = (P', T', B', M'_0)$ as follows: $T' = \{t \in T : \exists p \in P' : (t, p) \in B \vee (p, t) \in B\}, B' = \{(p, t) : p \in P' \wedge t \in T' \wedge (p, t) \in B\} \cup \{(t, p) : p \in P' \wedge t \in T' \wedge (t, p) \in B\}, M'_0 = M_0$.

Note that there is a possibility that a particular component S contains unconnected input or output transitions. Such a situation is caused by the fact, that some places of the decomposed system are common to more than one component. Colouring of a concurrency graph just groups vertices (places) into separate sets, but does not assure common places for different components. Therefore, additional NOPs ought to be supplied to the components with unconnected input or output transitions (we shall explain such a situation in more detail by an example in Section V).

Algorithm 4 presents a method that supplements the components with the non-operational places. Additionally, the algorithm verifies whether the net can be properly decomposed and whether the NOPs can be supplied.

Algorithm 4 Supplementation of components by NOPs

Input: A well-formed Petri net $N = (P, T, B, M_0)$, its structural concurrency graph $G_C = (P, ||^A)$, set of SM-nets $\mathcal{S}$ obtained by means of minimum coloring of G_C
Output: If N can be decomposed: set $\mathcal{S}$ with the SM-nets supplemented by NOPs

1: $i \leftarrow 1$
2: **for each** $S \in \mathcal{S}$ [where $S = (P', T', B', M'_0)$] **do**
3: $I \leftarrow \emptyset$
4: $O \leftarrow \emptyset$
5: **for each** $p \in S$ **do**
6: **for each** $t \in p\bullet$ such that $t\bullet = \emptyset$ in S **do** $O \leftarrow O \cup \{t\}$
7: **for each** $t \in \bullet p$ such that $\bullet t = \emptyset$ in S **do** $I \leftarrow I \cup \{t\}$
8: **end for**
9: **while** $O \neq \emptyset$ **do**
10: select arbitrary transition t from O
11: $\{p\} \leftarrow \bullet t$
12: $t' \leftarrow t$
13: $V \leftarrow \emptyset$
14: $Q \leftarrow \emptyset$ *//where Q is a stack*
15: marked$\leftarrow$false
16: Q.push(t')
17: **repeat**
18: $t' \leftarrow Q$.pop
19: **if** $(t'\bullet \cap M_0) \setminus P' \neq \emptyset$ **then** marked$\leftarrow$true
20: $V = V \cup \{t'\}$
21: **for each** $t'' \in T$ such that $[t'' \in (t'\bullet)\bullet$ and $t'' \notin (T' \setminus I)]$ **do**
22: **if** $[\exists p' \in t'\bullet : (p, p') \notin ||^A$ and $t'' \notin V]$ **then**
23: Q.push(t'')
24: **end if**
25: **end for**
26: **for each** $t'' \in T$ such that $[t'' \in \bullet(\bullet t)$ and $t'' \notin (T' \setminus I)]$ **do**
27: **if** $[\exists p' \in t'\bullet : (p, p') \notin ||^A$ and $t'' \notin V]$ **then**
28: Q.push(t'')
29: **end if**
30: **end for**
31: **until** Q.empty
32: **if** $V \cap I = \emptyset$ **then**
33: **write:** "The net cannot be decomposed."
34: **return**
35: **end if**
36: $P' \leftarrow P' \cup \{NOP_i\}$
37: **for each** $t \in V \cap O$ **do** $t\bullet \leftarrow \{NOP_i\}$
38: **for each** $t \in V \cap I$ **do** $\bullet t \leftarrow \{NOP_i\}$
39: **if** [marked] **then** $M'_0 \leftarrow M'_0 \cup \{NOP_i\}$
40: $O \leftarrow O \setminus (V \cap O)$
41: $I \leftarrow I \setminus (V \cap I)$
42: $i \leftarrow i + 1$
43: **end while**
44: **end for**
45: **return** $\mathcal{S}$

Let us briefly explain the functionality of *Algorithm 4*. The proposed method uses DFS, taking into account the structural concurrency relation. Initially, strong connectedness of each of the obtained components is checked. The method searches for the unconnected transition outputs and the unconnected

transition inputs. If there are such transitions in component S, then S should be supplemented with the NOPs. For each unconnected transition output, an adequate unconnected transition input is searched. If a proper input cannot be found, then the net cannot be decomposed by means of the proposed technique (such a situation may be caused by the difference between real and structural concurrency relations, see Subsection IV-B1).

Finally, a NOP is supplied to the component. Note that if a NOP replaces at least one initially marked place, then it should also be initially marked.

Let us estimate the computational complexity of *Algorithm 4*. The outer *for each* loop is executed at most $|P|$ times, because each place $p \in P$ belongs to only one component $S \in \mathcal{S}$. Executions of the inner *for each* and *while..do* loops are disjoint, thus we shall analyse their complexity separately.

The inner *for each* loop (lines 5-8) is executed at most $|P|$ times and checks every transition at most twice, hence its execution time is bounded by $O(|P|+|T|)$. The *while..do* loop involves inner *repeat..until* and two *for each* loops. Clearly, the *for each* loops (lines 37, 38) are executed at most $|T|$ times. The *repeat..until* loop is performed at most $|T|$ times, since V holds all the transitions to be visited. It performs a DFS with additional checking of the structural concurrency relation, which can be executed in $O((|P|+|T|)|P|)$ time. Finally, the outer *while..do* loop is repeated for all the transitions in the set O, thus it is bounded by $O(|T|)$. Therefore, the execution of the whole *while..do* loop is bounded by $O(|P||T^2|(|P|+|T|))$. Now, we can formulate the following statement:

Statement 2: Supplementation of the decomposed components with non-operational places by *Algorithm 2* takes time $O(|P|^2|T|^2(|P|+|T|))$.

From the above analysis the total computational complexity of the whole decomposition process can be calculated. Let us recall all the partial results and their complexities:

- Formation of the graph G_C: $O((|P|+|T|)^5)$.
- Transitive orientation of G_C: $O(|E|\delta(G))$.
- Coloring of G_C: $O(|P|\delta(G))$.
- Formation of the set $\mathcal{S}$: $O(|P|^2|T|^2(|P|+|T|))$.

Let us conclude from the partial results the final statement regarding computational complexity and applicability of the proposed decomposition method.

Statement 3: Decomposition of a concurrent control system specified by an interpreted Petri net based on the application of comparability graphs can be done in $O((|P|+|T|)^5)$ time if the following conditions are satisfied:

- The structural concurrency graph of the net is transitively orientable (see Subsection IV-B2).
- The obtained components can be supplemented by NOPs.

Analysis of numerous benchmarks demonstrates that most of the parallel logical control algorithms satisfy the mentioned conditions (see Section VI).

C. Modelling, validation, synthesis and final implementation

After the system is decomposed, it should be modelled in a hardware description language (HDL) such as Verilog [61] or VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) [12], [75]. Each of the achieved SMCs forms a separate finite state machine (FSM). Therefore, each component is described independently. Please note that proper synchronization of SMCs must be assured by additional internal signals [21], [31], [48], [49], [66]. There are no restrictions regarding language or style used to describe the components, thus they can be modelled as traditional FSMs [6], or as microprogrammed controllers [67], depending on the designer needs [65].

The decomposed system described in an HDL can be validated using a software simulation. Finally, the system is synthesized and implemented in order to create a bit-stream (that is, a portion of data to be sent to configure an FPGA).

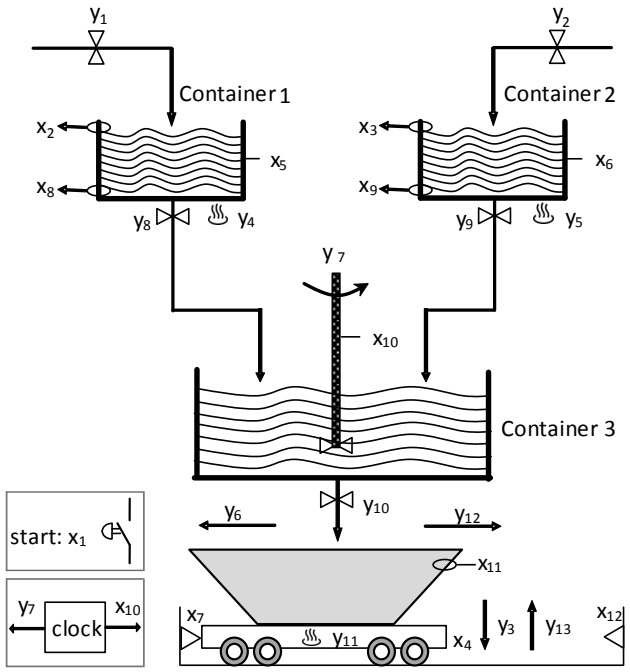


Fig. 1. A beverage production and distribution concurrent control system

V. Example

Let us illustrate the whole prototyping process by an industrial example. We shall use a modified version of a real-life concurrent control system, initially described in [63],- i.e. a beverage production and distribution machine presented in Fig. 1.

The system works as follows. Initially, the machine remains in an idle state. Pressing of a button on the operator console (illustrated as input signal x_1) starts the production process. Two valves (y_1 and y_2) are opened. Two containers (*Container 1* and *Container 2*) are filled with the liquid ingredients. When the upper limit of *Container 1* is reached (which is signalized by sensor x_2), the liquid in this container is warmed up (y_4). Similarly, reaching of the upper bounds of *Container 2* (signalized by x_3) initializes the warm-up process of the second container (y_5).

Simultaneously to the above procedure, a cup is placed on the cart (y_3), which is signalized by sensor x_4. When the

cup is placed, the cart moves to the left, until reaching the leftmost position, indicated by sensor x_7. The cup on the cart is being warmed up (y_{11}), waiting until the beverage is ready for distribution.

When the liquid reaches the required temperature in any of the containers (signalized by sensor x_5 for *Container 1* and x_6 for *Container 2*), the output valve of this container is opened (outputs y_8 and y_9, respectively). The ingredients from both containers are mixed (output y_7) in the third one (*Container 3*). The process is controlled by a clock. When the remaining time has elapsed (x_7) and additionally both upper containers are empty (x_8, x_9), the production is finished and the beverage is ready for distribution.

Finally, the output valve of *Container 3* is opened (y_{10}) and the liquid is poured to the cup located on the cart. When the upper level of the cup is reached (sensor x_{11}), the cart moves to the right (y_{12}). The moment when the cart reaches the appropriate position is signalized by the sensor x_{12}. Then the cup is taken from the cart and the system returns to the initial state.

An interpreted Petri net that specifies the described concurrent control system is shown in Fig. 2. There are 16 places and 13 transitions in the net. There are 12 input signals and 13 output signals. The net is live and safe, therefore it is well-formed.

To clarify the explanation of the proposed prototyping idea we shall use the reduction technique called *fusion of series of places* [50]. Such an action simply joins the series of places into single *macroplace*, as presented in Fig. 3 (left). In the presented example, there are eight macroplaces (denoted by the consecutive letters of the alphabet): macroplace a joins places p_2, p_5, p_8, macroplace b includes p_3, p_6, p_9, and so on. The reduced net (denoted as $\mathcal{N}_1$) preserves the properties of the initial net N, therefore it is well-formed.

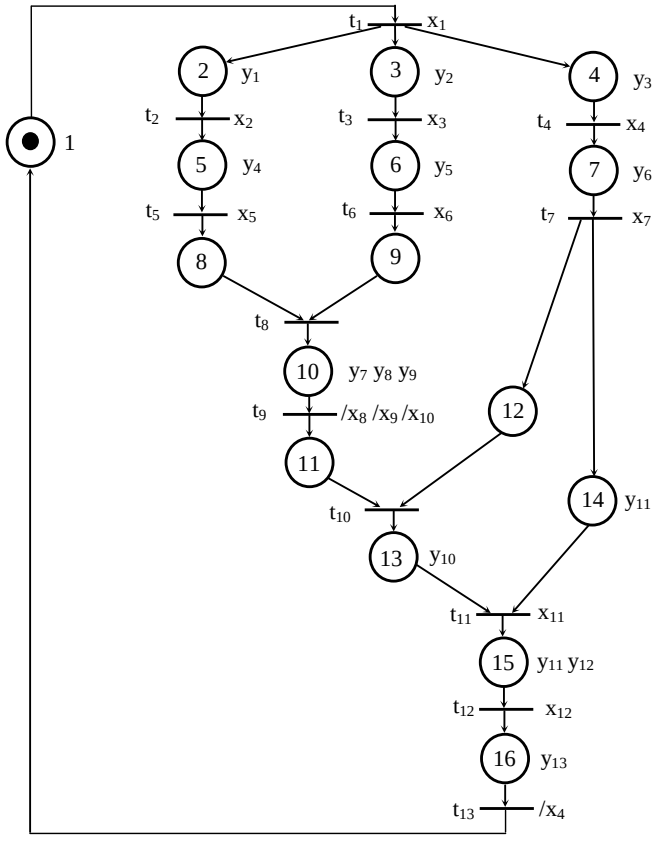


Fig. 2. An interpreted net N_1 (beverage production and distribution system)

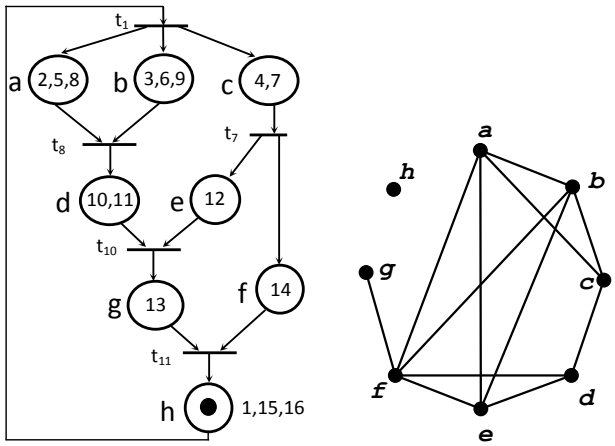


Fig. 3. Reduced net $\mathcal{N}_1$ (left) and its concurrency graph G_{C_1} (right)

Note that reduction of N_1 does not affect the results of decomposition. However, such a technique reduces the number of structural concurrency graph vertices and edges. More general cases of the sets of places which can be replaced by the macroplaces for simplifying analysis of the Petri nets are described in [36], [37].

Let us decompose $\mathcal{N}_1$ with the use of the proposed method. Initially, the structural concurrency graph G_{C_1} is formed. Such a graph is obtained according to *Algorithm 1*. The structural concurrency graph for the net $\mathcal{N}_1$ is shown in Fig. 3 (right). There are eight vertices in G_{C_1} that refer to the places of $\mathcal{N}_1$.

After the structural concurrency graph is obtained, *Algorithm 2* is applied, which searches for a transitive orientation F of the graph. We shall use the alphabetical order of the vertices to clarify the presentation. *Algorithm 2* starts with the edge ab, which is transitively oriented as $\overrightarrow{ab}$. Further orientation of edge $\overrightarrow{ac}$ forces orientation $\overrightarrow{bc}$. Similarly, orientation of edges $\overrightarrow{ae}$ and $\overrightarrow{af}$ forces orientation $\overrightarrow{ef}$, and so on. Figure 4 (left) presents the final transitive orientation of G_{C_1}.

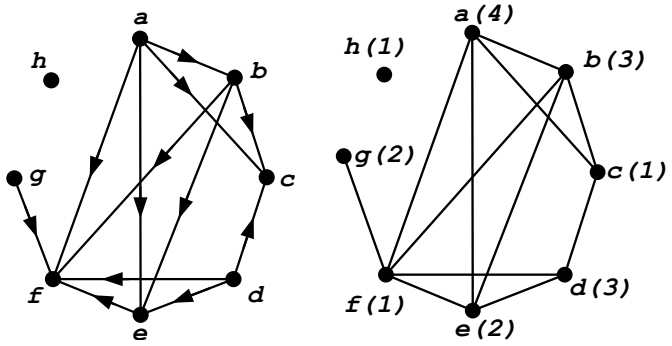


Fig. 4. Transitive orientation of G_{C_1} (left) and its minimum coloring (right)

Since there exists a TRO in G_{C_1}, such a graph can be optimally coloured by means of *Algorithm 3*. Let us present the coloring process in more detail. There are three sink vertices in the graph: c, f and h. According to the proposed algorithm, these sink vertices have the height equal to 1, thus they receive the first colour. Further computation assigns the second colour to vertices e and g, since their height is equal to 2. Consequently, vertices d and b are coloured by the third colour, while the remaining vertex a achieves the last, fourth colour. Summarizing, colouring of G_{C_1} results in four colours, as presented in Fig. 4 (right).

The obtained colours split the set of vertices into four sets, that correspond to the state machine components $\mathcal{S}$=$\{S_1, \ldots, S_4\}$: S_1=$\{c, f, h\}$, S_2=$\{e, g\}$, S_3=$\{b, d\}$, S_4=$\{a\}$. Figure 5 (left) illustrates the current step. Note that some of the input and output arcs of the transitions in three of the components remain unconnected (Fig. 5 left, components b-d).

Let us now apply *Algorithm 4* to finalize the decomposition process. Clearly, the first net (a), shown in Fig. 5 (left), does not require any changes, since it forms a proper state machine component. However, set S_2 contains the unconnected arcs to transition t_7 and from t_{11}. Similarly, S_3 also contains the unconnected arcs (a gap between transitions t_9 and t_6), and so does S_4 (a gap between transitions t_8 and t_1). At the beginning *Algorithm 4* searches for such unconnected transitions. Two sets are formed. The first one O=$\{t_8, t_{10}, t_{11}\}$ holds the transitions with unconnected outgoing arcs. The second set I=$\{t_1, t_1, t_7\}$ contains the transitions with unconnected incoming arcs. Note that t_1 occurs twice in the set I. In the further steps the algorithm tries to pair the transitions, by matching an output from O with an input from I. In our example, t_8 is associated with t_1, t_{10} with t_1, and t_{11} with t_7. The additional non-operational places are inserted between the unconnected arcs, as shown in Fig. 5 (right). Note that all three NOPs are initially marked. Since all the elements from the set O have been associated with the proper input transitions from I, *Algorithm 4* finishes successfully. This means that the net has been properly decomposed.

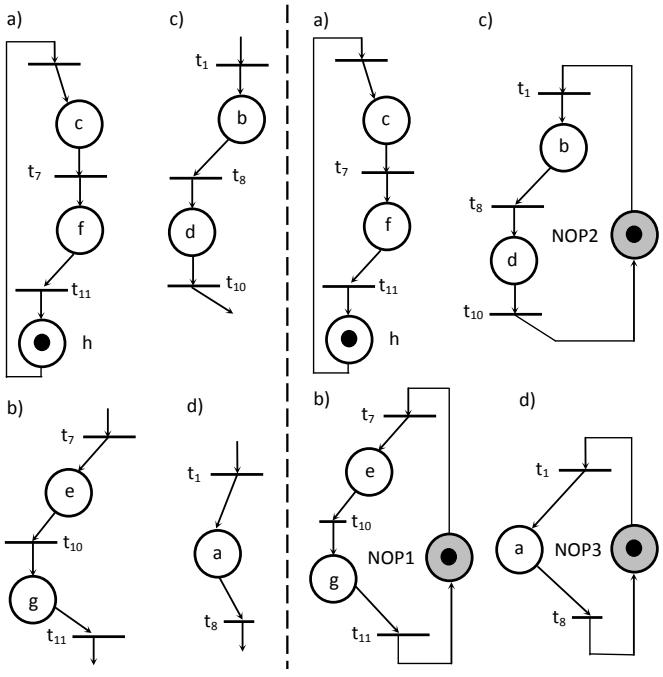


Fig. 5. Decomposed net $\mathcal{N}_1$ before and after execution of *Algorithm 4*

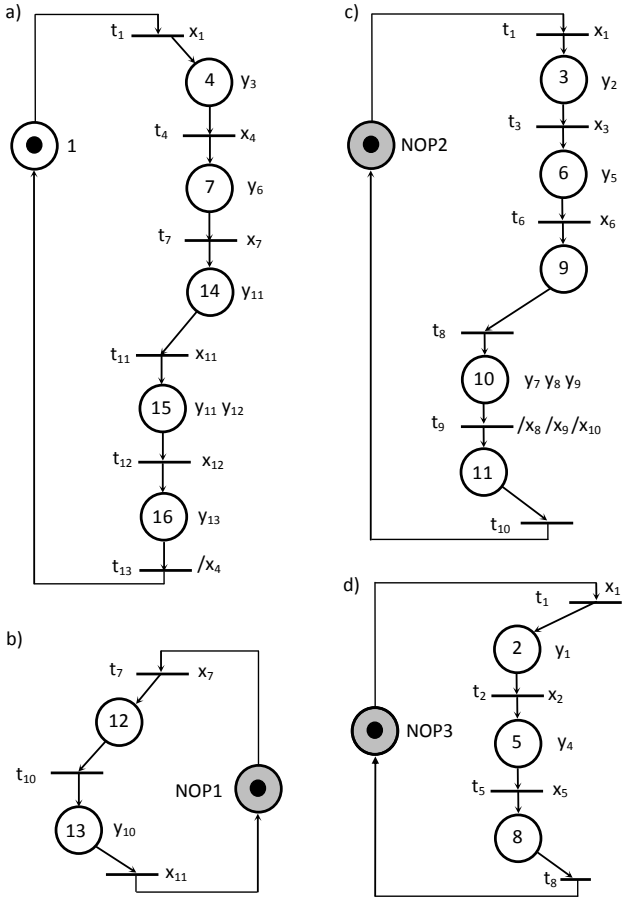


Fig. 6. Decomposed interpreted net N_1

When the decomposition process is finished, all the macroplaces in the obtained components are replaced by the original places. This operation is a reversal of the reduction process. The set of decomposed components consists of four SMCs $\mathcal{S}$=$\{S_1, \ldots, S_4\}$ that contain the following places:

- S_1=$\{p_1, p_4, p_7, p_{14}, p_{15}, p_{16}\}$,
- S_2=$\{p_{12}, p_{13}, NOP_1\}$,
- S_3=$\{p_3, p_6, p_9, p_{10}, p_{11}, NOP_2\}$,
- S_4=$\{p_2, p_5, p_8, NOP_3\}$.

Figure 6 presents the results obtained during the decomposition process. Let us briefly analyse them. The obtained components split the concurrent system into four modules. Each of them forms a sequential automaton. The first component (S_1) mainly controls the cart (its movement, warming-up of the cup placed on the cart, loading and unloading of the cup). The second one is in charge of proper transferring of the liquid from *Container 3* to the cup on the cart. The two remaining modules are in response of proper functionality of *Container 2* (component S_3) and *Container 1* (S_4). Additionally, S_3 controls mixing of the ingredients.

Next, each of the obtained SMCs is modelled independently in a hardware description language. In the presented example *Verilog* language was used. All four modules were described as synchronous automata. The *clock* signal (active high) synchronizes all the components, while *reset* returns them to the initial state (corresponding to the initially marked places). The schema of the decomposed system is shown in Fig. 7. Note that decomposed modules are synchronized by internal signals, according to the algorithms presented in [66]. Such signals are left out of the figures and diagrams in order to clarify the presentation of the proposed idea.

The results of simulation of the system, performed by the tool *Active-HDL* from *Aldec, Inc.*, is shown in Fig. 8. The behavior of the controller is represented by output signals $y_1, \ldots, y_{13}$, which are stimulated by the inputs $x_1, \ldots, x_{12}$. The values of the signals are represented in a graphical form, as they change in time. For example, pressing of the button (active signal x_1) starts the process and runs three operations (y_1, y_2 and y_3). After reaching the upper level of a particular container (x_2, x_3), the warming-up of the ingredients is launched (y_4,y_5). Meanwhile, the cart is moving to the left (y_3) until reaching the required position (x_4), and so on.

Finally, the system was implemented in a programmable device. The *XC7A100T* FPGA (*Artix-7* family) from *Xilinx* was used. The device was selected due to its integration with the board *Nexys 4 DDR*. The input values of the prototyped beverage production machine (set X) were specified by the integrated switches, while outputs (set Y) were assigned to the LEDs of the board. The clock signal was connected to the internal source (450 MHz). The logic synthesis and implementation were performed by the tool *Xilinx ISE 14.7*.

Table I shows the utilization of the implemented system. Particular components were realized as FSMs according to the Xilinx recommendations [70]. Additionally, internal states of FSMs were encoded using the Gray code to optimize the number of used logic blocks. Indeed, the number of occupied slices was reduced by one element in comparison to the traditional natural binary encoding.

TABLE I
UTILIZATION OF THE FPGA RESOURCES

FPGA (XC7A100T)	
Number of Slice Registers	10
Number of Slice LUTs	16
Number of occupied Slices	9
Number of IOs	27

VI. RESULTS OF EXPERIMENTS

The effectiveness of the proposed idea has been verified experimentally. To perform this, 111 benchmarks were checked. The library of test modules contains safe Petri nets that describe hypothetical and real devices, processes and actions, such as a controller of a volumetric feeder, a system for a chemical reactor, a concrete mixer, a process for picture frame manufacturing, a control system for a speedway tournament etc. All the verified nets can be found

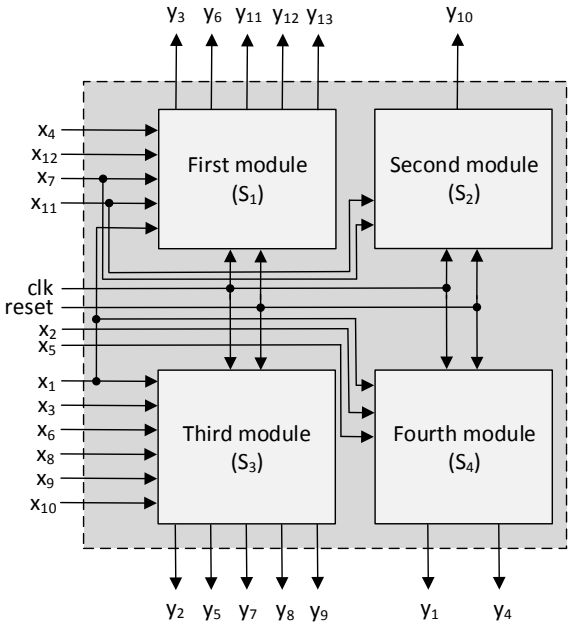


Fig. 7. Schema of the system (the synchronization signals are not shown)

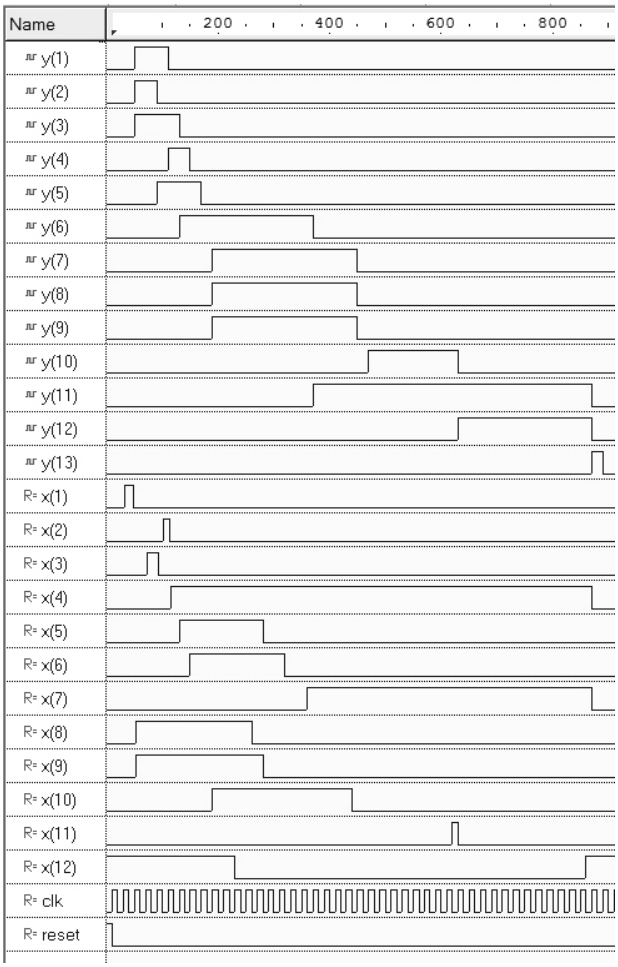


Fig. 8. Simulation results of the decomposed system

at: *www.hippo.iee.uz.zgora.pl* and *gres.uninova.pt/IOPT-Tools/* (username: *models*, password: *models*), where the source (reference), short description, incidence matrix and graphical representation are presented for every net. The aim of the experiments was to check the effectiveness of the proposed method, that is, the execution time of the whole process, presented in Section IV-B.

Table II shows the results of performed experiments (only benchmarks containing at least ten places are listed). The particular columns contain the following values:

- *Benchmark* - name of a net (sometimes truncated).
- $|P|$ - number of places in the net.
- $|T|$ - number of transitions in the net.
- *Number of SMCs* - number of components in the decomposition.
- *Number of NOPs* - number of supplied NOPs.
- *Time* - run-time of the whole process.

If the concurrency graph of the net is not a comparability graph, the proposed method cannot decompose the net, so in such cases, the number of SMCs is undetermined ($--$).

The obtained results indicate that the structural concurrency graph belongs to the class of comparability graphs in case of 94 benchmarks (out of 111). Furthermore, the non-operational places cannot be supplied for only two benchmarks. Such a situation was caused by a difference between structural and behavioral concurrency relations. Summarizing, 92 of the examined nets (83%) can be prototyped with the use of the proposed method. In the rest of the considered cases (17%) the concurrency graphs do not belong to the class of comparability graphs; in such cases one of the known methods should be applied (e.g., described in [10], [20]–[22], [31], [64], [73]).

VII. Conclusions

An idea for prototyping of concurrent control systems specified by the interpreted Petri nets is presented in the article. The main advantage of the presented method is the obtaining of minimum decomposition of a net into the sequential components in polynomial time. The computational complexity of the whole method is bounded by $O((|P| + |T|)^5)$.

The proposed technique applies comparability graph theory and includes new algorithms. The method can be applied to the nets with a concurrency graph belonging to the class of comparability graphs. The performed experiments show that most of the examined concurrent control systems satisfy this condition.

The main original contributions and results presented in the paper can be summarized as follows:

- The formulation of a novel algorithm for recognition of comparability graphs (*Algorithm 2*), supplemented by its computational complexity analysis.
- The formulation of a novel supplementation algorithm (*Algorithm 4*) of SMCs by NOPs and its computational complexity analysis.
- The formulation of a novel polynomial time decomposition method of concurrent control systems specified by the interpreted Petri nets and its computational complexity analysis.

Future work is going to be directed to the application of the obtained results to prototyping of concurrent systems implemented as reconfigurable controllers. The newest FPGA devices offer a mechanism called *partial reconfiguration* that allows reprogramming of selected portions of the system, without stopping the execution of the entire device [65], [71]. Such a process requires decomposition of the controller, which can be handled by the method proposed in this paper.

TABLE II
The results of experiments

Benchmark name	$\|P\|$	$\|T\|$	Number of SMCs	Number of NOPs	Time [ms]
np5	10	5	–	–	–
pascal_reach	10	8	4	4	0,23
speedway	10	7	3	2	0,13
dataflow_computation	11	7	–	–	–
lnet_p5n1	11	9	3	2	0,21
oil_sep_cov_s_net_v3	11	11	2	2	0,13
pnbrexpl_06	11	9	2	1	0,14
pnbrexpl_07	11	12	3	2	0,30
pnbrexpl_13	11	9	3	2	0,25
four_philosophers	12	8	–	–	–
inv_exp_3_4	12	4	3	0	0,23
lnet_p10n1	12	9	4	3	0,40
lnet_p6n1	12	11	3	2	0,34
frame_manufact	13	10	4	3	0,26
lnet_p2n2	13	7	4	6	0,22
lnet_p5n2	13	11	3	2	0,40
pn_campos_silva2	13	12	–	–	–
pn_campos_silva7	13	13	3	2	0,39
pnbrexpl_02	13	10	4	3	0,29
pnbrexpl_04	13	13	3	2	0,28
pnbrexpl_08	13	11	3	2	0,23
pnbrexpl_11	13	11	3	1	0,47
pnbrexpl_15	13	12	3	2	0,49
lnet_p9n1	14	16	3	2	0,57
philosophers_2	14	10	–	–	–
philosophers_2_rev_2	14	10	–	–	–
pn_silva_03	14	10	–	–	–
pnbrexpl_12	14	13	4	3	0,67
IEC	15	12	3	3	0,40
lnet_p8n4	15	16	3	2	0,67
oil_sep_cov_s_net_v2	15	11	4	5	0,25
mixer_one_cup	16	13	4	3	0,58
pnbrexpl_09	16	13	4	3	0,67
state_space16	16	16	8	0	1,55
s_net_frame_man_v1	17	16	4	3	0,32
pn_campos_silva6	18	11	–	–	–
mixer	19	15	–	–	–
s_net_f_man_v2_mod	19	18	4	3	0,49
milling	20	16	4	3	1,20
mixer_mod1	20	16	–	–	–
philosophers_5	20	15	–	–	–
lnet_p8n3	21	17	5	4	1,20
pn_fernandez3	21	20	4	4	0,42
oil_sep_cov_alfa_net	27	21	4	6	0,81
lnet_p8n2	28	17	9	11	2,46
oil_sep_cov_s_net	29	25	4	7	1,03
well_built_alfa_subpr	30	25	4	5	4,22
s_net_cpy_mm_subpr	31	28	4	4	2,88
alfa_net_cmms_syn	32	27	4	5	4,33
state_space32	32	32	16	0	13,14
lnet_p4n1	37	41	1	0	0,24
lnet_p7n1	41	32	9	9	13,24
state_space48	48	48	24	0	52,75
cn_crr7	56	15	28	21	48,77
cn_crr15	120	31	60	45	687,8
cn_crr25	200	51	100	75	4367
Average	**11,3**	**8,78**	**3,31**	**2,85**	**0,36**

ACKNOWLEDGMENT

The authors would like to thank the Associate Editor and anonymous reviewers for their constructive comments and helpful suggestions.

REFERENCES

[1] P. S. Andrews and J. Timmis. Inspiration for the next generation of artificial immune systems. In *Artificial Immune Systems*, pages 126–138. Springer, 2005.

[2] M. Augin, F. Boeri, and C. Andre. Systematic method of realization of interpreted Petri nets. *Digital Processes*, 6:55–68, 1980.

[3] A. Aybar and A. Iftar. Overlapping decompositions and expansions of Petri nets. *IEEE Transactions on Automatic Control*, 47:511–515, 2002.

[4] Z. Banaszak, J. Kuś, and M. Adamski. *Petri Nets: Modeling, Control and Synthesis of Discrete Systems*. Higher School of Engineering Publishing House, Zielona Góra, 1993. in Polish.

[5] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 1st edition, 2007.

[6] A. Barkalov and L. Titarenko. *Logic synthesis for FSM-based control units*, volume 53 of *LNEE*. Springer, Berlin, 2009.

[7] F. Basile and P. Chiacchio. On the implementation of supervised control of discrete event systems. *IEEE Transactions on Control Systems Technology*, 15(4):725–739, July 2007.

[8] G. Berthelot. Checking properties of nets using transformation. In *Advances in Petri Nets'85, Lecture Notes in Computer Science*, volume 222, pages 19–40. Springer-Verlag, 1986.

[9] B. Berthomieu, F. Peres, and F. Vernadat. Model checking bounded prioritized time Petri nets. In *ATVA'07 Proceedings of the 5th international conference on Automated technology for verification and analysis*, 2007.

[10] K. Bilinski, M. Adamski, J. Saul, and E. Dagless. Petri-net-based algorithms for parallel-controller synthesis. *IEE Proceedings - Computers and Digital Techniques*, 141(6):405–412, 1994.

[11] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. Roux. The expressive power of time Petri nets. *Theoretical Computer Science*, 474:1–20, 2013.

[12] S. Brown and Z. Vernesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw Hill, New York, USA, 2000.

[13] G. Bruno, R. Agarwal, A. Castella, and M. P. Pescarmona. CAB: an environment for developing concurrent applications. In *Applications and Theory of Petri Nets 1995, 16th International Conference (ICATPN 1995)*, June 1995.

[14] J. Carmona and J. Cortadella. State encoding of large asynchronous controllers. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pages 939–944, 2006.

[15] M. Chen and R. Hofestädt. Quantitative Petri net model of gene regulated metabolic networks in the cell. *In silico biology*, 3(3):347–365, 2003.

[16] Y. Chen, Z. Li, and A. Al-Ahmari. Nonpure Petri Net Supervisors for Optimal Deadlock Control of Flexible Manufacturing Systems. *IEEE Transactions on SMC: Systems*, 43(2):252–265, 2013.

[17] C.-N. Chou, Y.-S. Ho, C. Hsieh, and C.-Y. Huang. Symbolic model checking on SystemC designs. In *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 327–333, 2012.

[18] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *The Computer*, 43(3):224–242, Apr. 2000.

[19] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. The MIT Press, 1999.

[20] A. Costa, P. Barbosa, L. Gomes, F. Ramalho, J. Figueiredo, and A. Junior. Properties preservation in distributed execution of Petri nets models. *Emerging Trends in Tech. Innovation*, 314:241–250, 2010.

[21] A. Costa and L. Gomes. Petri net partitioning using net splitting operation. In *INDIN'2009 - 7th IEEE International Conference on Industrial Informatics*, pages 24–26, Cardiff, UK, June 2009.

[22] H. Costelha and P. Lima. Petri net robotic task plan representation: Modelling, analysis and execution. In V. Kordic, editor, *Autonomous Agents*, pages 65–89. InTech, 2010.

[23] L. Dai and W. Guo. Concurrent subsystem-component development model (CSCDM) for developing adaptive e-commerce systems. In *Computational Science and Its Applications*, pages 81–91. Springer, 2007.

[24] R. Diestel. *Graph Theory*. Springer-Verlag New York, 4th edition, 2010.

[25] A. Ellis. System and method for maintaining n number of simultaneous cryptographic sessions using a distributed computing environment, Nov. 19 2002. US Patent 6,484,257.

[26] E. Emerson. The beginning of model checking: A personal perspective. In *O. Grumberg, H. Veith (ed.), 25 Years of Model Checking: History, Achievements, Perspectives*, pages 27–45. Springer Verlag, 2008.

[27] M. C. Golumbic. The complexity of comparability graph recognition and coloring. *Computing*, 18(3):199–208, 1977.

[28] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, New York, 1980.

[29] I. Grobelna. Formal verification of embedded logic controller specification with computer deduction in temporal logic. *Przeglad Elektrotechniczny*, 87:47–50, 2011.

[30] I. Grobelna, M. Wiśniewska, R. Wiśniewski, M. Grobelny, and P. Mróz. Decomposition, validation and documentation of control process specification in form of a Petri net. In *7th International Conference on Human System Interactions - HSI 2014*, pages 232–237, Lisbon, Portugal, 2014.

[31] I. Grobelna, R. Wiśniewski, M. Grobelny, and M. Wiśniewska. Design and verification of real-life processes with application of Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 2016. DOI (to be activated): http://dx.doi.org/10.1109/TSMC.2016.2531673.

[32] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *TCS*, 234(1–2):59 – 84, 2000.

[33] R. Hadjidj and H. Boucheneb. On-the-fly TCTL model checking for time Petri nets. *Theoretical Computer Science*, 410(42):4241–4261, 2009.

[34] H. Hu, M. Zhou, and Z. Li. Supervisor design to enforce production ratio and absence of deadlock in automated manufacturing systems. *IEEE Transactions on SMC, Part A*, 41(2):201–212, March 2011.

[35] A. Karatkevich. *Dynamic analysis of Petri net-based discrete systems*. Lecture Notes in Control and Information Sciences, 356. Springer-Verlag, Berlin, 2007.

[36] A. Karatkevich. On macroplaces in Petri nets. In *Proceedings of IEEE East-West Design & Test Symposium - EWDTS' 08*, pages 418–422, Lviv, Ukraina, 2008. Kharkov National University of Radioelectronics, Lviv, The Institute of Electrical and Electronics Engineers, Inc.

[37] A. Karatkevich. Reduction of SM-components in Petri nets. *Telecommunication Review and Telecommunication News*, 6:806–808, 2008.

[38] A. Karatkevich and R. Wiśniewski. Relation between SM-covers and SM-decompositions of Petri nets. In *11th Int. Conf. of Computational Methods in Sciences and Engineering - ICCMSE'15*, volume AIP Conference Proceedings, Vol. 1702, pages 1–4, Athens, Greece, 2015.

[39] A. Kovalyov. Concurrency relation and the safety problem for Petri nets. In *Proc. of the 13th Int. Conf. on Application and Theory of Petri Nets 1992, LNCS*, volume 616, pages 299–309. Springer-Verlag, June 1992.

[40] A. Kovalyov. *Hardware Design and Petri Nets*, chapter A Polynomial Algorithm to Compute the Concurrency Relation of a Regular STG, pages 107–126. Springer US, Boston, MA, 2000.

[41] A. Kovalyov and J. Esparza. A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. In *In Proc. of the International Workshop WODES*, pages 1–6, 1995.

[42] M. Kubale. *Discrete Optimization: models and methods of graph coloring*. Wydawnictwa Naukowo Techniczne, Warszawa, 2002. in Polish.

[43] G. Łabiak, M. Adamski, M. Doligalski, J. Tkacz, and A. Bukowiec. UML modelling in rogorous design methodology for discrete controllers. *Int. Journal of Electronics and Telecommunications*, 58:27–34, 2012.

[44] M. Ma. Model Checking for Protocols Using Verds. In *5th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 231–234, 2011.

[45] F. Maffray. On the coloration of perfect graphs. In B. Reed and C. Sales, editors, *Recent Advances in Algorithms and Combinatorics*, volume 11 of *CMS Books in Mathematics*, pages 65–84. Springer-Verlag, 2003.

[46] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In *Selected Papers from the European Workshop on App. and Theory of Petri Nets*, pages 301–310, London, UK, 1982. Springer-Verlag.

[47] T. Miyamoto, H. Nogawa, S. Kumagai, et al. Autonomous distributed secret sharing storage system. *Systems and Computers in Japan*, 37(6):55–63, 2006.

[48] L. Montano, F. García-Izquierdo, and J. Villarroel. Using the time Petri net formalism for specification, validation, and code generation in robot-control applications. *Int. Journal of Robotics Research*, 19:59–76, 2000.

[49] F. Moutinho and L. Gomes. Asynchronous-channels within Petri net-based gals distributed embedded systems modeling. *Industrial Informatics, IEEE Transactions on*, 10(4):2024–2033, Nov 2014.

[50] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of IEEE*, volume 77, pages 548–580, 1989.

[51] T. Nishi and R. Maeno. Petri net modeling and decomposition method for solving production scheduling problems. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 1(2):262–271, 2007.

[52] W. Penczek and A. Półrola. *Advances in verification of Time Petri Nets and timed automata*. Springer, 2006.

[53] A. Ramirez-Trevino, I. Rivera-Rangel, and E. Lopez-Mellado. Observability of discrete event systems modeled by interpreted Petri nets. *IEEE Transactions on Robotics and Automation*, 19(4):557–565, Aug 2003.

[54] A. Ramirez-Trevino, E. Ruiz-Beltran, I. Rivera-Rangel, and E. Lopez-Mellado. Online fault diagnosis of discrete event systems. a Petri net-based approach. *IEEE Transactions on Automation Science and Engineering*, 4(1):31–39, Jan 2007.

[55] O. Ribeiro and J. Fernandes. Translating synchronous Petri nets into PROMELA for verifying behavioural properties. In *International Symposium on Industrial Embedded Systems SIES '07*, pages 266–273, 2007.

[56] I. Rivera-Rangel, A. Ramírez-Treviño, and E. López-Mellado. Building reduced Petri net models of discrete manufacturing systems. *Math. Comput. Model.*, 41(8-9):923–937, Apr. 2005.

[57] A. Santone, V. Intilangelo, and D. Raucci. Application of equivalence checking in a loan origination process in banking industry. In *Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 292–297. IEEE, 2013.

[58] L. Stefanowicz, M. Adamski, and R. Wiśniewski. Application of an exact transversal hypergraph in selection of SM-components. In *Technological innovation for the internet of things*, pages 250–257. Heidelberg - Dordrecht, Springer, 2013.

[59] J. Suk, Y. Lee, S. Kim, H. Koo, and J. Kim. System identification and stability evaluation of an unmanned aerial vehicle from automated flight tests. *KSME International Journal*, 17(5):654–667, 2003.

[60] M. Szpyrka, A. Biernacka, and J. Biernacki. Methods of translation of Petri nets to NuSMV language. In *Int. Workshop on Concurrency, Specification and Programming (CS&P)*, pages 245–256, 2014.

[61] D. Thomas and P. Moorby. *The Verilog Hardware Description Language*. Kluwer Academic Publishers, Norwell, MA, 5^{th} edition, 2002.

[62] A. Tzes, S. Kim, and W. McShane. Applications of Petri networks to transportation network modeling. *Vehicular Technology, IEEE Transactions on*, 45(2):391–400, May 1996.

[63] R. Valette. Comparative study of switching representation tool with GRAFCET and Petri nets. *Nouv. Autom.*, 23(12):377–382, Dec. 1978.

[64] W. van der Aalst. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[65] R. Wiśniewski. *Synthesis of compositional microprogram control units for programmable devices*, volume 14 of *LNCCS*. University of Zielona Góra Press, Zielona Góra, 2009.

[66] R. Wiśniewski. *Prototyping of Concurrent Control Systems Implemented in FPGA Devices*. Advances in Industrial Control. Springer International Publishing, 2017.

[67] R. Wiśniewski, A. Barkalov, L. Titarenko, and W. Halang. Design of microprogrammed controllers to be implemented in FPGAs. *Int. Journal of Applied Mathematics and Computer Science*, 21(2):401–412, 2011.

[68] R. Wiśniewski, Ł. Stefanowicz, A. Bukowiec, and J. Lipiński. Theoretical aspects of Petri nets decomposition based on invariants and hypergraphs. In *Lecture Notes in Electrical Engineering: Multimedia and Ubiquitous Engineering*, volume 308, pages 371–376, 2014.

[69] R. Wiśniewski, Ł. Stefanowicz, M. Wiśniewska, and D. Kur. Exact cover of states in the discrete state-space system. In *AIP Conference Proceedings*, volume 1702, 2015.

[70] Xilinx FSM style implementation: http://www.xilinx.com/support/documentation/university/ise-teaching/hdl-design/14x/nexys3/verilog/docs-pdf/lab10.pdf. Accessed: 2016-02-27.

[71] Xilinx partial reconfiguration tutorial: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf.

[72] D. A. Zaitsev. Compositional analysis of Petri nets. *Cybernetics and Systems Analysis*, 42(1), 2006.

[73] A. Zakrevskij, Y. Pottosin, and L. Cheremisinova. *Design of Logical Control Devices*. TUT Press, Moskov, 2009.

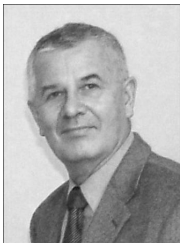
[74] R. Zurawski and M. Zhou. Petri nets and industrial applications: A tutorial. *IEEE Trans. on Industrial Electronics*, 41(6):567–583, 1994.

[75] M. Zwolinski. *Digital system design with VHDL*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

Remigiusz Wiśniewski received his doctoral degree in computer science from the University of Zielona Góra in 2008. His general research interests include design of the concurrent systems, graphs and hypergraphs, cryptology, FPGAs.

In 2000-2003 he worked for Aldec, Inc. (with a placement at headquarter, Henderson, NV, USA), where he conducted specialized trainings (for companies such as Xilinx and Intel). Since 2003, he has been an assistant professor of the University of Zielona Góra (currently with the Institute of Electrical Engineering). Co-founder and coordinator of the research project *Hippo*: www.hippo.iee.uz.zgora.pl

Andrei Karatkevich received his M.Sc. and Ph.D degrees in computer science from the Belarusian State University of Informatics and Radioelectronics in 1998 and D.Sc. degree in computer science from the Ilmenau University of Technology.
Currently he is a professor at the Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra. His main research interests focus on formal analysis of Petri nets and other parallel discrete systems, applications of the graph theory in logical design and Boolean algebra.



Marian Adamski received his M.Sc. degree in Electrical Engineering from Poznan Technical University, Poland, the Ph.D. degree in Control and Computer Engineering from Silesian Technical University, Gliwice, Poland, and Habilitated Doctor (D.Sc.) degree in Computer Engineering from Warsaw University of Technology, Poland. Currently, he is Professor of Computer Engineering.

His main research interests includes mathematical logic and Petri nets in digital systems design, formal development of logic controller programs, VHDL, FPLD and FPGA in industrial applications.

Anikó Costa received her engineer degree in electrical engineering from Czech Technical University in Prague in 1992, her MsC degree in informatics and PhD degree in electrical engineering from Nova University of Lisbon in 2003 and 2010 respectively. Currently she is an assistant professor at Faculty of Science and Technology of Nova University of Lisbon. Her main research interest are model based development of embedded systems using Petri nets and statechart, digital system design and hardware-software co-design.

Luís Gomes received his Electrotech. Eng. degree from UniversidadeTécnica de Lisboa, Lisbon, Portugal, in 1981, and a Ph.D. degree in digital Systems from Nova University of Lisboa, in 1997. He is an associate professor at the Electrical Engineering Department, Faculty of Sciences and Technology of Nova University of Lisboa, Portugal and a researcher at UNINOVA Institute, Portugal. From1984 to 1987, he was with EID, a Portuguese medium enterprise in the area of electronic system design. His main interests include the usage of Petri nets and other concurrency models applied to reconfigurable and embedded systems co-design.