10.1515/acsc-2016-0019

*Archives of Control Sciences*
*Volume* 26(LXII), 2016
*No.* 3, *pages 339–365*

# Tools and methods for RTCP-nets modeling and verification

MARCIN SZPYRKA, JERZY BIERNACKI and AGNIESZKA BIERNACKA

RTCP-nets are high level Petri nets similar to timed colored Petri nets, but with different time model and some structural restrictions. The paper deals with practical aspects of using RTCP-nets for modeling and verification of real-time systems. It contains a survey of software tools developed to support RTCP-nets. Verification of RTCP-nets is based on coverability graphs which represent the set of reachable states in the form of directed graph. Two approaches to verification of RTCP-nets are considered in the paper. The former one is oriented towards states and is based on translation of a coverability graph into nuXmv (NuSMV) finite state model. The later approach is oriented towards transitions and uses the CADP toolkit to check whether requirements given as $\mu$-calculus formulae hold for a given coverability graph. All presented concepts are discussed using illustrative examples

**Key words:** RTCP-nets, Petri nets, model checking, coverability graphs, nuXmv, CADP.

## 1. Introduction

Colored Petri nets (CP-nets [13], [14]) is one of the most widespread classes of high level Petri nets. They provide a discrete-event modeling language combining capabilities of Petri nets [19], [18] with the capabilities of a high-level programming language that provides the primitives for the definition of data types, variables, expressions for describing data manipulation etc. CP-nets are aimed towards modeling a very broad class of systems including information technology, automatics, electronics, biology, chemistry, medicine etc. All those systems can be characterized as concurrent ones. Equipped with a time model, CP-nets may be used to model real-time systems. A survey of time extensions of Petri nets can be found for example in [20]. There are two main time models for CP-nets, one based on time stamps attached to tokens described in [13] and the other called interval time Petri nets proposed in [1]. Due to the CPN Tools [15], the most often used modeling environment for CP-nets, the former model is more popular. Unfortunately, reachability graphs for timed CP-nets are usually infinite, thus verification of

The Authors are with AGH University of Science and Technology, Department of Applied Computer Science, Al. Mickiewicza 30, 30-059 Krakow, Poland. M Szpyrka is the corresponding author, e-mail: mszpyrka@agh.edu.pl.

Received 25.07.2016.

such nets using model checking techniques is limited. To overcome the problem, reachability graphs with equivalence classes can be used [13], but it is necessary to define equivalence relations for each model individually. Moreover, when reachability graphs with equivalence classes are used, possibilities of verification of some properties can be lost.

RTCP-nets (real-time colored Petri nets, [21], [22]) are based on CP-nest but are equipped with different time model that makes them suitable for modeling real-time systems. Analysis of RTCP-nets may be carried out with coverability graphs [21]. Such a graph may be used to analyze boundedness, liveness and timing properties of the corresponding RTCP-net. If the set of reachable markings of an RTCP-net is finite, it is possible to construct a finite coverability graph that represents the set of all reachable states regardless of whether the set is finite or infinite.

The paper focuses on practical aspects of using RTCP-nets. It contains a survey of tools and verification methods developed for RTCP-nets recently. The paper is organized as follows. Section 2 contains the formal definition of RTCP-nets and description of basic ideas connected with them. Rules that must be obeyed while modeling RTCP-nets with CPN Tools are presented in Section 3. Section 4 deals with reachability and coverability graphs. Section 5 presents RTCP-net Compiler used to generate Java representation for models. This representation is used to generate reachability and coverability graphs. A method of verification of RTCP-nets with the nuXmv tool is considered in Section 6. An algorithm of translation of reachability/coverability graphs into nuXmv states machines is also given in the section. Action-based verification of RTCP-nets with the CADP toolbox is considered in Section 7. All methods considered in the paper are illustrated using the example presented in Section 8. A short summary is given in the final section.

## 2. RTCP-nets

RTCP-nets [21], [22] are high level Petri nets similar to CP-nets. We use tokens data types (colors), variables and expressions to deal with the tokens flow. Due to the fact that we use CPN Tools [15] for designing RTCP-nets, the inscription language is CPN ML [14]. Let $\mathcal{E}$ denote the set of expressions provided by the inscription language. For $x \in \mathcal{E}$, $\mathcal{V}(x)$ denotes the set of all free variables in $x$ (i.e. not bound in the local environment of the expression), and $\mathcal{T}(x)$ denotes the *type* of $x$, i.e. the type of values obtained by evaluating the expression. Let $V'$ be a set of variables. The set of expressions $x \in \mathcal{E}$ such that $\mathcal{V}(x) \subseteq V'$ is denoted $\mathcal{E}_{V'}$.

Let *Bool* denote the boolean type (containing the elements $\{false, true\}$, and having the standard operations from propositional logic). For an arc $a$, $P(a)$ and $T(a)$ denotes the *place node* and the *transition node* of the arc, respectively.

**Definition 3** *An* RTCP-net *is a tuple* $\mathcal{N} = (P, T, A, \Sigma, V, C, G, I, E_M, E_S, M_0, S_0)$*, where:*

1. *P is a non-empty finite set of* places.

2. $T$ *is a non-empty finite set of* transitions *(*$P \cap T = \emptyset$*).*
3. $A \subseteq (P \times T) \cup (T \times P)$ *is a* flow relation.
4. $\Sigma$ *is a non-empty finite set of non-empty* types.
5. $V$ *is a finite set of* variables *such that* $\forall_{v \in V} \mathcal{T}(v) \in \Sigma$.
6. $C\colon P \to \Sigma$ *is a* type function.
7. $G\colon T \to \mathcal{E}_V$ *is a* guard function, *such that* $\forall_{t \in T} \mathcal{T}(G(t)) = Bool$.
8. $I\colon T \to \mathbb{N} \cup \{0\}$ *is a* priority function.
9. $E_M\colon A \to \mathcal{E}_V$ *is an* arc expression function, *such that* $\forall_{a \in A} \mathcal{T}(E_M(a)) = C(P(a))$.
10. $E_S\colon A \to \mathcal{E}_V$ *is an* arc time expression function, *such that* $\forall_{a \in A} \mathcal{T}(E_S(a)) = \mathbb{Q}^+ \cup \{0\}$.
11. $M_0$ *is an* initial marking, *such that* $\forall_{p \in P} M_0(p) \in 2^{C(p)^*}$, *where* $2^{C(p)^*}$ *denotes the set of all multi-sets over the set* $C(p)$.
12. $S_0\colon P \to \mathbb{Q}$ *is an* initial time function, *which maps each place to a rational value called* initial place time.

RTCP-nets are an adaptation of CP-nets to make modeling and verification of real-time systems easier and more efficient. In comparison to CP-nets the set of arcs is defined as a relation (multiple arcs are not allowed). Two expressions are attached to each arc: a weight expression and a time expression. For any arc, each evaluation of the arc weight expression must yield a single token belonging to the corresponding type; and each evaluation of the arc time expression must yield a non-negative rational value. Finally, each place has its own local clock attached that measures the place time. Any positive value of a place time describes how long tokens in the corresponding place will be inaccessible for any transition. Negative values represent tokens age (all tokens in a given place share the clock value). It is possible to specify how old a token should be so that a transition may consume it.

**Definition 4** *Let an RTCP-net* $\mathcal{N}$ *be given. A* marking *of* $\mathcal{N}$ *is a function M defined on the set of places P, such that:* $\forall_{p \in P} M(p) \in 2^{C(p)^*}$. *A* time function *is a function S defined on P, such that:* $\forall_{p \in P} S(p) \in \mathbb{Q}$. *A* state *is a pair* $(M,S)$*, where M is a marking and S is a time function. The* initial state *is the pair* $(M_0, S_0)$.

Let $P$ be an ordered set, $P = \{p_1, p_2, \ldots, p_n\}$. Both a marking $M$ and a time function $S$ can be represented by vectors with $|P|$ entries.

For any node $x \in X = P \cup T$ we define the set of *input nodes* $In(x) = \{y \in X\colon (y,x) \in A\}$ and the set of *output nodes* $Out(x) = \{y \in X\colon (x,y) \in A\}$.

Let $\mathcal{V}(t)$ be the set of variables that occur in the expressions of arcs surrounding a transition $t$ and in the guard of the transition. A binding of a transition $t$ is a substitution $b$ that replaces each variable of $\mathcal{V}(t)$ with a value of the corresponding type, such that the guard evaluates to *true*. For a transition $t$ and its binding $b$, $G(t)_b$ denotes the evaluation of the guard expression in the binding $b$ and $E_M(p,t)_b$ and $E_S(p,t)_b$ denote the evaluation of the weight and the time expression in the binding $b$, respectively.

**Definition 5** *A transition $t \in T$ is* enabled *in a state $(M,S)$ in a binding $b$ iff the following conditions hold:*

$$\forall_{p \in In(t)} E_M(p,t)_b \in M(p) \wedge E_S(p,t)_b \leqslant -S(p), \tag{1}$$

$$\forall_{p \in Out(t)} S(p) \leqslant 0. \tag{2}$$

*and for any transition $t' \neq t$ that satisfies the above conditions, $I(t') \leqslant I(t)$ or $In(t) \cap In(t') = Out(t) \cap Out(t') = \emptyset$.*

It means that a transition is enabled if all input places contain suitable tokens and have suitable time (age of tokens), all output places are accessible and no other transition with a higher priority strives for the same input or output places. If a transition $t \in T$ is enabled in a state $(M_1,S_1)$ in a binding $b$ it may *fire*, changing the state $(M_1,S_1)$ to another state $(M_2,S_2)$, such that:

$$M_2(p) = \begin{cases} M_1(p) - \{E_M(p,t)_b\} \cup \{E_M(t,p)_b\} & \text{for } p \in In(t) \cup Out(t), \\ M_1(p) \cup \{E_M(t,p)_b\} & \text{for } p \in In(t) - Out(t), \\ M_1(p) - \{E_M(p,t)_b\} & \text{for } p \in Out(t) - In(t), \\ M_1(p) & \text{otherwise.} \end{cases} \tag{3}$$

and

$$S_2(p) = \begin{cases} E_S(t,p)_b & \text{for } p \in Out(t), \\ 0 & \text{for } p \in In(t) - Out(t), \\ S_1(p) & \text{otherwise.} \end{cases} \tag{4}$$

We write $(M_1,S_1) \xrightarrow{(t,b)} (M_2,S_2)$ to denote the change of states. Moreover, every time the clock goes forward, all time stamps are decreased by the same value.

**Definition 6** *Let $(M,S)$ be a state and $e = (1,1,\ldots,1)$ a vector with $|P|$ entries. The state $(M,S)$ is changed into a state $(M',S')$ by a* passage of time $\tau \in \mathbb{Q}^+$*, denoted by $(M,S) \xrightarrow{\tau} (M',S')$, iff $M = M'$ and the passage of time $\tau$ is possible, i.e., no transition is enabled in any state $(M,S'')$, such that: $S'' = S - \tau' \cdot e$, for $0 \leqslant \tau' < \tau$.*

For the sake of simplicity, we will assume that there is one passage of time (sometimes equal to 0) between firings of two consecutive transitions. A *firing sequence* of an RTCP-net $\mathcal{N}$ is a sequence of pairs $\alpha = (t_1, b_1), (t_2, b_2), \ldots$, such that $b_i$ is a binding of the transition $t_i$, for $i = 1, 2, \ldots$ The firing sequence is *feasible* from a state $(M_1, S_1)$ iff there exists a sequence of states such that:

$$(M_1, S_1) \xrightarrow{\tau_1} (M_1, S_1') \xrightarrow{(t_1, b_1)} (M_2, S_2) \xrightarrow{\tau_2} \cdots \xrightarrow{(t_n, b_n)} (M_{n+1}, S_{n+1}) \xrightarrow{\tau_{n+1}} \ldots \tag{5}$$

A firing sequence may be finite or infinite. A state $(M', S')$ is *reachable* from a state $(M, S)$ iff there exists a finite firing sequence $\alpha$ feasible from the state $(M, S)$ and leading to the state $(M', S')$. The set of all states that are reachable from $(M, S)$ is denoted by $\mathcal{R}(M, S)$.

## 3. Design of RTCP-nets with CPN Tools

*CPN Tools* [15] is a modeling environment for editing, simulating, and analyzing colored Petri nets [14], [12]. Due to syntax differences CPN Tools cannot be use to simulate or verify RTCP-nets. However, the industrial-strength modeling environment can be used for constructing RTCP-net models, which can be later processed with the *RTCP-net Compiler* (see Section 5). An RTCP-net designed with the CPN Tools is presented in Fig. 1. The following editing rules must be obeyed while designing RTCP-nets:

- A place initial state is written *marking@clock_value*.
- The notation for multisets is $m_1(e_1) + m_2(e_2) + \ldots$, where $m$ stands for multiplicity and $e$ stands for element. The multiplicity equal to 1 and the corresponding brackets can be omitted.
- Expressions for single direction arcs are written *weight_expression@time_expression*.
- Expressions for double direction arcs are written *input_weight_exp@input_time_exp* | *output_weight_exp@output_time_exp*, where arcs are consider input or output from the places point of view.
- Time expressions equal to 0 are omitted.

An example of an RTCP-net is given in Fig. 1. The model uses two token types: $P$ with elements $a$ and $b$ and a singleton $R$ with element $r$. The model represents two types of processes (represented by the $a$ and $b$ tokens), which compete for some resources stored in places $p6$ and $p7$. The $a$ process reaches two states, which are represented by placing the $a$ token in places $p1$ and $p2$ respectively. Similarly, each of the two $b$ processes reaches three states, which are represented by placing the $b$ tokens in places $p3$, $p4$ and $p5$ respectively. Let us consider expressions for selected arcs.

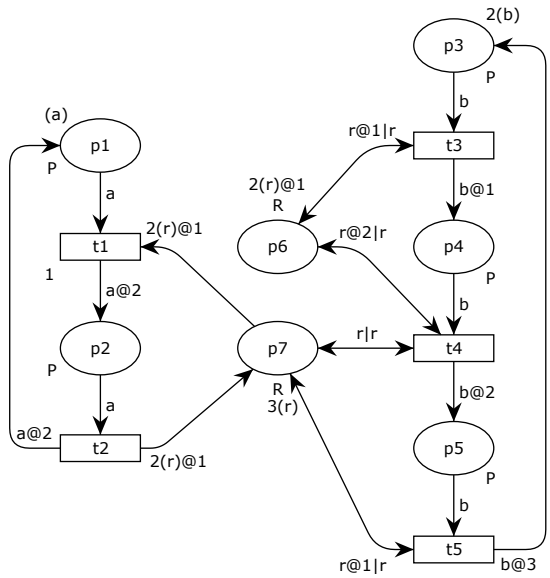



Figure 1: RTCP-net example (shared resources).

- $(p1,t1)$ – Transition $t1$ takes one token $a$ from place $p1$. The place time must be less or equal to 0. The transition sets the place time to 0.
- $(t1,p2)$ – Transition $t1$ adds one token $a$ to place $p2$ and sets the place time to 2.
- $(p7,t1)$ – Transition $t1$ takes one token $r$ from place $p7$. The place time must be less or equal to $-1$. The transition sets the place time to 0.
- $\{p7,t5\}$ (double arc) – Transition $t5$ takes one token $r$ from place $p7$. The place time must be less or equal to 0. Then the transition adds one token $r$ to the place and sets its time to 1.

A small part of a simulation log for the considered RTCP-net is shown in Table 8. Tokens in place $p6$ are inaccessible in the initial state $(M_0,S_0)$; transition $t1$ must take one-time unit old token from place $p7$, thus there is no enable transition in the initial state. However, there is possible the passage of time with $\tau = 1$ that leads to state $(M_1,S_1)$ with two enable transitions $t1$ and $t3$. Due to the lack of variables there are only empty bindings in the model, so they are omitted in firing sequences. The states presented in Table 8 have been taken from the following firing sequence.

$$\begin{aligned}&(M_0,S_0) \xrightarrow{1} (M_1,S_1) \xrightarrow{t_1} (M_2,S_2) \xrightarrow{t_3} (M_3,S_3) \xrightarrow{1} \\ &(M_4,S_4) \xrightarrow{t_3} (M_5,S_5) \xrightarrow{1} (M_6,S_6) \xrightarrow{t_4} (M_7,S_7) \xrightarrow{t_2} \\ &(M_8,S_8) \xrightarrow{2} (M_9,S_9)\end{aligned} \tag{6}$$

Table 8: RTCP-net simulation log

| | p1 | p2 | p3 | p4 | p5 | p6 | p7 |
|---|---|---|---|---|---|---|---|
| $M_0$ | a | – | 2(b) | – | – | 2(r) | 3(r) |
| $S_0$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $M_1$ | a | – | 2(b) | – | – | 2(r) | 3(r) |
| $S_1$ | -1 | -1 | -1 | -1 | -1 | 0 | -1 |
| $M_2$ | – | a | 2(b) | – | – | 2(r) | 2(r) |
| $S_2$ | 0 | 2 | -1 | -1 | -1 | 0 | 0 |
| $M_3$ | – | a | b | b | – | 2(r) | 2(r) |
| $S_3$ | 0 | 2 | 0 | 1 | -1 | 1 | 0 |
| $M_4$ | – | a | b | b | – | 2(r) | 2(r) |
| $S_4$ | -1 | 1 | -1 | 0 | -2 | 0 | -1 |
| $M_5$ | – | a | – | 2(b) | – | 2(r) | 2(r) |
| $S_5$ | -1 | 1 | 0 | 1 | -2 | 1 | -1 |
| $M_6$ | – | a | – | 2(b) | – | 2(r) | 2(r) |
| $S_6$ | -2 | 0 | -1 | 0 | -3 | 0 | -2 |
| $M_7$ | – | a | – | b | b | 2(r) | 2(r) |
| $S_7$ | -2 | 0 | -1 | 0 | 2 | 2 | 0 |
| $M_8$ | a | – | – | b | b | 2(r) | 3(r) |
| $S_8$ | 2 | 0 | -1 | 0 | 2 | 2 | 1 |
| $M_9$ | a | – | – | b | b | 2(r) | 3(r) |
| $S_9$ | 0 | -2 | -3 | -2 | 0 | 0 | -1 |

Hierarchical RTCP-nets are based on the construct used for hierarchical CP-nets. Substitution transitions and fusion places [13] are used to combine pages but they are a mere designing convenience. The former idea allows the user to refine a transition and its surrounding arcs to a more complex net, which usually gives a more precise and detailed description of the activity represented by the substitution transition. In comparison with CP-nets each socket node must have only one port node assigned and vice versa. Thus, a hierarchical net can be easily ”squashed” into a non-hierarchical one.

A fusion of places allows users to specify a set of places that should be considered as a single one. It means, that they all represent a single conceptual place, but are drawn as separate individual places (e.g. for clarity reasons). The places participating in such a *fusion set* may belong to several different pages. They must have the same types and

initial markings. Hierarchical RTCP-nets designed with CPN Tools can be processed with RTCP-nets Compiler.

## 4. Reachability and coverability graphs

Analysis of RTCP-nets may be carried out by the use of reachability graphs. The set of reachable states $\mathcal{R}(M_0, S_0)$ is represented as a weighted, directed graph. Each node corresponds to a unique state, consisting of a net marking and a time vector, such that the state is the result of firing of a transition. Each arc represents a change from a state $(M_i, S_i)$ to a state $(M_j, S_j)$ resulting from a passage of time $\tau \geqslant 0$ and the firing of a transition $t$ in one of its bindings. A label of such a graph is written $(t, b)/\tau$.

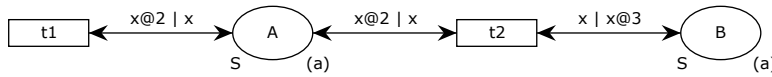



Figure 2: RTCP-net example.

A reachability graph for an RTCP-nets may be infinite even though the set of the net markings is finite. Let's consider the RTCP-net presented in Fig. 2, where type $S$ is a singleton with value $a$ and $x$ is a variable of type $S$. The following firing sequence is feasible in the initial state $((a,a),(0,0))$:

$$\begin{aligned}
&((a,a),(0,0)) \overset{(t1,(a/x))}{\rightarrow} ((a,a),(2,0)) \xrightarrow{2} \\
&((a,a),(0,-2)) \overset{(t1,(a/x))}{\rightarrow} ((a,a),(2,-2)) \xrightarrow{2} \\
&((a,a),(0,-4)) \overset{(t1,(a/x))}{\rightarrow} ((a,a),(2,-4)) \xrightarrow{2} \\
&((a,a),(0,-6)) \overset{(t1,(a/x))}{\rightarrow} ((a,a),(2,-6)) \xrightarrow{2} \ldots
\end{aligned} \tag{7}$$

This example shows that transition $t2$ can be starved by the other one. The firing sequence represents an infinite path in the RTCP-net reachability graph. All states in this sequence share the same marking, but the time of place $B$ is infinitely decreasing, which generates the infinite set of possible states.

However, we can point out sets of states that can be treated as indistinguishable from the formal analysis point view. For example, let us consider states $((a,a),(2,-4))$ and $((a,a),(2,-6))$. The same transitions are enabled in both states and the same sequences of actions are feasible from the states. Both states have the same markings and the same *level* of tokens accessibility, i.e. we have to wait 2 time-units to take the token from the place $A$ and the token in the place $B$ is already accessible. The token in the place $B$ is accessible if its age is at least 3 time-units, i.e. the value of the time stamp is equal to or less than $-3$. It makes no difference whether the time stamp is equal to $-4$, $-6$, etc.

The states are said to *cover* each other and only one node in the coverability graph will be used to represent them [21].

To define the coverability relation formally we need two numbers that characterize the considered place time. Let $p \in P$ be a place of an RTCP-net $\mathcal{N}$. The *minimal accessibility age* of a place $p$ ($\delta_{\min}(p)$) denotes the age when tokens in the place become accessible for at least one output transition of the place (for at least one of its binding). The *maximal accessibility age* of a place $p$ ($\delta_{\max}(p)$) denotes the age when tokens in the place become accessible for all output transitions of the place (considering any binding of these transitions).

**Definition 7** *Let $\mathcal{N}$ be an RTCP-net and let $(M_1,S_1)$ and $(M_2,S_2)$ be states of the net. The state $(M_1,S_1)$ is said to* cover *the state $(M_2,S_2)$ ($(M_1,S_1) \simeq (M_2,S_2)$) iff $M_1 = M_2$ and the following condition holds:*

$$\forall_{p\in P}(S_1(p) = S_2(p)) \vee (S_1(p) \leqslant -\delta_{max}(p) \wedge S_2(p) \leqslant -\delta_{max}(p)). \tag{8}$$

The *coverability relation* $\simeq$ is an equivalence relation on $\mathcal{R}(M_0,S_0)$. It has been proven [21] that for any two states such that $(M_1,S_1) \simeq (M_2,S_2)$ it also holds $\mathcal{L}(M_1,S_1) = \mathcal{L}(M_2,S_2)$.

The reachability and coverability graphs are constructed in a similar way. They differ only about the way a new node is added to the graph. For the coverability graph, after calculating a new node, we check first whether there already exists a node that covers the new one. If so, we add only a new arc that goes to the found state (if it does not already exist) and the node is omitted. Otherwise, the new state is added to the coverability graph together with the corresponding arc. The coverability graph contains only one node for each equivalence class of the coverability relation.

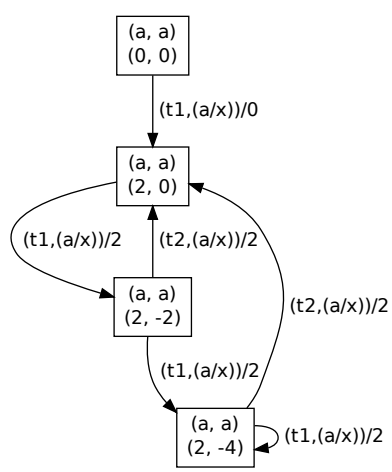



Figure 3: Coverability graph for model from Fig. 2.

The coverability graph for the RTCP-net from Fig. 2 is shown in Fig. 3. Let us consider the graph and the firing sequence from equation (7). After calculating the state $((a,a),(2,-6))$ we affirm that there already exists the state $((a,a),(2,-4))$ that covers it. Therefore, we add only an arc that goes back to the state $((a,a),(2,-4))$.

## 5. RTCP-net Compiler

The scheme of the modeling and verification process with RTCP-nets is shown in Fig. 4. It starts from designing a model using CPN Tools environment as presented in Section 3. The designed model is stored using XML file format (a *.cpn* file). *RTCP-net Compiler* is a tool that takes an RTCP-net XML file and generates a Java application, which can be used to simulate and generate the coverability graph for the consider model. The Java JDK environment (version 1.6 or higher) is necessary to use the compiler.

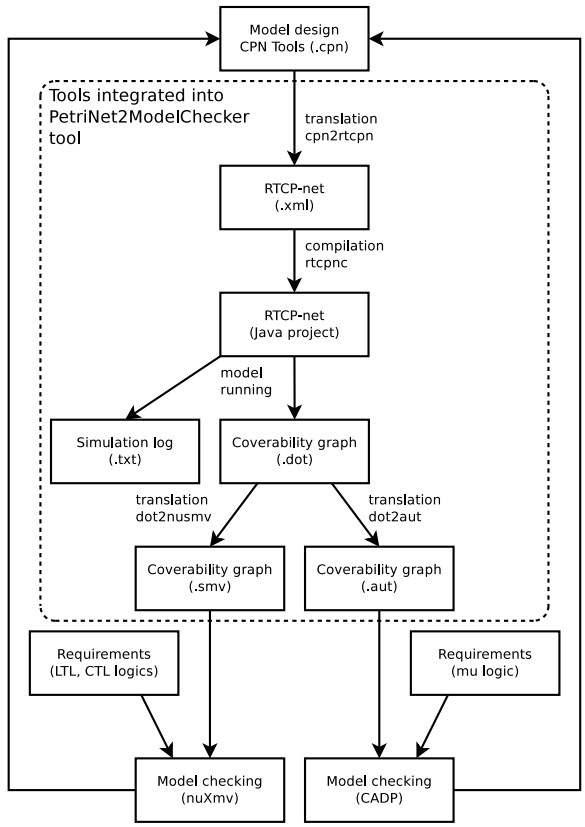



Figure 4: Modeling and verification process for RTCP-nets

The XML file format used by the RTCP-net Compiler differs from the XML file format used by the CPN Tools [15]. The *cpn2rtcpn* translator is used to convert *cpn* files into XML format accepted by the compiler. The translator has been implemented in the Java language and is used as follows:

```
java –jar cpn2rtcpn model.cpn –o model.xml
```

Then, the received XML file is transformed into a Java application:

```
java –jar rtcpnc model.xml model
```

where *model* is the name of the generated application. It is placed inside *model* subdirectory, which contains *model.jar* file. All steps performed by the compiler are logged. By default the log is written to standard output, however it may be redirected to a file.

The first stage of the compiler work is reading an RTCP-net from a file. The log contains information about pages, places, transitions and arcs read by the compiler. Then the RTCP-net is converted into equivalent non-hierarchical one. The compiler removes substitution transitions replacing them with corresponding subpages and merges fusion places. If necessary, some names may be adjusted to guarantee unique names for the whole model. In the next stage, Java code is generated for the model. This stage is based on predefined Java templates. Finally, the compiler generates files necessary for project compilation, compiles the project and prepares the *jar* file. After successful compilation the model can executed:

```
java –jar model.jar time
```

where *time* denotes the simulation period. The result of a model simulation is text log as shown in Fig. 5. If more than one transition is enabled in a given state, simulator takes one of them at random.

To generate reachability or coverability graph instead of running the simulation, -rg (reachability graph) and -cg (coverability graph) options can be used:

```
java –jar model.jar time –rg
java –jar model.jar time –cg
```

Generated graph is stored in a dot format file and located in the working directory of the simulator process. DOT is a graph description language that is both easily processed by computer programs and understandable by humans. It is a part of an open source graph visualization software Graphviz [10]. Graph saved in dot file can be easily rendered into a PDF using one of Graphviz tools:

```
dot –Tpdf coverability_graph.dot > cg.pdf
```

In the case of our example RTCP-net, the rendered PDF file of its coverability graph is presented in the Fig. 3. Generated graphs can be also previewed using various external interactive dot format viewing tools like xdot or ZGRViewer.

PetriNet2ModelChecker [2] [3] is a tool that deals with the problem of translation of RTCP-net reachability or coverability graphs into the nuXmv language and Aldebaran format. Thus, it provides the possibility of formal verification using model checking

```
================================================================
Simulation time: 0
================================================================
Simulation time: 1
Transition 't1' fired with binding: ()
  -->> token 'a' removed from place 'p1' by arc expression: a@0
  <<-- token 'a' added to place 'p2' by arc expression: a@2
  -->> token 'r' removed from place 'p7' by arc expression: r@1
State of 'p1' place:
State of 'p2' place: (a)@2
State of 'p7' place: 2(r)
Transition 't3' fired with binding: ()
  -->> token 'b' removed from place 'p3' by arc expression: b@0
  <<-- token 'b' added to place 'p4' by arc expression: b@1
  -->> token 'r' removed from place 'p6' by arc expression: r@0
  <<-- token 'r' added to place 'p6' by arc expression: r@1
State of 'p3' place: (b)
State of 'p4' place: (b)@1
State of 'p6' place: 2(r)@1
================================================================
Simulation time: 2
...
```

Figure 5: Simulation result for model from Fig. 1.

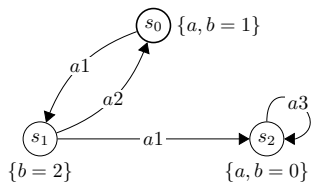
techniques and mainstream model checkers – nuXmv for LTL and CTL temporal logics (see section 6), CADP Evaluator for regular alternation-free $\mu$-calculus (see section 7). The tool integrates *cpn2rtcpn* and RTCP-net Compiler software and therefore allows to load RTCP nets modeled in CPN Tools and generate their coverability graphs or simulators in one application. PetriNet2ModelChecker also provides for automatic coverability graph rendering to a pdf file. These features significantly simplify the process of RTCP-nets modeling and verification. The tool was previously known as PetriNet2NuSMV [23] and enables automatic translation for other classes of Petri nets, such as place-transition and colored Petri nets.

## 6. Verification with nuXmv

The nuXmv tool [4] (the previous version known as NuSMV) is one of the most popular model checkers for temporal logic. Given a finite state model and a temporal logic formula, nuXmv can be used to check automatically whether or not the model satisfies the formula. Formulas can be treated as a specification of requirements for a given model and can be expressed using LTL [7] or CTL [7], [8] temporal logics. In the nuXmv approach, the verified system is modeled as a *finite state transition system* [5] usually called *Kripke structure* [16].

**Definition 8** *A* finite state transition system *is a tuple* $TS = (S, I, \rightarrow, L)$*, where:*

- $S$ *is a finite set of* states*,*
- $I \subseteq S$ *is the set of* initial states*,*
- $\rightarrow \subseteq S \times S$ *is the* transition relation*, specifying the possible transitions from state to state,*
- $L$ *is the* labeling function *that labels states with* atomic propositions *that hold for the given state.*




```
MODULE main
IVAR
  action: {a1,a2,a3};                    } set of actions
VAR
  s : {s0, s1, s2};                      } set of states
  a : boolean;                           } atomic propositions
  b : 0 .. 2;                            } given implicitly
ASSIGN
  init(s) := {s0, s2};                   } initial states

  next(s) := case
    s = s0 & action = a1: s1;
    s = s1 & action = a2: s0;
    s = s1 & action = a1: s2;            } transition relation
    s = s2 & action = a3: s2;
    TRUE: s;
  esac;

  a := case
    s = s0 : TRUE;
    s = s2 : TRUE;
    TRUE   : FALSE;
  esac;

  b := case                              } labelling function
    s = s0 : 1;
    s = s1 : 2;
    TRUE   : 0;
  esac;
TRANS s = s0 -> (action = a1)
TRANS s = s1 -> (action = a1
                 | action = a2)          } transitions' labels
TRANS s = s2 -> (action = a3)
```

Figure 6: Finite state transition system written with SMV language

The nuXmv tool equipped with a dedicated modeling language (the SMV language), which is used to define finite state transition systems [5]. An example of such a model is given in Fig. 6. A nuXmv model consists of three sections: Variables definitions (`VAR` and `IVAR`), `ASSIGN` and specification of transitions' labels (`TRANS`). The `IVAR` section contains definition of an input variable `action`, containing transitions' labels. In the `VAR` section standard variables are defined, including set of states and atomic propositions variables. The `ASSIGN` section is composed of three main parts. The first one is the initialization of the state variable. The second part is responsible for defining

transitions between the states. The last part assigns values to the atomic propositions for specific states. The set of atomic propositions is given implicitly using variables and their domains. For example, the following expressions can be considered as atomic propositions: $a$ (i.e. $a = TRUE$), $!a$, $b = 0$, $b > 1$ etc. The `TRANS` lines specify which transitions (actions) are available in specific states. For instance, line `TRANS s = s1 -> (action = a1) | (action = a2)` determines that when the system is in state `s1`, the only available actions are `a1` and `a2`.

A translation algorithm of an RTCP-net coverability graph into the nuXmv code is presented in Fig. 7. It extends the translation algorithm presented in [2] through adding transition's names as edges' labels in the nuXmv transition system. This enhancement enables using transition names in verification process (in LTL forumalae) and giving substantial information during the model simulation.

The flowchart is divided into six main parts corresponding to the abstract model of a finite state transition system in nuXmv. In the adopted notation, $\triangleright$ indicates generated nuXmv code, # represents string concatenation and $m_X(y)$ is the number of $y$ elements in the $X$ multiset.

The first block of the flowchart is responsible for generation of a declaration of an input variable `action`. This variable contains the name of the fired transition. First two lines generate module name line and the beginning of the IVAR section. The *action* set is initialized with a single element *NOP*, representing empty action. Then, every transition label is added to the set. The next step defines the domain of the state variable $s$. The variable value represents the current node of the RTCP-net coverability graph. For every state that is reachable from the initial $(M_0, S_0)$ state, the state name is included into the set of nuXmv states.

The third section of the algorithm contains steps required to define variables which represent markings and clocks' values of places. At the begining, $L_M$ and $L_S$ sets are initialised. They contain labels of markings and clocks, respectively. After initialization of the two sets of labels there are two consecutive loops, containing three blocks each. For every pair (*place name*, *color*) from the reachable markings, the proper variable label is generated by concatenating these two values. The variable identified by this label (`l_mark`) is a bounded integer that contains the information about the number of tokens of the specified color in the corresponding place. Similarly, bounded integer clock variable (`l_stamp`) is generated for every place of the net. Its name is the concatenation of the place's name and the `time` keyword.

The fourth step starts with adding of the beginning of the ASSIGN section and initialization of the $s$ variable with the name of the initial state. Then the transition relation switch statement is opened. $V_{ik}$ is the set of successors of $si$ state, reachable by firing of the transition $tk$. In the nested loops that follow, successors lists for every reachable state are generated.

The next sections of the algorithm contains two similar loops. Each generates labeling functions – the first one for marking variables and the second one for clock variables defined in the VAR section. Labeling functions are basically switch statements in which the proper value is assigned to the variable depending on the current state of the system.

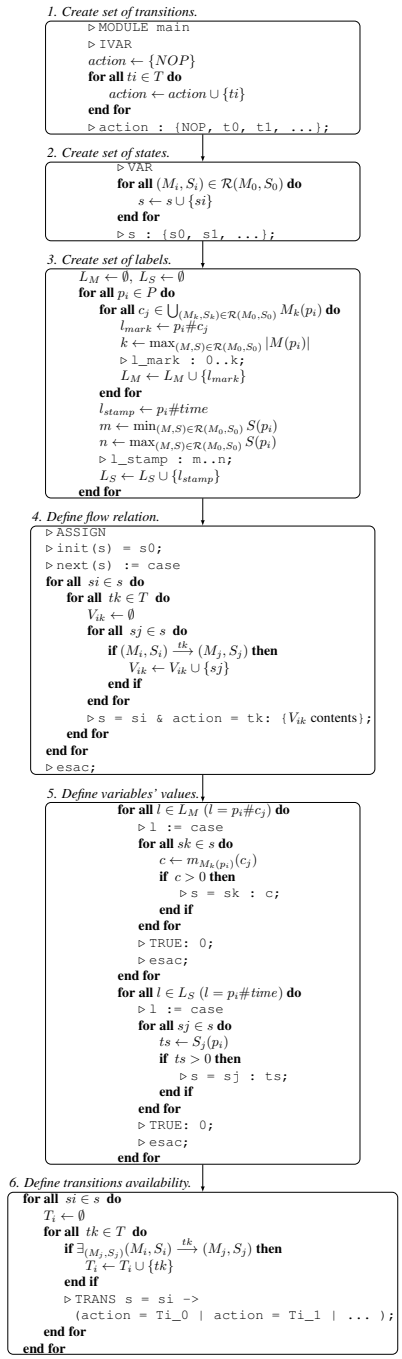



Figure 7: Coverability graph into nuXmv translation algorithm.

The last block defines availability of the transitions. It is determined by the value of the *si* variable. For every reachable state, TRANS line is generated containing a list of available transitions. One can notice that *NOP* action is never listed there. This is because it is only available when no successor exists and indicates a terminal state of the system.

The algorithm was implemented as one of the features of the PetriNet2ModelChecker tool and thus allows conversion of a coverability graph of an RTCP-net stored in .dot file into nuXmv code automatically. The generated nuXmv model is stored as a text file. To include temporal logic formulas into the file the `LTLSPEC` and `CTLSPEC` statements are used. In case of the LTL logic, the temporal operators G (globally), F (finally), X (next), U (until) can be used. Moreover, the propositional logic operators are represented by: `!` (not), `&` (and), `|` (or), `xor` (exclusive or), `->` (implies) and `<->` (equivalence). In case of CTL following temporal logic operators can be used: EG (exists globally), EX (exists next state), EF (exists finally), AG (forall globally), AX (forall next state), AF (forall finally), E[U] (exists until), A[U] (forall until). Satisfaction of each specified formula is automatically verified with the nuXmv tool. If a modeled system does not satisfy a given formula, a proper counterexample is presented. It is finally worth mentioning that nuXmv can verify systems of high complexity, i.e. containing more than $10^{20}$ states. These features make nuXmv useful and convenient tool for finite automata verification.

The usage of the presented approach is illustrated on an example of a fire alarm control panel in Section 8.

## 7. Verification with CADP

The verification process described in the previous section is oriented towards states i.e. the temporal logic formulas describe some properties of reachable states. On the other hand, coverability graphs can be verified using the CADP toolbox [11] and the actions oriented approach. One of CADP tools called *evaluator* provides on-the-fly model checking of regular alternation-free $\mu$-calculus formulas [9], [17]. In such approach, a specification of requirements is given as a set of $\mu$-calculus formulas and the tool is used to check whether the LTS graph satisfies them. The $\mu$-calculus formulas concern actions labels and describes sequences of performed actions.

We use *dot2aut* translator (included in the PetriNet2ModelChecker tool) to convert a coverability graph into Aldebaran format. An example of a finite state transition system coded using the Aldebaran format is given in Fig. 8. Then, such a graph can be converted into BCG (Binary Coded Graphs) format that is one of acceptable input formats for CADP Toolbox. The conversion method (from Aldebaran to BCG) is provided by one of CADP tools. The BCG format is independent from any particular model-based verification technique; it can be used either by tools performing graph comparison and reduction modulo equivalence relations, or by tools checking properties expressed in temporal logics.

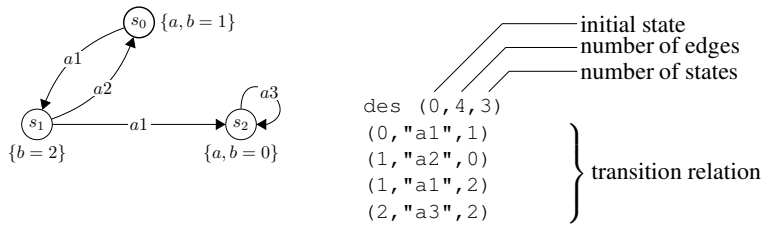



Figure 8: Finite state transition system written using Aldebaran format

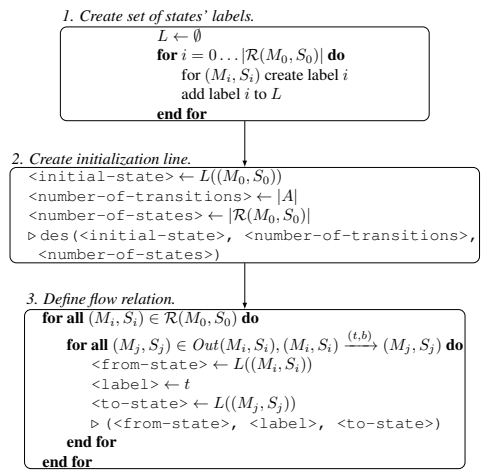



Figure 9: RTCP coverability graph into Aldebaran format translation algorithm

A translation algorithm of an RTCP-net coverability graph into Aldebaran format is presented in the Fig. 9. The first step of the algorithm is an initialization the set of defined labels for reachable states. This set is denoted with letter $L$. An appropriate label is created for every reachable state $(M_i, S_i)$. The label of a state is its order number. The next block appends a template content to the output file. Keywords of the template are replaced with proper values, i.e. the label of the initial state, number of arcs and number of states. In the last section there are two nested loops. They are responsible for processing each edge of the coverability graph and appending template lines containing information about the transition which firing is denoted by the given edge. $(M_i, S_i)$ corresponds to the input node and $(M_j, S_j)$ corresponds to the output node of the edge. $Out(M_i, S_i)$ stands for the set of successor nodes of $(M_i, S_i)$ in a reachability or coverability graph. As previously, template line's keywords are replaced with appropriate values, i.e. the label of the input node, the label of the transition and the label of the output node. The detailed version of the algorithm can be found in [3]. It has also been implemented in the PetriNet2ModelChecker tool.

The input language of the CADP *evaluator* tool is an extension of the alternation-free $\mu$-calculus. The logic is built from three types of formulas: *action, regular* and *state formulas*. An action formula is built from action names (text put into quotation marks), regular expressions substituted for action names (text put into apostrophes), Boolean constants *true* and *false* and the propositional logic operators: *not, or, and, implies* and *equ* (equivalence). Regular formulas represent regular expressions over action sequences. The $\mu$ logic uses *nil* to denote the empty word and the following regular expression operators: . (dot) – concatenation operator, | – choice operator, $*$ – the transitive and reflexive closure operator, and $+$ – the transitive closure operator. Finally, a state formula is built of: propositional variables, Boolean constants *true* and *false*, the propositional logic operators, the *possibility* ($\langle\,\rangle$) and *necessity* ([ ]) *modal operators* and *minimal* ($\mu$) and *maximal* ($\nu$) *fixed point operators*. For more details on formulas syntax see the CADP *evaluator* site `http://cadp.inria.fr/man/evaluator.html`.

The *evaluator* tool takes a coverability graph encoded in the BGC format and a file with a $\mu$-calculus formula and checks whether the formula holds for the graph. The tool is equipped with diagnostic generation algorithms, which construct both examples and counterexamples for a given formula.

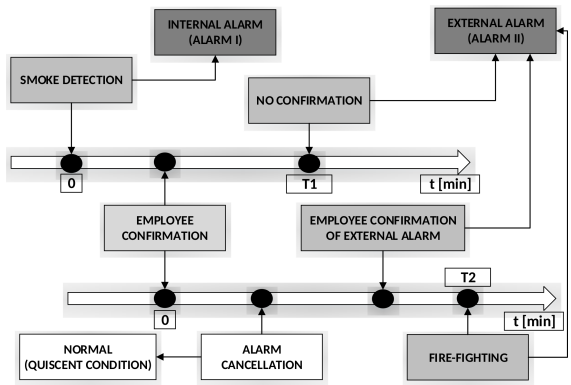



Figure 10: Fire alarm control panel scheme.

## 8. Case study

A model of a fire alarm control panel is used to illustrate the process of RTCP-nets modeling and verification. It is the model of an actual solution designed by the SITP association [6]. According to the SITP, alarm variants usage is a common practise in construction of fire alarm control panels. This method aims at the reduction of false fire alarms. Its most popular variant is two-stage alarming which scheme is presented in Fig. 10. The designed RTCP-net is shown in Fig. 11.

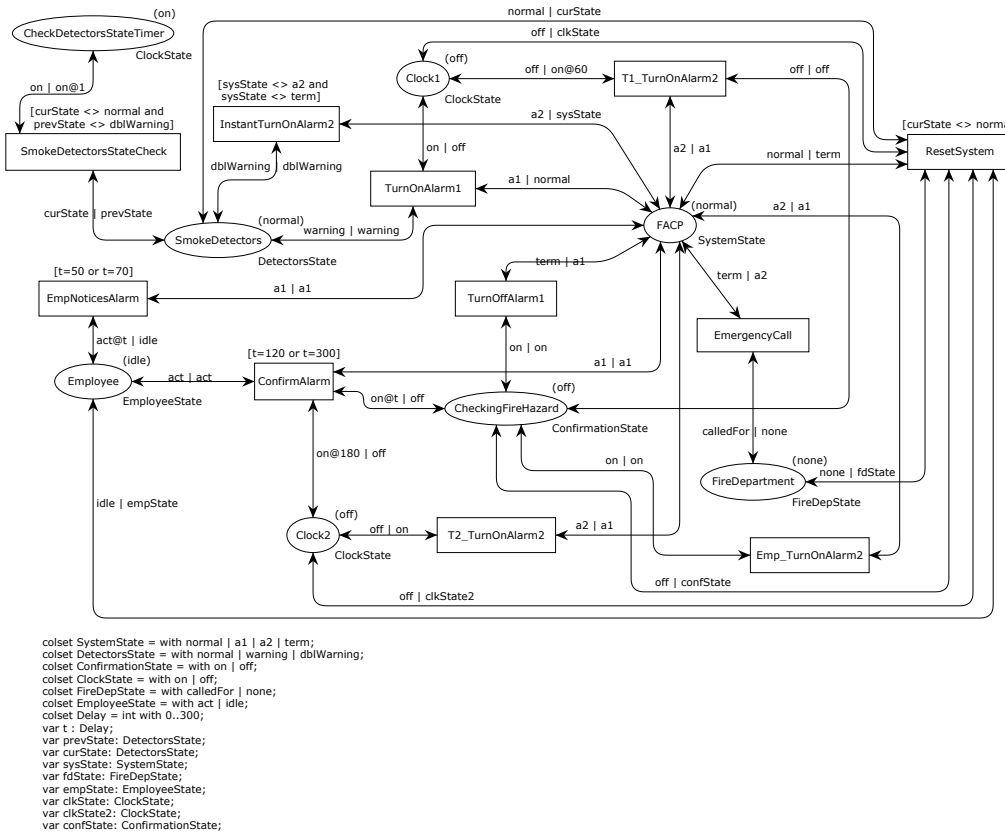



Figure 11: RTCP-net model of fire alarm control panel.

There are four possible states of the system: normal, internal alarm, external alarm and terminal. The current state of the panel is defined by the color of the token in place *FACP*. Firing of *SmokeDetectorsStateCheck* transition with *defState* variable set to *warning* corresponds to smoke detection by only one of smoke detectors. It results in raising of the internal alarm (*TurnOnAlarm1*). Personnel participation is required in this approach. It has strictly defined role of the operator in alarm verification process. Internal alarm calls in the operator to identify the fire hazard. *T1* determines the time for operator to confirm reception of the notification. In the presented model *T1* is set to 60 time units. *Clock1* place performs a role of the timer which activates external alarm at the expiry of the deadline (*T1_TurnOnAlarm2*). External alarm indicates that fire department assistance is required and launches fire emergency procedures. Acknowledgement of the internal alarm (*ConfirmAlarm*) results in activation of a second timer (*Clock2*). Operator has *T2* time units to assess the threat and verify the alarm. If the internal fire alarm proves to be false, the operator can turn it off and reset the panel to the normal state. If the threat is real, personnel can push manual call point button (*EmpTurnOnAlarm2*) which auto-

matically turns on the external alarm. If the fire is manageable, the operator can attempt to extinguish it using available fire fighting equipment. *T2* is the time limit defined for personnel to get the situation under control and to turn off the internal alarm (*TurnOffAlarm1*). Otherwise, the external alarm is raised (*T2_TurnOnAlarm2*). This limit is set to 180 time units. Coincidence detection is one of the most effective ways of elimination false fire alarms. In this case external alarm is instantly raised upon fire detection by at least two smoke detectors (*InstantTurnOnAlarm2*). Fire detection by one smoke detector raises the internal alarm (*TurnOnAlarm1*) which can be verified and handled by the personnel.

Fire alarm system is an excellent example of a safety critical system. Its failures always cause major losses. If it raises the alarm too late, many people can perish or become seriously injured. Yet false alarms result in high costs due to, inter alia, the stoppage of technological processes or activation of automatic extinguishing system. Hence, comprehensive formal verification of such systems is crucial. Coverability graph of the designed RTCP-net has 3077 states and 3986 edges. A small fragment of it, containing the shortest cycle in the considered system, is presented in Fig. 12.

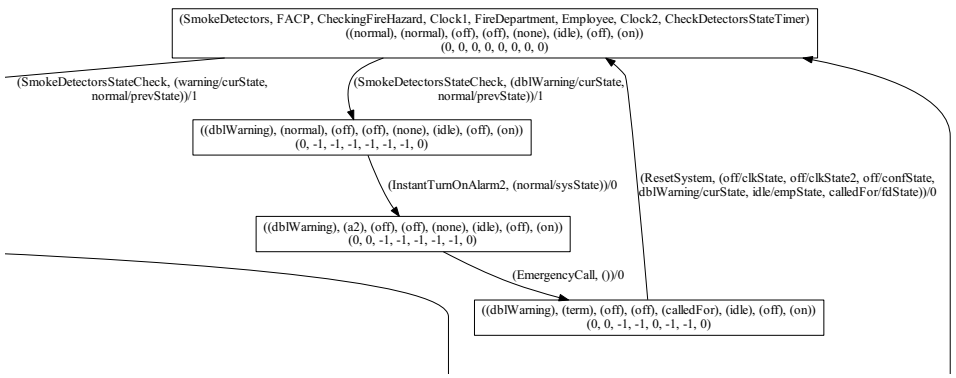



Figure 12: Fragment of a coverability graph for model of fire alarm control panel.

Analysis of a coverability graph of any Petri net modeling a real system is not trivial. Complexity of this task rapidly increases with the complexity of the modeled net. In this case manual verification is practically impossible. In the presented approach, the coverability graph of modeled system is automatically translated into nuXmv and Aldebaran formats. In the first case, satisfiability of formulae specified in LTL, CTL and RTCTL temporal logics can be automatically verified with nuXmv tool. In the second case, $\mu$-calculus formulae are verified with CADP Evaluator. Selected fragments of nuXmv code are shown in Listing 1.

Listing 1: Fragments of nuXmv code generated from the coverability graph of the model in Fig. 11.

```
MODULE main
IVAR
    action: {NOP, ConfirmAlarm, ..., TurnOnAlarm1};
VAR
    s: {s0, s1, s2, ..., s3075, s3076};
    ...
    FACP_normal : 0..300;
    FACP_a1 : 0..300;
    FACP_a2 : 0..300;
    FACP_term : 0..300;
    ...
    FACP_time : -350..300;
    ...
ASSIGN
    init(s) := s0;
    next(s) := case
        s = s0 & action = SmokeDetectorsStateCheck: {s1, s2};
        s = s1 & action = TurnOnAlarm1: s3;
        ...
        s = s3076 & action = ResetSystem: s0;
        TRUE: s;
    esac;
    ...
    FACP_normal := case
        s = s0 : 1;
        s = s1 : 1;
        s = s2 : 1;
        s = s30 : 1;
        s = s39 : 1;
        s = s40 : 1;
        TRUE : 0;
    esac;
    ...
    FACP_time := case
        s = s1 : -1;
        ...
        s = s3076 : -120;
        TRUE : 0;
    esac;
    ...
    TRANS s = s0 -> (action = SmokeDetectorsStateCheck)
    TRANS s = s1 -> (action = TurnOnAlarm1)
    TRANS s = s2 -> (action = InstantTurnOnAlarm2)
    ...
    TRANS s = s391 -> (action = ConfirmAlarm | action =
      ↪ SmokeDetectorsStateCheck)
```

```
...
TRANS s = s3076 -> (action = ResetSystem)
```

Three examples of LTL formulae representing properties of the modeled system are presented below.

Listing 2: Examples of LTL formulae for the model in Fig. 11.

```
LTLSPEC G FACP_normal > 0 -> FACP_a1 = 0 & FACP_a2 = 0 &
  ↪ FACP_term = 0
LTLSPEC G F FACP_normal > 0
LTLSPEC G((FACP_a1 > 0 &  X(action = TurnOffAlarm1)) -> FACP_a2
  ↪ = 0 U FACP_normal > 0)
```

The first formula is an example of an invariant property. It denotes that there can be no states in which system is both in normal and any other *SystemState*. The second formula is a liveness property that verifies whether system can always go back to the initial *SystemState*. Both of these formulae are proven true. The last formula is the most interesting one. It employs not only states of the coverability graph but also edges' labels. This approach is only possible thanks to the latest improvements in the translation algorithm (Fig. 7). This formula checks whether operator's action of internal alarm disarming is definitely preventing false external alarm from being raised. The modeled system satisfies this formula. However, if it wasn't the case, nuXmv would provide an counterexample that, compared to the previous version of translation algorithm, is much easier to understand and track. This is due to the fact that beyond just showing current values of system variables, nuXmv would show the fired transitions too. Furthermore, this huge improvement enhances the nuXmv simulation mode in the same way.

The other approach, strictly action-based, can also be used in the case of the presented system. It utilises popular CADP Evaluator tool. Selected fragments of the considered coverability graph translated into Aldebaran format accepted by the tool are presented below.

Listing 3: Fragments of aut file generated from the coverability graph of the model in Fig. 11.

```
des (0, 3986, 3077)
(0, "SmokeDetectorsStateCheck", 1)
(0, "SmokeDetectorsStateCheck", 2)
(1, "TurnOnAlarm1", 3)
(2, "InstantTurnOnAlarm2", 4)
(3, "EmpNoticesAlarm", 6)
(3, "EmpNoticesAlarm", 5)
(4, "EmergencyCall", 7)
...
(2107, "SmokeDetectorsStateCheck", 2120)
(2107, "Emp_TurnOnAlarm2", 2122)
(2107, "TurnOffAlarm1", 2121)
...
```

```
(3073, "SmokeDetectorsStateCheck", 3076)
(3074, "ResetSystem", 30)
(3075, "ResetSystem", 0)
(3076, "ResetSystem", 0)
```

Three examples of $\mu$-calculus formulae checking properties of the modeled system are presented below.

Listing 4: Examples of $\mu$-calculus formulae for the model in Fig. 11.

```
[true*."ConfirmAlarm".(not "ResetSystem")*."T1_TurnOnAlarm2".
  ↪ true*."ResetSystem"] false
[true*."InstantTurnOnAlarm2".(not "EmergencyCall")*."ResetSystem
  ↪ "] false
[true*."ConfirmAlarm".(not "ResetSystem" and not "
  ↪ T2_TurnOnAlarm2" and not "Emp_TurnOnAlarm2")*."
  ↪ EmergencyCall"] false
```

The first formula verifies whether acknowledgement of the internal alarm turns off the first timer, that is `Clock1`. Verification in CADP Evaluator proves, that even after expiration of `T1` time limit, the premature external alarm is not raised. The second formula checks whether fire detection by at least two smoke detectors always turns on the external alarm. It is also proven true. The last formula denotes that after employee confirmation of the internal alarm, the external alarm cannot be raised unless the operator turns it on manually or timer `T2` expires. This formula, however, is not satisfied. Second smoke detector can possibly activate at any given moment, automatically turning on the external alarm, what was verified by the second formula. It is also worth mentioning that a graph explaining the truth value of given formula can be generated using `-diag` option of the Evaluator tool.

Table 9: Examples of RTCP-net models.

| ID | Model | States | Edges |
|---|---|---|---|
| 1. | Fire alarm control panel | 3077 | 3986 |
| 2. | Railway switch | 3888 | 5817 |
| 3. | Producer-consumer | 3696 | 13330 |
| 4. | Communication protocol | 11001 | 11000 |
| 5. | Communication protocol 2 | 7390 | 22390 |

Translation and verification times were measured for the presented fire alarm control panel and a few other RTCP-net models (Table 9). Tests were performed on a PC with Intel Core i7 930 processor and 16 GB of RAM. Verification results are divided into two tables. The first one (Table 10) covers verification in nuXmv and the other (Table 11) verification in CADP Evaluator tool. Each table contains times (measured in seconds) of

Table 10: Measured times [s] of LTL and CTL formulae verification in nuXmv for the models from Table 9.

| ID | Trans. | Init. | I | II | III | Avg. |
|---|---|---|---|---|---|---|
| 1. | 0,7 | 29,86 | 14,72 | 85,89 | 53,91 | 51,51 |
| 2. | 1,0 | 25,21 | 14,2 | 13,15 | 16,76 | 14,7 |
| 3. | 1,4 | 19,95 | 18,92 | 29,63 | 14,66 | 21,07 |
| 4. | 5,3 | 214,56 | 3,6 | 20,98 | 56,62 | 27,07 |
| 5. | 6,0 | 130,9 | 10,56 | 0,81 | 67,46 | 26,28 |

Table 11: Measured times[s] of $\mu$-calculus formulae verification in CADP Evaluator for the models from Table 9.

| ID | Trans. | Init. | I | II | III | Avg. |
|---|---|---|---|---|---|---|
| 1. | 0,6 | 0,026 | 0,381 | 0,235 | 0,23 | 0,28 |
| 2. | 0,9 | 0,031 | 0,375 | 0,242 | 0,234 | 0,28 |
| 3. | 1,3 | 0,042 | 0,38 | 0,229 | 0,156 | 0,26 |
| 4. | 4,9 | 0,038 | 0,242 | 0,243 | 0,387 | 0,29 |
| 5. | 5,9 | 0,057 | 0,4 | 0,236 | 0,234 | 0,29 |

translation, initialization, verification of three formulae specific for the models and their average.

The measured times proved to be entirely satisfactory. Verification in nuXmv is visibly slower then in CADP Evaluator. But on the other hand nuXmv allows to use variables' values in the specification while in CADP one can only use edges' labels. These two approaches complete each other allowing formal verification of complex models.

## 9. Summary

The popularity of Petri nets results from the simplicity of the modeling language and the ability of adjusting them to users' needs. The diversity of Petri nets classes allows users either to find the class that is the most suitable for the given problem or to refine one. In the second case, it is necessary to provide suitable software supporting practical usage of the new-defined Petri net class.

RTCP-nets are the result of the adaptation of colored Petri nets to modeling and verification of real-time systems. The main difference between CP-nets and RTCP-nets is the new time model with local clocks attached to places in case of the latter class.

The main advantage of the time model is the possibility of representation of the set of reachable states with a finite coverability graph that stores all information necessary for the verification of time properties.

The paper presents a survey of tools and verification methods (algorithms) developed for RTCP-nets recently. We use CPN Tools, a mainstream environment for CP-nets modeling, for constructing RTCP-nets. To simulate behaviour of an RTCP-net and to compute its reachability and coverability graphs the rtcpnc compiler has been developed. For the formal verification of a model properties the model checking techniques are used. Two established model checkers are used for that purpose, nuXmv and CADP. The use of these tools requires the transformation of a reachability/coverability graph into the NuSMV or Aldebaran format respectively. Improved versions of translation algorithms have been presented in the paper. Moreover, all described tools used for RTCP-nets verification have been implemented and integrated into the PetriNet2ModelChecker tool. To summarize, the paper contains description of the key RTCP-nets' concepts and the state-of-the-art of the tools and verification methods developed for the nets recently. All these information are sufficient to use RTCP-nets in practise.

## References

[1] W.M.P. VAN DER AALST: Interval timed coloured Petri nets and their analysis. In *Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets*, **691** London, UK, (1993), 453-472.

[2] A. BIERNACKA, J. BIERNACKI and M. SZPYRKA: State-based verification of RTCP-nets with nuXmv. In *Int. Conf. of Computational Methods in Sciences and Engineering (ICCMSE 2015)*, **1702**, Athens, Greece, (2015), 100010–1–100010–4.

[3] J. BIERNACKI, A. BIERNACKA and M. SZPYRKA: Action-based verification of RTCP-nets with CADP. In *Int. Conf. of Computational Methods in Sciences and Engineering (ICCMSE 2015)*, **1702**, Athens, Greece, (2015), 100011–1–100011–4.

[4] R. CAVADA, A. CIMATTI, M. DORIGATTI, A. GRIGGIO, A. MARIOTTI, A. MICHELI, S. MOVER, M. ROVERI and S. TONETTA: The nuXmv symbolic model checker. In *Computer Aided Verification*, **8559** of *Lecture Notes in Computer Science*, Springer, 2014, 334–342.

[5] A. CIMATTI, E. CLARKE, F. GIUNCHIGLIA and M. ROVERI: NUSMV: a new symbolic model checker. *Int. J. on Software Tools for Technology Transfer*, **2**(4), (2000), 410-425.

[6] J. CISZEWSKI, K. KUNECKI, W. MARKOWSKI, J. SAWICKI and M. SOBECKI: *SITP Guideline WP-02:2010. Fire alarm systems. The design*, 2010.

[7] E.M. CLARKE, O. GRUMBERG and D.A. PELED: *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[8] E.A. EMERSON: Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, **B** Elsevier Science, 1990, 995-1072.

[9] E.A. EMERSON: Model checking and the mu-calculus. In *DIMACS Series in Discrete Mathematics*, *American Mathematical Society*, (1997), 185-214.

[10] EMDEN R. GANSNER and STEPHEN C. NORTH: An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, **30**(11), (2000), 1203–1233.

[11] H. GARAVEL, F. LANG, R. MATEESCU and W. SERWE: CADP 2006: A toolbox for the construction and analysis of distributed processes. In *Computer Aided Verification*, **4590** Springer-Verlag, 2007, 158-163.

[12] B. JASIUL, M. SZPYRKA AND J. ŚLIWA: Malware behavior modelling with colored Petri nets. In *Computer Information Systems and Industrial Management Proceedings of the 13th IFIP TC8 Int. Conf. CISIM 2014*, **8838** Springer-Verlag, 2014, 667-679.

[13] K. JENSEN: *Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1–3. Springer-Verlag, Berlin, Germany, 1992-1997.

[14] K. JENSEN and L.M. KRISTENSEN: Colored Petri nets. Modelling and Validation of Concurrent Systems. Springer, Heidelberg, 2009.

[15] K. JENSEN, L.M. KRISTENSEN and L. WELLS: Colored Petri nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. on Software Tools for Technology Transfer*, **9**(3-4), (2007), 213-254.

[16] S. KRIPKE: A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, **9** (1963), 67-96. Announced in *J. of Symbolic Logic*, **24** (1959), 323.

[17] R. MATEESCU and M. SIGHIREANU: Efficient on-the-fly model-checking for regular alternation-free $\mu$-calculus. Technical Report 3899, INRIA, 2000.

[18] T. MURATA: Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, **77**(4), (1989), 541-580.

[19] C. A. PETRI: Communication with automata. Technical report, New York, 1965. English translation of *Kommunikation mit Automaten*, PhD Dissertation, University of Bonn, 1962.

[20] S. SAMOLEJ and T. SZMUC: Time extensions of Petri nets for modelling and verification of hard real-time systems. *Computer Science*, **4** (2002), 55-76.

[21] M. SZPYRKA: Analysis of RTCP-nets with reachability graphs. *Fundamenta Informaticae*, **74**(2–3), (2006), 375-390.

[22] M. SZPYRKA: Analysis of VME-Bus communication protocol – RTCP-net approach. *Real-Time Systems*, **35**(1), (2007), 91-108.

[23] M. SZPYRKA, A. BIERNACKA and J. BIERNACKI: Methods of translation of Petri nets to NuSMV language. In *Proc. of the Concurrency Specification and Programming Workshop (CSP 2014)*, **1269** Chemnitz, Germany, (2014), 245-256.