A High Level Net for Modeling and Analysis Reconfigurable Discrete Event Control Systems

Ahmed Kheldoun, Kamel Barkaoui, Jiafeng Zhang, Malika Ioualalen

A high level net for modeling and analysis reconfigurable discrete event control systems

Ahmed Kheldoun[14], Kamel Barkaoui[2], JiaFeng Zhang[3], Malika Ioualalen[1]

[1] MOVEP, Computer Science Department, USTHB, Algiers, Algeria
ahmedkheldoun@yahoo.fr, mioualalen@usthb.dz
[2] CEDRIC-CNAM, 292 Rue Saint-Martin 75141, Cedex 03 Paris, France
kamel.barkaoui@cnam.fr
[3] School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China
zhangjiafeng628@gmail.com
[4] Sciences and Technology Faculty, Yahia Fares University, Medea, Algeria

Abstract. This paper deals with automatic reconfiguration of discrete event control systems. We propose to enrich the formalism of recursive Petri nets by the concept of $feature$ from which runtime reconfigurations are facilitated. This new formalism is applied in the context of automated production system. Furthermore, the enhanced recursive Petri net is translated into rewriting logic, and by using Maude LTL model-checker one can verify several behavioural properties related to reconfiguration.

Keywords: Reconfigurable control systems, Feature, Recursive Petri nets, Rewriting logic, Maude.

1 Introduction

The new generation of discrete event control models is adressing new creteria as flexibility and agility. The need of flexibility and adaptability leads to integrate reconfigurability features in these models, but it makes the system more complex and its development a hard task. Therefore, an approach for the design safe and reconfigurable systems is a crucial need. The Petri net formalism is one of the most used tools to model and analyse discrete event systems [2].

Recently, recursive Petri nets (RPNs) [3] are proposed to specify flexible concurrent systems where functionalities of discrete event systems such as abstraction, dynamicity, preemption, recursion are preponderant. In fact, RPNs have ability to model dynamic creation of threads which behave concurrently.

In this paper, we introduce the concept of $feature$ proposed in [13] to deal with reconfiguration at runtime. More precisely, the reconfiguration is modelled by combining the interruption and the activation/deactivation of transitions which is ensured by : $application\ condition$ and $update\ expression$.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related work. Section 3 recalls the syntax and semantic of the formalism RPNs. The formalism which enrich RPN by the concept of $feature$, named reconfigurable RPN

and denoted by R^2PN, is presented in Section 4. Section 5 presents a case study of a reconfigurable automated production system, and we present in Section 6 its modelling in terms of R^2PN. The verification of the obtained model is done by using the LTL model-checker of Maude [6] [11] and is described in Section 7. Section 8 concludes this paper and depicts further research work.

2 Related Work

Many researchers have tried to deal with formal modeling of control systems with potential reconfigurations. The author of [1] proposed self-modifying nets that can modify their own firing rules at runtime, however, most of the net basic properties such as reachability, boundedness and liveness become indecidable on these nets. In [4], the authors developped a Reconfigurable Petri Nets (RPN) for modeling adaptable multimedia and protocols that can self-modify during execution. They modelled the reconfiguration by introducing the concept of *modifier* places. The authors of [5] presented net rewriting systems (NRS) where a reconfiguration of the net is obtained by a rewriting rules execution. The rewriting rules are similar to production of graph grammars. However, the formalism of NRS is Turing powerful and, thus, automatic verification is no longer possible in that case. Recently, in [7], the authors proposed Reconfigurable timed net condition/event systems (R-TNCES) for modeling reconfigurable discrete event control systems. In this formalism, the system is represented by a set of control componnents and a reconfiguration is modelled by enabling/disabling some control components modules by changing condition/event signals among them.

In this paper, we present a new formalism named Reconfigurable RPN (R^2PN) enriches RPN by the concept of feature selection introduced in [13]. Indeed, in R^2PN, the reconfiguration is modelled by combining the interruption and refinement with the activation/deactivation of transitions which is ensured by : $application\ condition$ and $update\ expression$. Moreover R^2PN captures the behaviour of entire reconfigurable discrete event control system in a concise modular model, opening the way for efficient analysis and verification.

3 Recursive Petri nets

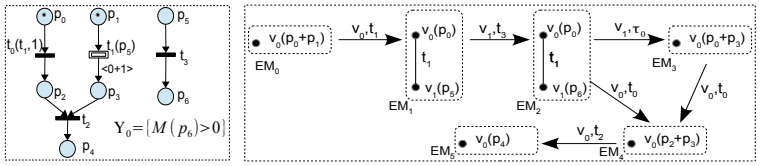
The formalism of RPN [3] consider two types of transitions : elementary and abstract. Moreover a starting marking is associated to each abstract transition and a semi-linear set of final markings is defined.

Definition 1. *(Recursive Petri Nets). A Recursive Petri Net [3] is defined by a tuple* $N = \langle P, T, Pre, Post, \Omega, I, \Upsilon, K \rangle$ *where:*

- P *is a finite set of places.*
- T *is a finite set of transitions where* $T = T_{el} \uplus T_{abs}$ *named respectively, the set of elementary and abstract transitions,*
- I *is a finite set of indices called termination indices,*
- Pre *is a mapping defined as :* $Pre : T \rightarrow P^{\oplus}$*, where* $P^{\oplus}$ *is the set of finite multi-sets over the set* P*,*

- *$Post$ is a mapping defined as : $Post : T_{el} \cup (T_{abs} \times I) \rightarrow P^{\oplus}$,*
- *Ω is a mapping $T_{abs} \rightarrow P^{\oplus}$ associating to each abstract transition an ordinary marking,*
- *Υ is a family indexed by I of termination sets, where each set represents a set of final markings (i.e. un element of $P^{\oplus}$),*
- *$K : T_{el} \rightarrow T_{abs} \times I$, maps a set of interrupted abstract trasitions, and their associated termination indexes, for every elementary transition.*

Example 1. Let's use the net presented in Fig.1(a) to highlight RPN's graphical symbols and associated notations. (i) An elementary transition is represented by a filled rectangle; its name is possibly followed by a set of terms $(t', i) \in T_{abs} \times I$. Each term specifies an abstract transition t', which is under the control of t, associated with a termination index to be used when aborting t' consequently to a firing of t. For instance, t_0 is an elementary transition where its firing preempts threads started by the firing of t_1 and the associate index is 1. (ii) An abstract transition t is represented by a double border rectangle; its name is followed by the starting marking $\Omega(t)$. For instance, t_1 is an abstract transition and $\Omega(t_1) = p_5$ means that any thread, named refinement net, created by firing of t_1 starts with one token in place p_5. (iii) Any termination set can be defined concisely based on place marking. For instance, Υ_0 specifies the final marking of threads such that the place p_6 is marked at least by one token. (iv) The set I of termination indices is deduced from the indices used to subscript the termination sets and from the indices bound to elementary transitions i.e. interruption. In this example, $I = \{0, 1\}$.



(a) A recursive Petri net N

(b) Two firing sequences of N

Fig. 1. A recursive Petri net and Two possible firing sequences

Informally, a RPN generates during its execution a dynamical tree of marked threads called an extended marking, which reflects the global state of a such net. This latter denotes the fatherhood relation between the generated threads (describing the inter-threads calls). Each of these threads has its own execution context.

Definition 2. *(Extended Marking). An extended marking [3] of a recursive Petri net is a labelled tree*
$EM = \langle V, M, E, A \rangle$ where:

- *V is the (possibly empty) finite set of nodes. When it is non empty, v_0 denotes the root of the tree,*

- *M is a mapping $V \rightarrow P^{\oplus}$ associating an ordinary marking for each node,*
- *$E \in V \times V$ is the set of edges,*
- *A is a mapping $E \rightarrow T_{abs}$ associating an abstract transition for each edge.*

Any ordinary marking can be seen as an extended marking composed by a unique node. The empty tree is denoted by $\perp$. Note contrary to ordinary nets, RPNs are often disconnected since each connected component may be activated by the firing of abstract transitions. In a RPN, we have two kinds of markings: extended markings and ordinary markings. An extended marking represents the state of the RPN. An ordinary marking represents an execution context of the thread as in Petri nets.

Definition 3. *(Enabled transition or cut step [3]).*

- *A transition is enabled in a node v of an extended marking $EM \neq \perp$ denoted by $EM \xrightarrow{v,t}$ if $\forall p \in P : M(v)(p) \geq Pre(p,t)$,*
- *A cut step τ_i is enabled in a node v if $M(v) \in \Upsilon_i$.*

The firing of an elementary transition updates the current marking using ordinary firing rule like in Petri nets. The firing of an abstract transition refines it by a new sub-net (i.e. creation of new thread, named its child) which starts its own token game, from a starting marking whose value is attached to the abstract transition. Once a final marking is reached, a cut step closes the corresponding sub net, kills its children and produces tokens, indicated by the $Post$ function, in the appropriate output places of the abstract transition. Formal definitions of firing rules are defined in [3]. Due to lack of space, we explain their principls through our illustrated example of Fig.1(a).

Example 2. Fig.1(b) highlights a firing sequences of RPN represented in Fig.1(a). The graphical representation of any extended marking EM is a tree where an arc $v_i(m_i) \xrightarrow{t_{abs}} v_j(m_j)$ means that v_j is a child of v_i created by firing the abstract transition t_{abs} and m_i (reps. m_j) is the marking of v_i (reps. v_j). Note that the initial extended marking EM_0 is reduced to a single node v_0 whose marking is $p_0 + p_1$. From the intial extended marking EM_0, the abstract transition t_1 is enabled; its firing leads to the extended marking EM_1 which contains a fresh node v_1 marked by the starting marking $\Omega(t_1)$. Then, the firing of the elementary transition t_3 from node v_1 of EM_1 leads to an extended marking EM_2, having the same structure as EM_1 but only the marking of node v_1 is changed. From node v_1 in EM_2, the cut step τ_0 is enabled; its firing leads to an extended marking EM_3 by removing the node v_1 and change the marking on its node predecessor i.e. v_0 by adding $Post(p_3, t_1, 0)$. Also, another way to remove nodes in extended marking is using the concept of preemption associated to the elementary transitions. For instance, from node v_0 in EM_2, the elementary transition t_0 with associated preemption $(t_1, 1)$ is enabled; its firing leads to an extended marking EM_4 by removing the node v_1.

4 Reconfigurable Recursive Petri nets

Reconfigurable Recursive Petri nets (R^2PNs) enriches RPN by the concept of feature selection introduced in [13]. In fact, R^2PNs extend RPN by associating transitions and

cut steps with application conditions and update expressions. An application condition is a logical formula over a set of features, describing the feature combinations to which the transition applies. It consitutes a necessary (although not sufficient) condition for the transition to fire. In fact, if the application condition is false, means that the transition is desactivated. An update expression, describes the feature selection evolves after firing a transition.

A feature is defined as a prominent or distinctive user-visible aspect, quality or characteristic of a system. A feature is defined in [13] as follows :

Definition 4. *(Feature [13]). A feature is an end-user visible characteristic of a system.*

The concept of feature has been introduced by the software design community to specify and distinguish products in product lines [9][13]. Now, let's define the set of application conditions over a set of features.

Definition 5. *(Application condition). An application condition φ [9] is a logical (boolean) contraint over a set of features F, defined by the following grammar: $\varphi ::= true \mid a \mid \varphi \wedge \varphi \mid \neg\varphi$, where $a \in F$. The remaining logical connectives can be encoded as usual. We write Φ_F to denote the set of all application conditions over F.*

Definition 6. *(Satisfaction of application conditions [9]). Given an application condition φ and a sub set of features FS, called a feature selection, we say that FS satisfies φ, written as $FS \models \varphi$, iff: (1) $FS \models true$ always; (2) $FS \models a$ iff $a \in FS$; (3) $FS \models \neg\varphi$ iff $FS \not\models \varphi$; (4) $FS \models \varphi_1 \wedge \varphi_2$ iff $FS \models \varphi_1$ and $FS \models \varphi_2$*

Definition 7. *(Update). An update[9] is defined by the following grammar: $u ::= noop \mid a\ on \mid a\ off \mid u; u$, where $a \in F$ and F is a set of features. We write U_F to denote the set of all updates over F. Given a feature selection $FS \subseteq F$, an update expression modifies FS according to the following rules:* r1: $FS \xrightarrow{noop} FS$; r2: $FS \xrightarrow{a\ on} FS \cup \{a\}$; r3: $FS \xrightarrow{a\ off} FS \setminus \{a\}$; r4: $\frac{FS \xrightarrow{u_0} FS' \quad FS' \xrightarrow{u_1} FS''}{FS \xrightarrow{u_0;u_1} FS''}$.

We are now in position to introduce R^2PNs.

Definition 8. *(Reconfigurable Recursive Petri nets). A R^2PNs is a tuple $EN = \langle N, F, f, u\rangle$, where :*

- *$N = \langle P, T, Pre, Post, \Omega, I, \Upsilon, K\rangle$ is RPN,*
- *F is a set of features,*
- *$f : T \cup \{\tau_i\} \to \Phi_F$ is a function associating to each transition and cut step with an application condition from Φ_F where $i \in I$,*
- *$u : T \cup \{\tau_i\} \to U_F$ is a function associating to each transition and cut step with an update from U_F where $i \in I$.*
  We write u_t resp. u_{τ_i} to denote the update expression $u(t)$ resp. $u(\tau_i)$ associated to a transition t resp. a cut step τ_i.

we write $FS \models f(t)$ if the feature selection FS satisfies the application condition associated with transition t. In the following, graphically, each transition of R^2PN is annotated by an application condition and an update expression in the following way:

$$\frac{application\ condition}{update\ expression}$$

Definition 9. *(A state of Reconfigurable RPN). A state of a Reconfigurable RPN* $EN = \langle N, F, f, u\rangle$ *is a tuple* $S = (EM, FS)$ *where* $EM = \langle V, M, E, A\rangle$ *is an extended marking and* $FS \subseteq F$ *is a feature selection.*

Definition 10. *(Enabled transition or cut step). Let* $S = (\langle V, M, E, A\rangle, FS)$ *be a state of* R^2*PN* $EN = \langle N, F, f, u\rangle$ *where* $N = \langle P, T, Pre, Post, \Omega, I, \Upsilon, K\rangle$*. Let a node* $v \in V$*.*

- *A transition* t *is enabled in a node* v*, if* $\forall p \in P : M(v)(p) \geq Pre(p, t)$ *and* $FS \models f(t)$*,*
- *A cut step* τ_i *is enabled in a node* v*, if* $M(v) \in \Upsilon_i$ *and* $FS \models f(\tau_i)$*.*

Definition 11. *(Firing rules of Reconfigurable RPN). Let* $S = (EM, FS)$ *be a state of* R^2*PN* $EN = \langle N, F, f, u\rangle$ *where* $N = \langle P, T, Pre, Post, \Omega, I, \Upsilon, K\rangle$*. Let a node* $v \in V$*.*

- *The firing of an elementary transition* t *from a node* v *leads to a state* $S' = (EM', FS')$ *where* $EM \xrightarrow{v,t} EM'$ *as* Definition12. *in [3]. and* $FS \xrightarrow{u(t)} FS'$*,*
- *The firing of an abstract transition* t *from a node* v *leads to a state* $S' = (EM', FS')$ *where* $EM \xrightarrow{v,t} EM'$ *as* Definition13. *in [3]. and* $FS \xrightarrow{u(t)} FS'$*,*
- *The firing of a cut step* τ_i *from a node* v *leads to a state* $S' = (EM', FS')$ *where* $EM \xrightarrow{v,\tau_i} EM'$ *as* Definition14. *in [3]. and* $FS \xrightarrow{u(\tau_i)} FS'$*.*

Therefore, the analysis of R^2PN is based on constructing its extended reachability graph, which is used for checking properties such as reachability, deadlock and liveness.

5 Case Study : Automated Production Systems

In this research work, we use a reconfigurable production devices called, FESTO[7] as a running example. We assume that the device may perform some particular reconfiguration scenarios according to well-defined conditions. FESTO is composed of three units: distribution, test and processing units. The distribution unit is composed of a pneumatic feeder and a converter to forward cylindrical work pieces from a stack to the testing unit which is composed of the detector, the tester and the elevator. The testing unit checks of work pieces for height, material type and color. Work pieces that successfully pass this check are forwarded to the rotating disk of the processing unit, where the drilling of the work piece is performed. We assume in this work two drilling machines $Dr1$ and $Dr2$ to drill pieces. The result of the drilling operation is next checked by a checking machine and the work piece is forwarded to another mechanical unit. Three production modes (called local configurations) can be performed by $FESTO$.

- $Light1$: For this production mode, only the drilling machine $Dr1$ is used;
- $Light2$: To drill work pieces for this production mode, only the drilling machine $Dr2$ is used;
- $High$: For this production mode, where $Dr1$ and $Dr2$ are used at the same time in order to accelerate the production.

$Ligth1$ is the default production mode of $FESTO$ and the system completely stops in the worst case if the two drilling machines are broken. We assume that $FESTO$ may perform four reconfiguration scenarios as shown in Fig.2.

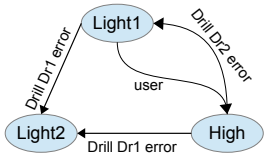


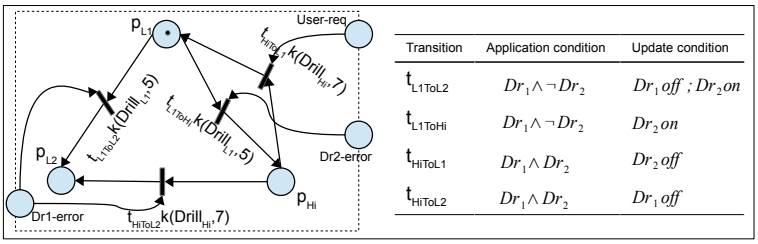
Fig. 2. Reconfiguration scenarios of $FESTO$

6 Modeling $FESTO$ using Reconfigurable RPN

The automated production system $FESTO$ is modelled as follows: $EN_{FESTO} = \langle EN_{Beh}, EN_{Adapt} \rangle$ where EN_{Beh} represents the behaviour module of $FESTO$ and EN_{Adapt} is the adaptor which represents possible reconfiguration scenarios may be applied by the reconfigurable control system $FESTO$.

The adaptor EN_{Adapt} of $FESTO$ is shown in Fig.3. It is represented by ERPN where each place specifies one beahviour. As shown in Fig.3, we have three places p_{L1}, p_{L2} and p_{Hi} which specify the three production modes $Light1$, $Light2$ and $High$. Each one of these places may contain at most one token and the marking of such place means that its associated production mode is currently applied by the production system $FESTO$. For instance, the place p_{L1} is marked, which means the current production mode applied by $FESTO$ is $Light1$ i.e.the initial production mode. The set of elemenatry transitions represent the set of reconfiguration scenarios of $FESTO$. For instance, the elementary transition t_{L1ToL2} models the reconfiguration scenario that allows the production system $FESTO$ to transform from the first production mode $Light1$ to the second production mode $Light2$ when drilling machine $Dr1$ is broken. In fact, the firing of this transition will interrupt the abstract transition $Drill_{L1}$, which models the first production mode $Light1$, and update the current feature selection FS by applying its associated update expression $Dr1\ off$;$Dr2\ on$ as shown in Fig.3.

The behaviour EN_{Beh} of $FESTO$ which is a union of multiple R^2PNs is formalised as follows: $EN_{Beh} = \bigcup_{i \in \overline{1..3}} EN_{Beh_i}$, with $EN_{Beh_i} = \langle P_i, T_i, Pre_i, Post_i, \Omega_i, I_i, \Upsilon_i, K_i, F_i, f_i, u_i \rangle$ is a R^2PN models one possible behaviour of reconfigurable control system of $FESTO$. Fig.4 models the behaviour of $FESTO$ using ERPN. All the transitions shown in Fig.4, where their $application\ condition$ and $update\ expression$ are omitted, are annotated by the term : $\frac{true}{noop}$. This means that this set of transitions are common to all behaviours of $FESTO$. The set of features F contains the set of drilling machines which may be used to select the proper behaviour of $FESTO$ i.e. $F = \{Dr1, Dr2\}$. As noted in Fig.4, the abstract transitions $Distribute$, $Test$ and $Process$ models the distribution, tester and processing unit. The firing of one of these transitions will create a thread representing the behaviour of associating unit.



Transition	Application condition	Update condition
t_{L1ToL2}	$Dr_1 \wedge \neg Dr_2$	$Dr_1 off\ ; Dr_2 on$
t_{L1ToHi}	$Dr_1 \wedge \neg Dr_2$	$Dr_2 on$
t_{HiToL1}	$Dr_1 \wedge Dr_2$	$Dr_2 off$
t_{HiToL2}	$Dr_1 \wedge Dr_2$	$Dr_1 off$

Fig. 3. R^2PN represents FESTO's adaptor

For instance, the firing of the abstract transition $Process$ creates a thread, models the behaviour of processing unit, which starts by one token i.e. workpiece in place p_{12}. The workpiece is then forwarded to the drilling machines by firing the elementary transition $Rotate$. After, three abstract transitions $Drill_{L1}$, $Drill_{L2}$ and $Drill_{Hi}$ may be enabled; they model the drilling's step according to the three production modes $Light1$, $Light2$ and $High$. But each one of these abstract transitions is associated an application condition which restricts its activation (firing) to the set of bound features F. As described above, the default production mode of $FESTO$ is $Light1$, where only the drilling machine $Dr1$ is used, so the initial feature selection $FS_0 = \{Dr1\}$. In this case, only the abstract transition $Drill_{L1}$ is enabled. The firing of this abstract transition will create a thread, models the drilling's step, which starts by one token in place p_{17}. Note that the created thread can use only the drilling machine $Dr1$ represented by the elementary transition $Dr1$-$L1$. Moreover, this thread presents two types of termination :

- $Properly\ termination$: it means that the workpiece is well drilled and the place p_{18} is marked. So, a final marking belongs to termination's set Υ_4 is reached, then the cut step τ_4 may be enabled. The firing of τ_4 terminates the current thread and puts a workpiece in the place p_{14} in order to perform the remains operations such as $Checker$ and $Evacuate$.
- $Termination\ by\ interruption$: this termination occured when the production system $FESTO$ applies a reconfiguration as described above for adaptor module. For instance,from Fig.3, firing the elementary transition t_{L1ToL2} will interrupt the thread created by the absract transition $Drill_{L1}$ with termination index 5 and update the feature selection FS_0. The new obtained feature selection $FS_1 = \{Dr2\}$. In fact, the workpiece is put it in the place p_{13} and only the abstract transition $Drill_{L2}$ may be enabled, which specify the drilling's step according to the second production mode $Light2$.

7 Verification of Reconfigurable control systems

In this section, we outline the conversion from R^2PNs to a Maude specification [6] and the use of its Linear Temporal Logic (LTL) model checker [11].

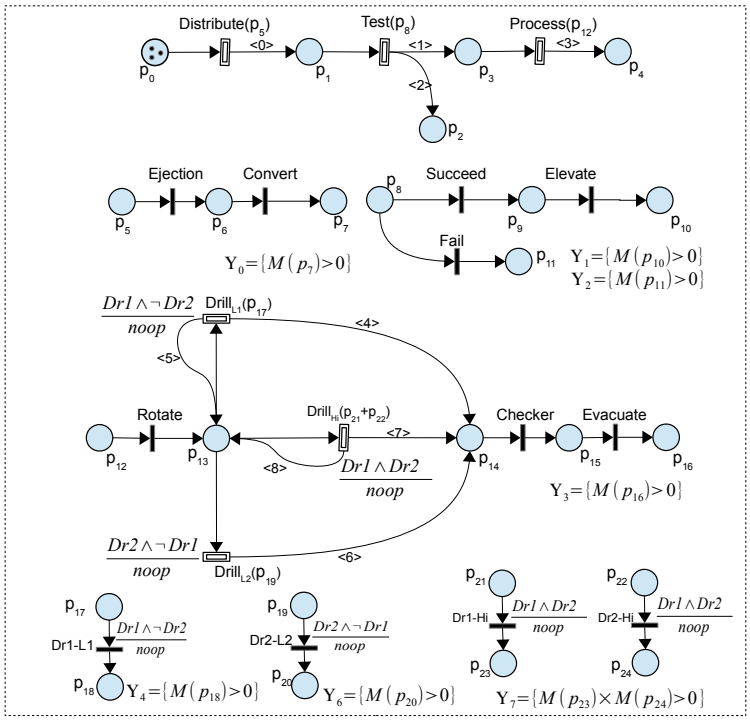


Fig. 4. R^2PN represents FESTO's behaviour

7.1 Maude and its Model-Checker

Maude is a high-performance reflective language and system supporting rewriting logic specification [12]. It has been developed at SRI (URL: http://maude.cs.uiuc.edu/) International for over two decades. A system, under Maude, is represented using membership equational logic describing its set of states and a set of rewrite rules representing its state transitions. Maude is stricly typed, where the types are called $sorts$ and can be built hierarchically using $subsorts$. Maude's basic programming statements are equations and rules, and have in both cases a simple rewriting semantics in which instances of the left-hand side pattern are replaced by corresponding instances of the right-hand side. One aim using Maude is its LTL model-checker which can be used to verify properties as reachability, deadlock or liveness for a specified model. Model checking can be used to prove properties, specified in LTL when the set of states reachable from an initial state in a system module is finite. In [11], the author presents more details about syntax and semantic of LTL.

7.2 Conversion of R^2PN to a Maude specification

Like in [8], the state of a R^2PN is described by a term $State(EM, fs)$ of sort $STATE$ where:

– EM is an extended marking represented, in a recursive way, as a dynamical tree by the term $[M_{Th}, tabs, ThreadChilds]$ of sort $Thread$ where M, of sort $Marking$, represents the internal marking of Th. The term $tabs$ represents the name of the abstract transition whose firing (in its thread father) gave birth to the thread Th. Note that the root thread is not generated by any abstract transition, so the abstract transition which gave birth to it, is represented by the constant $nullTrans$. The term $ThreadChilds$ represents a finite multiset of threads generated by the firing of abstract transitions in the thread Th. We denote by the constant $nullThread$, the empty thread.
– fs is a feature selection, of sort FS, represented by a list of terms of sort $Term$. We denote by the constant $empty$, the empty list of feature selection.

We have also impleneted two functions in the module $FeatureSel$ needed by our formalism R^2PN. The first function is $SATAC(ac : AC, fs : FS) : Bool$ which checks the truth value of an application condition ac, of sort AC, for a given feature selection fs. The second function is $UPDATE(u : UE, fs : FS) : FS$ which returns the new feature selection after applying the update expression u, of sort UE, for a given feature selection fs.

Moreover, each transition firing and cut step execution is formally specified in Maude by a labelled rewrite rule as follows :

– Rule associated to an elemetary transition t with $K(t) = \phi$, application condition $ac(t)$ and update expression $ue(t)$

```
crl[t]: State(<p; N+Pre(p,t)> (*) <p'; M> , fs) =>State(<p; N
    )> (*) <p'; M + Post(p',t)> , UPDATE(ue(t), fs) if SATAC(
    ac(t), fs).
```

– Rule associated to an elemetary transition t with $K(t) = \{(t_{absi}, k), (t_{absj}, m), ..\}$, application condition $ac(t)$ and update expression $ue(t)$

```
crl[t]: State([<p;N+Pre(p,t)>(*)<p';M>(*)<p'_i;A>(*)<p'_j;B>,
    absTrans,Thread],fs) => State([<p; N>(*)<p'; M+Post(p',t)
    >(*)<p'_i; A+Post(p'_i, t_absi,k)>(*)<p'_j; B+Post(p'_j,t_absj,m)>,
    absTrans, DeleteThread(t_absi,t_absj,...,Thread)], UPDATE(ue(t
    ), fs) if SATAC(ac(t), fs).
```

– Rule associated to an abstract transition t with starting marking $\Omega(t)$, application condition $ac(t)$ and update expression $ue(t)$

```
crl[t]: State([<p;N+Pre(p,t)>, absTrans, Thread] , fs) =>
    State([<p; N)>, absTrans, Thread[<p'; Ω(t)>, t, nullThread
    ]] , UPDATE(ue(t), fs) if SATAC(ac(t), fs).
```

– Rule associated to a cut step τ_i with application condition $ac(\tau_i)$ and update expression $ue(\tau_i)$

```
crl[τ_i]: State([<p;N>, absTrans, Thread[<p';N'> , tabs,
    Thread1]] , fs) => State([<p; N+Post(p, tabs, i)>,
    absTrans, Thread, UPDATE(ue(τ_i), fs) if (Υ_i and SATAC(ac(τ_i
    ), fs)).
```

7.3 Implementation using the Maude Tool

Since we give a Maude specification for the formalism R^2PN, we can benefit from the use of the LTL model-checker of the Maude system for verification purpose where the generated state space must be finite. For instance, one can check the liveness property over EN_{FESTO} for its initial behaviour $Ligth1$. We suppose that the system starts by100 tokens i.e. workpieces, this is specified in Maude by : $eqinitialState = State(< p0; 100 > (*) < pL1; 1 >, Dr1)$. A liveness condition is : *each workpiece must reach (from all reachable markings) the final state where the place* p_4 *is marked*.This can be phrased as "*For all paths and from all states,* $State(< p4; 100 > (*) < pL1; 1 >, Dr1)$ *can finally be reached*". In Maude, this is stated by $[] <> State([< p4; 100 > (*) < pL1; 1 >, nullTrans, nullThread], Dr1)$., and proven to be $valid$ by its model checker in Fig.5(a). We suppose in this case that there is no $fail$ during the workpieces's test process.

Let take another example and we focus on the case, when an error occurs, whether the control module can respond and select a proper behaviour. We define the following LTL formula : $\alpha : [](Behaviour(Light1)/\ Drill - Down(Dr1) =>$
$<> Behaviour(Light2)))$, where, the predicate $Behaviour$ allows to know the current behaviour applied by the production system $FESTO$. The predicate $Drill$-$Down$ indicates which among drilling machines $Dr1$ ord $Dr2$ is break-down.

This LTL formula means that, always, if the current production mode of $FESTO$ is $Light1$, drill machine $Dr1$ is broken, the production system $FESTO$ will eventually select the production mode $Light2$. This LTL formula is proved to be $valid$ in Fig.5(b).

Now, let's define a LTL proprty β by replacing in the formula α the production mode $Light2$ by $High$. In Fig.5(c), this formula is proved to be $not\ valid$ and the model-checker returns the expected $counterexample$.

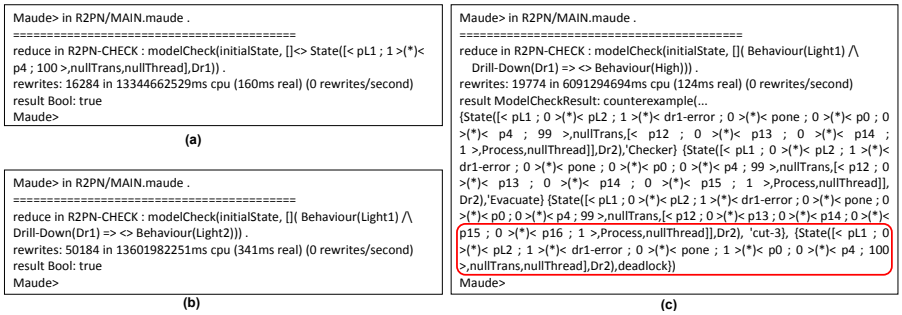


Fig. 5. (a) Model checking of the liveness condition for first production mode of $FESTO$, **(b)** Model checking of the LTL property α and **(c)** $Counterexample$ generated by model checking of the LTL property β

8 Conclusion and Future Work

This research work copes with the reconfiguration issue of discrete control systems. We have proposed Renconfigurable RPN (R^2PN) which enriches RPN by the concept of $feature$ to deal with reconfigurations at runtime. R^2PN allows instance of threads in RPN to be renconfigurable. We have shown the efficiency of R^2PN through a case study represented by a reconfigurable production system. A verification method for R^2PN has also been presented by using the LTL model-cheker of Maude.

In the future, we will plan to extend our formalism in order to model time constraints which are of great importance in real-time systems. Therefore, one can verify some properties with respect to time constraints using Real-Time Maude model-checker [10].

References

1. R. Valk, "Self-modifying nets, a natural extension of petri nets," in *Proceedings of the Fifth Colloquium on Automata, Languages and Programming*, 1978, pp. 464–476.
2. T. Murata, "Petri nets: Properties, analysis and applications," Proceedings of the IEEE, vol. 77, no. 4, Apr 1989, pp. 541–580.
3. S. Haddad and D. Poitrenaud, "Recursive Petri nets – Theory and application to discretee event systems," Acta Informatica, vol. 44, no. 7-8, Dec. 2007, pp. 463–508.
4. S.-U. Guan and S.-S. Lim, "Modeling adaptable multimedia and self-modifying protocol execution," Future Gener. Comput. Syst., vol. 20, no. 1, Jan. 2004, pp. 123–143.
5. M. L. E. Badouel and J. Oliver, "Modeling concurrent systems: Reconfigurable nets," in In Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 03). CSREA Press, 2003, pp. 1568–1574.
6. M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada, "Maude: specification and programming in rewriting logic," Theoretical Computer Science, vol. 285, no. 2, 2002, pp. 187 – 243, rewriting Logic and its Applications.
7. J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. Al-Ahmari, "R-tnces: A novel formalism for reconfigurable discretee event control systems," Systems, Man, and Cybernetics: Systems, IEEE Transactions on, vol. 43, no. 4, 2013, pp. 757–772.
8. K. Barkaoui and A. Hicheur, "Towards analysis of flexible and collaborative workflow using recursive ecatnets," in Business Process Management Workshops, ser. Lecture Notes in Computer Science, A. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds. Springer Berlin Heidelberg, 2008, vol. 4928, pp. 232–244.
9. R. Muschevici, "Modelling Diversity in Software Product Lines," PhD thesis, KU Leuven university, Belgium. Dec-2013.
10. P.C. Ölveczky, and J. Meseguer, " Real-Time Maude: A tool for simulating and analyzing real-time and hybrid systems". In *3rd International Workshop on Rewriting Logic and its Applications (WRLA'00)*, volume 36 of *Electronic Notes in Theoretical Computer Science*. 2000.
11. S. Eker, J. Meseguer, and A. Sridharanarayanan, "The maude LTL model checker," Electronic Notes in Theoretical Computer Science, vol. 71, no. 0, 2004, pp. 162 – 187.
12. J. Meseguer, "Conditioned rewriting logic as a united model of concurrency." Theor. Comput. Sci., vol. 96, 1992, pp. 73–155.
13. K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-021, 1990.