

A high-level nets based approach for reconfigurations of distributed control systems

Ahmed Kheldoun¹, JiaFeng Zhang², Kamel Barkaoui³, and Malika Ioualalen⁴

¹ Sciences and Technology Faculty, Yahia Fares University, Medea, Algeria
ahmedkheldoun@yahoo.fr

² School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China
zhangjiafeng628@gmail.com

³ CNAM, 292 Rue Saint-Martin 75141, Cedex 03 Paris, France
kamel.barkaoui@cnam.fr

⁴ MOVEP, Computer Science Department, USTHB, Algiers, Algeria
mioualalen@usthb.dz

Abstract. The paper deals with automatic reconfiguration problems of distributed control systems that are composed of a group/set of networked reconfigurable devices. A multi-agent architecture is proposed such that each device of a distributed control system has a special self-governed agent named reconfiguration sub-controller agent to manage its local reconfiguration. Accordingly, a communication protocol is developed to handle interaction style among these agents for the purpose of consistency. The proposed architecture is modeled by RECAT-Nets and the distributed reconfiguration process is modeled by an ECATNet. Furthermore, in order to check the correctness of the proposed approach, the model checker Maude is applied, where required properties are specified by Linear Temporal Logic. Finally, a virtual distributed control system contains two reconfigurable devices is applied to illustrate this work.

Keywords: Distributed control systems, Reconfiguration, Agent-based system, Recursive ECATNet.

1 Introduction

Distributed reconfigurable control systems (DRCSs) receive more and more attention from academic and industry for their broad application prospects [1],[2]. In industry, the development of safe DRCSs is not an obvious activity because of their dynamic reconfigurations that are implemented by the dynamic addition/removal of software/hardware components, the modification of logic relationship among components, and the adjustment of system states and data [3],[4]. This type of systems manages a set of networked reconfigurable devices that should cooperate with each other, since any uncontrolled automatic reconfiguration applied in a device may cause serious disturbance on others and thus on the safety and correctness of the whole system.

Many researchers have tried to deal with the formal modeling of control systems with potential reconfigurations. The author of [5] proposed self-modifying nets that can

modify their own firing rules at runtime. However, most of the basic decidable properties of Petri nets such as reachability, place boundedness are lost for this extended model. In [6], the authors developed a Reconfigurable Petri Nets (RPN) for modeling adaptable multimedia and protocols that can self-modify during execution, where the reconfiguration behaviour in Petri nets are modeled by novel modifier places. The work in [7] presented net rewriting systems that are an extension of Petri nets. The concept of rewriting rules was depicted, where the execution of a rewriting can change the configuration of a Petri net. Reconfigurable timed net condition/event systems (R-TNCESs) are first developed in [4]. The authors aim to optimal model reconfigurable discrete event control systems (RDECS). An R-TNCES is modular and the reconfigurable control ideas are applied in the formalism directly [8]. All these methods are efficient in their concerned fields. However, most of the proposed formal models lack of modularity, cannot allow the compact and concise modeling of self-reconfigurability of complex systems. Most importantly, none of them is qualified in modeling DRCSs.

We mention the works of [3] and [9], where the authors defined a multi-agent architecture for distributed reconfigurable embedded systems. For each device, a reconfiguration agent modeled by a nested state machine is associated in order to handle its local reconfiguration scenarios. Furthermore, for the purpose of coherent reconfigurations of distributed devices, an inter-agents communication protocol was developed, which is based on well-defined coordination matrices held by a coordination agent. Nevertheless, the proposed protocol is limited to treat only one reconfiguration requirement in the case of multiple requirements arising simultaneously. Therefore, in this work, we propose a novel multi-agent architecture and a new communication protocol in order to handle all possible reconfiguration scenarios in a DRCS including treating multiple concurrent requirements.

By using the novel multi-agent architecture, each reconfigurable device is assigned with a special agent named reconfiguration sub-controller agent to manage its local automatic reconfigurations. With the new communication protocol that regulates the interaction among agents, the consistency of agents is solved without any mechanism such as a special coordinator. Besides, each reconfiguration sub-controller agent handles a set of *Decision Matrices* for treating with concurrent reconfiguration requirements from different agents. The concept of *Decision Matrix* was inspired from the concept of *Coordination Matrix* defined in [9].

In this paper, the RECATNet [10] is applied to model each reconfiguration sub-controller agent, which is a high-level algebraic net dedicated to the modeling and the analysis of interactions in collaborative and distributed systems with dynamic structure. In addition, the distributed reconfiguration processes are modeled by ECATNet [10]. To check the rationality and correctness of the proposed approach, the model checker Maude [11] is applied to check the linear temporal logic (LTL) based properties. Moreover, the proposed approach is illustrated by a virtual distributed reconfigurable production system that is composed of two physically reconfigurable benchmark production devices: FESTO and EnAS. Their prototypes are available at Martin Luther University [3],[4]<http://aut.informatik.uni-halle.de/>.

The remainder of this paper is organized as follows. Section 2 recalls the basic concepts of Recursive ECATNet. The virtual reconfigurable distributed control system that

we use as running example is introduced in Section 3 before the explanation of the proposed approach in Section 4 including the RECATNet and ECATNet based modeling. After that, the verification of the obtained models using the model checker Maude is described in Section 5. Finally, section 6 concludes this paper and depicts further research plans of the authors.

2 Recursive ECATNet Review

Recursive ECATNets (abbreviated RECATNets) are a kind of high level algebraic Petri nets combining the expressive power of abstract data types and Recursive Petri nets [12]. Each place in such a net is associated to a sort (i.e. a data type of the underlying algebraic specification associated to this net). The marking of a place is a multiset of algebraic terms (without variables) of the same sort of this place. Moreover, transitions in RECATNet are partitioned into two types : elementary (Fig.1) and abstract transitions (Fig.1(b)). Each abstract transition is associated to a starting marking represented graphically in a frame. A capacity associated to a place p specifies the number of algebraic terms which can be contained in this place for each element of the sort associated to p . As shown in Fig.1, the places p and p' are respectively associated to the sorts s

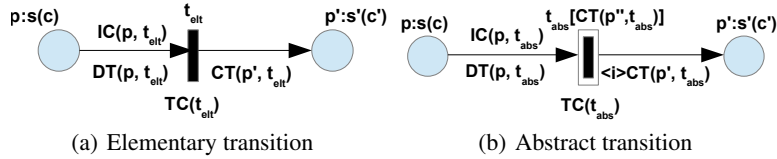


Fig. 1. Transition types in RECATNets

and s' and to the capacity c and c' . An arc from an input place p to a transition t (elementary or abstract) is labelled by two algebraic expressions $IC(p, t)$ (Input Condition) and $DT(p, t)$ (Destroyed Tokens). The expression $IC(p, t)$ specifies the partial condition on the marking of the place p for the enabling of t (see Table.1). The expression $DT(p, t)$ specifies the multiset of terms to be removed from the marking of place p when t is fired. Also, each transition t may be labelled by a Boolean expression $TC(t)$ which specifies an additional enabling condition on the values taken by contextual variables of t (i.e. local variables of the expressions IC and DT labelling all the input arcs of t). When the condition $TC(t)$ is omitted, the default value is the term *True*. For an elementary transition t , an output arc (t, p') connecting this transition t to a place p' is labelled by the expression $CT(t, p')$ (*Created Tokens*). However, for an abstract transition t , an output arc (t, p') is labelled by the expression $\langle i \rangle CT(t, p')$ (*Indexed Created Tokens*). These two algebraic expressions specify the multiset of terms to produce in the output place p' when the transition t is fired. In the graphical representation of RECATNets, we note the capacity of a place regarding an element of its sort only if this number is finite. If $IC(p, t) =_{def} DT(p, t)$ on input arc (p, t) (e.g. $IC(p, t) = a^+$ and $DT(p, t) = a$), the expression $DT(p, t)$ is omitted on this arc.

In what follows, we note $Spec = (S, E)$ an algebraic specification of an abstract data type associated to a RECATNet, where $\Sigma = (S, OP)$ is its multi-sort signature (S is a finite set of sort symbols and OP is a finite set operations, such $OP \cap S = \phi$). E is the set of equations associated to $Spec$. $X = (X_s)_{s \in S}$ is a set of disjoint variables associated to $Spec$ where $OP \cap X = \phi$ and X_s is the set of variables of sort s . We denote by $T_{\Sigma, s}(X)$ the set of S -sorted S -terms with variables in the set X . $[T_{\Sigma}(X)]_{\oplus}$ denotes the set of the multisets of the Σ -terms $T_{\Sigma}(X)$ where the multiset union operator (\oplus) is associative, commutative and admits the empty multiset ϕ as the identity element. For a transition t , $X(t)$ denotes the set of the variables of the context of this transition and $Assign(t)$ denotes the set of all the possible affectations of this variables set, i.e. $Assign(t) = \{sub : X(t) \rightarrow T_{\Sigma}(\phi) \mid x_i \in X(t) \text{ of sort } s, sub(x_i) \in T_{\Sigma, s}(\phi)\}$

Definition 1. (Recursive ECATNets). A recursive ECATNet is a tuple $RECATNet = (Spec, P, T, sort, Cap, IC, DT, CT, TC, I, \Upsilon, ICT)$ where:

- $Spec = (\Sigma, E)$ is a many sorted algebra where the sorts domains are finite (with $\Sigma = (S, OP)$), and $X = (X_s)_{s \in S}$ is a set of S -sorted variables
- P is a finite set of places.
- $T = T_{elt} \cup T_{abs}$ is finite set of transitions ($T \cap P = \phi$) partitioned into abstract and elementary ones. T_{abs} and T_{elt} denoted the set of abstract and elementary transitions.
- $sort: P \rightarrow S$, is a mapping called a sort assignment.
- Cap : is a P -vector on capacity places: $p \in P$, $Cap(p): T_{\Sigma}(\phi) \rightarrow N \cup \{\infty\}$,
- $IC : P \times T \rightarrow [T_{\Sigma}(X)]_{\oplus}^*$ where $[T_{\Sigma}(X)]_{\oplus}^* = \{\alpha^+\} \cup \{\alpha^-\} \cup \{\alpha^0\}$, such that $\alpha \in [T_{\Sigma, sort(p)}(X)]_{\oplus}$
- $DT : P \times T \rightarrow [T_{\Sigma}(X)]_{\oplus}$, such that $DT(p, t) \in [T_{\Sigma, sort(p)}(X)]_{\oplus}$,
- $CT : P \times T \rightarrow [T_{\Sigma}(X)]_{\oplus}$, such that $CT(p, t) \in [T_{\Sigma, sort(p)}(X)]_{\oplus}$,
- $TC : T \rightarrow [T_{\Sigma, bool}(X)]$,
- I is a finite set of indices, called termination indices,
- Υ is a family, indexed by I , of effective representation of semi-linear sets of final markings,
- $ICT : P \times T_{abs} \times I \rightarrow [T_{\Sigma}(X)]_{\oplus}$, where $ICT(p, t, i) \in [T_{\Sigma, sort(p)}(X)]_{\oplus}$.

Table 1. Different forms of Input Condition $IC(p,t)$

IC(p,t)	Enabling Condition
a^0	The marking of the place p must be equal to a (e.g. $IC(p, t) = \phi^0$ means the marking of p must empty)
a^+	The marking of the place p must include a (e.g. $IC(p, t) = \phi^+$ means condition is always satisfied)
a^-	The marking of the place p must not include a , with $a \neq \phi$
$\alpha 1 \wedge \alpha 2$	Conditions $\alpha 1$ and $\alpha 2$ are both true
$\alpha 1 \vee \alpha 2$	$\alpha 1$ or $\alpha 2$ is true

Informally, a RECATNet generates during its execution a dynamical tree of marked threads called an extended marking, which reflects the global state of 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 of a RECATNet is a labelled rooted tree denoted $Tr = \langle V, M, E, A \rangle$ where:

- V is the set of nodes (i.e. threads),
- M is a Mapping $V \rightarrow [T_{\Sigma}(\phi)]_{\oplus}$ associating an ordinary marking with each node of the tree, such that $\forall v \in V, \forall p \in P, M(v)(p) \leq Cap(p)$,
- $E \subseteq V \times V$ is the set of edges,
- A is a mapping $E \rightarrow T_{abs}$ associating an abstract transition with each edge.

Example 1. Fig2(a) illustrates the characteristic features of RECATNet. (1) An abstract transition t is followed by the starting marking $CT(p, t)$. For instance $SendRequest$ is an abstract transition and $CT(SendRq) = (Request, Rq)$ where Rq represents, for instance, the request of client product. The firing of $SendRequest$ will create a thread that starts with one token Rq in place $Request$. (2) Any termination set can be defined concisely based on place marking, for instance, γ_0 specifies the final marking of threads such that the place $EndRequest$ contains at least one token. (3) The index of termination ($\langle 0 \rangle$ or $\langle 1 \rangle$) specifies the place $ResultOk$ or $ResultNotOk$, respectively, will be marked after the termination of the created thread.

Note that contrary to ordinary nets, RECATNet are often disconnected since each connected component may be activated by the firing of abstract transitions.

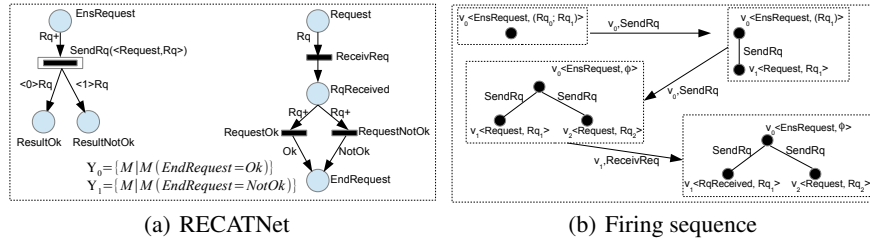


Fig. 2. Example of a RECATNet and one possible firing sequence

Example 2. Fig2(b) highlights a possible firing sequence of the RECATNet represented in Fig2(a). The graphical representation of any extended marking Tr is a tree where an arc $v_1(m_1) \rightarrow v_2(m_2)$ labeled by t_{abs} means that v_2 is a child of v_1 created by firing abstract transition t_{abs} and m_1 (resp. m_2) is the marking of v_1 (resp. v_2). Note that the initial extended marking is Tr_0 is reduced to a single node v_0 whose marking is $\langle EnsRequest, (Rq1; Rq2; Rq3) \rangle$. More details about RECATNets such as firing transitions and generating extended reachability graph are presented in [10][13]. The usefulness of the formalism of RECATNet is: (1) modeling and analysis of interactions in distributed systems having a dynamic structure and (2) its semantic may be defined

in terms of conditional rewriting logic [14] therefore, the model-checker MAUDE [11] can be used to check its behavioural properties. This paper applies RECATNet to model the sub-controllers in the proposed multi-controller based multi-agent architecture.

3 Reconfiguration of Automated Production Systems

In this research work, we use a virtual distributed system composed of two physically reconfigurable production devices: FESTO and EnAS as a running example. We assume that the two devices cooperate to manufacture workpieces and some particular reconfiguration scenarios can be applied to them according to well-defined conditions.

3.1 FESTO

It 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 performs 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. Four production modes (called local reconfigurations) 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;
- *Medium*: Medium production mode, where $Dr1$ and $Dr2$ are used alternatively;
- *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 only light and medium production modes are interchangeable, so are medium and high production modes. The high production mode can be transformed into light production mode directly, but the reverse is not allowed.

3.2 EnAS

It transports work pieces from FESTO into storage units. The work pieces shall be placed inside tins to close with caps afterwards. The EnAS is mainly composed of a belt, two jack stations (J1 and J2) and two gripper stations (G1 and G2). The jack stations place new drilled work pieces from FESTO and close tins with caps, whereas the gripper stations remove charged tins from the belt into storage units. Initially, the belt moves a particular pallet containing a tin and a cap into the first jack station J1. Three production modes can be performed by EnAS.

- *Policy1*: The jack station J1 places a new work piece from FESTO in a tin before closing the tin with the cap. In this case, the gripper station G1 removes the tin from the belt into a storage station;
- *Policy2*: When the jack station J1 is broken, an empty tin is displaced to the jack station J2, where a work piece and a cup are put. The closed tin is displaced thereafter on the belt to the gripper station G2 for an evacuation to a storage station;
- *Policy3*: The jack station J1 places just a drilled work piece in the tin that is moved thereafter into the jack station J2 to place a second new work piece. Once J2 closes the tin with a cap, the belt moves the pallet into the gripper station G2 to remove the tin (with two work pieces) into a storage station.

Policy1 is the default behaviour mode of EnAS and the system completely stops in the worst case if the two jack stations are broken. We assume that only *Policy1* and *Policy3* are interchangeable. We suppose that the two benchmark production systems FESTO and EnAS are logically linked to coordinate their tasks. The allowed compositions of behaviour modes of the two systems are defined in Tab.2. In fact, when a

Table 2. Possible behaviour compositions of FESTO and EnAS

(FESTO, EnAS)	(FESTO, EnAS)
(Light1, Policy1)	(Medium, Policy1)
(Light1, Policy2)	(Medium, Policy2)
(Light2, Policy1)	(Medium, Policy3)
(Light2, Policy2)	(High, Policy3)

local reconfiguration is planned to be applied to one of these two devices, the other one should have a proper reaction as a response to the planned reconfiguration in order to guarantee the coherence of the whole system.

4 Modelling Distributed Reconfigurable Control System

4.1 Multi-Agent Architecture For Distributed Reconfigurable Control Systems

We define a multi-agent architecture for distributed reconfigurable systems where each reconfigurable sub-controller agent is assigned to a device in order to control its local reconfiguration. In fact, any reconfiguration scenario cannot be applied within a device until its sub-controller receives an acceptance command from the others. The sub-controller may coordinate through a communication network, since any uncontrolled automatic reconfiguration applied to a device may cause serious disturbance on others. Fig.3 shows our proposed architecture. Assume that the architecture manages n networked reconfigurable devices. A multi-agent architecture Π is defined by: $\Pi = (\Lambda, \rho)$ Where $\Lambda = (A_1, \dots, A_n)$ is a set of n sub-controllers agents and ρ represents the communication protocol that defines the interaction rules between the set of sub-controllers agents.

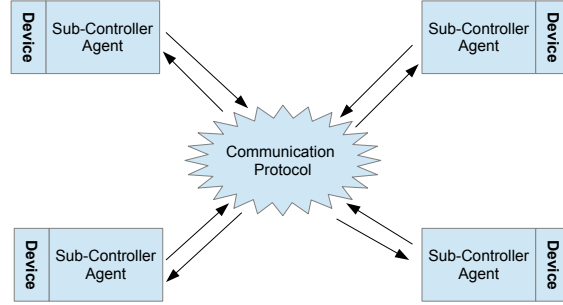


Fig. 3. Architecture of Distributed Control Systems

4.2 Communication Protocol

In this section, we show the main function of communication protocol ρ used in order to coordinate between the sub-controller agents.

When a particular sub-controller agent A_i would to reconfigure itself from the c_i^{th} configuration into the g_i^{th} configuration, it sends, broadcasts, the following request to the other sub-controller agents:

$$request(A_i, A_j, g_i). \text{ where } j = \overline{1, n} \text{ and } j \neq i$$

For each request received by the sub-controller agent A_j , it selects the *Decision Matrix* that will be applied. Let DM_j be such a matrix. It is $m \times 2$ integer, where m is the number of possible local configurations of A_j while the sub-controller agent A_i is in the g_i^{th} configuration. A row vector, denoted by row_j , of DM_j corresponds to a particular configuration of A_i and A_j . For the determinacy of the local configuration of A_j , only one of the row vector in DM_j is finally chosen. We assume, in this current research, that all row vectors of DM_j have the same priority.

Example 3 (Decision Matrix). We show bellow a decision matrix built when receives possible reconfiguration requirements from A_1 .

$$DM_{2,L1} = \begin{pmatrix} L1 & P1 \\ L1 & P2 \end{pmatrix}$$

The matrix $DM_{2,L1}$ is applied when sub-controller agent A_1 requires to transform FESTO into *Light1*. In fact, if *Light1* is activated in A_1 , then *Policy1* or *Policy2* of A_2 can be activated.

Let denote by $row_k^j = (row_{k1}^j, row_{k2}^j)$, where $row_{k1}^j = g_i$, the k^{th} row vector to be applied by the sub-controller A_j . Let denote the current behaviour of each sub-controller agent A_j by c_j . The possible results of sub-controller agent A_j are as follows:

- If no row can be selected in DM_j , then A_j sends the following message to the sub-controller agent A_i : $reject(A_j, A_i, c_j)$. It means that the sub-controller agent A_j reject the reconfiguration requirement of A_i .

- Else
 - If $r_{k2}^j = c_j$, then the sub-controller agent Λ_j sends the following message to the sub-controller agent Λ_i : $accept(\Lambda_j, \Lambda_i, c_j)$. It means that the sub-controller Λ_j agent accept the reconfiguration requirement sent by Λ_i without need to change its local behaviour.
 - If $r_{k2}^j \neq c_j$, and suppose that there exist some rows (suppose the number is w) in the matrix holt by Λ_j in which $r_{k_l 2}^j \neq c_j$ where $l = \overline{1, w}$, then the sub-controller agent Λ_j sends the following message to the sub-controller agent Λ_i : $possible_reconfigure(\Lambda_j, \Lambda_i, r_{k_1 2}^j, r_{k_2 2}^j, \dots, r_{k_w 2}^j)$. It means that the sub-controller agent Λ_j accepts the reconfiguration requirement sent by Λ_i , and needs to apply one of the set of possible reconfigurations requirement $r_{k_l 2}^j$. Let denote by β_{apply} the set of sub-controller agents need to apply a local reconfiguration during a distributed reconfiguration process.

When the sub-controller Λ_i receives a response from all the other sub-controller agents:

- If there is one reject message received, then it sends to the sub-controller agents of the set β_{apply} the following message: $refuse(\Lambda_i, \Lambda_j, r_{k2}^j)$;
- Else, a set of new distributed configurations is formed by combining all the received possible reconfigurations from sub-controllers Λ_j , let denote it by D_Π .
 - If there is a such distributed reconfiguration $d = (r_{k2}^1, \dots, g_i, \dots, r_{k2}^n) \in D_\Pi$ that can be deduced from its Decision Matrices then, Λ_i sends to the sub-controller agents of the set β_{apply} the following message : $apply(\Lambda_i, \Lambda_j, r_{k2}^j)$;
 - Else, Λ_i sends $refuse(\Lambda_i, \Lambda_j, r_{k2}^j)$.

Example 4. We show in Fig.5 required interactions between sub-controller agents when FESTO would to apply *Ligth2* production policy when drilling machine *Dr1* breaks down. The row vector $(L2, P2)$ is finally selected by the sub-controller agent of EnAS i.e. $row_2 = (L2, P2)$. From the selected row, EnAS needs to change its local behavior, so it sends a message $possible_reconfigure(\Lambda_2, \Lambda_1, P2)$ to FESTO. Therefore, there is no reject message received in FESTO, so FESTO sends to EnAS message $apply(\Lambda_1, \Lambda_2, P2)$ in order to apply its new local reconfiguration.

We model each possible distributed reconfiguration process by ECATNet composed of $(\beta_{apply} + 1)$ traces where each trace is composed of places and transitions corresponds to coordination result of a sub-controller agents. Each trace starts with a place p , ends with a place p' , and two transitions between the two places. We distinguish two types of traces:

- A trace which corresponds to the result of a sub-controller that sends a reconfiguration requirement. In this case, a trace is denoted by $p, t_{accept}/t_{reject}, p'$, where t_{accept} and t_{reject} are in conflict. The firing of t_{accept} means that the required reconfiguration is accepted, whereas firing of t_{reject} means that the reconfiguration requirement is rejected at least by one sub-controller agent.
- A trace which corresponds to the result of a sub-controller that does not send a reconfiguration requirement. In this case, a trace is denoted by $p, t_{apply}/t_{refuse}, p'$, where t_{apply} and t_{refuse} are in conflict. The firing of t_{apply} means that the sub-controller agent receives an order to apply the required reconfiguration, whereas

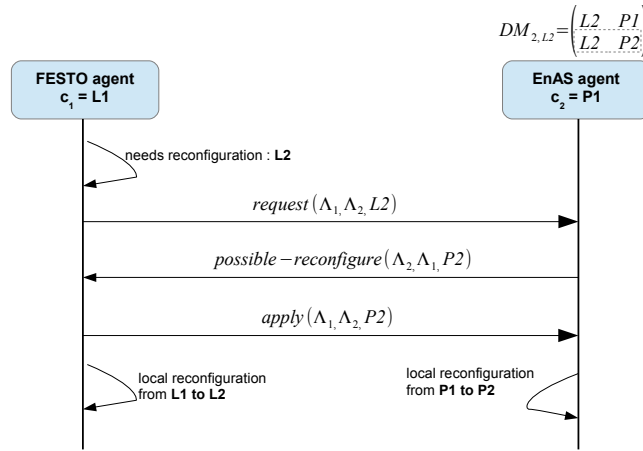


Fig. 4. Coordination between FESTO and EnAS when drilling machine *Dr1* is broken

the firing of t_{refuse} means that the sub-controller agent receives an order to not apply the required reconfiguration.

Example 5. Fig.5 shows the ECATNet based model of the distributed reconfiguration process in *Example 4*. The model of the selected vector $(L2, P2)$ has two traces: the first $p_0, t_{accept}/t_{reject}, rep_1$ corresponds to the result of the reconfiguration requirements of FESTO and the second $p_1, t_{apply}/t_{refuse}, rep_2$ corresponds to the result of the reconfiguration requirements of EnAS according to the reconfiguration requirements of FESTO. For the first trace, according to selected row vector, the sub-controller agent of EnAS Accepts the reconfiguration requirement of FESTO. In this case, the transition t_{accept} will be enabled because its associated condition $(b1 = b2)$ is *true*. For the second trace, EnAS needs to apply the new selected reconfiguration. In this case, the transition t_{apply} will be enabled i.e. its associated condition $(g1 = g2)$ is *true*.

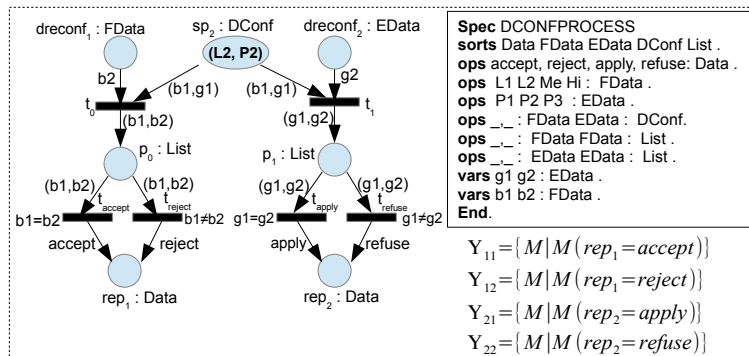


Fig. 5. ECATNet model of a distributed reconfiguration process

4.3 Sub-Controller Agent

The main aim of a sub-controller agent is controlling the local reconfigurations of associated device. This agent should be self-reconfigurable. In fact, the agent may offer multiple behaviours and dynamic transformations of them according to changed environment while keeps the correctness and safety. In fact, and in order to specify the dynamic structure of each sub-controller agent $A_i, i = \overline{1, n}$ we use a marked RECAT-Net, i.e. $A_i = (RN_i, M_{0i})$ where $RN_i = (Spec_i, \dots, ICT_i)$ is a Recursive ECATNet that represents the dynamic structure of this agent and M_{0i} is the initial marking which represents its initial local behaviour.

Example 6 (FESTO). We present in FIG.6(a). the RECATNet of our sub-controller agent A_1 of FESTO. As described above, the sub-controller agent can perform four

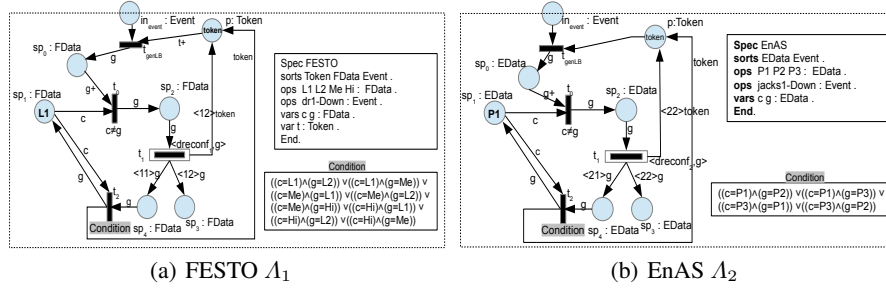


Fig. 6. RECATNets model of the sub-controller agents

policies *Ligth1*, *Ligth2*, *Medium* and *High*. They can be modelled as terms, according to our algebraic specification, by $L1$, $L2$, Me and Hi . The initial marking of this agent is $M_{01} = \langle sp_1, L1 \rangle$ that represents the initial local behaviour *Ligth1*. As shown in Fig.7, the specification includes on input place (*event*). By means of the input place, events are received from FESTO (i.e drill machine *Dr1* is break down,...) or user requirements. For each new received event, the sub-controller agent generates its associate local state in place sp_0 . For instance, when drill machine *Dr1* is break down, the place *event* will receive a token *dr1 – Down* and transition t_{genLB} can be fired and generate the local behaviour *L2* where the sub-controller needs to transform. Marking of place p allows to treat one by one received events. In order to apply the new generated local behaviour, the sub-controller will fire the set of transitions t_0, t_1 and t_2 which defines the self-reconfiguration of the sub-controller agent. The enabling of transition t_0 allows to generate the requirement behaviour g that must be different of the current behaviour c where $c, g \in \{L1, L2, Me, Hi\}$. The abstract transition t_1 allows to check if the sub-controller can be applied the new requirement. When the sub-controller receiving a response, two cases can be distinguished: The first one represented by the termination index $\langle 12 \rangle$ means that the new requirement of the sub-controller is rejected. The second one represented by the termination index $\langle 11 \rangle$ means that the new requirement is accepted and the sub-controller can apply the new requirement by enabling the transition t_2 . This last transition represents the local reconfiguration

transformation. We associate to this transition an additional enabling condition which corresponds to eight possible local reconfiguration scenarios can be applied to FESTO. For instance, condition $(c = L1 \wedge g = L2)$ means that the sub-controller agent may be transformed from local behaviour $L1$ to $L2$.

Example 7 (EnAS). The RECATNet model of the sub-controller agent A_2 for EnAS, as shown in Fig.8, is similar to that of the sub-controller agent A_1 for FESTO, except that A_2 can perform three policies *Policy1*, *Policy2* and *Policy3* modelled by a closed terms $P1$, $P2$ and $P3$. The initial marking of this agent is $M_{02} = \langle sp_1, P1 \rangle$ that represents the initial local behaviour *Policy1*. Four different reconfiguration scenarios can be applied to EnAS represented by conditions associated to transition t_2 . The specification of EnAS includes one input place (*event*). It means that the sub-controller can receive in this place events produced by the jack station J1 or J2 or user requirements.

5 Verification of DRCS

For analysing the obtained RECATNet, we have expressed its semantic into rewriting logic [14], the input of the model-checker Maude. The checked properties have to be expressed using Linear Temporal Logic (LTL).

5.1 RECATNet semantics in terms of rewriting logic

Since we choose to express the RECATNet semantics in terms of rewriting logic, we define each RECATNet as a conditional theory like in [13], where transitions firing and cut step execution are formally expressed by labelled rewrites rules. Each extended marking Tr is expressed, in a recursive way, as a term $[MTh, tabs, ThreadChilds]$, where MTh represents the internal marking of Th , $tabs$ 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. Consequently, the extended marking Tr represented in Fig.7 is expressed as a general term of sort $Thread$ with the following recursive form: $[MTh_0, nullTrans, [MTh_1, tabs_1, [MTh_{11}, tabs_{11}, \dots] \dots [MTh_{1n}, tabs_{1n}, \dots] \dots [MTh_n, tabs_n, [MTh_{n1}, tabs_{n1}, \dots] \dots [MTh_{nn}, tabs_{nn}, \dots]]]]$ where MTh_0 is the marking of the root thread.

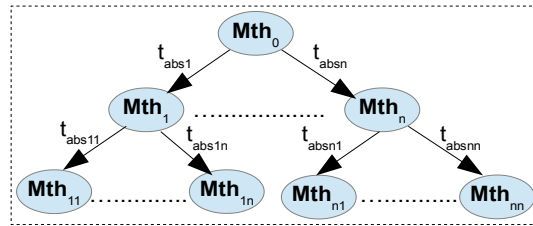


Fig. 7. General form of an extended marking

The structure of distributed states of RECATNets is formalized by the following equational theories.

```
fmod THREAD is
pr MARKING .
sorts Thread Trans TransAbs .
subsort TransAbs < Trans .
op nullTrans : -> TransAbs .
op nullThread : -> Thread .
op [_,_,_]:Marking TransAbs Thread ->Thread .
op _ _ :Thread Thread ->Thread [assoc comm id: nullThread] .
endfm
```

Where module *Marking* is a set of *Place – Marking* defined in rewriting logic as follows:

```
fmod MARKING is
pr PLACE-MARKING .
sort Marking .
subsort Place-marking < Marking .
op em : -> Marking . --- empty marking
op _(*)_ : Marking Marking -> Marking [assoc comm id: em] .
endfm
fmod PLACE-MARKING is
sorts Place Multiset Place-marking .
op ems : -> Multiset . --- empty Multiset
op _(+)_ : Multiset Multiset -> Multiset [assoc comm id: ems] .
op <_;> : Place Multiset -> Place-marking .
endfm
```

Each firing step in a RECATNet is expressed by a rewrite rule of the form $M \Rightarrow M'$ if $Cond$ which means that a fragment of the RECATNet state fitting pattern M can change to a new local state fitting pattern M' if the condition $Cond$ holds. We give the general form of the rewrite rules describing the behaviour of RECATNets firing steps:

– Rule associated to an elementary transition

```
crl[telt]:<p, mp (+) DT(p, telt)> (*) <p', mp'> =>
<p, mp> (*) <p', mp' (+) CT(p', telt)> if (InputCond and TC(telt)
and Nbr(mp' (+) CT(p', telt) < Cap(p')))
```

– Rule associated to an abstract transition

```
crl[tabs]:[M (*)<p, mp (+) DT(p, tabs)> , T, Th] =>
[M (*)<p, mp> , T, Th [<p'', CT(p'', tabs) >, tabs, nullThread]]
if (InputCond and TC(tabs)) .
```

– Rule associated to a cut step

```
crl[cut]:<Mf (*) <p', mp'> , T, mThf [M (*) <pfinal, mpfinal>,
tabs, mTh]> =>[Mf (*) <p', mp' (+) ICT(p', tabs, i), T, mThf>]
if (  $\mathcal{I}i$  and Nbr(mp' (+) ICT(p', tabs, i) < Cap(p')))
```

For instance, if we consider the RECATNet of Fig.6(a), the rewrite rule describing the elementary transition labelled t_0 is given as follows:

```
cr1[t0]:<sp1 ; c>(*)<sp0 ; g> => <sp1;c>(*)<sp2;g> if c /= g .
```

Another example, the rewrite rule describing the abstract transition labelled t_1 is given as follows:

```
r1[t1]:[<sp2 ; g>,T,Th] => [M,T,Th [<ctrl ;g>,t1,nullThread]] .
```

After defining the semantics of RECATNet in terms of rewriting rules, the model-checker Maude can be used.

5.2 Implementation using the Maude Tool

In this section, we plan to check the correct response of the whole system to the reconfiguration request and the valid behavior of sub-controller agents after the application of a distributed reconfiguration scenario.

Example 8. Let take the example shown in Fig.4. The initial distributed configuration is $d = (L1, P1)$, where $L1$ represents the local behavior *Lighth1* of FESTO and $P1$ represents the local behavior *Policy1* of EnAS. This distributed configuration is specified in rewriting logic as follows:

```
eq initDistState = < sp11 ; L1 >(*)< sp12 ; P1 > .
```

From Fig.4, when drill machine $Dr1$ is broken, the sub-controller agent of FESTO needs to change its local behaviour to $L2$. The new selected distributed configuration from *Decision Matrices* is $d = (L2, P2)$.The following equation, in term of rewriting logic, specifies the new distribute state of the whole system to be reached.

```
eq < sp11 ; L2 >(*)< sp12 ; P2 > |= newDistState = true .
```

In order to check the valid behavior of sub-controller agents, we call the model-checker of Maude as follows:

```
Model-Check (initDistState, <> newDistState) .
```

This means that starting from an initial distribute state, every possible execution path reaches the new distributed state. The symbol $\langle \rangle$ denotes the temporal operator *Eventually*. In Fig.8, verification shows that the specified formula is *true*.

```
> load RDCS/modelCheck.maude .
=====
reduce in RECATNET-CHECK : modelCheck(initialState, <> newDistState) .
rewrites: 2934 in 13503670060ms cpu (29ms real) (0 rewrites/second)
result Bool: true
Maude>
```

Fig. 8. Model Checking under Maude

Example9. Let take another example and we focus on the case, if a response command is received, whether the sub-controller FESTO can respond and select a proper behaviour. First, we define predicates:

```

op Medium-Mode Ligth1-Mode : -> Prop .
ops Drill-Down Reply : Multiset -> Prop .
eq <sp11 ; Me> (*) M |= Medium-Mode = true .
eq < sp11 ; L1>(*) M |= Ligth1-Mode = true .
eq <repl;accept>(*) M |= Reply(accept)=true .
eq <in1-event ; dr2-down> (*) M |= Drill-Down(dr2-down)=true .

```

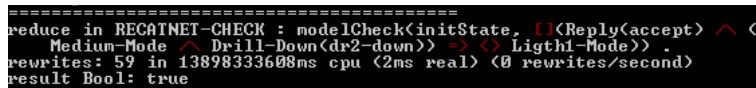
Now, let's consider the following LTL property:

```

[] ( Medium-Mode /\ Drill-Down(dr2-down) /\ Reply(accept)
    => <> Ligth1-Mode ).

```

This LTL formula means that, always, if FESTO is in *Medium* behaviour, drill machine *Dr2* is broken and command *accept* is received, FESTO will eventually select the *Ligth1* behaviour. Finally, we call Maude LTL model checker to check this property (see Fig.9).



```

=====
reduce in RECATNET-CHECK : modelCheck<initState, []<Reply(accept) /\ <
  Medium-Mode /\ Drill-Down(dr2-down)> -> <> Ligth1-Mode>> .
rewrites: 59 in 13898333608ms cpu <2ms real> <0 rewrites/second>
result Bool: true

```

Fig. 9. Model checking of the property

6 Conclusion and Future Works

This research work copes with the reconfiguration and coordination issues of DRCSs. A novel multi-agent architecture is proposed such that a device has a particular reconfigurable sub-controller agent to especially manage its local reconfiguration. The problem of concurrent reconfiguration requirements are solved by proposing a new communication protocol where *Decision Matrices* are applied by the sub-controller agents. The proposed approach is modeled by the formalism of RECATNets and checked by using the MAUDE model-checker. A virtual reconfigurable distributed production system is applied as a running example to illustrate the whole work.

In the future, we will plan to extend our work in order to model and analyse time-constrained DRCSs. In this case, we will use an extended version of our formalism called time-RECATNets (T-RECATNets)[13]. Therefore, one can verify some properties with respect to time constraints using Real-Time MAUDE model-checker[15].

References

1. Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Uiso, and H. V. Brussel, "Reconfigurable manufacturing systems," *CIRP Annals- Manufacturing Technology*, vol. 48, pp. 527–540, 1999.

2. M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.
3. M. Khalgui and H. Hanisch, "Automatic nces-based specification and sesa-based verification of feasible control components in benchmark production systems," *International Journal of Modeling, Identification and Control*, vol. 12, no. 3, pp. 223–243, 2011.
4. J. Zhang, M. Khalgui, Z. Li, O. Mosbahi, and A. Al-Ahmari, "R-tnces: A novel formalism for reconfigurable discrete event control systems," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, vol. 43, no. 4, pp. 757–772, 2013.
5. 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.
6. S. U. Guan and S.-S. Lim, "Modeling adaptable multimedia and self-modifying protocol execution," *Future Generation Comp. Syst.*, vol. 20, no. 1, pp. 123–143, 2004.
7. E. Badouel, M. Llorens, and J. Oliver, "Modeling concurrent systems: Reconfigurable nets," in *International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA*. CSREA Press, 2003, pp. 1568–1574.
8. H. M. Hanisch, J. Thieme, A. Luder, and O. Wienhold, "Modeling of plc behavior by means of timed net condition/event systems," in *Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97., 1997 6th International Conference on*, 1997, pp. 391–396.
9. M. Khalgui and O. Mosbahi, "Intelligent distributed control systems," *Inf. Softw. Technol.*, vol. 52, no. 12, pp. 1259–1271, 2010.
10. 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. Hofstede, B. Benatallah, and H.-Y. Paik, Eds., 2008, vol. 4928, pp. 232–244.
11. M. Clavel and al., "Maude manual (version 2.3)," 2007. [Online]. Available: <http://maude.cs.uiuc.edu>
12. S. Haddad and D. Poitrenaud, "Recursive petri nets: Theory and application to discrete event systems," *Acta Inf.*, vol. 44, no. 7, pp. 463–508, Nov. 2007.
13. K. Barkaoui, H. Boucheneb, and A. Hicheur, "Modelling and analysis of time-constrained flexible workflows with time recursive ecatnets," ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5387, pp. 19–36.
14. R. Bruni and J. Meseguer, "Semantic foundations for generalized rewrite theories," *Theor. Comput. Sci.*, vol. 360, no. 1, pp. 386–414, Aug. 2006.
15. 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.