

# Coloured Reconfigurable Nets for Code Mobility Modeling

Kahloul Laid, and Chaoui Allaoua

**Abstract**—Code mobility technologies attract more and more developers and consumers. Numerous domains are concerned, many platforms are developed and interest applications are realized. However, developing good software products requires modeling, analyzing and proving steps. The choice of models and modeling languages is so critical on these steps. Formal tools are powerful in analyzing and proving steps. However, poorness of classical modeling language to model mobility requires proposition of new models. The objective of this paper is to provide a specific formalism “*Coloured Reconfigurable Nets*” and to show how this one seems to be adequate to model different kinds of code mobility.

**Keywords**—Code mobility, modeling mobility, labeled reconfigurable nets, Coloured reconfigurable nets, mobile code design paradigms.

## I. INTRODUCTION

NOWADAYS, code mobility is one of the attracting fields for computer science researchers. Code mobility technology seems an interest solution for distributed applications facing bandwidth problems, users' mobility, and fault tolerance requirement. Numerous platforms were been developed [18]. Such platforms allow the broadcasting of this technology in many domains (information retrieving [10], e-commerce [12], network management [23], ...). Software engineering researches have provided some interest design paradigms influencing the development of the field. The most recognized paradigms [7] are: code on demand, remote evaluation, and mobile agent. To avoid ad-hoc development for code mobility software, many works attempt to propose methodologies and approaches ([17], [22], [15], ...). Indeed, these approaches are mostly informal. They lack in analyzing and proving system proprieties. Enhancing development process with formal tools was an attractive field in code mobility researches.

Traditional formal tools which were massively used to model and analyze classical systems seem to be poor to deal with inherent proprieties in code mobility systems. Works on formal tools attempt to extended classical tools to deal with code mobility proprieties. The most important proposition can

be found in process algebra based model and state transition model. For the first one,  $\pi$ -calculus [14] is the famous one, and for the second, high-level Petri net (with many kinds) can be considered the good representative.  $\pi$ -calculus is an extension for CCS (communicating concurrent systems) [13]. CCS allows modeling a system composed of a set of communicating process. This communication uses names (gates) to insure synchronization between processes. In  $\pi$ -calculus information can be exchanged through gates. The key idea is that this information can be also a gate. With this idea, process can exchange gates. Once these gates received, they can be used by the receiver to communicate. In an extension of  $\pi$ -calculus,  $HO\pi$ -calculus [16], processes can exchange other processes through gates (the exchanged processes called agents).

To model mobility with Petri nets, high level PNETs were proposed. The most famous are Mobile Nets (variant of coloured Petri nets) [1] and Dynamic Petri nets. In mobile Petri nets, names of places can appear as tokens inside other places. Dynamic Petri nets extend mobile Petri nets. In this last one, firing a transition can cause the creation of a new subnet. With high-level Petri nets, mobility in a system is modeled through the dynamic structure of the net. A process appearing in a new environment is modeled through a new subnet created in the former net by firing a transition. Many extensions have been proposed to adapt mobile Petri net to specific mobile systems: Elementary Object Nets [19], reconfigurable nets [3], Nested Petri Nets [11], HyperPetriNets [2], ... With respect to [21], all these formalisms lack in security aspect specification. To handle this aspect in code mobility, recently Mobile Synchronous Petri Net (based on labeled coloured Petri net) are proposed [20].

The objective of this work is to present a new formalism based on Petri nets. Our formalism “Coloured reconfigurable nets” as an extension for our work “Labeled Reconfigurable Nets” [8]. We attempt to propose to model mobility in an intuitive and an explicit way. Mobility of code (a process or an agent) will be directly modeled through reconfiguration of the net. We allow adding and deleting of places, arcs, and transitions at run time. In this formalism, we introduce two kinds of specific transitions: calculi transitions and reconfigure transitions. A calculi transition takes as input a set of tokens (of type nets), it computes a set of places and arcs, and it outputs a set of tokens (of types: nets, places and arcs). The objective of this kind of transition is to prepare the reconfiguration of the net (migration of a net). A reconfigure





Manuscript received September 14, 2007.

Kahloul Laid is with Computer Science Department, Biskra University, Algeria (phone: 213 75 17 90 20; fax: +213 33 74 31 61; e-mail: kahloul2006@yahoo.fr).

Chaoui Allaoua is with LIRE laboratory, Constantine University, Algeria (phone: +213 5 82 87 64; e-mail: a\_chaoui2001@yahoo.com).

transition takes as input tokens (of types: nets, places and arcs), it reconfigure the net by moving some subnets, places and arcs from one net towards another net. We propose that these two kinds of transition allow modeling mobility in an explicit and more sophisticated manner. In this model we consider that nets, places or arcs can play as tokens. These tokens move from one place to another when some transitions are fired.

## II. DEFINITION OF COLOURED RECONFIGURABLE NETS (CRN)

Coloured reconfigurable nets are an extension of labeled reconfigurable nets. Informally, a coloured reconfigurable net is a set of *environments* (blocs of *units*). Connections between these environments and their contents can be modified during runtime. A *unit* is a specific Petri net. A unit can contain three kinds of transitions (a unique *start* transition: , a set of ordinary transitions: , a set of calculi transition:  and a set of reconfigure transitions: ).

Preconditions and post-conditions to fire a start or an ordinary transition are the same that in Petri nets. When a *reconfigure transition* is fired, a net  $N$  will be (re)moved from an environment  $E$  towards another environment  $E'$ . The net  $N$ , the environment  $E$  and  $E'$  are defined by a *calculi transition* which must always precedes this one. After firing a *reconfigure transition*, the structure of the coloured reconfigurable net will be updated (i.e some places, arcs, and transitions will be deleted or added). Here after we give our formal definitions of the concepts: unit, environment and coloured reconfigurable net. After the definition, we present the dynamic aspect of this model.

To define coloured reconfigurable nets, we introduce firstly the definition of units and environment.

### Definition 1 (Unit) :

A unit is a net  $U=(\Sigma, P, T, A, C, E)$ .

$\Sigma$  : a finite set of types (colors); we denote by *expr* the set of expression that can be written using variables in sets of  $\Sigma$ .

$P$ : a finite set of places;

$T$ : a finite set of transitions. We have  $T=T \cup C \cup R$ . Where

$T$ : a set of ordinary transitions,  $T=\{t_1, \dots, t_n\}$ . This set must contain a unique transition that we call a *start* transition. We denote this transition as *strt*,

$C$ : a set of calculi transitions,  $C=\{c_1, \dots, c_m\}$ ,

$R$ : a set of reconfigure transitions,  $R=\{r_1, \dots, r_p\}$ .

$A$ : a set of arcs

$C$ : a color mapping from  $P$  to  $\Sigma$ .  $C$  joins to each place  $p$  a color  $c$  that we note  $C(p)$ .

$E$ : an expression mapping from  $A$  to *expr*.

### Definition 2 (Environment):

An environment  $E$  is a quadruplet  $E=(GP, RP, U, A)$ .

- $GP = \{gp_1, gp_2, \dots, gp_s\}$  a finite set of specific places : "guest places";

- $RP = \{rp_1, rp_2, \dots, rp_s\}$  a finite set of specific places : "resource places";

- $U = \{N_1, N_2, \dots, N_k\}$  a set of nets. where  $T_1, T_2, \dots, T_k$  are the sets of their transitions and  $StrT = \{strt^1, strt^2, \dots, strt^k\}$  is the set of their start transitions.

- $A$  : a set of arcs,  $A \subseteq GP \times StrT \cup RP \times T$ . Such that:  
 $T = T_1 \cup T_2 \cup \dots \cup T_k$

*Remark:* we say that a unit  $U$  is in an environment  $E$  iff the net  $U$  is a subnet of the net  $E$ .

### Definition 3 (Coloured reconfigurable nets):

A coloured reconfigurable nets (CRN) is couple  $N=(E, A)$ , such that:

$E$ : a finite set of environments;

$A$ : a finite set (probably empty) of arcs; these arcs connect places (resp. transitions) from one environment to other transitions (resp. places) in another environment.

### Dynamic of coloured reconfigurable nets:

To introduce the dynamic of CRN we consider three types (colors):  $P$ (set of places),  $N$ (set of nets), and  $B$ (set of arcs). We denote respectively by  $P^*, N^*, B^*$  the three multisets of types  $P, N, B$ . We focus on the semantic of *calculi* and *reconfigure* transition.

#### Semantic of calculi transition:

A calculi transition must take as input three tokens of type  $N$ (two environments and one unit, the unit must be in one and only one of the two environments). Firing the calculi transition provides a token in the multi-sets  $\langle N^*, P^*, B^* \rangle$ . We can say that a calculi transition uses a set of nets to computes some arcs and places. At the output, it provides a composite token of the input nets and the computed arcs and places. In general, this token is used by a reconfigure transition.

If  $t$  is a calculi transition, and  $E_1, E_2, U$  are the input nets ( $U$  is in  $E_1$ ), once  $t$  is fired it produce a token  $\langle U+E_1+E_2, P, A \rangle$  such that  $P$  and  $A$  are two multi-sets that can be defined like this:  $P = \{p \in P_{E_1} / p \notin P_U \text{ and } \exists t \in T_U \text{ such that } (p,t) \in A_{E_1} \text{ or } (t,p) \in A_{E_1}\}$ , and

$$A = \{a \in A_{E_1} / a \notin A_{E_1} \text{ and } \exists (t,p) T_{E_1} \times T_U \cup T_U \times T_{E_1}\}.$$

Where  $P_N, A_N$  and  $T_N$  denote respectively places, arcs and transitions of a net  $N$ .

#### Semantic of reconfigure transition:

The objective of a reconfigure transition is to reconfigure the structure of the net. To be fired, a reconfigure transition takes as input a token in the multi-sets :  $\langle N^*, N, P^*, B^* \rangle$ . Firing a reconfigure transition will update the structure of the coloured reconfigurable nets that contains this transition in the following semantic:

If  $rt$  is a reconfigure transition and  $\langle U+E_1, E_2, P, A \rangle$  is an input token, to fire  $rt$  we impose that there exists a *free* place  $p_g$  in  $GP_{E_2}$ ; which means: for each  $t \in strT_{E_2}, (p_g, t) \notin A_{E_2}$ .

Once this condition is satisfied, firing  $rt$  changes  $N$  structurally such that:

If  $E_1$  and  $E_2$  denote the same environment then  $N$  will be not changed;  
 Else:

- 1) The net  $U$  is removed from the net  $E_1$ :  
 $U_{E2} \leftarrow U_{E2} \cup \{U\}$ ;
- 2) The net  $U$  is added to the environment  $E_2$ :  
 $U_{E1} \leftarrow U_{E1} / \{U\}$ ;
- 3)  $A_{E2} \leftarrow A_{E2} \cup (p_g, strt)$ ; such that  $strt$  is the start transition for  $U$ .
- 4) Some elements of  $P$  are transformed from  $E_1$  towards  $E_2$ , some other are cloned and some other will not be changed (resp for elements in  $A$ ). These elements depend on the modeling case. In section 3, we show how these elements can be defined depending on the mobile code design paradigm to model.

### III. MODELING CODE MOBILITY WITH CRN

A mobile code system is composed of execution units (EUs), resources, and computational environments (CEs). EUs will be modeled as units and computational environments as environments. Modeling resources requires using a set of places.

Reconfigure transitions model mobility actions. The key in modeling mobility is to identify the unit to be moved, the target computational environment and the types of binding to resources and their locations. This information is supposed to be known before mobility. We use calculi transition as computing actions that compute this information. After computing these elements, the reconfigure transition updates the net by moving a unit from one environment to another. This moving must respect requirement for bindings to resources to insure the reliability of components on their new locations. Information concerning units, environments and bindings will be defined according to the resources types and to the three design paradigms: remote (REV) evaluation, code on demand (COD), and mobile agent (MA).

#### A. Remote Evaluation

In remote evaluation paradigm, an execution unit  $EU_1$  sends another execution unit  $EU_2$  from a computational environment  $CE_1$  to another one  $CE_2$ .

**Example 4.1:** Let us consider two computational environments  $E_1$  and  $E_2$ . Firstly,  $E_1$  contains two execution units  $EU_1$  and  $EU_2$ ;  $E_2$  contains an execution unit  $EU_3$ . The three execution units execute infinite loops.  $EU_1$  executes actions  $\{a_{11}, a_{12}\}$ ,  $EU_2$  executes actions  $\{a_{21}, a_{22}, a_{23}\}$ , and  $EU_3$  executes actions  $\{a_{31}, a_{32}\}$ .  $a_{21}$  requires a transferable resource  $TR_1$  and a non-transferable resource bound by type  $PNR_1$  which is shared with  $a_{11}$ .  $a_{22}$  and  $a_{12}$  share a transferable resource bound by value  $VTR_1$ , and  $a_{23}$  requires a non-transferable resource  $NR_1$ . In  $E_2$ ,  $EU_1$  requires a non-

transferable resource bound by type  $PNR_2$  to execute  $a_{31}$ .  $PNR_2$  has the same type of  $PNR_1$ .

The system will be modeled as a coloured reconfigurable net  $N$ .  $N$  contains two environments  $E_1, E_2$  that model the two computational environments ( $CE_1$  and  $CE_2$ ). Units  $EU_1$  and  $EU_2$  will model execution units  $EU_1$  and  $EU_2$ , respectively. In this case, the unit  $EU_1$  will contain a *reconfigure transition*  $rt$  and a calculi transition  $ct$ .

1.  $E_1 = (RP_1, GP_1, U_1, A_1)$ ;  $RP_1 = \{TR_1, PNR_1, VTR_1, NR_1\}$ .  $U_1 = \{EU_1, EU_2\}$ ;
2.  $E_2 = (RP_2, GP_2, U_2, A_2)$ ;  $RP_2 = \{PNR_2\}$ .  $GP_2 = \{PEU_1\}$ .
3.  $ct$  will take as input tokens :  $E_1, EU_2$  and  $E_2$ .  $ct$  will provide the token :  $\langle EU_2 + E_1, E_2, P, A \rangle$ . such that  $P = TR_1 + VTR_1$   
 $A = (NR_1, a_{23}) + (PNR_2, a_{21})$
4.  $rt$  takes as input  $\langle EU_2 + E_1 + E_2, P, A \rangle$  and will remove  $EU_2$  and places in  $P$  from  $E_1$  towards  $E_2$ . Arcs in  $A$  will be added to the  $N$ .

In the Fig.1, the types of places  $P^1, P^2, P^3$  is  $N$ (set of nets).  $P^1$  contains  $EU_1$ ,  $P^2$  contains  $E_2$  and  $P^3$  contains  $E_2$ . The type of place  $P_{11}$  is  $\langle N^*, P^*, B^* \rangle$ .

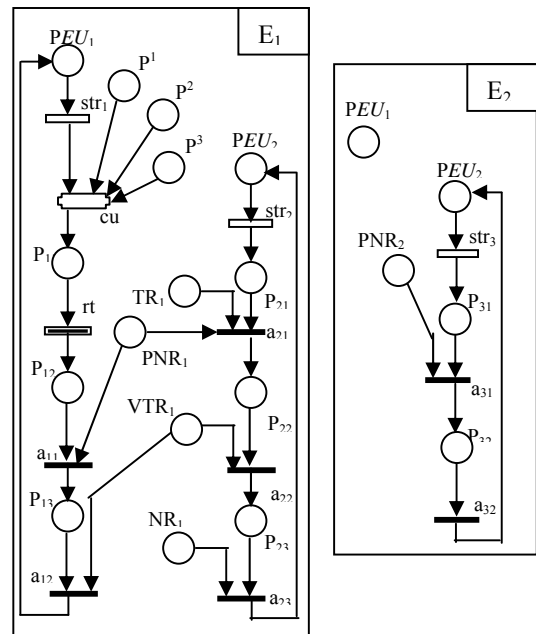


Fig. 1 REV-model before firing  $rt$

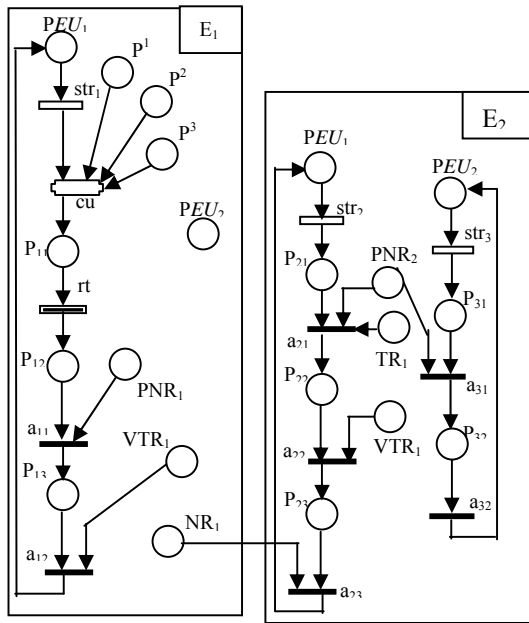


Fig. 2 REV-model after firing *rt*

**B. Code on Demand**

In code-on-demand paradigm, an execution unit  $EU_1$  fetches another execution unit  $EU_2$ . The reconfigure transition  $rt$  is contained in the unit modeling  $EU_1$ . If we reconsider the above example, the unit  $EU_1$  will contain a reconfigure transition  $rt$ . Fig. 3 shows the model proposed to model this system. In the Fig. 1, the types of places  $P^1, P^2, P^3$  is  $N$ (set of nets).  $P^1$  contains  $EU_1$ ,  $P^2$  contains  $E_2$  and  $P^3$  contains  $EU_2$ . The type of place  $P_{11}$  is  $\langle N^*, P^*, B^* \rangle$ . The transition  $cu$  will provide  $\langle EU_2 + E_2, E_1, P, A \rangle$ . Where  $P$  and  $A$  are the same as in the above example.

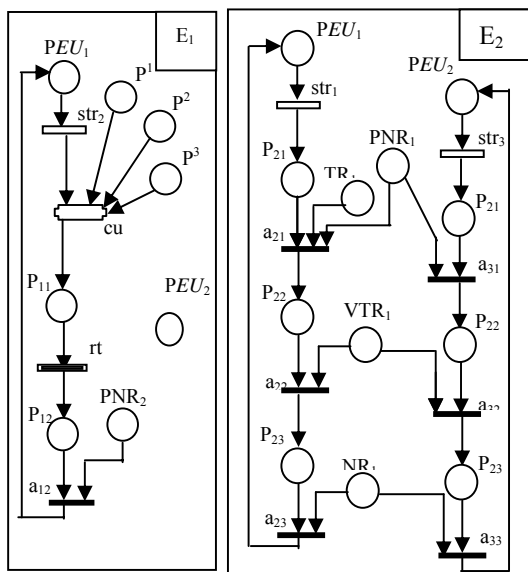


Fig. 3 COD-model before firing *rt*

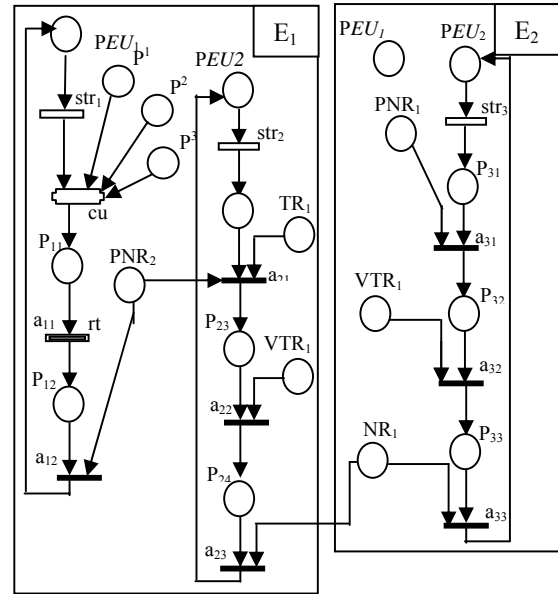


Fig. 4 COD-model after firing *rt*

**C. Mobile Agent**

In mobile agent paradigm, execution units are autonomous agents. The agent itself triggers mobility. In this case,  $rt$  –the reconfigure transition- is contained in the unit modeling the agent.

**Example 4.2:** let  $E_1$  and  $E_2$  two computational environments.  $E_1$  contains two agents, a mobile agent MA and a static agent  $SA_1$ ;  $E_2$  contains a unique static agent  $SA_2$ . The three agents execute infinite loops. MA executes actions  $\{a_{11}, a_{12}, a_{13}\}$ ,  $SA_1$  executes actions  $\{a_{21}, a_{22}, a_{23}\}$ , and  $SA_2$  executes actions  $\{a_{33}, a_{32}\}$ . To be executed,  $a_{11}$  require a transferable resource  $TR_1$  and a non-transferable resource bound by type  $PNR_1$  witch is shared with  $a_{21}$ .  $a_{12}$  and  $a_{22}$  share a transferable resource bound by value, and  $a_{13}$  and  $a_{23}$  share a non-transferable resource  $NR_1$ . In  $E_1$ ,  $SA_2$  requires a non-transferable resource bound by type  $PNR_2$  to execute  $a_{32}$ .  $PNR_2$  has the same type of  $PNR_1$ .

The system will be modeled as a coloured reconfigurable net  $N$ .  $N$  contains two environments  $E_1, E_2$  that model the two computational environments ( $CE_1$  and  $CE_2$ ). Units  $A_1, A_2$  and  $A_3$  will model MA,  $SA_1$  and  $SA_2$ , respectively. In this case, the unit  $A_1$  will contain a reconfigure transition  $rt$  and a calculi transition  $cu$ .

1.  $E_1 = (RP_1, GP_1, U_1, A_1)$ ;  $RP_1 = \{TR_1, PNR_1, VTR_1, NR_1\}$ .  $U_1 = \{EU_1, EU_2\}$ ;
2.  $E_2 = (RP_2, GP_2, U_2, A_2)$ ;  $RP_2 = \{PNR_2\}$ .  $GP_2 = \{PEU_1\}$ .
3.  $cu$  will take as input tokens :  $E_1, A_1$  and  $E_2$ .  $cu$  will provide the token :  $\langle A_1 + E_1, E_2, P, A \rangle$ . such that  

$$P = TR_1 + VTR_1$$

$$A = (NR_1, a_{23}) + (PNR_2, a_{21})$$
4.  $rt$  takes as input  $\langle A_1 + E_1, E_2, P, A \rangle$  and will remove  $A_1$  and places in  $P$  from  $E_1$  towards  $E_2$ . Arcs in  $A$  will be added to the  $N$ .

In the Fig. 1, the types of places  $P^1, P^2, P^3$  is  $N(\text{set of nets})$ .  $P^1$  contains  $A_1$ ,  $P^2$  contains  $E_2$  and  $P^3$  contains  $E_2$ . The type of place  $P_{11}$  is  $\langle N^*, N, P^*, B^* \rangle$ . Fig. 5 shows the model of this system.

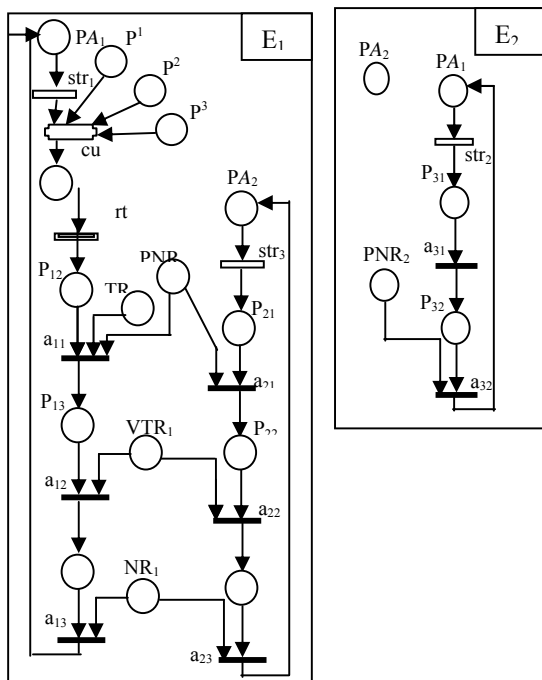


Fig. 5 MA-model before firing  $rt$

The Fig. 6 shows the configuration after firing  $rt$ .

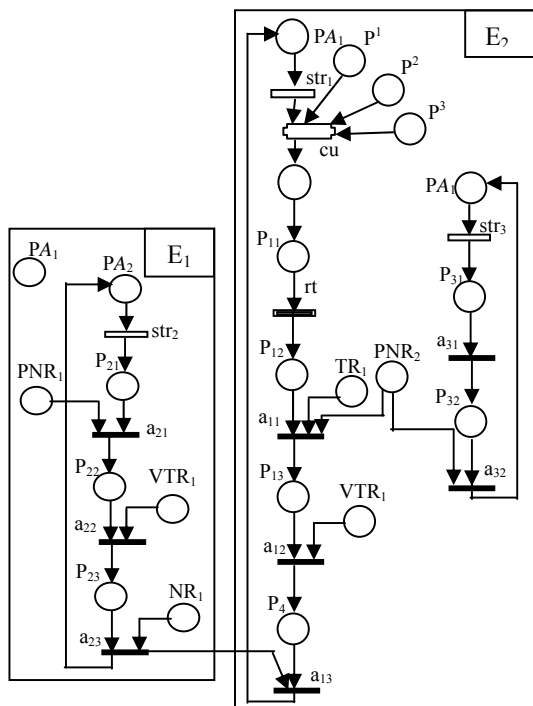


Fig. 6 MA-model after firing  $rt$

#### IV. RELATED WORKS

In [4], the authors proposed PrN (Predicate/Transition nets) to model mobility. They use concepts: agent space witch is composed of a mobility environment and a set of connector nets that bind mobile agents to mobility environment. Agents are modeled through tokens. So these agents are transferred by transition firing from a mobility environment to another. The structure of the net is not changed and mobility is modeled implicitly through the dynamic of the net. In [20], authors proposed MSPN (Mobile synchronous Petri net) as formalism to model mobile systems and security aspects. They introduced notions of nets (an entity) and disjoint locations to explicit mobility. A system is composed of set of localities that can contain nets. To explicit mobility, specific transitions (called autonomous) are introduced. Two kinds of autonomous transition were proposed: *new* and *go*. Firing a *go* transition move the net form its locality towards another locality. The destination locality is given through a token in an input place of the *go* transition. Mobile Petri nets (MPN) [1] extended colored Petri nets to model mobility. MPN is based on  $\pi$ -calculus and join calculus. Mobility is modeled implicitly, by considering names of places as tokens. A transition can consumes some names (places) and produce other names. The idea is inherited from  $\pi$ -calculus where names (gates) are exchanged between communicating process. MPN are extended to Dynamic Petri Net (DPN) [1]. In DPN, mobility is modeled explicitly, by adding subnets when transitions are fired. In their presentation [1], no explicit graphic representation has been exposed.

In nest nets [9], tokens can be Petri nets them selves. This model allows some transition when they are fired to create new nets in the output places. Nest nets can be viewed as hierarchic nets where we have different levels of details. Places can contain nets that their places can also contain other nets et cetera. So all nets created when a transition is fired are contained in a place. So the created nets are not in the same level with the first net. This formalism is proposed to adaptive workflow systems.

In [3], authors studied equivalence between the join calculus [6] (a simple version of  $\pi$ -calculus) and different kinds of high level nets. They used "reconfigurable net" concept with a different semantic from the formalism presented in this work. In reconfigurable nets, the structure of the net is not explicitly changed. No places or transitions are added in runtime. The key difference with colored Petri nets is that firing transition can change names of output places. Names of places can figure as weight of output arcs. This formalism is proposed to model nets with fixed components but where connectivity can be changed over time.

In this work, we attempt to provide a formal and graphical model for code mobility. We have extended Petri net with reconfigure transitions that when they are fired reconfigure the net. Mobility is modeled explicitly by the possibility of adding or deleting at runtime arcs, transitions and places. Modification in *reconfigure transition's* label allows modeling

different kinds of code mobility. Bindings to resources can be modeled by adding arcs between environments. It is clear that in this model created nets are in the same level of nets that create them. Creator and created nets can communicate. This model is more adequate for modeling mobile code systems. Instead of using label associated to reconfigure transition in "Labeled Reconfigurable Nets"[8], which gives information about mobility, we have introduced colors (types) and a calculi transition that must compute this information. Rigidity due to labels is now avoided in this formalism. This last advantage will make modeling easier.

## V. CONCLUSION

Proposed initially to model concurrency and distributed systems, Petri nets attract researchers in mobility modeling domain. The ordinary formalism is so simple with a smart formal background, but it fails in modeling mobility aspects. Many extensions were proposed to treat mobility aspects. The key idea was to introduce mechanisms that allow reconfiguration of the model during runtime. The most works extend coloured Petri nets and borrow  $\pi$ -calculus or join calculus ideas to model mobility. The exchanging of names between processes in  $\pi$ -calculus is interpreted as exchanging of place's names when some transitions are fired. This can model dynamic communication channels. In much formalism, mobility of process is modeled by a net playing as token that moves when a transition is fired. All these mechanisms allow modeling mobility in an implicit way. We consider that the most adequate formalisms must model mobility explicitly. If a process is modeled as a subnet, mobility of this process must be modeled as a reconfiguration in the net that represents the environment of this process.

In this paper, we have presented a new formalism "Coloured reconfigurable nets". This formalism allows explicit modeling of computational environments and processes mobility between them. We have presented how this formalism allows, in a simple and an intuitive approach, modeling mobile code paradigms. We have focused on bindings to resources and how they will be updated after mobility. We believe that the present formalism is an adequate model for all kinds of code mobility systems. In our future works we plan to focus on modeling and analyzing aspects. In modeling aspects, we are interested to handle problems such that modeling multi-hops mobility, process's states during travel, birth places and locations. On the analysis aspect, we are thinking about an encoding of our model in maude or mobile maude [5] in order to analyze automation of our models.

## REFERENCES

[1] Andrea Asperti and Nadia Busi. "Mobile Petri Nets". Technical Report UBLCS-96-10, Department of Computer Science University of Bologna, May 1996.  
[2] M.A. Bednarczyk, L. Bernardinello, W. Pawlowski, and L. Pomello. "Modelling Mobility with Petri Hypernets". 17th Int. Conf. on Recent Trends in Algebraic Development Techniques, WADT'04. LNCS vol. 3423, Springer-Verlag, 2004.

[3] M. Buscemi and V. Sassone. "High-Level Petri Nets as Type Theories in the Join Calculus". In Proc. of Foundations of Software Science and Computation Structure (FoSSaCS '01), LNCS 2030, Springer-Verlag.  
[4] Dianxiang Xu and Yi Deng. "Modeling Mobile Agent Systems with High Level Petri Nets". 0-7803-6583-6/00/ © 2000 IEEE.  
[5] Francisco Durán, Steven Eker, Patrick Lincoln and José Meseguer. "principles of mobile maude". In D.Kotz and F.Mattern, editors, Agent systems, mobile agents and applications, second international symposium on agent systems and applications and fourth international symposium on mobile agents, ASA/MA 2000 LNCS 1882, Springer Verlag, Sept 2000.  
[6] Cédric Fournet Georges Gonthier, "The Join Calculus: a Language for Distributed Mobile Programming". In Applied Semantics. International Summer School, APPSEM 2000, Caminha, Portugal, September 2000, LNCS 2395, pages 268--332, Springer-Verlag. August 2002.  
[7] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna, "Understanding Code Mobility". IEEE transactions on software engineering, vol. 24, no. 5, may 1998.  
[8] Kahloul Laid and Chaoui Allaoua, "Labeled reconfigurable nets for modeling code mobility", accepted and to appear in the proceeding of The International Arab Conference for Information technology (ACIT) 26-28/11/2007 in Syria.  
[9] Kees M. van Hee, Irina A. Lomazova, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, Marc Voorhoeve: "Nested Nets for Adaptive Systems". 14 EE. ICATPN 2006: 241-260.  
[10] P. Knudsen, "Comparing Two Distributed Computing Paradigms, A Performance Case Study"; MS thesis, Univ. of Tromsø 1995.  
[11] I.A. Lomazova. "Nested Petri Nets"; Multi-level and Recursive Systems. Fundamenta Informaticae vol.47, pp.283-293. IOS Press, 2002.  
[12] M. Merz and W. Lamersdorf, "Agents, Services, and Electronic Markets: How Do They Integrate?"; Proc. Int'l Conf. Distributed Platforms, IFIP/IEEE, 1996.  
[13] R. Milner. "A Calculus of Communicating Systems". Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.  
[14] R. Milner, J. Parrow, and D. Walker. "A calculus of mobile processes". Information and Computation, 100:1-77, 1992.  
[15] Reinhartz-Berger, I., Dori, D. and Katz, S. (2005) "Modelling code mobility and migration: an OPM/Web approach", Int. J. Web Engineering and Technology, Vol. 2, No. 1, pp.6-28.  
[16] D. Sangiorgi and D. Walker. "The  $\pi$ -Calculus: A Theory of Mobile Processes". Cambridge University Press, 2001.  
[17] Athie L. Self and Scott A. DeLoach. "Designing and Specifying Mobility within the Multiagent Systems Engineering methodology" Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at the 18th ACM Symposium on Applied Computing (SAC 2003). Melbourne, Florida, USA, 2003.  
[18] Tommy Thom, "Programming languages for mobile code". Rapport de recherche INRIA, N° 3134, Mars, 1997.  
[19] R. Valk. "Petri Nets as Token Objects: An Introduction to Elementary Object Nets". Applications and Theory of Petri Nets 1998, LNCS vol.1420, pp.1-25, Springer-Verlag, 1998.  
[20] F. Rosa Velardo, O. Marroqñ Alonso and D. Frutos Escrig. "Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems". In 1st Int. Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, MTCoord'05. ENTCS, No 150.  
[21] Fernando Rosa-Velardo. "Coding Mobile Synchronizing Petri Nets into Rewriting Logic", this paper is electronically published in Electronic Notes in Theoretical Computer science URL: www.elsevier.nl/locate/entcs.  
[22] Sutandiyó, W., Chhetri, M. B., Loke, S.W., and Krishnaswamy, S. "mGaia: Extending the Gaia Methodology to Model Mobile Agent Systems", Accepted for publication as a poster in the Sixth International Conference on Enterprise Information Systems (ICEIS 2004), Porto, Portugal, April 14-17.  
[23] D.J. Wetherall, J. Guttag, and D.L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols" Technical Report, MIT, 1997, in Proc. OPENARCH'98.