

# LRN/R-MAUDE BASED APPROACH FOR MODELING AND SIMULATION OF MOBILE CODE SYSTEMS

Laid Kahloul<sup>1</sup> and Allaoua Chaoui<sup>2</sup>

<sup>1</sup>Computer Science Department, Biskra University, Algeria  
kahloul2006@yahoo.fr

<sup>2</sup>Computer Science Department, Constantine University, Algeria  
a\_chaoui2001@yahoo.com

## 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 formal approach based on LRN and R-Maude. LRN (Labeled Reconfigurable Nets) is a specific formalism that we propose to model different kinds of code mobility. R-Maude (Reconfigurable Maude) is a system that we develop to encode and simulate LRN-models.

Keywords: code mobility, modeling mobility, LRN, R-Maude.

## 1 INTRODUCTION

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 [17]. Such platforms allow the broadcasting of this technology in many domains (information retrieving [9], e-commerce [11], network management [22], ...). 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 ([16], [21], [14], ...). 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 witch 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 [13] 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) [12]. 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 [15], 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 [18], reconfigurable nets [3], Nested Petri Nets [10], HyperPetriNets [2], ... With respect to [20], 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 [19].

The objective of this work is to treat to aspects of code mobility: modeling and simulation. We try to propose a formal approach in witch we define two formalisms : Labeled Reconfigurable Nets (LRN) and Reconfigurable Maude (R-Maude). Firstly, LRN will be used to model the system then R-Maude will encode and simulate this model. Our formalism “labeled reconfigurable nets” with a different semantic from the one presented in [3] is dedicated to model code mobility systems. 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. R-Maude is an extension for Maude that we propose and prototype in order to encode and simulate LRN models.

The rest of this paper is organized as follows. Section 2 starts by presenting the definition of the formalism LRN. In section 3 we show how LRN can be used to model the three mobile code paradigms: “remote evaluation”, “code on demand”, and “mobile agent”. Section 4 presents the idea and foundation of R-Maude, and section 5 discusses the prototype and shows an example. In section 6, we present some related works. We conclude this work and give some perspectives, in section 7.

## 2 LABELED RECONFIGURABLE NETS

Labeled reconfigurable nets are an extension of Petri nets. Informally, a labeled 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:  $\square$ , a set of ordinary transitions:  $\text{---}$ , and a set of reconfigure transitions:  $\text{▬}$ ).

Preconditions and post-conditions to fire a start

nets. Reconfigure transitions are labeled with labels that influence their firing. 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 in the label associated to the transition. After firing a

reconfigure transition, the structure of the labeled 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:

Formal Definition:

Let  $N_1, N_2, \dots, N_k$  be a set of nets.

for each  $i: 1, \dots, n: N_i = (P_i, T_i, A_i)$ , such that :

1.  $P_i = \{p_1^i, p_2^i, \dots, p_n^i\}$  a finite set of places,
2.  $T_i = ST_i \cup RT_i$ 
  - $ST_i = \{st_1^i, st_2^i, \dots, st_r^i\}$  a finite set of standard (ordinary) transitions,
  - $RT_i = \{rt_1^i, rt_2^i, \dots, rt_r^i\}$  a finite (eventually empty) of “reconfigure transitions”,
3.  $A_i \subseteq P_i \times T_i \cup T_i \times P_i$ .

Definition 1 (Unit): a unit  $UN$  is a net  $N_i$  that has a specific transition  $st_j^i$  denoted  $start^i$ . So

$T_i = \{start^i\} \cup ST_i \cup RT_i$ .

Définition 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.
- $A \subseteq GP \times StrT \cup RP \times T$ . Such that :  $StrT = \{start^1, start^2, \dots, start^k\}$  and  $T = ST_1 \cup RT_1 \cup ST_2 \cup RT_2 \cup \dots \cup ST_k \cup RT_k$

Definition 3 (Labeled reconfigurable net):

A labeled reconfigurable net LRN is a set of environments.  $LRN = \{E_1, E_2, \dots, E_p\}$  such that

- There exist at least one net  $N_i$  in LRN such that  $RT_i \neq \emptyset$ ;
- For each  $rt_j^i \in RT_i$ ,  $rt_j^i$  has a label  $\langle N, E_e, E_g, \psi, \beta \rangle$ , such that  $N$  is a unit,  $E_e$  and  $E_g$  are environments,  $\psi$  a set of places,  $\beta$  a set of arcs.

Dynamic of labeled reconfigurable nets:

Let  $LRN = \{E_1, E_2, \dots, E_p\}$  be a labeled reconfigurable net,

Let  $E_i = (GP^i, RP^i, U^i, A^i)$  be an environment in LRN,

- $GP^i = \{gp_1^i, \dots, gp_s^i\}$ ;
- $RP^i = \{rp_1^i, \dots, rp_s^i\}$ ;
- $U^i = \{N_1^i, N_2^i, \dots, N_k^i\}$ ;

where:

$Sarts^i = \{start^1, start^2, \dots, start^k\}$  and  $T^i = \{ST^i$

Let  $RT_j^i = \{ST_1, ST_2, \dots, ST_k\} \cup \{RT_1, RT_2, \dots, RT_k\}$

$RT_j^i$  be the non empty set of reconfigure transitions associated with the net  $N_j^i$ .

Let  $rt_j^i = \{rt_1^j, rt_2^j, \dots, rt_r^j\}$ .

$RT^i = \langle N, E_e, E_g, \psi, \beta \rangle$  be a reconfigure transition in unit, environment and labeled reconfigurable

net. After the definition, we present the dynamic aspect of this model.

$j$ , such that :

- $E_e=(GP^e, RP^e, U^e, A^e)$ ;
- $N=(P, T, A)$  and  $N \in U^e$ ;
- $E_g=(GP^g, RP^g, U^g, A^g)$ ;

- $\psi \subseteq \text{RP}^e$ ;  $\psi = \psi_r \cup \psi_c$ . ( $\psi_r$  denotes removed places and  $\psi_c$  denotes cloned places).
- $\beta$  is a set of arcs.  $\beta \subseteq \text{RP}^e \times \text{TURP}^e \times \text{T}$ .

Let  $\text{str}_t$  be the start transition of  $N$ .

Conditions to fire  $\text{rt}_{m \in N, E_e, E_g, \psi, \beta}^j$ :

In addition to the known conditions, we impose that there exists a free place  $p_g$  in  $\text{GP}^e$ ; witch means: for each  $t \in \text{starts}^e$ ,  $(p_g, t) \in A^e$ .

After firing  $\text{rt}_m^j$ :

In addition to the known post-condition of a transition firing, we add the following post-condition:

LRN will be structurally changed such that:

If  $E_e$  and  $E_g$  denote the same environment then

LRN will be not changed;

Else:

- 1)  $U^e \dot{\Delta} U^e \cup \{N\}$ ;  $U^e \dot{\Delta} U^e / \{N\}$ ;
- 2)  $A^e \dot{\Delta} A^e \cup (p_g, \text{str}_t)$ ;
- 3) Let  $DA = \{(a, b) \in A^e / (a \in \psi \text{ and } b \in \psi) \text{ and } ((a \in N \text{ and } b \in N) \text{ or } (a \in N \text{ and } b \in N))\}$ ,  
 $A^e = A^e - DA$   
DA. DA –deleted arcs- to be deleted after moving  $N$ .
- 4)  $\text{RP}^e \dot{\Delta} \text{RP}^e \cup \psi$ ;  $\text{RP}^e \dot{\Delta} \text{RP}^e / \psi_r$
- 5) if  $A_{\text{LRN}}$  is the set of arcs in LRN,  
 $A_{\text{LRN}} \dot{\Delta} A_{\text{LRN}} \cup \beta$ .

### 3 MODELING MOBILITY PARADIGMS WITH LABELED RECONFIGURABLE NETS

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 label associated with the reconfigure transition. We must identify the unit to be moved, the target computational environment and the types of binding to resources and their locations. This label depends on the kind of mobility.

In general, a reconfigure transition  $\text{rt}$  is always labeled  $\langle \text{EU}, \text{CE}, \text{CE}', \psi, \beta \rangle$ , such that:

- EU: the execution unit to be moved.
- CE, CE': respectively, resource and target computational environments.
- $\psi$ : will be used to model transferable resources. So  $\psi$  is empty if the system has no transferable resource.
- $\beta$ : models bindings after moving.

The execution unit that contains  $\text{rt}$  and the EU that represents the first argument in the label will be defined according to the three design paradigms: remote evaluation (REV), code on demand (COD), and mobile agent (MA).

#### 3.1. Remote Evaluation

In remote evaluation paradigm, an execution unit  $\text{EU}_1$  sends another execution unit  $\text{EU}_2$  from a computational environment  $\text{CE}_1$  to another one  $\text{CE}_2$ . The reconfigure transition  $\text{rt}$  is contained in the unit modeling  $\text{EU}_1$ , and  $\text{EU}_2$  will be the first argument in  $\text{rt}$ 's label.

Example 4.1: Let us consider two computational environments  $E_1$  and  $E_2$ . Firstly,  $E_1$  contains two execution units  $\text{EU}_1$  and  $\text{EU}_2$ ;  $E_2$  contains an execution unit  $\text{EU}_3$ . The three execution units execute infinite loops.  $\text{EU}_1$  executes actions  $\{a_{11}, a_{12}\}$ ,  $\text{EU}_2$  executes actions  $\{a_{21}, a_{22}, a_{23}\}$ , and  $\text{EU}_3$  executes actions  $\{a_{31}, a_{32}\}$ .  $a_{21}$  requires a transferable resource  $\text{TR}_1$  and a non-transferable resource bound by type  $\text{PNR}_1$  witch is shared with  $a_{11}$ .  $a_{22}$  and  $a_{12}$  share a transferable resource bound by value  $\text{VTR}_1$ , and  $a_{23}$  requires a non-transferable resource  $\text{NR}_1$ . In  $E_2$ ,  $\text{EU}_1$  requires a non-transferable resource bound by type  $\text{PNR}_2$  to execute  $a_{31}$ .  $\text{PNR}_2$  has the same type of  $\text{PNR}_1$ .

The system will be modeled as a labeled reconfigurable net LRN. LRN contains two

environments  $E_1, E_2$  that model the two computational environments ( $\text{CE}_1$  and  $\text{CE}_2$ ). Units  $\text{EU}_1$  and  $\text{EU}_2$  will model execution units  $\text{EU}_1$  and  $\text{EU}_2$ , respectively. In this case, the unit  $\text{EU}_1$  will contain a reconfigure transition  $\text{rt}_{\langle \text{EU}_2, E_1, E_2, \psi, \beta \rangle}$ ; such that:

1.  $E_1 = (\text{RP}_1, \text{GP}_1, U_1, A_1)$ ;  $\text{RP}_1 = \{\text{TR}_1, \text{PNR}_1, \text{VTR}_1, \text{NR}_1\}$ .  $U_1 = \{\text{EU}_1, \text{EU}_2\}$ ;
2.  $E_2 = (\text{RP}_2, \text{GP}_2, U_2, A_2)$ ;  $\text{RP}_2 = \{\text{PNR}_2\}$ .  $\text{GP}_2 = \{\text{PEU}_1\}$ .
3.  $\psi_r = \{\text{TR}_1\}$ ,  $\psi_c = \{\text{VTR}_1\}$ ;
4.  $\beta = \{(\text{PEU}_1, \text{str}_2), (\text{PNR}_2, a_{21}), (\text{NR}_1, a_{23})\}$ .

Fig. 1 shows the configuration before firing  $\text{rt}$ , and Fig. 2 shows the configuration after the firing.

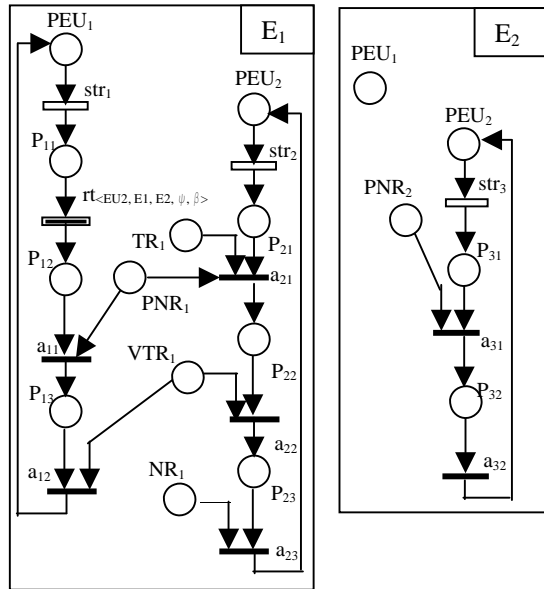


Figure 1: REV-model before firing  $rt$

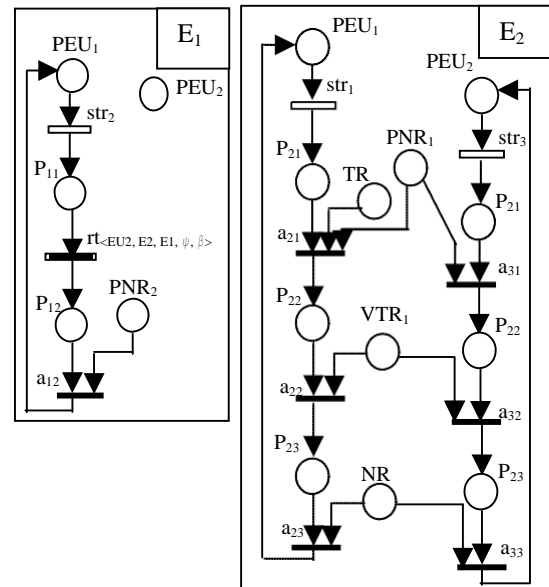


Figure 3: COD-model before firing  $rt$

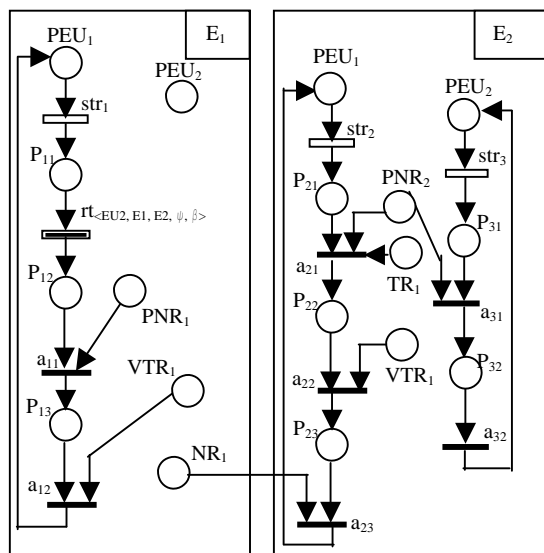


Figure 2: REV-model after firing  $rt$

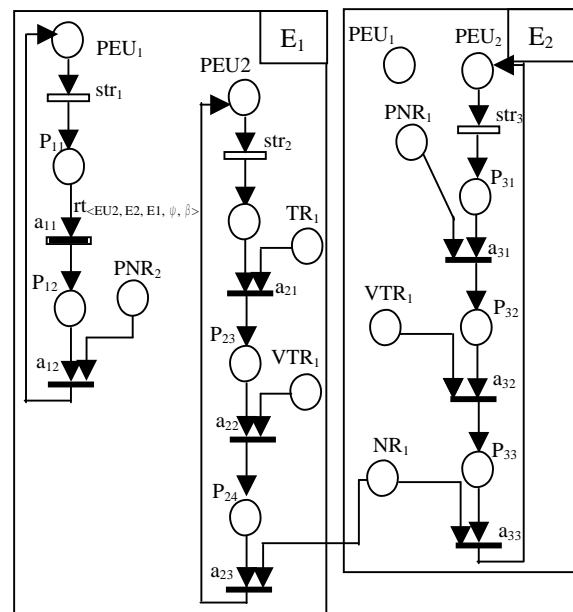


Figure 4: COD-model after firing  $rt$

### 3.2. 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  $EU_1$ , and  $EU_2$  will be the first argument in  $rt$ 's label. If we reconsider the above example, the unit  $EU_1$  will contain a reconfigure transition  $rt_{\langle EU_2, E2, E1, \psi, \beta \rangle}$ . Fig. 3 and Fig. 4 shows the model proposed for this system.

The transition  $rt_{\langle EU_2, E2, E1, \psi, \beta \rangle}$  means that  $EU_1$  will demand  $EU_2$  to be moved from  $E_2$  to  $E_1$ . In this case,  $\psi = \{TR_1, VTR_1\}$ ,  $\beta = \{(PEU_2, str_2), (PNR_2, a_{21}), (NR_1, a_{23})\}$ . Fig.3 shows the configuration before firing  $rt$ , and Fig.4 shows the configuration after the firing.

### 3.3. 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 and  $EU$  (the first argument) is also this 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  $\{a_{31}, a_{32}\}$ . To be executed,  $a_{11}$  require a transferable resource TR1 and a non-transferable resource bound by type PNR<sub>1</sub> 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<sub>1</sub>. In  $E_2$ ,  $SA_2$  requires a non-transferable resource bound by type PNR<sub>2</sub> to execute  $a_{32}$ . PNR<sub>2</sub> has the same type of PNR<sub>1</sub>.

The system will be modeled as a labeled reconfigurable net LRN. LRN contains two environments  $E_1$  and  $E_2$  that model the two computational environments. In this case the unit A that models the mobile agent A will contain a reconfigure transition  $rt < A, E_1, E_2, \psi, \beta >$ ; such that:

1.  $E_1 = (RP_1, GP_1, U_1, A_1)$ ;  $RP_1$  contains at least four places that model the four resources. Let  $TR_1, NR_1, PNR_1$  and  $VTR_1$  be these places.  $GP_1$  contains at least a free place  $PA_1$  modeling that A can be received, and  $U_1 = \{A\}$ .
2.  $E_2 = (RP_2, GP_2, U_2, A_2)$ ;  $RP_2 = \{PNR_2\}$ ,  $GP_2 = \{PA_2\}$ .
3.  $\psi_i = \{TR_1\}, \psi_c = \{VTR_1\}$ ;
4.  $\beta = \{(PA_2, str_1), (PNR_2, a_{11}), (NR_1, a_{13})\}$ .

Fig. 5 shows the configuration before firing  $rt$ .

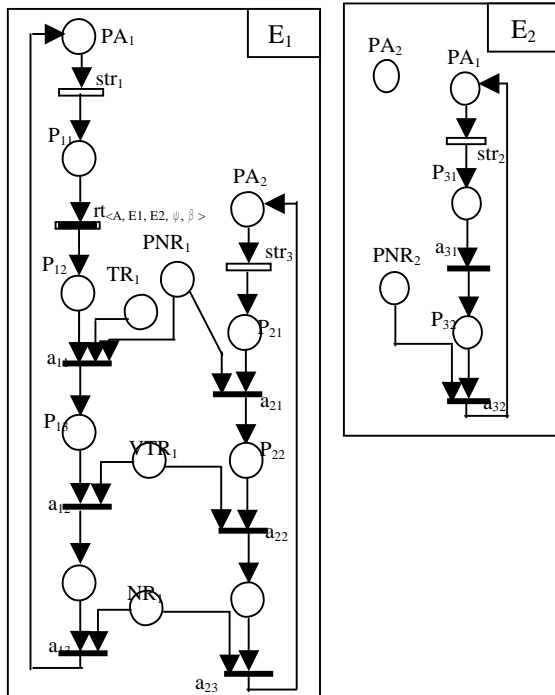


Figure 5: MA-model before firing  $rt$

Fig. 6 shows the configuration after the firing.

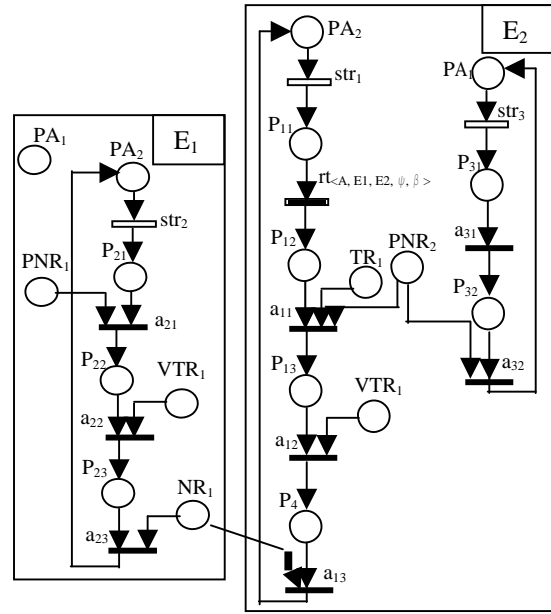


Figure 6: MA-model after firing  $rt$ .

#### 4 RECONFIGURABLE MAUDE

Maude [23] is a high-level language and high-performance system supporting executable specifications and declarative programming in rewriting logic [24]. Maude also supports equational specification, since rewriting logic contains equational logic. The underlying equational logic chosen for Maude is membership equational logic, that has sorts, subsorts, operator overloading, and partiality definable by membership and equality conditions. Modules of Maude are theories in rewriting logic. The most general Maude module are called system modules. A rewrite theory is a triple  $T = (\Omega, E, R)$ , where  $\Omega$  is a signature,  $E$  a set of equations and  $R$  a set of rewriting rules. The equations  $E$  in the equational theory  $(\Omega, E)$  are presented as a union  $E = A \cup E'$ , with  $A$  a set of equational axioms introduced as attributes of operators in the signature  $\Omega$ .  $E'$  is a set of Church-Rosser equations assumed to be terminating modulo the axioms  $A$ . Considering the Maude syntax, a system module has the form  $\text{mod } T \text{ endmod}$ . Maude contains a sublanguage of functional modules and object modules. A functional module have the form  $\text{fmod } = (\Omega, E) \text{ endfmod}$  and an object module have the same form as system module. Maude can be used as a tool for the specification and verification of distributed systems. The dynamic of the distributed system is specified by rewrite rules. Rewrite rules model transitions from one state to another state, during the execution of the distributed system.

Maude has been extended to deal with some aspects not considered in former version. Real time

Maude [25] is a system to specify and analyze real time and hybrid systems. Mobile Maude [5] is an extension of Maude for mobile systems specification. Mobile Maude is an object oriented high level language with asynchronous message passing. The key feature of mobile Maude is that it deals with various security aspects. Cryptographic authentication protects machine from malicious mobile code, redundant checks insure reliability and integrity of computations, and public-key infrastructure service protects communications.

In this section we propose an encoding of Labeled Reconfigurable Nets in a Maude-based language. We call the inspired language "Reconfigurable Maude" (R-Maude). We want to profit from the powerful of Maude (as a meta-language). We extend Maude to support the translation of LRN and their simulation. R-Maude enrich Maude with new kind of rewriting rules. These rules are called Reconfigurable rules (R-Rules). The semantic of these rules is similar to that of Reconfigurable transition in LRN. When a R-rule is executed, the R-Maude specification will be updated in different ways, this will depend on label associated with this rule.

A specification in R-Maude is a set of Reconfigurable rewrite theories (R-theories). An R-theory  $\mathcal{RT}$  is a triple  $(\mathcal{L}, E, R)$  as like a rewrite theory. The different resides in the set  $R$ .  $R$  will contain two kinds of rules: standard Rules S-Rules (well known rules of Maude) and Reconfigurable rules R-Rules. A R-Rule  $r_\lambda$  is composed of a label  $\lambda = \langle d, RT1, RT2, S \rangle$  and a rule  $t \dot{\lambda} t'$ . In the label  $\lambda$ ,  $RT1$  and  $RT2$  are two R-Theorie,  $S$  is a segment of a theory, and  $d$  a specific parameter. The segment  $S$  can be a set of sorts, rules, variables, operators that can be an R-theory or not. When  $r_\lambda$  is fired, the specification can be updated in several ways. Updating specification means that their R-theories will be changed. This change depends on  $\lambda$ . In general, when  $r_\lambda$  is fired, the segment  $S$  will move from  $RT1$  to  $RT2$  or the inverse. The  $d$  parameter can be used to express direction of this move.

## 5 PROTOTYPING R-MAUDE

We have prototyped R-Maude. The prototype is a system composed from a text editor and an interpreter. The editor is used to enter the specification and commands. The interpreter executes commands, and through this updates specifications. The system was experimented on a LAN (Local Area Network), composed of a few machines. The system is installed on all hosts. So the specifications are edited ever where. On every hosts, commands can be executed. The execution of commands will create the system dynamic. This dynamic can be shown as migration of specification's part (or ever else at whole) through

the LAN. The specification (or their parts) are transferred in messages between machines, using UDP protocol.

The interpreter realized for R-Maude can be used to interpret Maude specifications. The major different is that in this newest interpreter, we have added the interpretation of R-Rules. The label of an R-Rule precedes the rule, and it has the form  $[MT] L | IP@ | S]$ . Semantics of these parameters is :  $MT$ : mobility type (MA, COD, REV, ...),  $L$ : a multi-set of operations and rules to be moved, cloned or removed from or to the local host,  $IP@$ : IP address of distant host,  $S$ : sources to move or to remove from or to the local host. When specifications (or part of them) are moved, some resources (R) necessary to firing some rules become far (on an other host). IP address of the far host appears with the concerned resource in the form:  $R[IP@]$ .

To encode LRN we adopt the same rules proposed to translate Petri Nets into Maude in [23]. The newest is that R-transition will be translated in R-Rules. Here after, we present the encoding of Fig.5's example in R-Maude prototype. We consider that the two environments  $E1$ ,  $E2$  are specified as two specifications on two hosts (Host1 and Host2). Host1 has the IP address : 192.168.0.1, and Host2 has the IP address : 192.168.0.2.

On Host1, we have the specification:

```
mod E1
sort Place Marking .
subsort Place << Marking .
op _ , _ : Marking Marking ->
Marking .
ops PA1, P11, P12, P13, P14, PA2, P21,
P22, P23, TR1, VTR1, PNR1, NR1 :-> Place
rl [str1] : PA1 => P11 .
rl [rt][MA|192.186.0.2|{{P11-
P14}}, {str1-
a13}}|{TR1, VTR1}}] :
P11 => P12 .
rl [a11] : P12, TR1, PNR1 => P13 .
rl [a12] : P13, VTR1 => P14 .
rl [a13] : P14, NR1 => PA1 .
rl [str2] : PA2 => P21 .
rl [a21] : P21, PNR1 => P22 .
rl [a22] : P22, VTR1 => P23 .
rl [a23] : P23, NR1 => PA2 .
endmod
```

and on Host2, we have the specification :

```
mod E2
sort Place Marking .
subsort Place << Marking .
op _ , _ : Marking Marking ->
Marking .
ops PA1, PA2, P31, P32, PNR2 : -> Place.
rl [str3] : PA2 => P31 .
rl [a31] : P31 => P32 .
rl [a32] : P32, PNR2 => PA2 .
endmod
```

As an example of a command, we have "rw PA<sub>1</sub>" on Host1. The execution of this command

will produce respectively on Host1, and Host2 the two specifications:

```

mod E1
sort Place Marking .
subsort Place << Marking .
op _ , _ : Marking Marking ->
    Marking .
ops PA1, PA2, P21, P22, P23,
    VTR1, PNR1, NR1 :-> Place
rl [str1] : PA2 => P21 .
rl [a21] : P21, PNR1 => P22 .
rl [a22] : P22, VTR1 => P23 .
rl [a23] : P23, NR1 => PA2 .
endmod

```

and

```

mod E2
sort Place Marking .
subsort Place << Marking .
op _ , _ : Marking Marking ->
    Marking .
ops PA1, PA2, P31, P32, PNR2 :-> Place.
ops VTR1, TR1 :-> Place.
ops P11, P12, P13, P14 :-> Place.
rl [str3] : PA2 => P31 .
rl [a31] : P31 => P32 .
rl [a32] : P32, PNR2 => PA2 .
rl [str1] : PA1 => P11 .
rl [rt][MA|192.186.0.2|{{P11-
    P14}}, {str1-
    a13}}|{TR1, VTR1}] :
    P11 => P12 .
rl [a11] : P12, TR1, PNR2 => P13 .
rl [a12] : P13, VTR1 => P14 .
rl [a13] :
    P14, NR1[192.168.0.1] => PA1 .
endmod

```

Finally, the state of the marking will be : "P<sub>12</sub>" on the Host2. At this point, the two specifications continue their execution on the two hosts where they reside.

## 6 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 [19], 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 [8], tokens can be Petri nets themselves. 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 have attempted to provide a formal and graphical model for code mobility. We have extended Petri net with reconfigure labeled 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. We propose also an extension for Maude, that we call R-Maude. R-Maude extends Maude with Reconfigurable rules (R-rules). When an R-Rule is fired, R-Maude specifications are reconfigured on a LAN (Local Area Net). We use R-Maude to encode and simulate LRN models.

## 7 CONCLUSION

Proposed initially to model concurrency and distributed systems, Petri nets attract searchers 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 been proposed to treat mobility aspects. The key idea was to introduce mechanisms that allow reconfiguration of the model during runtime. The most works extends 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 "labeled 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 have presented an extension for Maude : reconfigurable Maude (R-Maude). R-Maude is a distributed system. This system can be used to specification and simulation of mobile code system. A prototype for this system has been realized. We use this prototype to encode and simulate LRN models. 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 working on a denotational semantics for LRN. For R-Maude, the current R-Maude can be used only to simulate Models. Future works will handle specification analyzing. As a future extension, we think to adapt Maude model-checker to reconfigurable Maude. In [26], we have proposed extensions for LRN "Temporal LRN", and in [27], we proposed Coloured LRN. In this sense, we focus on using R-Maude to simulate models of these extensions.

## 8 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] Kees M. van Hee, Irina A. Lomazova, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, Marc Voorhoeve: Nested Nets for Adaptive Systems. IEEE. ICATPN 2006: 241-260.
- [9] P. Knudsen: Comparing Two Distributed Computing Paradigms, A Performance Case Study; MS thesis, Univ. of Tromso 1995.
- [10] I.A. Lomazova: Nested Petri Nets. Multi-level and Recursive Systems. Fundamenta Informaticae vol.47, pp.283-293. IOS Press, 2002.
- [11] M. Merz and W. Lamersdorf: Agents, Services, and Electronic Markets: How Do They Integrate?. Proc. Int'l Conf. Distributed Platforms, IFIP/IEEE, 1996.
- [12] R. Milner: A Calculus of Communicating Systems. Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.
- [13] R. Milner, J. Parrow, and D. Walker: A calculus of mobile processes. Information and Computation, 100:1-77, 1992.
- [14] Reinhartz-Berger, I., Dori, D. and Katz, S.: Modelling code mobility and migration: an OPM/Web approach. Int. J. Web Engineering and Technology, Vol. 2 (2005), No. 1, pp.6-28.
- [15] D. Sangiorgi and D. Walker: The  $\pi$ -Calculus: A Theory of Mobile Processes. Cambridge University Press, 2001.
- [16] 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.
- [17] Tommy Thorn: Programming languages for mobile code. Rapport de recherche INRIA, N ° 3134, Mars, 1997.
- [18] 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.
- [19] F. Rosa Velardo, O. Marroqn Alonso and D. Frutos Escrig: Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems. In 1<sup>st</sup> Int. Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, MTCoord'05. ENTCS, No 150.
- [20] 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](http://www.elsevier.nl/locate/entcs).
- [21] Sutandiyo, 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.
- [22] 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.
- [23] M. Clavel, F.Durán, S.Eker, P.Lincoln, N.Marti-Oliet, J.Meseguer, and J. Quesada: Maude: specification and programming in rewriting logic. SRI International, Januray 1999, <http://maude.csl.sri.com>.
- [24] J. Meseguer: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science, 96 (1):73-155, 1992.
- [25] P.C.Ölveczky, J. Meseguer: Real-Time Maude : A tool for simulating and analyzing real-time and hybrid systems. In K. Futatsugi, editor, Third International Workshop on Rewriting Logic and its Applications, volume 36 of Electronic Notes in Theoretical Computer Science. Elsevier, 2000. <http://www.elsevier.nl/locate.entcs/volume36.html>.
- [26] Laïd Kahloul, Allaoua Chaoui: Temporal Labeled Reconfigurable Nets for Code Mobility Modeling. The International Workshop on Trustworthy Ubiquitous Computing (TwUC 2007) associated to the iiWAS2007 conference.
- [27] Laïd Kahloul, Allaoua Chaoui: Coloured reconfigurable nets for code mobility modeling. In the Proceedings of World Academy of Science, Engineering and Technology, Volume 25, November 2007 with ISSN: 1307-6886. WASET-XXV International Conference Venice, Italy.