Maude Based Patterns for Mobile Agents Itineraries

Faiza Bouchoul, Mohammed Mostefai
Ferhat Abbas University
Sétif. Algeria
{faizabouchoul, Mostefai}@univ-setif.dz

ABSTRACT: *Mobile agent based systems are so complex that the usage of formal tools for simulation and prototyping to facilitate the modelling of such systems is of great interest. Improved methods are needed to insure their correctness. In particular predefined patterns seems to give suitable solution to deal with complexity of formal methods and to enable reusability and consistency. In another hand executable specification are a powerful technique to prototype and analyse complex systems The aim of this paper is to propose a number of patterns sin MAUDE language to model mobile agents itineraries and to ease their analysis and/or their model checking through MAUDE based executable specification.*

Keywords: Mobile agents, Formal verification, Rapid prototyping, Rewriting logic, Maude, Design patterns

Received: 3August 2011, Revised 9 October 2011, Accepted 12 October 2011

1. Introduction

Mobile agents are programs that, with varying degrees of autonomy, can move between hosts across a network. Mobile agents combine the notions of mobile code, mobile computation, and mobile state. They are location aware and can move to new network locations through explicit mobility operations. Mobile agents realize the notion of moving the computation to the data as opposed to moving the data to the computation, which is an important paradigm for distributed computing. Mobile agents are effective in operating in networks that tend to disconnect, have low bandwidth, or high latency [18].

Mobile agents systems exhibit a high degree of complexity so that the use of formal tools seems to be mandatory, design patterns are predefined model structures that can ease the use of formal tools.

2. Mobile Agents Based Systems

Agent systems consist of several autonomous entities, each of them possibly developed independently, and with capabilities to communicate in order to achieve a common goal, they have has some important characteristics, such as distribution, autonomy, interaction and openness, which are helpful to transform traditional architectures into a distributed and cooperative architecture in intelligent systems [12] . Using agents to build complex system functions is an emerging field where researchers are exploring the ability of agents to improve process integration, interoperability, reusability and adaptability.

Agent mobility reinforce performances of complex systems: A mobile agent can autonomously migrate from one platform to another to interact with other agents and to do specific tasks; it can for example, perform local processing, or retrieve information and bring back the results. Mobile agents are advantageous in particular in mobile environments where there is intermittent connectivity, low bandwidth and limited local storage; and for information retrieval in heterogeneous networks. According to [13]. there are seven good reasons to use mobile agents: They reduce the network load; they overcome network latency; they encapsulate protocols; they execute asynchronously and autonomously; they adapt dynamically; they are naturally heterogeneous, and finally they are robust and fault tolerant, Furthermore, many works proved that agent technology in general outperforms client server technology [21]. and that mobile-agents systems can in most conditions outperform static agent systems [19].

3. Executable specification and design patterns

A theoretical model of a system allows formal reasoning about the system. Formal reasoning can be used to establish guarantees about the behaviour of the system. Understanding the semantics of mobile computation is essential for reasoning about mobile agents. Reasoning about mobility can, in turn, yield guarantees about the correctness of mission critical software.

Agent-based models (ABMs) try to capture both the essential characteristics and the complete detail to a agent based systems. In particular pattern based modelling (POM) is a promising techniques dealing with such challenges [11], (POM) is a set of strategies for using patterns observed in the systems to ensure that an ABM captures the right "*essence*" of the system. POM starts with identifying multiple patterns of behaviour in the real system and its agents that seem to capture the essential internal mechanisms for the problem being modelled. These patterns are then used as submodels to build more complex models. In another hand [18] the use of executable specification seems gaining the interest of numerous researchers. Agents are generally specified using a logical description and then this description is *directly executed* in order to implement the agent's behaviour. a logical description of an agent provides an unambiguous and (if appropriate logics are chosen) concise specification of the agent's behaviour [8].

4. Related works

In literature, only few works dealing with formal specification of mobile agents migrations can be found: In [15]. the authors discuss how mobile agent enabled interorganizational workflows can be usefully modelled using advanced Petri Net techniques such as Interorganizational Workflow Nets. This model provides a means to verify the correctness (and, so the viability) of the itineraries of agents used in enacting interorganizational workflows. The authors used a special class of Petri nets, called Workflow nets (WF-nets) [27]. In [5]. the paper presents results of a performance comparative study of the three mobile agents design patterns presented in [6] the Itinerary, Star-Shaped and Branching migration patterns were investigated as solutions to a distributed information retrieval system. The itinerary assumes the migration of the agent over a sequence of hosts to do a given job and then return back to the source agency. In the Star-Shaped pattern, the agent migrates to the first destination agency in the list, executes the relevant job and comes back to the source agency, the agent repeats this cycle until the last agency on its list is visited. In the Branching pattern, the agent clones itself according to the numbers of agencies in the defined itinerary; each clone migrates to one agency, executes its job and notifies the source agency when the job is completed. The three solutions were modelled with timed Coloured Petri Nets.

In [24]. An agent itinerary specification language is defined; the specification language is basically inherited from those of existing process algebras, e.g., CCS and π - calculus, because, according to the authors, they provide well-studied foundations. In [16]. Another itinerary language, *MAIL*, is introduced to model the mobile behaviour of proactive agents. The language is structured and compositional so that an itinerary can be constructed recursively from primitive itineraries. Special constructs to support the specifications of different kinds of itineraries are provided: sequential, conditional, concurrent, and loop itineraries. The operational semantics of the language is defined in terms of a set of inference rules. *MAIL* is amenable to formal methods to reason about mobility and verifies correctness and safety properties.

In [29]. The authors present a new approach to ensure the secure execution of itinerary-driven mobile agents, in which the specification of the navigational behaviour of an agent is separated from the specification of its computational behaviour. Each host is empowered with an access control policy so that the host will deny the access from an agent whose itinerary does not conform to the host's access control policy. A host uses model checking algorithms to check if the itinerary of the agent conforms to its access control policy written in μ -calculus.

In [22]. The goal was to provide both a specification language to model and support reasoning about the coordination of mobile agent systems, and an executable language to control their deployment and operation at runtime. Agents can migrate and thus "*hop*" from host to host in order to perform distributed computations. This is modelled through the p-calculus notion of channel mobility.

Another work [9] presents a State charts-based development process for mobile agents which allows for a seamless transition from the specification of mobile agent behaviour to its implementation. [10] Proposes a method aiming to optimize MA itineraries. In [3] a high-level language for mobile agents with timed interactions and explicit locations is presented. Authors in [4] introduce and study process algebra able to model the systems composed of processes (agents) which may migrate within a distributed environment comprising a number of distinct locations.

5. Theoretical background: the rewriting logic

The rewriting logic [17] is a powerful unifying paradigm for most of formal models of concurrency. It is a computational logic that can be efficiently implemented and that has good properties as a general and flexible logical and semantic framework, in which a wide range of models of computation can be represented. In particular, for programming language semantics [25].

5.1 Rewriting logic semantics

In rewriting logic the rules are similar to those of equational logic but have a completely different significance [17]. A rule $T \rightarrow T'$ do not mean any more T equal T' but T becomes T '. The rule is a basic action allowing the transition of the system from one state to another. The rewriting logic describes the changes of the system so that the state is represented by an algebraic term; the transition becomes a rewriting rule and the distributed structure, an algebraic structure modulo a set of axioms *E*.

Syntax in rewriting logic is given by a signature (Σ, E) where Σ is a set of functions and E a set of axioms. A rewriting theory $T = (\Sigma, E, L, R)$ in rewriting logic is composed of a signature (Σ, E) and by a set of labelled rules R with labels in L. These rules describe the behaviour of the system and the rewritings are performed on the classes of equivalences of the terms modulo the axioms E. The states of the system are specified as algebraic data types, the basic changes which may occur in the system and in parallel are specified by rewriting rules, the set of axioms E capture the structural properties of the system, a possible state of the system is represented by an equivalence class $[\,t\,]$of a term $\boldsymbol{t}$ modulo the structural axioms E.

Given a rewrite theory R, we say that R entails a sequent $[\,t\,] \rightarrow [\,t'\,]$ and write $R \mid\!- [\,t\,] \rightarrow [\,t'\,]$ if and only if $[\,t\,] \rightarrow [\,t'\,]$ can be obtained by finite application of the following rules of deduction:

(1) Reflexivity

$$\forall [\,t\,] \in T_{\Sigma, E(X)} \qquad \overline{[\,t\,] \rightarrow [\,t'\,]} \tag{1}$$

(2) Congruence

For each f (function symbol) $\in \Sigma_n$ $n \in \mathrm{IN}$

$$\frac{[\,t\,] \rightarrow [\,t'\,] \ldots [\,t_n\,] \rightarrow [\,t'_n\,]}{f[\,t_1,\ldots,t_n\,] \rightarrow f[\,t'_1,\ldots,t'_n\,]} \tag{2}$$

(3) Replacement

For each rewrite rule

r : $[t(\bar{x})] \rightarrow [t'(\bar{x})]$ We have

$$\frac{[\,w_1\,] \rightarrow [\,w_1'\,] \ldots [\,w_n\,] \rightarrow [\,w'_n\,]}{[\,t(\bar{w}/\bar{x})\,] \rightarrow [\,t'(\overline{w'}/\bar{x}')]} \tag{3}$$

(4) Transitivity

$$\frac{[\,t_1] \rightarrow [\,t_2]\,[\,t_2] \rightarrow [\,t_3]}{[\,t_1] \rightarrow [\,t_3]} \tag{4}$$

Notice that equational logic is obtained by adding the next symmetry rule, this rule has no meaning in rewriting logic because changes in time are irreversible.

$$\frac{[\,t\,] \rightarrow [\,t'\,]}{[\,t'\,] \rightarrow [\,t\,]} \tag{5}$$

5.2 The MAUDE Language

MAUDE [7]. is a fully reflective programming language and development environment based on rewriting logic and its equational logic sublanguage to specify formal executable environments. *Full MAUDE* is an object-oriented formal language derived from *MAUDE*. In *Full MAUDE* a class defines the structure of an object, and objects are specific instances of a class.

A class consists of a class identifier (Cid), which is a sort, and a list of attributes that are sorts, including class or object identifiers. Classes also support multiple inheritance with subclasses that inherit all the attributes of parent classes.

class C | attribute$_1$*: Sort*$_1$*, … , attribute*$_n$*: Sort*$_n$ *.*
< Oid_Name : C | attribute$_1$: variable$_1$, … , attribute$_n$: variable$_n$ >
msg syntax : Oid Sort$_1$ *Sort*$_n$ *-> Msg .*

Each object has an object identifier (Oid), a class identifier, and a list of attributes. Each message type has a name and a list of arguments. Rewrite rules in *Full MAUDE* transition the system from a configuration of objects and messages to a new configuration of objects and messages, so that they define all the possible transitions for the concurrent system.

In *MAUDE*, the general form required of rewrite rules used to specify the behaviour of an object-oriented system is as follows:

$\mathbf{m_1,...,m_n} < O_1 : C_1 / atts_1 > \;...\; < O_m : C_m / atts_m > \;=>$
$< O_{i1} : C_{i1} / atts_{i1} > \;...\; < O_{ik} : C_{ik} / atts_{ik} >$
$< Q_1 : D_1 / \mathbf{atts''_1} > \ldots < O_p : D_p / \mathbf{atts''_p} > \; \mathbf{m'_1}, \ldots, \mathbf{m'_q} \;\; if\ C$

where the $m_1,\ldots,m_n$ are message expressions,

$O_1,\ldots,O_n$; $O_{i1}, \ldots ,O_{ik}$ and $Q_1,\ldots, Q_n$ are objects such as
$\{O_{i1}, \ldots ,O_{ik}\} \subset \{O_1,\ldots,O_n\}$

C is the rule's condition. A rule of this kind expresses a communication event in which *n* messages and *m* distinct objects participate. The outcome of such an event is as follows:

- The messages $\mathbf{m_1, \ldots, m_n}$ disappear;
- The state and possibly even the class of the objects $\mathbf{O_{i1}, \ldots ,O_{ik}}$ may change;
- All other objects vanish;
- New objects $\mathbf{Q_1,\ldots, Q_n}$ are created;
- New messages $\mathbf{m'_1,..., m'_q}$ are sent.

6.1 Algebra of itineraries for mobile-agents

This proposition is based on the algebra of itineraries proposed in [15]. modified and enhanced with an operational semantics in term of rewriting logic rules :

Let $\{A^{\{i\}*}\}$ where {i}* denotes strings of integers identifying an agent or a clone of an agent be a finite set of mobile agents. For generating names for agents and their clones the following assumption is done: When an agent $A^{\{i\}}$ is cloned k times, its clones are named $A^{\{i\}1}, \ldots , A^{\{i\}k}$ and let *A* ={a1,a2 , … ,an} be a finite set of activities and *S* ={s1,s2 , … ,sn} a finite set of sites to be visited. Itineraries (denoted by *I*) are formed as follows representing the null activity, atomic activity, parallel, sequential, nondeterministic, conditional nondeterministic behaviour, and have the following syntax:

- The null activity : $\varnothing$
- The atomic activity, A_s^a which means the agent *A* moves to site *s* and executes the action *a*
- The parallel behaviour : $I \ \Pi \ I'$ which means the itinerary *I* in parallel with the itinerary *I'*
- Sequential behaviour : $I \oplus I'$ which means itineray *I* then the itinerary *I'*
- Nondeterministic behaviour : $I \mid I'$ which means *I* or *I'*
- Conditional nondeterministic behaviour : $\frac{I \mid I'}{c}$ which means if condition *C* is true then *I* else *I'* , *C* is a Boolean expression.

6.2 Formal patterns for Mobile agent based systems
One approach that can simplify and improve the development of complex systems based on mobile agents are the use of design patterns similar to those proposed in [1] ,[2], [5], [20], [26].

In [15]. It is shown that itineraries are associative with an element identity $\varnothing$, so it is easy to conclude that itineraries are a special case of strings rewriting theory modulo a set of axioms *E=AI.*

Under these considerations we can propose a rewriting logic based operational semantics in term of *MAUDE* rules.The configuration is in this case a set of agents and messages, messages will be used to mark important steps of the itinerary such as the achievement of activities.

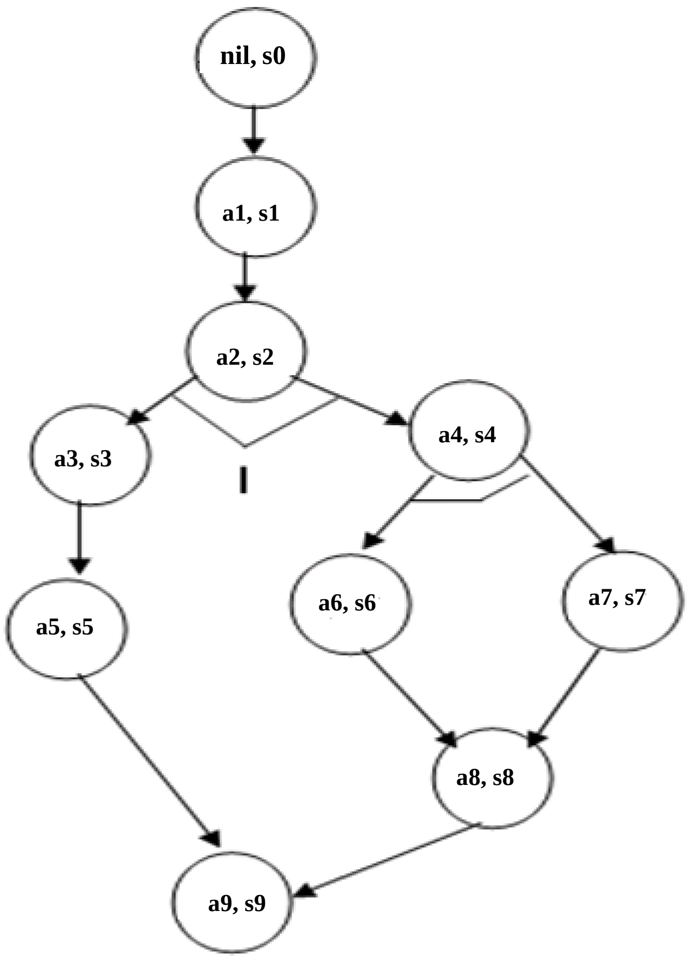
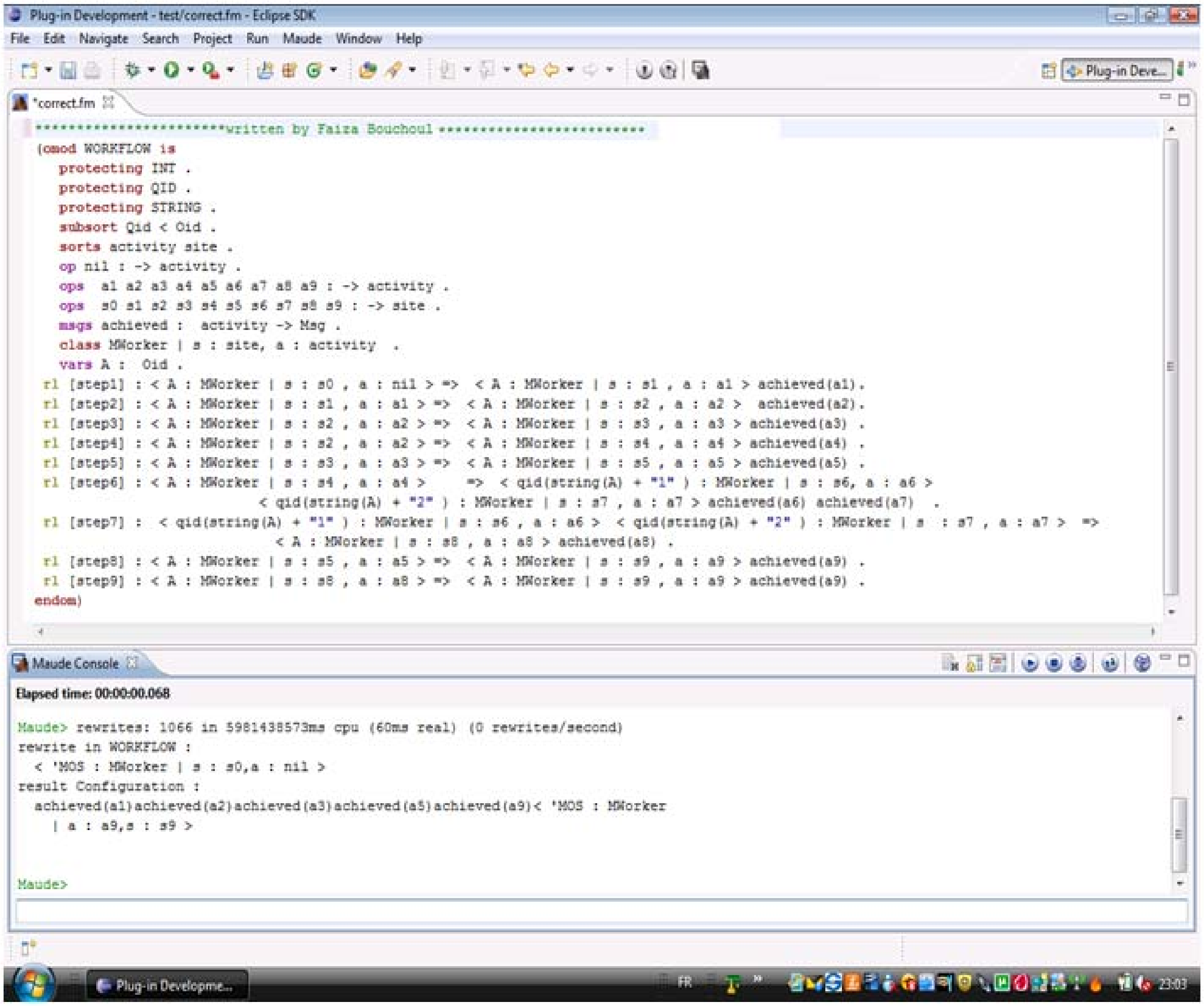


Figure 1. An example of a mobile agent itinerary


Plug-in Development - test/correct.fm - Eclipse SDK
File Edit Navigate Search Project Run Maude Window Help
Plug-in Deve...
*correct.fm
Maude Console
Plug-in Developme...
FR
23:03

```
***********************written by Faiza Bouchoul *************************
(omod WORKFLOW is
   protecting INT .
   protecting QID .
   protecting STRING .
   subsort Qid < Oid .
   sorts activity site .
   op nil : -> activity .
   ops  a1 a2 a3 a4 a5 a6 a7 a8 a9 : -> activity .
   ops  s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 : -> site .
   msgs achieved :  activity -> Msg .
   class MWorker | s : site, a : activity  .
   vars A :  Oid .
 rl [step1] : < A : MWorker | s : s0 , a : nil > =>  < A : MWorker | s : s1 , a : a1 > achieved(a1).
 rl [step2] : < A : MWorker | s : s1 , a : a1 > =>  < A : MWorker | s : s2 , a : a2 >  achieved(a2).
 rl [step3] : < A : MWorker | s : s2 , a : a2 > =>  < A : MWorker | s : s3 , a : a3 > achieved(a3) .
 rl [step4] : < A : MWorker | s : s2 , a : a2 > =>  < A : MWorker | s : s4 , a : a4 > achieved(a4) .
 rl [step5] : < A : MWorker | s : s3 , a : a3 > =>  < A : MWorker | s : s5 , a : a5 > achieved(a5) .
 rl [step6] : < A : MWorker | s : s4 , a : a4 >     =>  < qid(string(A) + "1" ) : MWorker | s : s6, a : a6 >
                      < qid(string(A) + "2" ) : MWorker | s : s7 , a : a7 > achieved(a6) achieved(a7)  .
 rl [step7] :  < qid(string(A) + "1" ) : MWorker | s : s6 , a : a6 >  < qid(string(A) + "2" ) : MWorker | s  : s7 , a : a7 >  =>
                       < A : MWorker | s : s8 , a : a8 > achieved(a8) .
 rl [step8] : < A : MWorker | s : s5 , a : a5 > =>  < A : MWorker | s : s9 , a : a9 > achieved(a9) .
 rl [step9] : < A : MWorker | s : s8 , a : a8 > =>  < A : MWorker | s : s9 , a : a9 > achieved(a9) .
endom)
```

Elapsed time: 00:00:00.068

```
Maude> rewrites: 1066 in 5981438573ms cpu (60ms real) (0 rewrites/second)
rewrite in WORKFLOW :
  < 'MOS : MWorker | s : s0,a : nil >
result Configuration :
  achieved(a1)achieved(a2)achieved(a3)achieved(a5)achieved(a9)< 'MOS : MWorker
    | a : a9,s : s9 >


Maude>
```

Figure 2. Verification in MAUDE of a mobile agent itinerary

Additionally two sets (Site) for places to be visited by the mobile agent and (Activity) for the activities to be performed by the mobile agent are assumed to be predefined.

A class *MWorker* (for mobile agents as mobile workers) with two attributes : '*site*' and '*activity*' is also defined. A mobile agent will be specified by the term < *A:Mworker s:s0* , *a:a0*> which at the right-side of a rule means : The mobile agent A moves to the site s0 to perform the activity a0, and at the left-side of a rule it means that the activity a0 is already performed by the agent

A in the side s0. The activity a0 is essentially built upon the invocation of a Web-process. Since the aim of the study is the behavioural aspect of the itinerary enactment and for the sake of simplicity, the state of the agent is not taken into account. It can be modelled in *MAUDE* easily by a set of additional attributes.

A particular itinerary enactment is performed by one agent and starts at a source site and ends at a target site, at intermediary steps, the agent may clone itself to perform some parallel tasks so that the configuration may sometimes involve more than one agent. After achieving parallel tasks, clones have to declone themselves so that the itinerary terminate with only one agent which is exactly the original one who launches the itinerary at the source site. In what follows the different itinerary constructs are modelled as *MAUDE* rules.

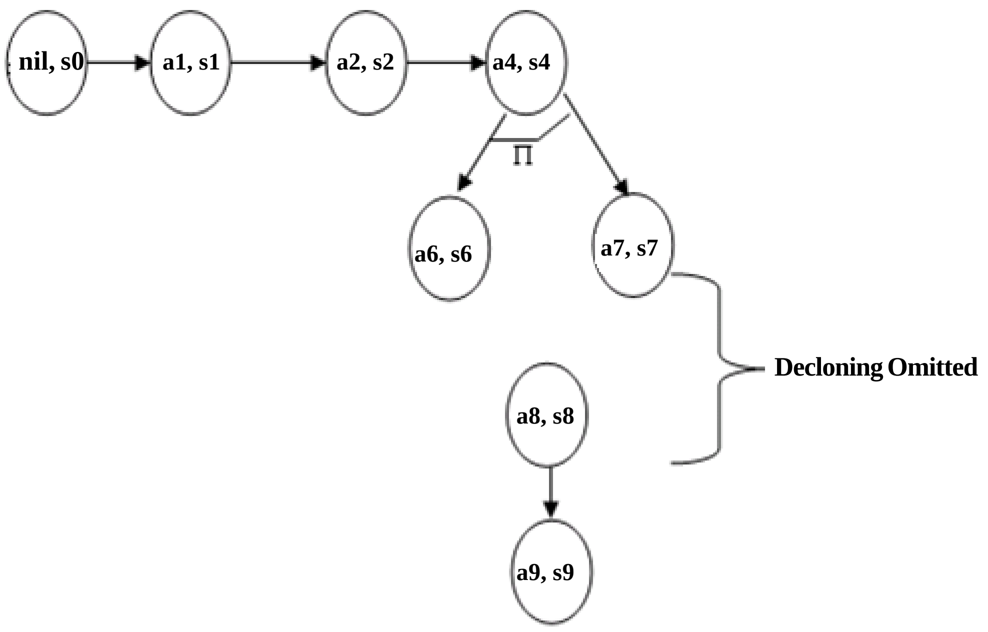


Figure 3. A wrong itinerary for a mobile agent

(1) Agent basic Movement

Different cases have to be envisaged for agent movements:

The agent is in site s1 with no activities to perform here and has to move to site s2 to perform activity a1

<A: MWorker |s:s1, a: nil >=> < A: MWorker| s: s2, a:a1>

After achieving activity a1 in site s1, the agent moves to site s2 to perform activity a2

<A: MWorker| s1:site, a1: activity >=> < A: MWorker| s2:site, a2: activity>

The agent have to perform sequentially two activities a1 then a2 in the same site s1

<A: MWorker| s1:site, a1: activity >=> < A: MWorker| s1:site, a2: activity>

(2) Sequential behaviour.

The basic rule presented above can be used to specify a sequence of more than one activity, so that for n steps (n sequential activities), n rules are necessary, for example an agent *A* is in a site s0 and have to perform sequentially 3 activities (a1,a2, a3) in three sites respectively s1,s2 and s3 ; this can be expressed as follows.

$$A_{s0}^{nil} \oplus A_{s1}^{a1} \oplus A_{s2}^{a2} \oplus A_{s3}^{a3}$$

This sequence can be modelled in *MAUDE* like this

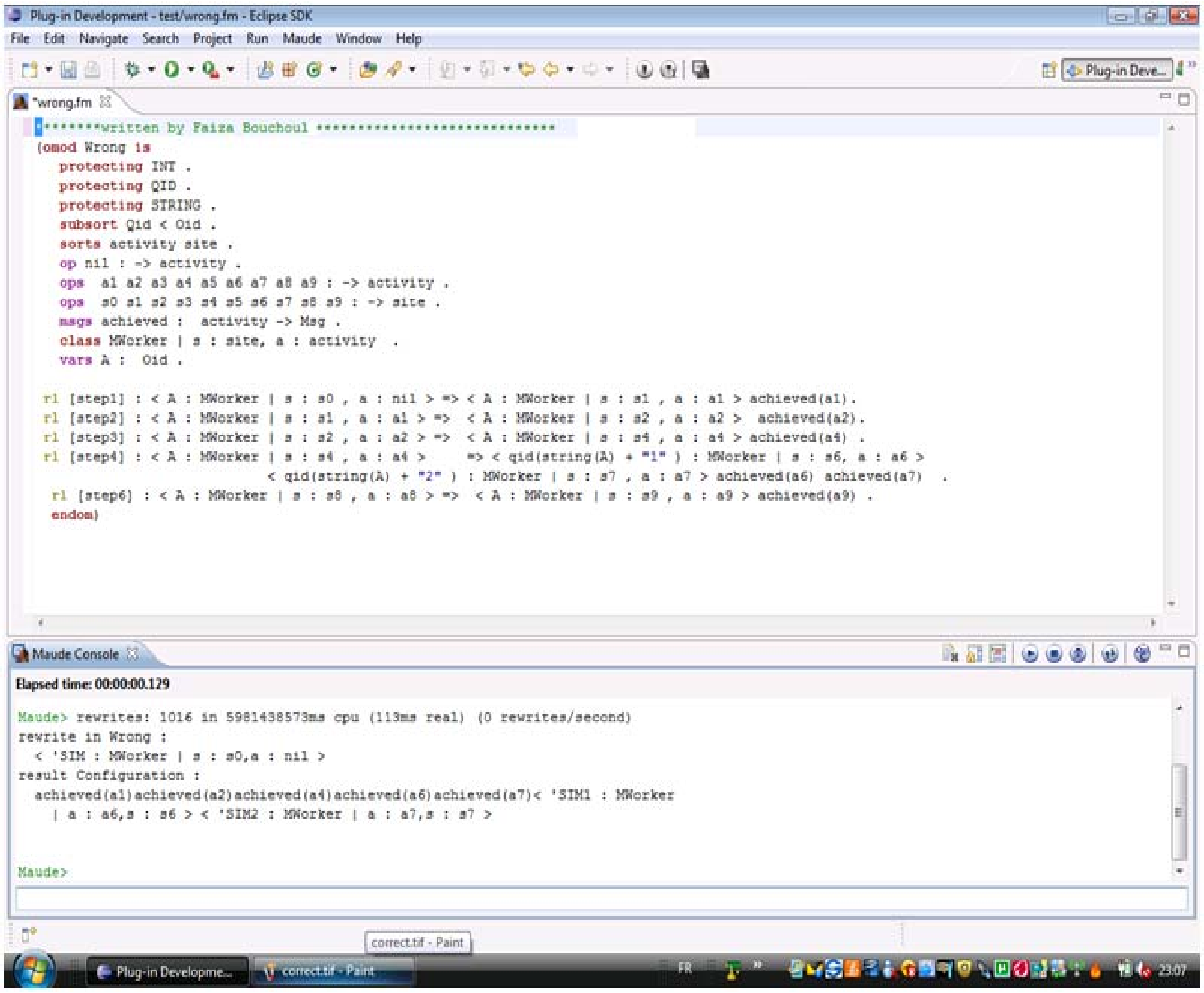
<A: MWorker| s:s0, a: nil>=> <A: MWorker|s: s1, a: a1>

<A: MWorker| s:s1, a: a1>=>

<A: MWorker|s: s2, a: a2><A: MWorker| s:s2, a: a2>

<A: MWorker|s: s3, a: a3>

(3) Parallel Composition.


Plug-in Development - test/wrong.fm - Eclipse SDK
File Edit Navigate Search Project Run Maude Window Help
Plug-in Deve...
*wrong.fm
Maude Console
correct.tif - Paint
Plug-in Developme...
correct.tif - Paint
FR
23:07

```
*******written by Faiza Bouchoul *****************************
(omod Wrong is
   protecting INT .
   protecting QID .
   protecting STRING .
   subsort Qid < Oid .
   sorts activity site .
   op nil : -> activity .
   ops  a1 a2 a3 a4 a5 a6 a7 a8 a9 : -> activity .
   ops  s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 : -> site .
   msgs achieved :  activity -> Msg .
   class MWorker | s : site, a : activity  .
   vars A :  Oid .

 rl [step1] : < A : MWorker | s : s0 , a : nil > => < A : MWorker | s : s1 , a : a1 > achieved(a1).
 rl [step2] : < A : MWorker | s : s1 , a : a1 > =>  < A : MWorker | s : s2 , a : a2 >  achieved(a2).
 rl [step3] : < A : MWorker | s : s2 , a : a2 > =>  < A : MWorker | s : s4 , a : a4 > achieved(a4) .
 rl [step4] : < A : MWorker | s : s4 , a : a4 >     => < qid(string(A) + "1" ) : MWorker | s : s6, a : a6 >
                         < qid(string(A) + "2" ) : MWorker | s : s7 , a : a7 > achieved(a6) achieved(a7)  .
  rl [step6] : < A : MWorker | s : s8 , a : a8 > =>  < A : MWorker | s : s9 , a : a9 > achieved(a9) .
  endom)
```

Elapsed time: 00:00:00.129

```
Maude> rewrites: 1016 in 5981438573ms cpu (113ms real) (0 rewrites/second)
rewrite in Wrong :
  < 'SIM : MWorker | s : s0,a : nil >
result Configuration :
  achieved(a1)achieved(a2)achieved(a4)achieved(a6)achieved(a7)< 'SIM1 : MWorker
    | a : a6,s : s6 > < 'SIM2 : MWorker | a : a7,s : s7 >


Maude>
```

Figure 4. Verification in MAUDE of a wrong itinerary enactment

Two itineraries composed by "*P*" are executed in parallel. $A_{s1}^{a1} \Pi A_{s2}^{a2}$ means that the agent *A* has to perform activities a1 in site s1 and a2 in site s2 concurrently. Parallelism imply cloning of agents. Since agent *A* has to perform actions at both s1 and s2 in parallel. The agent *A is* assumed in site s0 and have to perform the itinerary $A_{s0}^{nil} \oplus (A_{s1}^{a1} \Pi A_{s2}^{a2})$

The *MAUDE* rules for this itinerary are as follows :

<A: MWorker| s:s0, a: nil> => <A1: MWorker |s:s1, a:a1><A2: MWorker |s:s2,a: a2>
<A1: MWorker| s:s1, a:a1><A2: MWorker| s:s2,a: a2> => <A: MWorker| s:s0, a:nil>

The second rule is the declonning rule, it may be used in different manners and to represent different situations, the clones can before declonning themselves perform complex itineraries then declone themselves in a predefined site from where the rebuilt agent can continue its itinerary.

(4) Nondeterministic Behaviour.

In this proposition the nondeterminism presented in [15] is extended to more than two choices, since *MAUDE* has an interleaving semantics, we can specify nondeterministic behaviour in a very natural manner.

As example we assume this itinerary:

$$A_{s0}^{nil} \oplus (A_{s1}^{a1} | A_{s2}^{a2} | A_{s3}^{a3}) \tag{6}$$

The *MAUDE* specification can be like this

<A:MWorker|s:s0,a:nil>=><A1: MWorker| s:s1, a:a1>

<A: MWorker|s:s0,a:nil>=><A1:MWorker| s:s2, a:a2>

<A: MWorker|s:s0,a:nil>=><A1:MWorker| s:s3, a:a3>

Executing any one of these three rules in a nondeterministic manner will disable automatically the two others.

(5) Conditional nondeterministic behaviour.

A conditional nondeterministic choice between two activities a1 and a2 can be for example expressed like this

$$A_{s0}^{nil} \oplus \left(\frac{A_{s1}^{a1} \mid A_{s2}^{a2}}{c} \right) \tag{7}$$

Since *MAUDE* rules are conditional this expression can be specified easily as follows:

<A: MWorker| s:s0, a: nil> =>

<A1: MWorker| s:s1, a:a1> if c

<A: MWorker| s:s0, a: nil> => <A1: MWorker| s:s2, a:a2> if not c

We used Maude for windows under Eclipse, next listing (figure 2) is the source of the specification in *MAUDE* as an object oriented module of this itinerary. *MAUDE* offers a number of commands for verifying properties of the modelled system, the most powerful are *rewrite* command and *search* command.

In this example we used the rewrite command to verify that an enactment of the itinerary of the previous example by a mobile agent "*Mos*" is correct i.e. the correct activities are performed in the right order and at the final stage we have the original mobile agent that launched the enactment at the home site (s0). rewrite command causes a specified term to be rewritten using the rules and equations, in the given module. In figure 4 we tried with a wrong specification (see figure 3 for the itinerary graph) where a decloning of two clones is omitted, the itinerary is launched by a mobile agent '*Sim*' but at the last stage not only some activities are not performed but we end the itinerary with two clones 'Sim1' and 'Sim2' rather than 'Sim' alone which is the initiator of the itinerary.

7. Conclusion

This work tries to show how rewriting logic can be used as a powerful tool for verification and rapid prototyping of complex systems such as mobile agents based systems, especially the MAUDE language is a reflective language with very interesting possibilities in complex systems modelling. This work is a part of a number of other ones dealing with algebraic specification of complex systems and automated code generation from rewriting logic based formal models.

References

[1] Al-Shrouf, F. M. (2008). Facilitator Agent Design Pattern of Procurement Business Systems, 32nd Annual IEEE International Computer Software and Applications Conference, compsac, p. 505-510.

[2] Aridor, Y., Lange, D. B. (1998). Agent design patterns: Elements of agent application design, *In*: Proceedings of the Second International Conference on Autonomous Agents, ACM Press.

[3] Ciobanu, G., Juravle, C. (2010). Mobile Agents with Timers, and Their Implementation . Essaaidi et al. (Eds.): Intelligent Distributed Computing IV, SCI 315, p. 229–239. springerlink.com c_ Springer-Verlag Berlin Heidelberg (ref5)

[4] Ciobanu1, G., Koutny, M. (2011). Timed Migration and Interaction with Access Permissions Butler, M., Schulte, W. (Eds.): FM 2011, LNCS 6664, p. 293–307.

[5] De Araújo Lima, E . F., De Figueiredo, J. C. A., Guerrero, D. D. S. (2004). Using Coloured Petri Nets to Compare Mobile Agent Design Patterns, *Electronic Notes in Theoretical Computer Science* 95, p. 287–305.

[6] De Araújo Lima, E. F., De Lima Machado, P. D., De Figueiredo, J. C. A., Sampaio F. R. (2003). Implementing Mobile Agent Design Patterns in the JADE framework, *In*: Special Issue on JADE of the TILAB Journal EXP.

[7] Duran, F., Meseguer, J. (2007). MAUDE's module algebra, *Science of Computer Programming*, 66 (2) 125-153.

[8] Fisher, M., Ghidini, C. (2010). Executable specifications of resource-bounded agents Auton Agent Multi-Agent Syst 21 p. 368–396.

[9] Fortinoa, G., Russoa, W., Zimeob, E. (2004). A statecharts-based software development process for mobile agents Information and Software Technology 46, 907–921.

[10] Gavalas, A., Politi, C. T. (2006). Low-cost itineraries for multi-hop agents designed for scalable monitoring of multiple subnets Computer Networks 50, 2937–2952.

[11] Grimm, V., Railsback, S. F. (2012). Designing, Formulating, and Communicating Agent-Based Models Heppenstall, A. J., et al. (eds.), *Agent-Based Models of Geographical Systems*, 361.

[12] Guo, Q., Zhang, M. (2009). A novel approach for Multi-Agent-Based Intelligent Manufacturing System. *Information Sciences*, 179 (18) 3079-3090.

[13] Lange, D. B., Oshima, M. (1999). Seven Good Reasons For Mobile Agents, *Communication of the ACM*, 42(3).

[14] Ling, S., Loke, S.W. (2002). Advanced Petri Nets for modelling mobile agent enabled interorganizational workflows. Proceedings of Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, ECBS'02 8-11, p. 245 – 252

[15] Loke, S. W., Schmidt, H., Zaslavsky, A. (1999). Programming the Mobility Behaviour of Agents by Composing Itineraries. *In:* Proceedings. of the 5th Asian Computing Science Conference on Advances in Computing Science, p. 214 – 226.

[16] Lu, S., Xu, C. (2005). A formal framework for agent itinerary specification, security reasoning and logic analysis. 25th IEEE International Conference on Distributed Computing Systems Workshops, June 6-10, p. 580 – 586

[17] Meseguer, J. (1992). Conditional rewriting logic as a unified model of concurrency, *Theoretical Computer Science*, 96 (1) 73-155.

[18] Niranjan Suri, N.,Vitek, J. (2009). Mobile Agents Book chapter, Encyclopedia of Complexity and Systems Science, p.5604-5618 (ref6)

[19] O'Malley, A., Athie, L. S., Deloach, S. A.(2000).Comparing performance of static versus mobile multiagent systems, *In* National Aerospace and Electronics Conference (NAECON) Dayton, OH, October 10-12.

[20] Ojha, A. C., Pradhan, S. K., Patra, M. R. (2007). Pattern-Based Design for Intelligent Mobile Agents , *In*: 4th International Conference on Innovations in Information Technology, November 18-20, p. 501-505.

[21] Patel, R. B., Garg, K. (2005). A Comparative Study of Mobile Agent and Client-Server technologies in a Real Application, *In*: 11th International Conference on Management of Data (COMAD 2005), Goa, India.

[22] Peschanski, F., Darrasse, A., Guts, N., Bobbio, J. (2007). Coordinating mobile agents in interaction spaces, *Science of Computer Programming*, 66, 246–265.

[23] Garg, R. B., K. (2005). A Comparative Study of Mobile Agent and Client-Server technologies in a Real Application, *In*:11th International Conference on Management of Data (COMAD 2005), Goa, India.

[24] Satoh, I. (2004). Selection of mobile agents, Proceedings of. *In*: 24th International Conference on Distributed Computing System, p. 484 – 493

[25] Serbanuta, T. F., Rosu, G., Meseguer, J. (2009). A rewriting logic approach to operational semantics, *Information and Computation*, 207 (2) 305-340.

[26] Tahara, Y., Ohsuga, A. (2001). Behaviour Patterns for Mobile Agent Systems from the Development Process Viewpoint , Proceedings of the 5th International Symposium on Autonomous Decentralized Systems, p. 239 – 242.

[27] Van Der Aalst, W. M. P. (1997). Verification of workflow nets. *In*: Proceedings of 18th International Conference on Application and Theory of Petri Nets, LNCS 1248, Springer-Verlag, p. 407–26

[28] Xu, D.,Wang, H. (2002). Multi-agent collaboration for B2B workflow monitoring, *Knowledge-Based Systems*, 15 (1) 485–491.

[39] Yang, Z., Lu, S.,Yang, P. (2006). Runtime Security Verification for Itinerary-Driven Mobile Agents. 2nd IEEE International Symposium on Dependable, *Autonomic and Secure Computing*. p.177 – 186.

[30] Zhuge, H. (2003). Workflow and agent-based cognitive flow management for distributed team cooperation. Information and Management, 40(1) 419–429.

Biographies

Faiza Bouchoul is a lecturer in software engineering at Ferhat Abbas University (Sétif – Algeria), she holds magister diploma at Mentoury university (Constantine – Algeria), and a doctoral thesis at Ferhat Abbas university (Sétif – Algeria), currently preparing HDR diploma and activates as a research member in LRSD laboratory in Ferhat Abbas university, she's working on paradigms and tools for specification and verification of complex systems.

Mohammed Mostefai is senior lecturer in automatism with the title of professor at Ferhat Abbas university – Algeria. He holds engineer diploma in Electronics in 1988 at Ferhat Abbas University, the Masters and then the PhD of Automatic and Industrial Software at Lille University– France in 1994. His research interests are in the area of monitoring of complex systems. He is currently at the head of the laboratory of Automatic and software engineering (LAS laboratory), he supervises many doctorate students and research projects