Computer Science Journal of Moldova, vol.**32**, no.1(94), 2024

Formal Analysis of Medical Systems using Multi-Agent Systems with Information Sharing

Bogdan Aman   Gabriel Ciobanu

Abstract

Improving safety is a main objective for medical systems. To assist the modelling and formal analysis of medical systems, we define a language for multi-agent systems handling information, timed communication, and timed migration. We use a simplified airway laser surgery scenario to demonstrate our approach. An implementation in Maude is presented; we use the strategies allowed by Maude to guide the rules application in order to decrease substantially the number of possible executions and results in the highly nondeterministic and concurrent multi-agent systems. Finally, we present how the executable specifications can be verified with the model-checking tools in Maude to detect the behavioural problems or desired properties of the agents.

Keywords: multi-agent systems, rewriting engine Maude, strategies and model-checking, example of airway laser surgery.

MSC 2020: 68Q42, 68Q60, 68Q85, 93A16.

ACM CCS 2020: Theory of computation → Equational logic and rewriting, Software and its engineering → Model checking, Theory of computation → Process calculi.

1 Introduction

Information about patients history, diagnostics, drugs, and treatment methods is growing very fast and became more distributed in various locations. The challenge is to collect properly the relevant information and to use it smarter. Software agents can be used in medicine to collect information from many different locations and provide relevant

doi:10.56415/csjm.v32.01

assistance by presenting an integrated view and unexpected relationships or treatment procedures. These agents support decision-making by accessing distributed resources and coordinating the actions in complex medical processes, and potentially avoid failures in medical procedures. Agent-based systems overcome the weakness of centralized systems, improving the performance and providing flexibility, scalability, and robustness.

Software agents work in heterogeneous and distributed networks. An agent operates without the direct intervention of humans and interacts with other agents. A multi-agent system is a collection of autonomous software agents coordinated to solve larger problems. Usually, the information and knowledge required to solve a large problem are distributed in several locations of the network; information is obtained by moving the agents from a location to another. Multi-agent systems coordinate the actions and interactions of these migrating agents to provide solutions for a complex problem.

There already exist articles presenting some advantages of the agent technology and case studies of multi-agent systems in real medical domains [1], [2]. In this article, we present something new: how to use execution strategies in multi-agent systems in medical environments to reduce their evolution, and how to verify automatically their behaviour.

In multi-agent systems, an agent can be characterized by several properties (e.g., cooperation, learning, mobility [3]). In this article, we consider a language of multi-agent systems named iMAS , which is an extension of TiMo [4] with timeouts for communication and mobility, located agents acting in parallel locations and able to migrate between locations. The interaction among agents is given by message-passing communication. Such a system has public information that can be accessed by all agents, while each agent has its private information.

We consider the next example taken from [5] which illustrates the multi-agent systems with information sharing; this example involves information sharing and mobility in space and time: "For airway laser surgery, there are two potential dangers: (1) an accidental burn if both laser and ventilator are activated; and (2) a low-oxygen shock if the Saturation of Peripheral Oxygen (SpO) level of the patient decreases below a given threshold (assume 95%). To prevent the potential dangers when the surgery starts, the airway laser turns on and notifies the ventilator to turn off; and when the patient's SpO level becomes below 95%, the ventilator turns on and notifies the airway laser to turn off."

We provide an implementation in the rewriting engine Maude, and use the strategies allowed by Maude to guide the rules application in order to decrease substantially the number of possible results in the highly nondeterministic and concurrent multi-agent systems. Moreover, we verify their behaviour by using Maude model-checking tools.

The structure of the paper is as follows: Section 2 presents the syntax and semantics of our language iMAS and illustrates them with a running example. Section 3 contains an implementation of iMAS in Maude. In Section 4, we use strategies available in Maude, and verify that the agents behave as intended by means of model-checking tools of Maude. Conclusion and references end the article.

2 Multi-Agent Systems with Information Sharing: Syntax and Semantics

We consider a language named iMAS , where 'i' stands for 'information' and 'MAS ' stands for 'multi-agent system'. Table 1 contains the *syntax* of iMAS, where:

∗ $\mathit{Loc}=\{l, l', \ldots\}$ is a location set, $\mathit{Chan}=\{a, b, \ldots\}$ is channel set, $\mathit{Id}=\{id, \ldots\}$ is a name set for recursive processes, and $\mathcal{N} = \{N, N', \ldots\}$ is a network set;

∗ $id(v) \overset{def}{=} P_{id}$, for all $id \in \mathit{Id}$, is a unique process definition; $t \in \mathbb{N}$ is an action timeout, $k \in \mathbb{Z}$ is a threshold, u is a variable, v is an expression over variables and values, f is a field, and p is the type of the accessed information: either *private* to indicate the information belonging to an agent or *public* to indicate the information belonging to the entire location. If $Q(u)$ is a process definition, where $Q \in \mathit{Id}$ and $v_1 \neq v_2$, then $Q(v_1)$ and $Q(v_2)$ are different.

An agent $P \rhd I$ behaves according to process P and has *private* information I. An agent $\mathsf{go}^t\, l$ then $P \rhd I$ cannot move for t units of

Table 1. Syntax of our Multi-Agent Systems

Processes	$P, Q ::=$	$\mathsf{go}^t\ l$ then P	(move)
	$\mid$	$a^{\Delta t}!\langle v\rangle$ then P else Q	(output)
	$\mid$	$a^{\Delta t}?(u)$ then P else Q	(input)
	$\mid$	if $test$ then P else Q	(branch)
	$\mid$	$stop$	(termination)
	$\mid$	$id(v)$	(recursion)
	$\mid$	$updA(p, f, v)$ then P	(asynch update)
	$\mid$	$updS(a, p, f, v)$ then P	(synch update)
Information	$I ::=$	$\emptyset \mid \langle f; v\rangle \mid I\ I$	
Tests	$test ::=$	$true \mid \neg test \mid test \wedge test$	
		$\mid get(p, f) > k \mid \ldots$	$p \in \{private, public\}$
Agents	$A, B ::=$	$P \triangleright I$	
Set of Agents	$\tilde{A} ::=$	$\mathbf{0} \mid \tilde{A} \parallel A$	
Networks	$N ::=$	$void \mid l[[I \triangleleft \tilde{A}]] \mid M \mid M$	

time; afterwards the agent $P \triangleright I$ moves at location l. Since l can be instantiated after communication, agents can adapt their behaviours.

An agent $a^{\Delta t}!\langle v\rangle$ then P else $Q \triangleright I$ waits for up to t units of time to communicate the value v on channel a to an agent $a^{\Delta t'}?(x)$ then P' else $Q' \triangleright I'$ awaiting at the same location a value to be written on variable x for up to t' units of time. If communication happens, the agents become $P \triangleright I$ and $\{v/x\}P' \triangleright I'$ (all the free occurrences of the variable x are replaced by value v in P') and are available at the same location. If the timers expire, then the agents become $Q \triangleright I$ and $Q' \triangleright I'$.

An agent if $test$ then P else $Q \triangleright I$ checks $test$ using the available information (*public* and *private*). If the *test* returns *true*, then the agent becomes $P \triangleright I$; otherwise, the agent becomes $Q \triangleright I$. For example, a test $get(private, f) > k$ returns *true* only if the value stored in the field f of the private information I is greater than k.

The agent $updA(p, f, v)$ then $P \triangleright I$ with $p \in \{private, public\}$ has two possible outcomes: (i) if the p information does not have a field f, then a new piece of information $\langle f; v\rangle$ is added to p; (ii) if the p information does have a field f, then its value is updated to v. The agent continues with the same process P.

The agent $\mathit{updS}(a,p,f,v)$ $\mathtt{then}$ $P \rhd I$ has a similar behaviour as $\mathit{updA}(p,f,v)$ $\mathtt{then}$ $P \rhd I$, except that the update is performed only if there exists another agent $\mathit{updS}(a,p',f',v')$ $\mathtt{then}$ $P' \rhd I'$ at the same location and using the same channel a that is ready to perform an update. $A = 0 \rhd I$ has no action to execute and terminates.

A network is composed of parallel locations of the form $l[[I \lhd \tilde{A}]]$, where l is a location with *public* information I and a set $\tilde{A}$ of agents; an empty location is denoted by $l[[\emptyset \lhd \mathbf{0}]]$.

The structural equivalence rearranges agents so that they interact. This is needed in the *operational semantics* presented in Table 2; we present only some of the rules as the others are similar. We denote by $N \xrightarrow{\Lambda} N'$ a network N that transforms into a network N' by executing the multiset of actions Λ.

In rule (Stop), we use $\not\rightarrow$ to denote that no action can be executed if no agents are available. Rule (Com) models two agents at the same location l able to communicate on the same channel a. After a successful communication, the agents become $P_1 \rhd I_1$ and $\{v/u\}P_2 \rhd I_2$.

Rule (Put0) is used for an agent to remove its current output action if its timer is zero. Afterwards, the agent can execute Q using the unchanged information I. A similar rule (Get0) is available for the input action. Since rule (Com) can be applied even when the timers are zero, it follows that one of the rules (Com), (Put0), and (Get0) is nondeterministically applied.

Rule (Move0) is used when an agent migrates from location l to location l' and becomes $P \rhd I$. Rule (IfT) is used when the test performed by an agent returns *true*. A similar rule (IfF) is available when the *test* is *false*.

Rules (CrtPr) and (UpdPr) are used when an agent extends or updates, in an asynchronous manner, its *private* or *public* information; afterwards the agent becomes $P \rhd I\langle f; v\rangle$. Similar rules are available for the synchronous updates *updS*.

Rule (Call) transforms an agent $id(v) \rhd I$ into $\{v/u\}P_{id} \rhd I$. Other rules are available to compose smaller subnetworks, and to apply the structural equivalence over networks.

Table 2. Operational Semantics for our Multi-Agent Systems

(Stop)	$l[[I \lhd \mathbf{0}]] \not\rightarrow$
(Com)	$l[[I_l \lhd a^{\Delta t_1}!\langle v\rangle \text{ then } P_1 \text{ else } Q_1 \rhd I_1$ $\;\|\|\; a^{\Delta t_2}?(u) \text{ then } P_2 \text{ else } Q_2 \rhd I_2 \;\|\|\; \tilde{A}]]$ $\xrightarrow{a!?@l} l[[I_l \lhd P_1 \rhd I_1 \;\|\|\; \{v/u\}P_2 \rhd I_2 \;\|\|\; \tilde{A}]]$
(Put0)	$l[[I_l \lhd a^{\Delta 0}!\langle v\rangle \text{ then } P \text{ else } Q \rhd I \;\|\|\; \tilde{A}]] \xrightarrow{a!^{\Delta 0}@l} l[[I_l \lhd Q \rhd I \;\|\|\; \tilde{A}]]$
(Move0)	$l[[I_l \lhd \text{go}^0\, l' \text{ then } P \rhd I \;\|\|\; \tilde{A}]] \mid l'[[I'_l \lhd \tilde{B}]]$ $\xrightarrow{l \rhd l'} l[[I_l \lhd \tilde{A}]] \mid l'[[I'_l \lhd P \rhd I \;\|\|\; \tilde{B}]]$
(IfT)	$test@(I\ I_l) = true$ implies $l[[I_l \lhd \text{if } test \text{ then } P \text{ else } Q \rhd I \;\|\|\; \tilde{A}]] \xrightarrow{true@l} l[[I_l \lhd P \rhd I \;\|\|\; \tilde{A}]]$
(CrtPr)	$\nexists \langle f; v'\rangle \in I$ implies $l[[I_l \lhd updA(private, f, v) \text{ then } P \rhd I \;\|\|\; \tilde{A}]]$ $\xrightarrow{create_{lf}@l} l[[I_l \lhd P \rhd I\langle f; v\rangle \;\|\|\; \tilde{A}]]$
(UpdPr)	$l[[I_l \lhd updA(private, f, v) \text{ then } P \rhd I\langle f; v'\rangle \;\|\|\; \tilde{A}]]$ $\xrightarrow{upd_{lf}@l} l[[I_l \lhd P \rhd I\langle f; v\rangle \;\|\|\; \tilde{A}]]$
(Call)	$l[[I_l \lhd id(v) \rhd I \;\|\|\; \tilde{A}]] \xrightarrow{call@l} l[[I_l \lhd \{v/u\}P_{id} \rhd I \;\|\|\; \tilde{A}]]$, where $id(u) \overset{def}{=} P_{id}$
(DStop)	$l[[I_l \lhd \mathbf{0}]] \overset{t}{\rightsquigarrow} l[[I_l \lhd \mathbf{0}]]$
(DPut)	$t \geq t' \geq 0$ implies $l[[I_l \lhd a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q \rhd I]]$ $\overset{t'}{\rightsquigarrow} l[[I_l \lhd a^{\Delta t-t'}!\langle v\rangle \text{ then } P \text{ else } Q \rhd I]]$
(DGet)	$t \geq t' \geq 0$ implies $l[[I_l \lhd a^{\Delta t}?(u) \text{ then } P \text{ else } Q \rhd I]]$ $\overset{t'}{\rightsquigarrow} l[[I_l \lhd a^{\Delta t-t'}?(u) \text{ then } P \text{ else } Q \rhd I]]$
(DMove)	$t \geq t' \geq 0$ implies $l[[I_l \lhd \text{go}^t\, l' \text{ then } P \rhd I]]$ $\overset{t'}{\rightsquigarrow} l[[I_l \lhd \text{go}^{t-t'}\, l' \text{ then } P \rhd I]]$

We denote by $N \overset{t}{\rightsquigarrow} N'$ a network N that transforms into a network N' after t units of time. In rule (DStop), the network $l[[I \triangleleft \mathbf{0}]]$ is not affected by the passing of time. To decrease action timers, we use the rules (DPut), (DGet), and (DMove), while other rules are used to compose smaller subnetworks and to apply the structural equivalence.

A derivation $N \xRightarrow{\Lambda,t} N'$, where $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$ is a multiset of actions and t is a timeout, denotes a complete computational step:

$$N \xrightarrow{\lambda_1} N_1 \ldots N_{k-1} \xrightarrow{\lambda_k} N_k \overset{t}{\rightsquigarrow} N'.$$

By $N \Longrightarrow^* N'$ we denote an iMAS network N that transforms into a network N' after zero or more action steps followed by a time step.

In our setting, the passing of time is deterministic. The following theorem claims that time passing does not introduce nondeterminism in the evolution of a network.

Theorem 1. *The next statements hold for any three networks N, N', and N'':*

1. *if $N \overset{0}{\rightsquigarrow} N'$, then $N = N'$;*
2. *if $N \overset{t}{\rightsquigarrow} N'$ and $N \overset{t}{\rightsquigarrow} N''$, then $N' = N''$.*

Proof. By induction on the structure of N, as in [6]. □

The following theorem claims that when only time rules can be applied for two time steps of lengths t and t'', then the rules can be applied also for a time step of length $t + t'$. This ensures that the evolution is smooth (without gaps).

Theorem 2. *If $N \overset{t}{\rightsquigarrow} N'' \overset{t'}{\rightsquigarrow} N'$, then $N \overset{t+t'}{\rightsquigarrow} N'$.*

Proof. By induction on the structure of N, as in [6]. □

Example 1. *Let us consider the example mentioned before. For this scenario, we use the following notations for the descriptions of the involved agents: sl (start laser), lv (laser ventilator), stateL (state laser), stateV (state ventilator), and SpO (Saturation of Peripheral Oxygen).*

The entire system can be described as a network:

$$
\begin{aligned}
&\mathit{LaserSurgerySystem} = \mathit{SurgeryRoom}[[\mathit{empty} \lhd \\
&\quad \mathit{Surgeon} \rhd \mathit{empty} \; || \mathit{Laser} \rhd \langle \mathit{stateL}; \mathit{0}\rangle \\
&\quad || \, \mathit{Ventilator} \rhd \langle \mathit{stateV}; \mathit{1}\rangle \langle \mathit{SpO}; \mathit{98}\rangle]] \\
&\; | \mathit{LockerRoom}[[\mathit{empty} \lhd \mathit{Zero}]]
\end{aligned}
$$

where:

$$
\begin{aligned}
&\mathit{Surgeon} = \mathit{sl}^{\Delta 1}!\langle \mathit{yes}\rangle \text{ then } \mathit{Surgeon} \text{ else } \mathit{Surgeon} \\
&\mathit{Laser} = \mathit{sl}^{\Delta 1}?(x) \text{ then } \mathit{updS}(\mathit{lv}, \mathit{private}, \mathit{stateL}, \mathit{1}) \\
&\qquad\qquad\qquad\qquad \text{ then } \mathit{updS}(\mathit{lv}, \mathit{private}, \mathit{stateL}, \mathit{0}) \text{ then } \mathit{Laser} \\
&\qquad\qquad\qquad \text{ else } \mathit{Surgeon} \\
&\mathit{Ventilator} = \mathit{updS}(\mathit{lv}, \mathit{private}, \mathit{stateV}, \mathit{0}) \text{ then } \mathit{Ventilator}' \\
&\mathit{Ventilator}' = \text{if } \mathit{get}(\mathit{private}, \mathit{SpO}) > \mathit{95} \\
&\quad \text{then } \mathit{updA}(\mathit{private}, \mathit{SpO}, \mathit{get}(\mathit{private}, \mathit{SpO}) - \mathit{1}) \\
&\qquad \text{then } \mathit{wait}^{\Delta 1}?(y) \text{ then } \mathit{stop} \text{ else } \mathit{Ventilator}' \\
&\quad \text{else } \mathit{updA}(\mathit{private}, \mathit{SpO}, \mathit{98}) \\
&\qquad \text{then } \mathit{updS}(\mathit{lv}, \mathit{private}, \mathit{stateV}, \mathit{1}) \text{ then } \mathit{Ventilator}
\end{aligned}
$$

3 Implementing Multi-Agent Systems with Information Sharing

We provide an implementation for our language iMAS . Maude is a high-level language and a rewriting platform; it is part of a high-performance system supporting executable specifications in rewriting logic. Rewriting logic [7] is basically a framework which combines term rewriting with equational logic. We use Maude [8] extended with time aspects taken from Real-Time Maude [9].

In order to implement the multi-agent systems of iMAS , we consider sorts corresponding to sets from our language:

```
sorts Location Channel Process GlobalSystem .
```

For the iMAS operators in Table 1, the attached attribute `ctor` marks a constructor, while attribute `prec` followed by a value marks a precedence among operators. Moreover, in real-time Maude we attach the attributes `comm` and `assoc` to mark commutative and associative operators. For example:

```
op < _ ; _ > : Field Nat -> Inf .
op _ |> _ : Process Inf -> Agent [ctor] .
op _||_ : Agent Agent -> Agent [ctor prec 5 comm assoc] .
```

Since some rules of Table 2 have hypotheses, the corresponding rules in Maude are conditional. To identify the corresponding rule of Table 2, we use similar names for each of the below rewrite rules. For example:

```
crl [UpdatePrivate] : k[[I <| ((update(private,f,v)
   then (P)) |> (I' < f ; v' >)) || A]]
=> k[[I <| (P |> (I'< f ; v >)) || A]] if A =/= Zero .
```

As Maude does not support infinite computations, the recursion operator of iMAS is not directly encodable into Maude. Thus, we encode each $id(v)$ into a construction $id(v, b)$, where b is a Boolean value b limiting the unfolding until the first occurrence of *not* b (to transform *not* b into b, an evolution rule should be used again). For example:

```
op Laser : Bool -> Process [ctor] .
ceq Laser(b) =  ((sl ^ 1 ? ( x ))
    then (updateS(lv,private,stateL,1)
        then (updateS(lv,private,stateL,0)
            then Laser(not b)))
    else Laser(not b) ) if b == true .

crl [UnfoldLaser] : k[[I1 <| ((Laser(b)) |> I2 ) || B]]
=> k[[I1 <| ((Laser(not b)) |> I2 ) || B]]
    if b == false /\  B =/= Zero .
crl [UnfoldLaser] : k[[I1 <| ((Laser(b)) |> I2 )]]
=> k[[I1 <| ((Laser(not b)) |> I2 )]]  if b == false .
```

The definitions for *Surgeon* and *Ventilator* are similar.

The rule `[tick]` models the maximum passage of time.

```
crl [tick] : {M} => {delta(M, mte(M))}
    if mte(M) =/= INF and mte(M) =/= 0 .
```

The rule `[tick]` decreases time by means of function `delta`, and the value of the maximal passed time is computed by the function `mte`.

We show that the transition system associated with the rewrite theory in our Maude specification coincides with the reduction semantics for the multi-agent system. Given a system M, we use $\psi(M)$ to denote its representation in Maude. Also, $\mathcal{R}_{\mathcal{D}}$ denotes the rewrite theory mentioned previously in this section by the rewrite rules, and also by the additional operators and equations of these rewrite rules.

The next result relates the structural congruence in our multi-agent language iMAS with the equational equality of the rewrite theory.

Lemma 1. *$M \equiv N$ if and only if $\mathcal{R}_D \vdash \psi(M) = \psi(N)$.*

Proof. $\Rightarrow$: By induction on the congruence rules of our language iMAS .
$\Leftarrow$: By induction on the equations of the rewrite theory $\mathcal{R}_D$. $\square$

The following result emphasizes the operational correspondence between the high-level systems and their translations into a rewrite theory. Generically, by $M \to N$ is denoted any rule of Table 2.

Theorem 3. *$M \to N$ if and only if $\mathcal{R}_D \vdash \psi(M) \Rightarrow \psi(N)$.*

Proof. $\Rightarrow$: By induction on the derivation $M \to N$.
$\Leftarrow$: By induction on the derivation $\mathcal{R}_D \vdash \psi(M) \Rightarrow \psi(N)$. $\square$

4 Strategies and Model-Checking Multi-Agent Systems with Information Sharing

A strategy controls the rewriting steps such that each step obeys the strategy. The result of applying a strategy is the subset of computations produced according to the strategy.

In Maude, a strategy language able to control explicitly the application of rules was presented in [10]. The command for executing a strategy expression `alpha` applied to a term `t` is `srewrite t using alpha`; its output enumerates the solutions that are obtained after this controlled rewriting. Multiple solutions are possible because strategies do not remove the nondeterminism.

The elementary building block of the strategy language is the application of a rule, and the most basic form is the strategy `all` that does not use any restriction when applying the rules. The iteration `(all)*` runs the strategy `all` zero or more times consecutively. For example, the command

```
srew {LaserSurgerySystem} using (all)* .
```

returns a number of 78 solutions. However, this number of solutions can be reduced if we consider a sort of priority on rules. In what follows, besides the iteration strategy α^*, we consider several others: (i) the strategy `idle` that returns the initial term; (ii) the disjunction (or alternative) strategy $\alpha \mid \beta$ that executes α or β, and (iii) the conditional strategy $\alpha ? \beta : \gamma$ that executes α and then β on its results; if α does not produce any result, it executes γ on the initial term.

The simplest strategy we consider is to apply the `[Input0]` and `[Output0]` rules only if any other rule is not applicable, and the time rule `[tick]` last. Formally:

```
sd step := ((Comm | IfT | IfF | UnfoldSurgeon
    | UnfoldLaser | UnfoldVentilator | UpdatePrivate
    | UpdatePublic | CreatePrivate | CreatePublic
    | UnfoldScheduleBus | Move)
    ? idle : (Input0 | Output0)) .
sd mtestep := (step ? idle : tick )* .
```

In this case, by running the system *LaserSurgerySystem* using the strategy `mtestep`, the number of solutions is decreasing to 53. This number of solutions can be further reduced by considering the strategy `mtestep1` obtained by removing the number of alternative strategies of the form $\alpha \mid \beta$; this leads to an important decrease in the state space, namely 28 solutions.

Since in iMAS systems, the number of possible applicable rules is high, it turns out to be necessary to use software verification. Various properties of the iMAS systems controlled by using strategies can be analyzed and verified automatically by using the unified Maude model-checking tool umaudemc [11]. In this way, we can check by using CTL*. CTL* [12] is a branching-time temporal logic that extends both LTL [13] and CTL [14]. We can check by formulae of the form: (i) E $\langle\ \rangle\ \phi$ (checks the reachability: there exists a path such that eventually ϕ is satisfied); (ii) A $[\ ]\ \phi$ and E $[\ ]\ \phi$ (checks the safety: something bad will never happen); (iii) A $\langle\ \rangle\ \phi$ and $\phi \rightsquigarrow \psi$ (checks the liveness: something good will eventually happen).

The command-line for the umaudemc tool is the following:

```
umaudemc check < file name > < initial term >
    < formula > [ < strategy > ] .
```

To illustrate such a verification, we check some CTL* properties of our running example. Namely, according to [5], the simplified airway laser surgery scenario has two safety properties, i.e., P1: the laser and the ventilator can not be activated at the same time; and P2: the patient's SpO level can not be smaller than 95%. In what follows, we show the outcome of verifying these properties:

```
$ umaudemc check iMASLaserSurgery.maude {LaserSurgerySystem}
    ' A [] ((InfInLocation( < stateL ; 1 > , SurgeryRoom)
    /\ InfInLocation( < stateV ; 0 > , SurgeryRoom))
    \/ (InfInLocation( < stateL ; 0 > , SurgeryRoom)
    /\ InfInLocation( < stateV ; 1 > , SurgeryRoom)))
    ' mtestep2
The property is satisfied in the initial state (56 system
states, 587 rewrites, holds in 56/56 states)

$ umaudemc check iMASLaserSurgery.maude {LaserSurgerySystem}
    ' A [] Saturation(SpO) ' mtestep2
The property is satisfied in the initial state (56 system
states, 475 rewrites, holds in 56/56 states)
```

5 Conclusion

Outperforming the advantages of the agent technology and case studies of multi-agent systems presented in real medical domains [1],[2], in this article we show how to use strategies in multi-agent systems in medical environments in order to reduce their evolution, and how to verify automatically their behaviour by using rewriting platform Maude. Both qualitative aspects (e.g., reachability, safety, liveness) and quantitative aspects (e.g., related to stored information) are presented.

The prototyping language iMAS can be viewed as a member of the TiMo family; this family is generated by a process calculus in which processes can migrate between explicit locations in order to perform local communications with other processes. The initial version of TiMo (presented in [4]) generated various extensions: with access permissions in perTiMo [15], with real-time in rTiMo [16], combining TiMo and the bigraphs [17] to obtain the BigTiMo calculus [18]. Related to the approach presented in this article, we mention [19], in which repuTiMo describes agents with reputation, and [6], in which knowTiMo deals with knowledge of agents represented as sets of trees whose nodes carry information. A related approach is presented in [20], where a Java-based software allows the agents to perform timed migration like in TiMo. In [21], it is given a translation of TiMo into a Real-Time Maude rewriting language.

References

[1] H. Lieberman and C. Mason, "Intelligent agent software for medicine," *Studies in Health Technology and Informatics*, vol. 80, pp. 99–109, 2002.

[2] D. Isern, D. Sánchez, and A. Moreno, "Agents applied in health care: A review," *International Journal of Medical Informatics*, vol. 79, no. 3, pp. 145–166, 2010.

[3] M. J. Wooldridge, *An Introduction to MultiAgent Systems, Second Edition.* Wiley, 2009.

[4] G. Ciobanu and M. Koutny, "Timed mobility in process algebra and Petri nets," *Journal of Logic and Algebraic Programming*, vol. 80, no. 7, pp. 377–391, 2011.

[5] C. Guo, Z. Fu, Z. Zhang, S. Ren, and L. Sha, "Design verifiably correct model patterns to facilitate modeling medical best practice guidelines with statecharts," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6276–6284, 2019.

[6] B. Aman and G. Ciobanu, "Knowledge dynamics and behavioural equivalences in multi-agent systems," *Mathematics*, vol. 9, no. 22, 2021.

[7] J. Meseguer, "Twenty years of rewriting logic," *Journal of Logic and Algebraic Programming*, vol. 81, no. 7-8, pp. 721–781, 2012.

[8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4350.

[9] P. C. Ölveczky and J. Meseguer, "The Real-Time Maude tool," in *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, 2008, pp. 332–336.

[10] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. L. Talcott, "Programming and symbolic computation in Maude," *Journal of Logical and Algebraic Methods in Programming*, vol. 110, 2020.

[11] R. Rubio, N. Martí-Oliet, I. Pita, and A. Verdejo, "Model checking strategy-controlled systems in rewriting logic," *Automated Software Engineering*, vol. 29, no. 1, p. 7, 2022.

[12] E. A. Emerson and J. Y. Halpern, ""Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic," *Journal of the ACM*, vol. 33, no. 1, pp. 151–178, 1986.

[13] A. Pnueli, “The temporal logic of programs,” in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, 1977, pp. 46–57.

[14] E. M. Clarke and E. A. Emerson, “Design and synthesis of synchronization skeletons using branching-time temporal logic,” in *Workshop on Logics of Programs*, ser. Lecture Notes in Computer Science, D. Kozen, Ed., vol. 131. Springer, 1981, pp. 52–71.

[15] G. Ciobanu and M. Koutny, “Timed migration and interaction with access permissions,” in *17th International Symposium on Formal Methods, FM 2011*, ser. Lecture Notes in Computer Science, M. J. Butler and W. Schulte, Eds., vol. 6664. Springer, 2011, pp. 293–307.

[16] B. Aman and G. Ciobanu, “Real-time migration properties of rTiMo verified in Uppaal,” in *11th International Conference on Software Engineering and Formal Methods, SEFM 2013*, ser. Lecture Notes in Computer Science, R. M. Hierons, M. G. Merayo, and M. Bravetti, Eds., vol. 8137. Springer, 2013, pp. 31–45.

[17] R. Milner, *The Space and Motion of Communicating Agents.* Cambridge University Press, 2009.

[18] W. Xie, H. Zhu, M. Zhang, G. Lu, and Y. Fang, “Formalization and verification of mobile systems calculus using the rewriting engine maude,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC 2018*, S. Reisman, S. I. Ahamed, C. Demartini, T. M. Conte, L. Liu, W. R. Claycomb, M. Nakamura, E. Tovar, S. Cimato, C. Lung, H. Takakura, J. Yang, T. Akiyama, Z. Zhang, and K. Hasan, Eds. IEEE Computer Society, 2018, pp. 213–218.

[19] B. Aman and G. Ciobanu, “Dynamics of reputation in mobile agents systems and weighted timed automata,” *Information and Computation*, vol. 282, p. 104653, 2022.

[20] G. Ciobanu and C. Juravle, “Flexible software architecture and language for mobile agents,” *Concurrency and Computation: Practice and Experience*, vol. 24, no. 6, pp. 559–571, 2012.

[21] B. Aman and G. Ciobanu, “Verification of multi-agent systems with timeouts for migration and communication,” in *16th International Colloquium on Theoretical Aspects of Computing, ICTAC 2019*, ser. Lecture Notes in Computer Science, R. M. Hierons and M. Mosbah, Eds., vol. 11884, 2019, pp. 134–151.

Bogdan Aman, Gabriel Ciobanu

Received February 14, 2024
Revised February 16, 2024
Accepted February 21, 2024

Bogdan Aman
ORCID: https://orcid.org/0000-0001-7649-8181
Institute of Computer Science, Romanian Academy, Iasi Branch
Str. Teodor Codrescu 2, 700481, Iaşi, Romania
E–mail: bogdan.aman@iit.academiaromana-is.ro

Gabriel Ciobanu
ORCID: https://orcid.org/0000-0002-8166-9456
Academia Europaea, www.ae-info.org/ae/Member/Ciobanu_Gabriel
E–mail: gabriel.ciobanu@iit.academiaromana-is.ro