Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 19 (2013) 1034 – 1039

Procedia Computer Science

The 3rd International Symposium on Frontiers in Ambient and Mobile Systems (FAMS)

Towards a Distributed Plan Execution Monitoring Framework

Yosr Jarraya[1a], Sujoy Ray[a], Andrei Soeanu[a], Mourad Debbabi[a], Mohamad Allouche[b], Jean Berger[b]

[a] *Concordia Institute for Information Systems Engineering*
Concordia University, Montreal, Quebec, Canada
[b] *Defence Research and Development Canada (DRDC) - Valcartier*
Quebec, Canada

Abstract

Distributed monitoring is challenging yet essential in order to address scalability issues observed in the context of large-scale plan execution. A formal framework can be very helpful in analyzing and reasoning about plan specification, execution, and monitoring. In this paper, we elaborate on a distributed monitoring calculus framework that allows specifying and executing plans for multi-agent systems in a distributed environment. The framework allows taking into account a highly dynamic and uncertain environment that can be a contributor to the changing conditions possibly disrupting and causing the plan to fail. Furthermore, the calculus provides sound foundations for designing and evaluating monitoring algorithms and protocols. In order to achieve effective monitoring, we propose an automata-based approach, inspired by runtime security verification research initiatives. The proposed automata allow enforcing monitoring properties while the given plan is executed at the agent's side.

Keywords: Distributed Monitoring, Muti-Agent Systems, Formal Reasoning, Automata, Process Calculus

1. Introduction

Monitoring is closely related automation when compared to other activities such as planning, even if a lot of efforts have been devoted for automatic planning and plan generation over the last decades. In fact, unlike planning, the monitoring process is a time-oriented process. If we consider the monitoring of a process as the detection of any deviation from a "normal" known behavior, then this detection is meaningful only if it happens before a certain point in time. Ideally, the earlier this detection happens, the better. Otherwise, this detection becomes useless since it does not fulfill the very basic objective of any monitoring process: alerting the right user, at the right time, about the right thing, which must be taken into account in building any monitoring capability. The development of such a capability is usually intended for a specific class of users, that is, the output of any monitoring process should take into account the role of the user. Thus, the system will always be presenting the monitoring results at the right level of details. The role of the

[1]Corresponding author. Tel.: +1-514-848-2424; fax: +1-514-848-3171. E-mail: y_ jarray@encs.concordia.ca

doi:10.1016/j.procs.2013.06.144

user should also determine the level of automation of the monitoring process. This is also related to the type of the monitored process. In a multi-agent system, plans are assigned to agents that are distributed over the environment to execute them in order to fulfill some common goals. To effectively coordinate the execution of the tasks, monitors are needed as they represent essential means to report on specific events and information. There exist two approaches to monitoring: centralized and distributed. In a centralized approach, parallel operations are executed in partially observed contexts and feedback is provided to a single system supervisor. In a distributed environment, monitoring is a distributed task among a team of cooperating supervisor agents. In order to achieve an effective plan execution monitoring, it is desirable to have customizable monitors tailored to plans constraints. The goal of this work is to elaborate a formal decision support framework that enforces well-defined monitoring policies on a distributed executing plan.

In this paper, we propose a framework that allows specifying plans and monitored properties generated from pre-compiled monitored policies. The contribution of this paper is as follows: (1) Elaborate a process calculus endowed with an operational semantics that allows specifying plans and executing them in both friendly and hostile environments. (2) Develop a model for execution monitoring enforcement, based on a newly developed type of automata inspired by the well-established security models, such as edit automata [1] and mandatory results automata [2]. (3) Elaborate the semantics of a monitored plan by composing (1) and (2). Different levels of automation can be associated with the different stages of any monitoring process. Automatic detection of deviation is generally recommended in any monitoring process. The response to correct this deviation can be automatic, semi-automatic or manual. The proposed monitoring automata are located within each agent between the executing plan and the real-world. The automaton enforces predetermined monitored properties and may provide appropriate adaptations, if plan deviation can be solved locally. Instances of monitored policies to enforce include illegal actions, action failure, cost modification or increase, order of action execution, plan validity, enforce plan execution and optimality. The remainder of this paper is organized as follows. Section 2 provides an overview of the related work for execution monitoring. Section 3 elaborates on our calculus where its syntax and semantics in adversary environment are presented. Therein, we also define our monitor and describe its semantics and the semantics of the monitored plan. Section 4 presents a case study involving situation of crisis used to demonstrate our approach. We conclude the paper in Section 5 by summarizes our contributions along with the possible future work.

2. Related Work

Al-Shaer [3] presented an active distributed monitoring strategy, in 1999, to define re-configurable and self-directed management tasks. These tasks can be modified automatically at run-time in order to track the system behavior by appropriate placement of a set of predicates to evaluate the expression in runtime. Veloso et al. [4] also introduced the concept of rationale based monitor that captures information about current plan being developed and the alternative choices that are not being pursued. The scope of distributed monitoring [4] lies in verifying sub-goals. Fichtner et al. [5] developed Fluent Calculus to express dynamic domains in first-order logic then this was later implemented as Flux [6]. Fabre et al. [7] discussed on distributed monitoring of plans in the context of concurrent and asynchronous systems using partial unfolding approach over a Petri-net. Lavine [8] studied monitoring algorithms and proposed a set of offline and online monitoring algorithms in the area of robotics. Micalizio [9] elaborated the autonomous recovery of multi-agent plans using control loop of three tasks plan monitoring, agent diagnosis, and the plan repair. Francalanza et al. [10] characterized agents interaction as a result of generating events across the location boundaries and guided by a monitor semantics through labelled transition systems. They formalized the aspect of partitioning monitors based on the locations and captured the evolution of the partitions during execution. A survey of execution monitoring techniques is presented in [13]. Fritz et. al. [14] identified some monitored properties such as: plan feasibility, validity, optimality, etc. and captured them as predicates to verify before and after every action. Table 1 presents a comparison of some of the aforementioned initiatives as well as other research works with respect to the supported architecture and the proposed technique. It also shows the difference between our proposed approach and the state of the art.

Table 1. Comparison Between State-of-the-Art Initiatives and our Proposal

Articles	Architecture	Technique	Type
Al-shaer [3]	Distributed	Logic-based Filtering	Inline
Fabre et al. [7]	Distributed	Petri-Net Unfolding	External
Fritz et al. [11]	Centralized	Action Theory & Situational Calculus	Inline
Lesser & Corkill [12]	Distributed	Knowledge-based Problem Solving	External
Steven J. Levine [8]	Centralized	Algorithmic	External
Ligatti et al. [2]	Centralized	Specialized automata	External
Our Proposal	Distributed	Process calculus & Monitoring automata	External/Inline

P, Q	::=	nil	*inactivity*	α	::=	α_x	*variable*	ϕ	::=	tt	
	|	$(\nu\, n)\, P$	*restriction*		|	$\phi\ ::a.\alpha$	*action prefix*		|	$e\ op\ e$	
	|	$P \mid Q$	*parallel*		|	$\alpha + \alpha$	*choice*		|	ϕ_x	
	|	$!P$	*replication*		|	$\alpha \mid \alpha$	*parallel*		|	$\neg\phi$	
	|	$\pi(\alpha, \phi).P$	*plan execution*		|	$\mu\alpha_x.\alpha$	*recursion*		|	$\phi \wedge \phi$	
	|	$A[P]$	*ambient*								
X	::=	x	*ar var.*	a	::=	ξ	*empty action*	E	::=	e	*ar. expr.*
	|	ϕ_x	*bool. var.*		|	$\overrightarrow{X} := \overrightarrow{E}$	*assign. of expr.*		|	ϕ	*bool. expr.*

Fig. 1. Monitoring Calculus Syntax

3. Distributed Monitoring Calculus

Our calculus is inspired from Mobile Ambients [15, 16]. An ambient [15] is a bounded place with a name and a collection of local processes representing computations running within its boundary. In our context, ambients allow modeling supervisors and agents. Due to lack of space, we provide in the following a subset of our calculus focusing on plan execution and monitoring. Fig. 1 shows the syntax composed of two main categories: process P and plan action α. Predicates denoted by ϕ are built over boolean formulas and they specify conditions including action precondition and goals. We also use boolean and arithmetic expressions E to model action outcomes. An outcome may be seen as changes in the environment variables X due to actions execution, which may also include, for instance, incurred costs.

(ACTION) $\quad (tt\ ::a).\alpha \xrightarrow{a} \alpha$

(CHOICE 1) $\quad \dfrac{\alpha_1 \xrightarrow{a} \alpha_1'}{\alpha_1 + \alpha_2 \xrightarrow{a} \alpha_1'}$

(CHOICE 2) $\quad \dfrac{\alpha_2 \xrightarrow{a} \alpha_2'}{\alpha_1 + \alpha_2 \xrightarrow{a} \alpha_2'}$

(INTERLEAVING-L) $\quad \dfrac{\alpha_1 \xrightarrow{a} \alpha_1'}{\alpha_1 | \alpha_2 \xrightarrow{a} \alpha_1' | \alpha_2}$

(INTERLEAVING-R) $\quad \dfrac{\alpha_2 \xrightarrow{a} \alpha_2'}{\alpha_1 | \alpha_2 \xrightarrow{a} \alpha_1 | \alpha_2'}$

(RECURSION) $\quad \dfrac{\alpha\{\mu\alpha_X.\alpha/\alpha_X\} \xrightarrow{a}_n \alpha'}{\mu\alpha_X.\alpha \xrightarrow{a} \alpha'}$

Fig. 2. Plan's Action Execution Rules

A process P is described using one of the following constructs: *nil* is the process that does nothing, $(\nu\, n)\, P$ denotes creating a new (unique) name n within a scope P, $P \mid Q$ is the parallel composition of two processes P and Q, $!P$ in the unbounded replication of the process P and it is equivalent to $P \mid !P$, $\pi(\alpha, \phi).P$ is the process that executes a plan $\pi(\alpha, \phi)$ defined by its planned actions α and its goal predicate ϕ and then continues as the process P, and finally, $A[P]$ is an ambient named A within which computation P happens. Actions denoted by α are built using the following constructs: α_X is a variable action, $\phi\ ::a.\alpha$ executes a if the preconditions ϕ evaluates to true and then behaves as α, $\alpha|\alpha$ is the parallel composition over actions, and finally $\mu\alpha_X.\alpha$ is the action that behaves as α with the (free) occurrences of α_X replaced by $\mu\alpha_X.\alpha$. An atomic action a is either an empty action or a vector of assignments consisting of integer and boolean expression valuation assigned to variables. If $\overrightarrow{X} = (x_1, x_2, \cdots, x_n)$ and $\overrightarrow{E} = (e_1, e_2, \cdots, e_n)$, $\overrightarrow{X} := \overrightarrow{E}$ is equivalent to the set of simultaneous assignments $\{x_1 := e_1 \cdots x_n := e_n\}$. Note that $op \in \{<, \leq, >, \geq, =, \neq\}$;

In the following, we elaborate on the operational semantics of our calculus, which represents a formal execution framework to specify plans execution and agents behavior. Our operational inference rules are

depicted in Fig. 2. In order to elaborate on the semantics of our calculus, we define first the notion of perceived environment. In a distributed system, agents evolve in independent contexts (that may overlap). We associate with each ambient A its perceived subset of the environment that we denote by $\varepsilon@A$. Agents interact with their respective environment either by executing actions and thus modifying the environment or by updating it (e.g. through sensors). We model an environment as a partial mapping from a set of variables to (either boolean or integers) values, i.e. $\varepsilon = \{x_1 \mapsto v_1, \cdots, x_n \mapsto v_n\}$. Let $\varepsilon(x)$ denote the value associated with the variable x in the environment ε. Let $\varepsilon\{x \mapsto v\}$ denote the new environment ε' where $\forall\ y \in Var(\varepsilon)\ .\ y \neq x$ we have $\varepsilon(y) = \varepsilon'(y)$ and $\varepsilon'(x) = v$. The environment may have a direct impact on the plan execution by either allowing actions to be successfully executed or by acting as an adversary and thus suppressing or inserting unplanned actions. The operational semantics of our calculus is described using the transition relation $\overset{\beta}{\hookrightarrow}$ satisfying inference rules presented in Fig. 3, where $\beta \in \{\alpha, \tau\}$ and τ is the silent action. Plan execution inference rules use the rules of plans' action execution $\xrightarrow{a}$ defined in Fig. 2. Structural congruence and Context inference are omitted for brevity.

$$\text{(ACTION ALLOWED)}\quad \frac{\alpha \xrightarrow{\vec{X} := \vec{E}} \alpha' \qquad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[\pi(\alpha, \phi_G).P] \xrightarrow{\vec{X} := \vec{E}}_n \varepsilon\{\vec{X} \mapsto [\![\vec{E}]\!]\}@n \vdash n[\pi(\alpha', \phi_G).P]}$$

$$\text{(ACTION SUPPRESSED)}\quad \frac{\alpha \xrightarrow{a} \alpha' \qquad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[\pi(\alpha, \phi_G).P] \overset{\tau}{\hookrightarrow}_n \varepsilon@n \vdash n[\pi(\alpha', \phi_G).P]}$$

$$\text{(ACTION INSERTED)}\quad \frac{[\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[\pi(\alpha, \phi_G).P] \xrightarrow{\vec{X} := \vec{E'}}_n \varepsilon\{\vec{X} \mapsto [\![\vec{E'}]\!]\}@n \vdash n[\pi(\alpha, \phi_G).P]}$$

$$\text{(PLAN SUCCESS)}\quad \varepsilon@n \vdash n[\pi(\alpha, tt).P] \overset{\surd}{\hookrightarrow}_n \varepsilon@n \vdash n[P]$$

$$\text{(PLAN FAILURE)}\quad \varepsilon@n \vdash n[\pi(tt :: \xi, ff).P] \overset{\wedge}{\hookrightarrow}_n \varepsilon@n \vdash n[P]$$

Fig. 3. Unmonitored Plan Execution Inference Rules with Environment Influence

Hereafter, we first define our proposed monitoring automaton and then incorporate it within our monitoring calculus in order to describe the semantics of monitored plans executions. The developed monitoring automata is inspired by mandatory results automata [2] and it is defined formally as follows:

Definition 1. *An Adaptive Recovery Monitoring Automaton is a 5-tuple* $M = (Q,\ q_0,\ \Sigma,\ \longrightarrow_m)$*, where: (1)* Q *is the finite or countably infinite set of possible states that is partitioned into two subsets* Q_p *and* Q_{env} *representing the plan and the environment states, respectively. Let* $Q = Q_p \cup Q_{env}$*, (2)* q_0 *is the initial state, (3)* Σ *is the alphabet over which the monitor operates, (4)* $\longrightarrow_m$*:* $Q \times \Sigma \to Q \times \Sigma$ *is the transition function that takes the current state of M and an input action and returns a new state of M and an output action.* □

If the source state is a state of the executing plan, the input is an action from the plan, otherwise it is an input from the environment. If the destination state is a program state, the output is a result output to the plan, otherwise it is an output to the environment. We model a configuration state as the pair (q, d) where $q = q_p \in Q_p$ or $q = q_{env} \in Q_{env}$ and d can be input/output of action denoted a or result denoted by r. We use subscripts $-_i$ and $-_o$ on top of both actions and results in order to denote input and output, respectively. The operational semantics of the monitor is defined in terms of single-step judgment of the form $(q, d) \xrightarrow{d}_m (q', d'), d \in \Sigma$. The inference rules defining all possible monitor transitions are as follows:

$$\frac{\alpha \xrightarrow{a} \alpha'}{(q_p, r) \xrightarrow{a_i}_m (q'_p, a)}\ \text{(input action)} \qquad \frac{next_{env} = r}{(q_{env}, a)_{env} \xrightarrow{r_i}_m (q'_{env}, r)}\ \text{(input result)}$$

$$(q_p, a) \xrightarrow{a_o}_m (q_{env}, a)\ \text{(output action)} \qquad (q_{env}, r) \xrightarrow{r_o}_m (q_p, r)\ \text{(output env result)}$$

$$(q_p, a) \xrightarrow{r_o}_m (q'_p, r)\ \text{(output M result)}$$

Before providing monitored plan semantics, we need to introduce the generalization of the transition function in the standard way and we write $(q, d) \overset{\sigma}{\Longrightarrow}_m (q', d')$, or simply $M \overset{\sigma}{\Longrightarrow}_m M'$, as the reflexive, transitive closure of the single-step transition, where σ represent a sequence of steps. Similarly, the plan semantics rules can be generalized using $\Rightarrow_n$. We also define a new operator that is a monitor operator denoted by $M \rhd \pi(\alpha, \phi)$. The monitored plan execution rules are presented in Fig. 4.

$$\text{(R1-ALLOWED AND SUCCESSFUL)}\quad \frac{\alpha \xrightarrow{a} \alpha' \qquad a = \overrightarrow{X} := \overrightarrow{E} \qquad M \overset{a_i.\sigma.a_o}{\Longrightarrow}_m M' \quad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[M \rhd \pi(\alpha,\phi_G).P] \overset{\overrightarrow{X}:=\overrightarrow{E}}{\Longrightarrow}_n \varepsilon\{\overrightarrow{X} \mapsto [\![\overrightarrow{E}]\!]\}@n \vdash n[M' \rhd \pi(\alpha',\phi_G).P]}$$

$$\text{(R2-ALLOWED AND FAILURE)}\quad \frac{\alpha \xrightarrow{a} \alpha' \qquad a = \overrightarrow{X} := \overrightarrow{E} \qquad M \overset{a_i.\sigma.(\overrightarrow{X}:=\overrightarrow{E'})_o}{\Longrightarrow}_m M' \quad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[M \rhd \pi(\alpha,\phi_G).P] \overset{\overrightarrow{X}:=\overrightarrow{E}}{\Longrightarrow}_n \varepsilon\{\overrightarrow{X} \mapsto [\![\overrightarrow{E'}]\!]\}@n \vdash n[M' \rhd \pi(\alpha',\phi_G).P]}$$

$$\text{(R3-SUPPRESSED)}\quad \frac{\alpha \xrightarrow{a} \alpha' \qquad a = \overrightarrow{X} := \overrightarrow{E} \qquad M \overset{a_i.\tau}{\Longrightarrow}_m M' \quad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[M \rhd \pi(\alpha,\phi_G).P] \overset{\tau}{\Longrightarrow}_n \varepsilon@n \vdash n[M' \rhd \pi(\alpha',\phi_G).P]}$$

$$\text{(R4-INSERTED)}\quad \frac{M \overset{r_i.r_o}{\Longrightarrow}_m M' \quad [\![\phi_G]\!] = ff}{\varepsilon@n \vdash n[M \rhd \pi(\alpha,\phi_G).P] \overset{r_0}{\Longrightarrow}_n \varepsilon'@n \vdash n[M \rhd \pi(\alpha,\phi_G).P]}$$

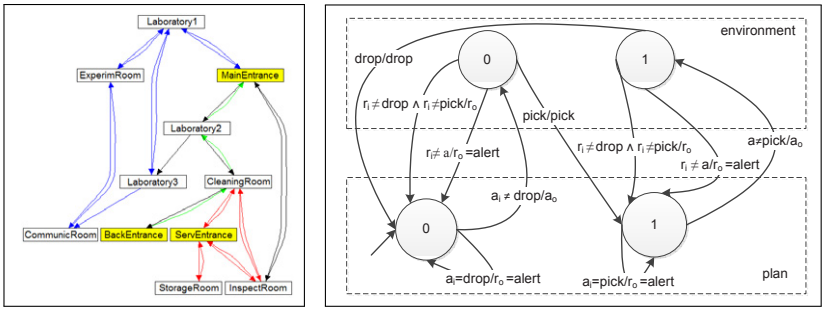
Fig. 4. Monitored Plan Semantics

4. Case Study

We consider a situation of crisis [17] that involves an accident on the premises of a research facility. Since the full extent of the damage (e.g. contamination level) is unknown, the main goal is to collect evidence on the premises of the research facility by sending robotic agents. Fig. 5(a) presents the schematic description of the research facility. The research facility has a main entrance (ME), a back entrance (BE) and a service entrance (SE). The facility also has three labs (L1), (L2), and (L3). Moreover there is a storage room (SR), a cleaning room (CR), an inspection room (IR), an experimental room (ER) and a communications room (TR). The global goal of the plan is to collect a piece of evidence from all the rooms that are not entrances and to drop these pieces at the entrances. The plan involves three distributed robotic agents initially respectively positioned at (ME), (BE), and (SE). We assume that the agent's actions capabilities are $\sigma = \{move(Loc), pick, drop\}$, where Loc is the destination location. We also assume a constraint for the robots that they cannot collect more than one piece of evidence at a time. For instance, after a piece of evidence is collected, one of the robots has to return back to SE, drop the evidence and then visit the unvisited rooms, namely Cleaning Room (CR), Inspect Room (IR) and StorageRoom (SR), until it collects and drops at SE a piece of evidence from each room. Using our calculus, A_3 plan can be described as follows:
$\pi(move(SE).\mu X(((\neg v(CR)) :: move(CR) + (\neg v(IR)) :: move(IR) + (\neg v(SR)) :: move(SR)).$
$pick.move(SE).drop.((\neg\phi_G) :: X + \xi))), \phi_G)$ where ϕ_G is the predicate denoting that all evidences have been collected from the rooms and have been dropped at the service entrance. In order to monitor the execution of the plan, the tactical team decided to monitor that the collection and the dropping of evidences are correctly executed. Since the agent can collect only one piece at a time, the first property P_M can be to check/prevent the agent from making two consecutive *pick* actions if no *drop* action have been performed in between. Monitor automata for the property P_M is illustrated in Fig. 5(b). In order to illustrate the impact of the monitor on the plan execution, we provide the following trace obtained by the execution of the unmonitored plan: $\sigma = move(SE).move(SR).pick.pick.move(SE).drop.$
$move(IR).pick.move(CR).move(SE).drop.drop.move(CR).pick.drop.move(SE)$. We suppose that the first *pick* of the trace actually failed in the environment. By executing the operational semantics of monitored plan execution, we obtain the following output trace: $move(SR).move(SE).alert.pick.move(SE).drop.move(IR).$
$pick.move(CR).move(SE).drop.alert.move(CR).pick.drop.move(SE)$.

5. Conclusion

In this paper, we proposed a process calculus for capturing and reasoning about plan execution monitoring performed by distributed agents. We have shown that by composing the plan and monitoring process it is possible to ensure that the execution respects pre-established monitoring properties. To this end, we presented how the plan dynamics can be formally captured using the syntax and the operational semantics of the proposed distributed monitoring calculus. The latter can be used as a foundation for a formal decision



(a) Robotic Agents Case Study (b) Monitor Automata for the Property P_M

Fig. 5. Adaptive Distributed Monitoring

support framework to enforce well-defined properties over a plan executed by distributed agents. As future work, we aim at elaborating a weaving procedure whereby the plan and the monitoring processes captured can be combined such that a plan can be self monitored during its execution.

Acknowledgements

This research is a result of a fruitful collaboration with Defence Research and Development Canada at Valcartier, Department of National Defense, MDA Corporation and Concordia University under the NSERC DND Research Partnership Program.

References

[1] J. Ligatti, L. Bauer, D. Walker, Edit automata: Enforcement mechanisms for run-time security policies, International Journal of Information Security 4 (1) (2005) 2–16.
[2] J. Ligatti, S. Reddy, A theory of runtime enforcement, with results, in: Proceedings of the 15th European conference on Research in computer security, ESORICS'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 87–100.
[3] E. S. Al-shaer, Programmable agents for active distributed monitoring, in: Tenth IFIP/ IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99, 1999.
[4] M. M. Veloso, M. E. Pollack, M. T. Cox, Rationale-based monitoring for planning in dynamic environments, in: AIPS, 1998, pp. 171–180.
[5] M. Fichtner, A. Grossmann, M. Thielscher, Intelligent execution monitoring in dynamic environments, Fundam. Inf. 57 (2-4) (2003) 371–392.
[6] M. Thielscher, Reasoning Robots: The Art and Science of Programming Robotic Agents (Applied Logic Series), Springer Netherlands, 2010.
[7] E. Fabre, A. Benveniste, S. Haar, C. Jard, Distributed monitoring of concurrent and asynchronous systems, Discrete Event Dynamic Systems 15 (1) (2005) 33–84.
[8] S. J. Levine, Monitoring the execution of temporal plans for robotic systems, Master's thesis, Massachusetts Institute of Technology (2011).
[9] R. Micalizio, A distributed control loop for autonomous recovery in a multi-agent plan, in: IJCAI, 2009, pp. 1760–1765.
[10] A. Francalanza, A. Gauci, G. J. Pace, Distributed system contract monitoring, in: FLACOS, 2011, pp. 23–37.
[11] C. Fritz, Monitoring the execution of optimal plans, in: The 17th International Conference on Automated Planning and Scheduling (ICAPS) Doctoral Consortium, 2007.
[12] V. Lesser, D. Corkill, The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks, AI Magazine 4 (3) (1983) 15–33.
[13] O. Pettersson, Execution monitoring in robotics: A survey, Robotics and Autonomous Systems 53 (2005) 73–88.
[14] C. Fritz, S. McIlraith, Monitoring plan optimality during execution, in: ICAPS, 2007.
[15] L. Cardelli, A. D. Gordon, Mobile Ambients, Theoretical Computer Science 240 (1) (2000) 177 – 213.
[16] Y. Jarraya, A. Eghtesadi, M. Debbabi, Y. Zhang, M. Pourzandi, Cloud calculus: Security Verification in Elastic Cloud Computing Platform, in: 2012 International Conference on Collaboration Technologies and Systems, CTS 2012, Denver, CO, USA, May 21-25, 2012, 2012, pp. 447–454.
[17] A. Soeanu, S. Ray, M. Debbabi, J. Berger, A. Boukhtouta, Multi-Agent Service Delivery Tour Planning Using Model Checking, in: the Proceedings of the International Conference on Information Systems, Logistics and Supply Chain, 2012.