Monitoramento de SMA Normativos – Uma Implementação para Agentes Reativos Simples em JAMDER 2.0

Francisco I. S. Cruz , Emmanuel S. S. Freire, Mariela I. Cortés e Nécio L. Veras

Grupo de Engenharia e Sistemas Inteligentes (GESSI)
Departamento de Computação – Universidade Estadual do Ceará (UECE)
Avenida Paranjana, 1700 – 60740-000 – Fortaleza – CE – Brazil

{israel.santos.cr, savio.essf, necioveras}@gmail.com,
mariela@larces.uece.br

Abstract. *Norms in multi-agent systems are used in order to regulate the behavior of agents to archive the system global goal. For this, the existence of a mechanism that allows to determine the performance of the agents in relation of the fulfillment of the system norms is essential. The present paper presents the implementation, in JAMDER 2.0, of an abstract architecture that allows the monitoring of the agent's behavior in relation to the fulfillment or the violation of norms. A case study considering the simple reactive agents in the vacuum cleaner world is presented to illustrate the approach of enforcement was implemented in this paper.*

Resumo. *Normas em sistemas multi-agente são utilizadas para regular o comportamento dos agentes de forma a alcançar os objetivos globais do sistema. Para isso, é crucial a existência de um mecanismo que permita determinar o desempenho dos agentes em relação ao cumprimento das normas especificadas no sistema. O presente artigo apresenta a implementação, em JAMDER 2.0, de uma arquitetura abstrata que possibilita o monitoramento do comportamento dos agentes em relação ao cumprimento ou violação de normas. Um estudo de caso considerando agentes reativos simples para o mundo do aspirador de pó é apresentado para ilustrar a abordagem de coesão (enforcement) implementada neste artigo.*

1. Introdução

Um sistema multi-agente (SMA) pode ser interpretado como uma sociedade onde entidades autônomas e heterogêneas podem cooperar ou competir para alcançar seus objetivos. A fim de lidar com a heterogeneidade nestes sistemas, mecanismos de governança são definidos por meio de um conjunto de normas que guiam o comportamento das entidades do sistema. Quando normas são inseridas para restringir as ações dos agentes por meio dos conceitos deônticos de obrigação, proibição e permissão [Figueiredo; Silva 2010], caracteriza-se um sistema multi-agente normativo (SMAN).

As normas são utilizadas para regular e coordenar o comportamento dos agentes [Modgil et al. 2009] e influenciam nos processos de tomada de decisão, no comportamento racional e, consequentemente, no desempenho do agente. As normas podem ser interpretadas como padrões de comportamento [Dybalova et al. 2013].

Portanto, é necessário observar o comportamento dos agentes diante do cumprimento e/ou violação das normas que regem o ambiente que tais agentes estão inseridos. Neste contexto, existem abordagens para implementar normas em SMAN. As principais são: regimentação (*regimentation*) [Jones; Sergot 1993] e coesão (*enforcement*) [Dastani et al. 2008].

Na regimentação, o comportamento do agente é restringido pelas normas, diminuindo a sua autonomia, pois o agente só pode executar as atividades permitidas pelas normas, eliminando consequentemente estados ou comportamentos proibidos pelas normas. Em contrapartida, as normas definidas na coesão podem influenciar o comportamento do agente informando as ações que podem, devem ou não podem ser executadas. Com isso, a última abordagem pode ser considerada mais flexível uma vez que o agente não é impedido de violar as normas, no entanto requer de um mecanismo normativo adicional para o monitoramento do comportamento dos agentes em relação ao cumprimento ou não das normas.

Em Modgil et al. (2009), é apresentada uma arquitetura genérica para a observação do comportamento dos agentes em sistemas multi-agente normativos, de forma a determinar se estão cumprindo ou violando as normas estabelecidas. Esta arquitetura prevê monitores que recebem informações de observadores, e processam estas informações através de redes de transição de normas individuais, de forma a determinar o cumprimento ou a violação de tais normas.

Em contrapartida, o *framework* JAMDER 2.0 [Cruz et al. 2014] permite a implementação de todos os elementos que compõem os sistemas multi-agente normativos, porém não contém a abordagem de coesão. O presente artigo implementa a abordagem de coesão utilizando o *framework* JAMDER 2.0 para o caso de agentes reativos simples e avalia o comportamento desses agentes em um sistema a partir de uma simulação para o mundo do aspirador de pó. Este trabalho está organizado da seguinte maneira: a Seção 2 apresenta os conceitos relacionados com o monitoramento em sistema multi-agente normativos. Em seguida, na Seção 3, é descrita a implementação da abordagem coesão utilizando o *framework* JAMDER 2.0. Um estudo de caso utilizando agente reativos simples no mundo do aspirador de pó é apresentado na Seção 4 e, finalmente, as conclusões e trabalhos futuros são descritos na Seção 5.

2. Monitoramento de Sistemas Multi-agente Normativos

A abordagem de coesão ou *enforcement* pode ser efetivamente utilizada para regular o comportamento dos agentes em um sistema multi-agente normativo desde que seja fornecido um mecanismo de monitoramento que possibilite o reconhecimento do cumprimento ou não das normas no ambiente [Modgil et al. 2009], de forma a viabilizar a aplicação de sanções de punição ou recompensa.

A abordagem para monitoramento utilizada no *framework* de Modgil et al. (2009) é baseada na técnica de *overhearing* [Kaminka; Pynadah; Tambe 2012], que consiste em observar a troca de mensagens entre os agentes com o objetivo de inferir o comportamento dos mesmos, em contrapartida a abordagens intrusivas [Jennings 1995] [Mazouzi; Seghrouchni; Haddad 2002] [Tambe 1997] as quais pressupõem que os estados mentais dos agentes são acessíveis à inspeção.

Em resumo, a técnica contempla a existência dos seguintes aspectos em relação ao monitoramento de comportamentos de agentes governados por normas:

- Modelo observador das mensagens dos agentes e demais estados de interesse de forma a prover subsídios para a aplicação adequada dos mecanismos de coesão.
- Normas são individualmente representadas por ATNs (*Augmented Transition Network,* um grafo que representa os três estados que uma norma pode assumir) independentes utilizadas para o monitoramento de comportamentos complexos de agentes em conjunto.
- Monitoramento de normas em sistemas multi-agente dinâmicos e abertos.

De acordo com a abordagem apresentada em Modgil et al. (2009), os agentes são tratados como caixas pretas e as transições de estados internos são invisíveis ao monitor. Um *mapper* mapeia as normas para representações ATN, as quais irão alimentar o processamento do monitor. Em tempo de execução, o monitor reporta ao mecanismo de observação sobre os estados ou ações de interesse identificados a partir dos componentes das normas.

Ainda em tempo de execução, o mecanismo de observação notifica o monitor sempre que for requerida a aplicação de uma sanção. Esse mecanismo considera a observação do comportamento dos agentes em relação aos estados e ações sendo regulamentados pelas normas segundo estabelecido anteriormente.

O monitor processa esta informação levando em conta a representação da ATN da norma para determinar se a norma se encontra ativa, foi atendida ou violada, ou expirou. Finalmente, o monitor sinaliza sobre a aplicação de uma sanção a um determinado agente, se for apropriado.

3. Implementação do *Framework* de Monitoramento em JAMDER 2.0

O *framework* JAMDER 2.0 contempla todos os elementos típicos que compõem um sistema multi-agente normativo. Entretanto, esse *framework* não possui um mecanismo para verificar se os agentes cumpriram ou não as normas definidas no contexto de uma organização, suborganização ou ambiente.

O conceito de normas utilizado em JAMDER 2.0 se refere ao monitoramento das ações dos agentes de forma a guiar a execução de ações. Com isso, as normas definidas em JAMDER 2.0 não garantem que as ações proibidas não sejam executadas nem que as ações obrigatórias sejam executadas. Portanto, a abordagem de coesão é mais adequada ao *framework*.

A solução implementada em JAMDER 2.0 contempla um agente de monitoramento, definido a partir da classe abstrata *jamder.monitoring.Monitor* que herda de jamder.agents.*GenericAgents* e que possui o método *percept(Object, Object)*. Esse método é chamado pelo agente vinculado ao monitor (instância de *jamder.monitoring.Monitor*) sempre que ele executa uma ação. Dessa forma, torna-se possível o monitoramento dos agentes considerando as normas do ambiente.

A classe *jamder.monitoring.Monitor* possui como atributos um conjunto de agentes (monitorados) e as ATNs que são geradas por meio das normas que restringem os agentes que são adicionados, ou quando uma nova norma passa a restringir um agente

que está sendo monitorado. A definição do comportamento do agente de monitoramento fica a cargo do usuário. Isso é importante, pois a literatura costuma divergir sobre como e quando as sanções devem ser aplicadas ao agente monitorado [Piunti et al., 2010]. A estrutura de monitoramento no *framework* JAMDER 2.0 é ilustrada na Figura 1.

Adicionalmente, o monitoramento em um sistema normativo requer uma abstração para representar os estados possíveis de uma norma. Para isso, utilizam-se *Augmented Transition Networks* (ATNs) [MODGIL et al. 2009]. Uma ATN é um grafo que representa os três estados que uma norma pode assumir. São eles: *INACTIVE*, *ACTIVE* e *COMPLIANCE-OR-INFRINGEMENT*. Para incorporar esse conceito em JAMDER 2.0, foram criados a classe *jamder.monitoring.ATN* e o *enumeration jamder.monitoring.ATNState* que representam as ATNs e seus estados, respectivamente. O *enumeration* é do tipo *String* e contém os três estados de uma ATN.

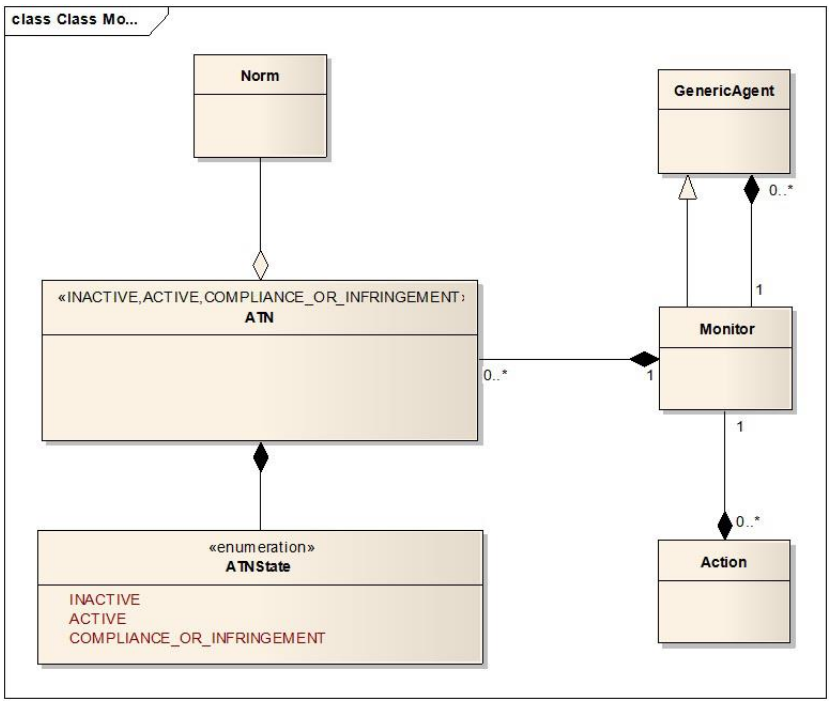


Figura 1. Estrutura de monitoramento no JAMDER 2.0

3.1 Agente monitor

As principais funções do agente de monitoramento são: (i) avaliar se um agente cumpriu ou violou uma norma e (ii) aplicar as sanções relacionadas às normas violadas ou cumpridas, quando apropriado.

A classe *jamder.monitoring.Monitor* (Figura 1) possui os seguintes métodos:

- **getAgents()** - que retorna os agentes que estão sendo monitorados pelo monitor;
- **addAgent(String, GenericAgent)** - que é utilizado para adicionar um agente que será monitorado;
- **addATN(Norm)** - adiciona um ATN por meio de uma norma. Este método é usado no método apply() da classe jamder.norms.Norm. Quando uma norma passa a

restringir um agente monitorado, uma nova ATN é criada para monitorar esse agente;

- **removeATN(String)** - remove uma ATN por meio de sua chave correspondente. Este método é usado no método disapply() da classe jamder.norms.Norm. Quando uma norma deixa de restringir um agente monitorado, a ATN correspondente é removida;
- **getAllAtns()** - que retorna a Hashtable que contém as ATNs vinculadas às normas que restringem os agentes que estão sendo monitorados;
- **punish(Norm)** - que aplica a punição (se ela existir) vinculada à norma que recebe como parâmetro;
- **reward(Norm)** - que aplica a recompensa (se ela existir) vinculada à norma que recebe como parâmetro.
- **percept(Object, Object)** - um método abstrato que é uma implementação do padrão GoF *Observer* [Vlissides et al. 1995]. Quando uma ação é executada, o agente envia um sinal para seu monitor através deste método.

A Figura 2 apresenta a classe abstrata do agente de monitoramento.

4. Estudo de caso

Nesta seção, é apresentado um estudo de caso baseado no Mundo do Aspirador de Pó Normativo. Tanto os agentes aspirador de pó como o ambiente foram implementados utilizando JAMDER 2.0, contemplando o agente monitor descrito na Seção 3. Este estudo de caso foi utilizado por Campos, Freire e Cortés (2012) para ilustrar sua abordagem.

Considerando um Mundo do Aspirador de Pó Normativo com somente duas salas, onde cada sala pode estar limpa ou suja, em nossos experimentos, a informação percebida do ambiente foi representada por estados do ambiente. Adicionalmente, o mundo possui normas que restringem o comportamento dos agentes para executar suas atividades. O Mundo do Aspirador de Pó Normativo é composto por:

- ***Place*** - um objeto que é um grafo de dois vértices. Estes vértices denotam a sala: *roomA* e *RoomB*;
- ***CleanserOrg*** - uma organização que possui os seguintes papéis de agente: *management e cleaner. O* primeiro é associado ao *ManagerAgent* (agente de monitoramento) e o outro é associado ao *VacuumCleaner* (agente aspirador);
- ***ManagerAgent*** **-** um agente de monitoramento que utiliza o papel de management na organização *CleanserOrg*. Sua função é monitorar o agente *VacuumCleaner* e modificar seu comportamento de acordo com as normas do ambiente. Suas ações são: *ActionMonitorCyclic* e *ActionMonitorReflex*. A implementação do agente é apresentada na Figura 3;
- ***NormativeVacuumCleaner*** **-** um agente reativo simples normativo (Campos, Freire and Cortés 2012) que utiliza o papel *cleaner* na organização

CleanserOrg. Suas ações são: *Right*, *Left*, *Suck* e sua percepção é *NextAction*. A implementação do agente é apresentada na Figura 4;

- ***VacuumCleaner*** - um agente reativo simples que utiliza o papel *cleaner* na organização *CleanserOrg*. Suas ações são: *Right*, *Left*, *Suck* e sua percepção é *NextAction*. A implementação do agente é apresentada na Figura 4.

```
public abstract class Monitor extends GenericAgent {
    private static final long serialVersionUID = 1L;
    private Hashtable<String, ATN> atns=new Hashtable<String, ATN>();
    private Hashtable<String, GenericAgent> agents=new Hashtable<String, GenericAgent>();


    public Monitor(String name, Environment environment, Organization owner) {
        super(name, environment, null);
        if (owner!=null && owner.getName()!=null)
            addOrganization(owner.getName(), owner);
    }

    public Hashtable<String, GenericAgent> getAgents() {
        return agents;
    }

    public void addAgent(String key, GenericAgent agent) {
        this.agents.put(key, agent);
        for (Norm nor: agent.getAllRestrictNorms().values()){
            ATN atn=new ATN(nor, nor.getName());
            atns.put(atn.getName(), atn);
        }
    }

    public void addATN(Norm nor){
        if (!atns.containsKey(nor.getName())){
            ATN atn=new ATN(nor, nor.getName());
            atns.put(atn.getName(), atn);
        }
    }
    public void removeATN(String key){
        atns.remove(key);
    }

    public Hashtable<String, ATN> getAllAtns() {
        return atns;
    }

    public void punish(Norm norm){
        Hashtable<String, Norm> punishments=norm.getSactionPunishment();
        for (Norm punishment: punishments.values()){
            if (punishment.isApply())
                punishment.disapply();
            Object context= norm.getContext();
            Object restrict=norm.getRestrict();
            punishment.setContext(context);
            punishment.setRestrict(restrict);
            punishment.apply();
        }
    }

    public void reward(Norm norm){
        Hashtable<String, Norm> rewards=norm.getSactionReward();
        for (Norm reward: rewards.values()){
            if (reward.isApply())
                reward.disapply();
            Object context= norm.getContext();
            Object restrict=norm.getRestrict();
            reward.setContext(context);
            reward.setRestrict(restrict);
            reward.apply();
        }
    }


    public abstract void percept(Object perception1, Object perception2);


}
```

Figura 2. Classe abstrata do agente de monitoramento.

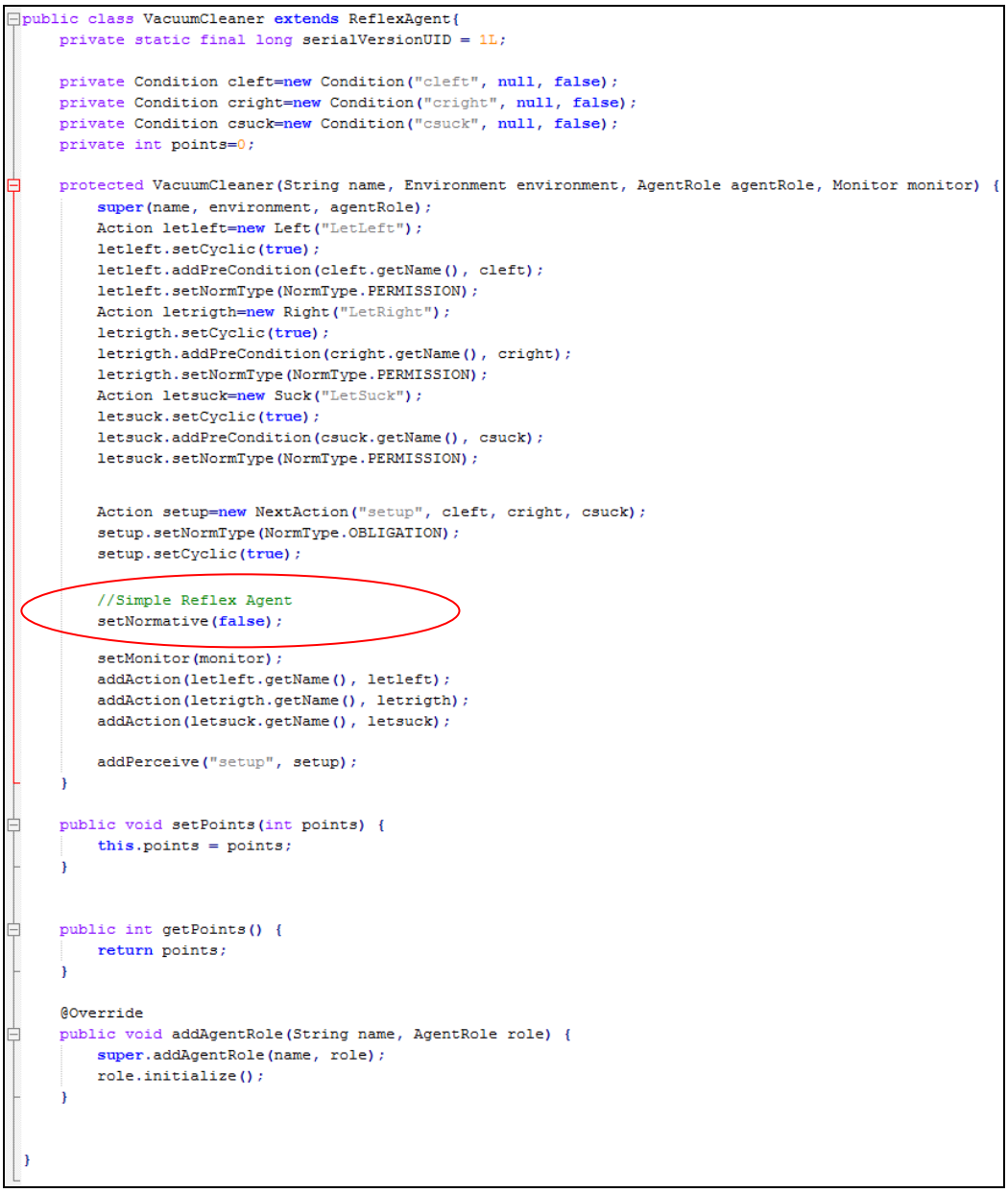
Adicionalmente, o mundo possui as seguintes normas para restringir o comportamento dos agentes para executar suas atividades:

- **N1** – O agente é obrigado a limpar a *roomB*;
- **N2** – O agente é obrigado a limpar a *roomA*. Sanção de recompensa da norma N1. Quando a norma N1 é cumprida, N2 é aplicada ao agente que nesse momento ganha 3 pontos por cumprir a norma N1.

No início de cada experimento, tanto o ***NormativeVacuumCleaner*** como o ***VacuumCleaner*** não conhecem a configuração do mundo em termos de salas sujas. Nós consideramos que quando o mundo está sem a presença de normas, a medida de desempenho oferece uma recompensa de um ponto para cada sala limpa (+1) e penaliza com a perda de um ponto para cada movimento (-1). No caso da presença de normas no mundo, a medida de desempenho precisa ser adaptada para considerar as recompensas (+pontos) e as penalidades (-pontos), que são consequências da escolha do agente em seguir ou rejeitar alguma norma.

```
public class ManagerAgent extends Monitor {


    private static final long serialVersionUID = 5670776386732896828L;

    public ManagerAgent(String name, Environment environment, Organization owner) {
        super(name, environment, owner);
        Action ac=new ActionMonitorCyclic("ActionMonitorCyclic");
        addAction("ActionMonitorCyclic",ac);

        //------Definição de papel do agente de monitoramento-----
        AgentRole ar=new AgentRole("management", owner, this);
        ar.addAction("ActionMonitorCyclic",ac);

        Calendar date1a=new GregorianCalendar(2014, GregorianCalendar.JANUARY, 29, 15, 8, 20);
        Calendar date2b=new GregorianCalendar(2014, GregorianCalendar.MARCH, 29, 15, 9, 40);
        NormConstraint cond=new Between(date1a, date2b);
        Hashtable<String, NormConstraint> constraints=new Hashtable<String, NormConstraint>();
        constraints.put("cond", cond);

        NormResource nre= new NormResource(ac);
        NormAction noa= new AtomicAction(AtomicActionType.AtomicExecute, nre);

        Norm no=new Norm("N1", NormType.OBLIGATION, ar, owner, noa, constraints);
        ar.addRestrictNorm(no.getName(), no);

        //Inicialização
        addAgentRole("management", ar);
        ar.initialize();
    }

    @Override
    public void percept(Object perception1, Object perception2) {
        if (perception1 instanceof ReflexAgent && perception2 instanceof Action){
            addBehaviour(new ActionMonitorReflex("ActionMonitor", (ReflexAgent)perception1, (Action)perception2));
        }
    }
}
```

Figura 3. ManagerAgent.



```
public class VacuumCleaner extends ReflexAgent{
    private static final long serialVersionUID = 1L;

    private Condition cleft=new Condition("cleft", null, false);
    private Condition cright=new Condition("cright", null, false);
    private Condition csuck=new Condition("csuck", null, false);
    private int points=0;

    protected VacuumCleaner(String name, Environment environment, AgentRole agentRole, Monitor monitor) {
        super(name, environment, agentRole);
        Action letleft=new Left("LetLeft");
        letleft.setCyclic(true);
        letleft.addPreCondition(cleft.getName(), cleft);
        letleft.setNormType(NormType.PERMISSION);
        Action letrigth=new Right("LetRight");
        letrigth.setCyclic(true);
        letrigth.addPreCondition(cright.getName(), cright);
        letrigth.setNormType(NormType.PERMISSION);
        Action letsuck=new Suck("LetSuck");
        letsuck.setCyclic(true);
        letsuck.addPreCondition(csuck.getName(), csuck);
        letsuck.setNormType(NormType.PERMISSION);


        Action setup=new NextAction("setup", cleft, cright, csuck);
        setup.setNormType(NormType.OBLIGATION);
        setup.setCyclic(true);

        //Simple Reflex Agent
        setNormative(false);

        setMonitor(monitor);
        addAction(letleft.getName(), letleft);
        addAction(letrigth.getName(), letrigth);
        addAction(letsuck.getName(), letsuck);

        addPerceive("setup", setup);
    }

    public void setPoints(int points) {
        this.points = points;
    }


    public int getPoints() {
        return points;
    }

    @Override
    public void addAgentRole(String name, AgentRole role) {
        super.addAgentRole(name, role);
        role.initialize();
    }


}
```

Figura 4. Quando o atributo *setNormative* é *true*, tem-se o NormativeVacuumCleaner. Caso contrário, tem-se o *VacuumCleaner*.

5. Experimentos

Uma vez definidas as entidades do estudo de caso seguiram-se os experimentos nos quais, incluímos os agentes *VacuumCleaner* e *NormativeVacuumCleaner* em mundos específicos com a presença ou não de normas.

5.1 *VacuumCleaner* em ambiente com normas inativas

Primeiramente, o agente *VacuumCleaner* foi inserido em um Mundo Aspirador de Pó sem normas. A Tabela 1 apresenta os resultados deste cenário. Observa-se que, como

não há normas, a pontuação só decresce quando o agente de desloca de uma sala a outra e cresce quando ele limpa uma sala suja.

Tabela 1 - Execução do agente *VacuumCleaner* em um ambiente com normas inativas.

	Estado			Ação	Pontos
	Onde está?	Estado roomA	Estado roomB		
1	*roomA*	Sujo	Sujo	*suck*	1
2	*roomA*	Limpo	Sujo	*right*	0
3	*roomB*	Limpo	Limpo	*suck*	1
4	*roomA*	Limpo	Limpo	*left*	0
5	*roomA*	Limpo	Limpo	*right*	-1
6	*roomB*	Limpo	Limpo	*left*	-2
7	*roomA*	Limpo	Limpo	*right*	-3

5.2 VacuumCleaner em ambiente com normas ativas

Em seguida, o agente *VacuumCleaner* foi inserido em um Mundo do Aspirador de Pó Normativo com a norma N1 ativa. A Tabela 2 apresenta os resultados desse cenário.

Tabela 2 - Execução do agente *VacuumCleaner* em um ambiente com normas ativas.

	Estado			Ação	Pontos
	Onde está?	Estado roomA	Estado roomB		
1	*roomA*	Sujo	Sujo	*suck*	1
2	*roomA*	Limpo	Sujo	*right*	0
3	*roomB*	Limpo	Limpo	*suck*	4
4	*roomA*	Limpo	Limpo	*left*	3
5	*roomA*	Limpo	Limpo	*right*	2
6	*roomB*	Limpo	Limpo	*left*	1
7	*roomA*	Limpo	Limpo	*right*	0

As linhas 1 e 2 descrevem o comportamento do agente no período que a norma está inativa. A linha 3 da tabela (destacada) ilustra o comportamento do agente quando a norma N1 está ativa, informando que o agente foi recompensado com três pontos por ações executadas durante esse período juntamente com um ponto por ter limpado a sala. Nas linhas 4 a 7, as normas foram desativadas.

5.3. NormativeVacuumCleaner em ambiente com normas inativas

Primeiramente, o agente *NormativeVacuumCleaner* foi inserido em um Mundo Aspirador de Pó sem normas. A Tabela 3 apresenta os resultados deste cenário. Observa-se que, como não há normas, a pontuação só decresce quando o agente de desloca de uma sala a outra e cresce quando ele limpa uma sala suja. O mesmo resultado é apresentado pelo VacuumCleaner quando inserido no mesmo ambiente com normas inativas.

Tabela 3 - Execução do agente *NormativeVacuumCleaner* em um ambiente com normas inativas.

	Estado			Ação	Pontos
	Onde está?	Estado roomA	Estado roomB		
1	*roomA*	Sujo	Sujo	*suck*	1
2	*roomA*	Limpo	Sujo	*right*	0
3	*roomB*	Limpo	Limpo	*suck*	1
4	*roomA*	Limpo	Limpo	*left*	0
5	*roomA*	Limpo	Limpo	*right*	-1
6	*roomB*	Limpo	Limpo	*left*	-2
7	*roomA*	Limpo	Limpo	*right*	-3

5.4. NormativeVacuumCleaner em ambiente com normas ativas

Em seguida, o agente *NormativeVacuumCleaner* foi inserido em um Mundo do Aspirador de Pó Normativo com a norma N1 ativa. A Tabela 4 apresenta os resultados desse cenário.

Tabela 4 - Execução do agente *NormativeVacuumCleaner* em um ambiente com normas ativas.

	Estado			Ação	Pontos
	Onde está?	Estado roomA	Estado roomB		
1	*roomA*	Sujo	Sujo	*suck*	1
2	*roomA*	Limpo	Sujo	*right*	0
3	*roomB*	Limpo	Limpo	*suck*	4
4	*roomA*	Limpo	Limpo	*suck*	3
5	*roomA*	Limpo	Limpo	*suck*	4
6	*roomA*	Limpo	Limpo	*suck*	5
7	*roomA*	Limpo	Limpo	*suck*	6

A norma N1 está ativa entre as linhas 1 e 3. Na linha 3, o agente monitor aplica a sanção de recompensa vinculada à norma N1 que é a norma N2. A partir da linha 4, a única norma ativa é N2. Através da aplicação de uma sanção, ou seja, a partir momento que a norma N2 foi aplicada ao agente (N2 passou a restringi-lo), o gente alterou seu comportamento enquanto a norma N2 esteve ativa. Uma modificação no ambiente por parte do monitor atuou redirecionando esse agente.

6. Conclusão e Trabalhos Futuros

A influência das normas na tomada de decisão dos agentes é essencial para regular e monitorar o comportamento de tais agentes quando inseridos em ambientes normativos. Neste contexto, esse artigo apresenta uma adaptação do *framework* JAMDER 2.0 para possibilitar a implementação de um agente de monitoramento seguindo os conceitos relacionados na abordagem de coesão ou *enforcement*. Adicionalmente, um exemplo baseado no Mundo do Aspirador de Pó Normativo foi utilizado para ilustrar a utilização da adaptação do JAMDER 2.0, demonstrando sua aplicabilidade para o desenvolvimento de sistemas multi-agente normativos.

Por meio do estudo de caso, pode-se perceber que os agentes reativos normativos quando inseridos em ambientes que não possuem normas ativas, se comportam como agentes reativos normais. Entretanto, quando inseridos em ambientes com normas ativas, a coesão (*enforcement*) garante que os agentes terão seu comportamento monitorado por alguma entidade do sistema. Com isso, a adaptação de JAMDER 2.0 permite a comparação do desempenho dos agentes quando inseridos ou não em ambientes normativos. Adicionalmente, o agente de monitoramento pode ser estendido para ser utilizado em novos cenários incluindo outras arquiteturas de agentes, como por exemplo, a arquitetura do agente baseado em modelos normativo [Campos et al. 2013] [Freire et al. 2013].

Como trabalhos futuros, pode-se relacionar: (i) a utilização do *framework* juntamente com o agente monitor em cenários que permitam a inclusão de normas de forma dinâmica, e (ii) a geração automática de código para os agentes reativos simples normativos e de monitoramento, baseada na adaptação de JAMDER 2.0 proposta nesse trabalho.

References

Campos, G. A.; Freire, E. S. S. and Cortés, M. I. (2012) Norm-based behavior modification in reflex agents. In: 14th International Conference on Artificial Intelligence (ICAI), 2012, Las Vegas, Nevada, USA, Proceedings of the 14th International Conference on Artificial Intelligence.

Campos, G. A. L. ; Freire, E. S. S. ; Cortés, M. I. ; Vasconcelos, W. W. (2013) An Approach for Norm-Based Behavior Modification in Model-Based Reflex Agents. In: 15th International Conference on Artificial Intelligence (ICAI), 2013, Las Vegas, Nevada, USA, Proceedings of the 15th International Conference on Artificial Intelligence. Cruz, F. I. S; Rocha Jr., R. M.; Freire, E. S. S. and Cortés, M. I. (2014) Norm-Based Behavior Modification in Reflex Agents - An Implementation in JAMDER 2.0. In: 16th International Conference on Enterprise Information Systems

(ICEIS), 2014, Lisboa, Portugal, Proceedings of the 16th International Conference on Enterprise Information Systems.

Dastani, M. D.; Grossi, J.-J. C.; Meyer, and Tinnemeier, N. (2018) Normative multi-agent programs and their logics. In Proc. Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS'08), pages 236–243.

Dybalova, D.; Testerink, B.; Dastani, M.; Logan, B.; Dignum, F. and Chopra, A. (2013) A Framework for Programming Norm-Aware Multi-Agent Systems In: 15th International Workshop on Coordination, Organisations, Institutions and Norms (COIN 2013).

Figueiredo, K. and Silva, V. T. (2010) NormML: A Modeling Language to Model Norms. In: 1st Workshop on Autonomous Software Systems, 2010, Salvador, Brazil.

Freire, E. S. S.; Campos, G. A. L.; Cortes, M. I.; Vasconcelos, W. W. (2013) Norm-Based Behavior Modification in Model-Based Reflex Agents. In: 2013 Brazilian Conference on Intelligent Systems (BRACIS), 2013, Fortaleza, Proceedings of the 2013 Brazilian Conference on Intelligent Systems. p. 38.

Jennings, N. R.. (1995) Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence, 75(2):195–240.

Jones, A. J. I. and Sergot, M. (1993) On the characterisation of law and computer systems: The normative systems perspective. In Deontic Logic in Computer Science: Normative System Specification, pages 275–307. John Wiley and Sons.

Kaminka, G., Pynadah, D. and Tambe, M. (2002) Monitoring teams by overhearing: A multi-agent plan-recognition approach. Journal of Artificial Intelligence Research, 17: p.83–135.

Mazouzi, H., Seghrouchni, A. E. F., and Haddad, S.. (2002) Open protocol design for complex interactions in multi-agent systems. In 1st Int. Joint Conference on Autonomous Agents and Multiagent Systems, pages 517–526.

Modgil, S.; Faci, N.; Meneguzzi, F.; Oren, N.; Miles, S. and Luck, M. (2009) A framework for monitoring agent-based normative systems. In: 8th International Conference on Autonomous Agents and Multiagent Systems, 2009, Budapeste, Hungria, Proceedings of the 8th International Foundation for Autonomous Agents and Multiagent Systems. Volume 1p. 153–160.

Piunti, M.; Ricci, A.; Boissier, O. and Hübner, J. F. (2010) Programming open systems with agents, environments and organizations. In: 11th Workshop nazionale 'Dagli Oggetti agli Agenti', 2010, Rimini, Italy, Proceedings of the WOA 2010 11th Workshop nazionale 'Dagli Oggetti agli Agenti'.

Tambe, M. (1997) Towards flexible teamwork. Journal of Artificial Intelligence Research, 7:83–124.

Vlissides, J.; Helm, R.; Johnson, R. and Gamma, E. (1995) Design patterns: Elements of reusable object-oriented software. Massachusetts: Addison-Wesley. p120.