PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SIMULAÇÃO DE AMBIENTES MULTIAGENTE NORMATIVOS

STEPHAN CHANG

Monografia apresentada como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Felipe Meneguzzi

Porto Alegre
2014

LISTA DE FIGURAS

Figura 3.1 – Interação de um agente e seu ambiente. 11
Figura 3.2 – Arquitetura simples de agente. 12
Figura 3.3 – Arquitetura de agente com estado interno. 12
Figura 3.4 – Arquitetura de subsunção de Brooks. 14
Figura 5.1 – Agentes Jason operando em ambiente centralizado. 21
Figura 5.2 – Agentes Jason operando em ambiente distribuído. 22
Figura 8.1 – Ao perceber a ativação de uma norma, o agente pode decidir se a aceita ou não. 35
Figura 8.2 – Ao perceber a destruição de uma norma, o agente restaura seus planos originais. 36
Figura 9.1 – Elementos do *framework* NormMAS 37
Figura 9.2 – Pacotes de desenvolvimento do *framework* NormMAS 39
Figura 9.3 – Agentes normativos estendem o agente CArtAgO, que por sua vez estende a arquitetura Jason padrão. 40
Figura 9.4 – Diagrama de atividades do processo de captura de ações. 41
Figura 9.5 – Diagrama de sequência para o fluxo de monitoramento. 42
Figura 9.6 – Diagrama de sequência para o fluxo de fiscalização *enforcing*. 43
Figura 10.1 – Utilidade de agentes corruptos cai com o aumento da fiscalização. 47
Figura 10.2 – Porcentagem de detecção e intensidade de monitoramento estão relacionados. 48
Figura 10.3 – Quantidade de violações decresce devido à suspensão de atividades dos agentes. 49

LISTA DE ALGORITMOS

Algoritmo 9.1 – Algoritmo para detecção de violações. 44
Algoritmo 9.2 – Algoritmo para comparação de contextos. 45
Algoritmo 9.3 – Algoritmo para análise de condições de ativação de uma norma. . . . 46

LISTA DE SIGLAS

AAMAS *International Conferece on Autonomous Agents & Multi-Agent Systems*

BDI *Belief, Desire, Intention*

CArtAgO *Common Artifact Agent Open environment*

HTN *Hierarchical Task Network*

IA Inteligência Artificial

JADE *Java Agent Development Framework*

MAS *Multi-Agent System*

MASON *Multi-Agent Simulator Of Neighbourhoods*

MASSim *Multi-Agent Systems Simulator*

MDP *Markov Decision Process*

MEU *Maximum Expected Utility*

STRIPS *Stanford Research Institute Problem Solver*

SUMÁRIO

1 INTRODUÇÃO **7**

2 OBJETIVOS **9**

2.1 OBJETIVO GERAL 9

2.2 OBJETIVOS ESPECÍFICOS 9

3 AGENTES **11**

3.1 ARQUITETURAS DE AGENTE 13

3.1.1 ARQUITETURA REATIVA 13

3.1.2 BDI 14

3.2 SIMULADORES DE AGENTES 15

4 PLANEJAMENTO **16**

4.1 PLANEJAMENTO DETERMINÍSTICO 16

4.2 PLANEJAMENTO PROBABILÍSTICO 17

4.2.1 APRENDIZADO POR REFORÇO 18

5 JACAMO **20**

5.1 JASON 20

5.2 CARTAGO 22

5.3 MOISE 23

6 CENÁRIO DE TESTE **24**

6.1 AGENTES DE IMIGRAÇÃO 24

6.2 MODELAGEM DO CENÁRIO EM JASON 25

7 RACIOCÍNIO NORMATIVO **26**

7.1 NORMAS 26

7.1.1 REPRESENTAÇÃO 26

7.1.2 IMPLEMENTAÇÃO 29

7.2 PODER NORMATIVO 29

7.3 NORMAS EM SIMULAÇÕES 30

8 MONITORAMENTO DE NORMAS **31**

8.1 CAPTURA DE AÇÕES 31

8.2 AGENTE MONITOR 33
8.3 AGENTE *ENFORCER* 34
8.4 MÓDULO NORMATIVO 34
8.5 PERCEPÇÃO DAS NORMAS 35

9 DIAGRAMAS E IMPLEMENTAÇÃO 37
9.1 ARQUITETURA GERAL DO SISTEMA 37
9.2 DIAGRAMA DE PACOTES DO SISTEMA 39
9.3 AGENTES NORMATIVOS 40
9.4 SISTEMA DE MONITORAMENTO 41
9.5 SISTEMA DE FISCALIZAÇÃO (*ENFORCEMENT*) 43
9.6 DETECÇÃO DE VIOLAÇÕES 44

10 EXPERIMENTOS E RESULTADOS 47

11 CONCLUSÃO 51

REFERÊNCIAS 54

APÊNDICE A – Agentes Jason **57**
A.1 AGENTE *OFFICER* 57
A.2 AGENTE *CORRUPT OFFICER* 58

1. INTRODUÇÃO

Em Inteligência Artificial (IA), o estudo de sistemas multiagentes, ou *Multi-Agent System* (MAS), tem como objetivo a resolução de problemas complexos por meio da integração de diferentes módulos de programa, ou agentes, que são responsáveis por resolver pequenas partes de um problema maior. Esses agentes podem ter funções variadas e, sozinhos, podem não conseguir resolver um problema completo, mas quando atuam no mesmo ambiente e há cooperação, podem encontrar a resolução de forma eficiente e coordenada.

Os agentes envolvidos nesses sistemas podem apresentar capacidade de raciocínio e aprendizado, de forma que é possível haver adaptação às mudanças ocorridas no ambiente sobre o qual operam. Com isso, é introduzida também a noção de autonomia, em que não há necessidade de um estímulo externo ou usuário para que o programa continue a executar, formando então a idéia de sistemas autônomos.

Fazendo a análise de sistemas multiagentes, é possível observar que suas estruturas e a maneira como funcionam lembram uma sociedade. Afinal, são compostos de vários módulos independentes que operam paralelamente, cada um com suas próprias percepções, formas de interagir e contribuições para o ambiente. Da mesma forma, é possível que haja cooperação entre os agentes do sistema ou, pelo contrário, que esses agentes interfiram direta ou indiretamente nas ações e estados uns dos outros. Essa característica levou à pesquisa e desenvolvimento de sistemas multiagentes normativos, que trabalham com um conjunto de regras, ou normas, que limitam o espaço de ação dos agentes, e uma entidade normativa cujo papel é observar o ambiente, as ações executadas sobre ele e detectar possíveis violações que comprometem o equilíbrio do ambiente, aplicando uma penalidade aos agentes transgressores.

Entretanto, a utilização de mecanismos de controle exige mais recursos de máquina para executar, sendo que essa exigência é diretamente proporcional ao tamanho do problema com o qual se está trabalhando. É necessário, portanto, fazer ajustes no sistema para que ele possa equilibrar sua eficiência e capacidade de detecção de violações. Uma das formas de obter esse equilíbrio é utilizando simulação.

A simulação permite a visualização de ambientes multiagente em ação, e é possível fazê-la atualmente pelo uso de simuladores como o MASSim [1][2], ou programada com o uso de linguagens de descrição de agentes como Jason [3] ou JADE [4]. Além disso, a simulação de ambientes multiagente permite o estudo de situações sociais, onde os agentes podem interagir de forma cooperativa ou competitiva. Para estes casos, a inclusão de um sistema de normas torna-se interessante para que essa interação seja mediada de forma realista, tornando o estudo mais preciso. Porém, não há ferramentas que permitam a simulação de ambientes sob influência de normas, o que reduz o número de situações que são possíveis de serem observadas quando traduzidas para um MAS. O objetivo deste trabalho é elaborar um *framework* de simulação com essa capacidade.

Este volume está organizado da seguinte forma. O capítulo 2 apresenta os objetivos do desenvolvimento deste trabalho. No Capítulo 3 é visto o que é um agente, como este interage com um ambiente e as maneiras com as quais pode raciocinar sobre o mundo. No Capítulo 4 são apresentados tópicos sobre diferentes formas de planejamento para agentes. O Capítulo 5 apresenta o *framework* para desenvolvimento de MASs JaCaMo. No capítulo 6 está descrito o cenário utilizado para a simulação de teste do *framework*. Os Capítulos 7 e 8 abordam os conceitos envolvidos em um MAS normativo, assim como mecanismos de reforço ao cumprimento de normas. O Capítulo 9 descreve a arquitetura e implementação do *framework*. No Capítulo 10, são mostrados os resultados da simulação de teste. Por fim, no Capítulo 11 é feita a compilação dos resultados, análise dos objetivos e indicações para trabalhos futuros.

2. OBJETIVOS

Na primeira etapa do projeto foi feito o estudo dos assuntos relacionados à simulação de ambientes multiagente e monitoramento e aplicação de normas e a conceitualização de um *framework* para o desenvolvimento de ambientes multiagente normativos utilizando as tecnologias Jason e CArtAgO. Para esta etapa, está estabelecido o objetivo de construir os elementos básicos desse *framework*, de forma que a teoria construída durante a primeira etapa possa ser validada utilizando uma simulação multiagente que inclua um contexto normativo.

2.1 Objetivo Geral

Criar um conjunto de classes e ferramentas que possibilitem a simulação de ambientes multiagentes normativos com monitoramento e aplicação de normas.

2.2 Objetivos Específicos

Os objetivos são classificados em três grupos: fundamentais, desejáveis e opcionais. Os objetivos fundamentais representam a base do projeto, sem a qual o sistema não possui a mínima funcionalidade para ser considerado útil. Os demais são incrementos que visam a flexibilização do sistema, como o ajuste da intensidade de monitoramento do sistema e a expansão da representação de normas. Alguns destes objetivos oferecem maior complexidade e, portanto, são vistos como opções de trabalho futuro.

1. Objetivos fundamentais
   (a) Implementar um workspace normativo, que contenha uma interface para monitoramento e uma para aplicação das normas.
   (b) Construir um agente que monitore as ações dos demais agentes do sistema.
   (c) Construir um agente que detecte e sancione violações de normas.
   (d) Construir um cenário de simulação para validar a implementação realizada.
   (e) Utilizar a linguagem de programação de agentes *Jason* e do *framework* de programação de ambientes CArtAgO.
2. Objetivos desejáveis
   (a) Adicionar controle de acesso à camada normativa.
   (b) Implementar um módulo de monitoramento de intensidade ajustável.
   (c) Implementar processamento de normas com Condições de Ativação e Expiração.

(d) Submeter um artigo para um *workshop* do *International Conferece on Autonomous Agents & Multi-Agent Systems* (AAMAS).

3. Objetivos opcionais (trabalhos futuros)

(a) Implementar processamento de normas com suporte à hierarquia de agentes.

(b) Implementar um mecanismo de aprendizado nos agentes para que estes possam aprender as normas do ambiente.

(c) Permitir a inclusão de agentes com poder normativo.

3. AGENTES

Conceitualmente, agentes são entidades capazes de receber estímulos de um ambiente por meio de sensores e responder a esses estímulos por intermédio de atuadores [5]. Essa abstração permite a utilização de agentes para representar tanto máquinas quanto humanos, ou qualquer outra entidade capaz de percepção e ação. A interação entre agente e ambiente é ilustrada na Figura 3.1. No contexto de simulação, os agentes tomam a forma de programas de computador [6], ou *software agents* [7]. Isso é possível pela associação feita entre entradas de dados de programas e percepções vindas do ambiente. Da mesma forma, os comandos executados pelo programa estão associados às ações do agente, que por sua vez dependem da função desempenhada por ele. Cada agente no sistema pode ter uma função diferente, sendo que para isso são utilizados tipos diferentes de agentes.

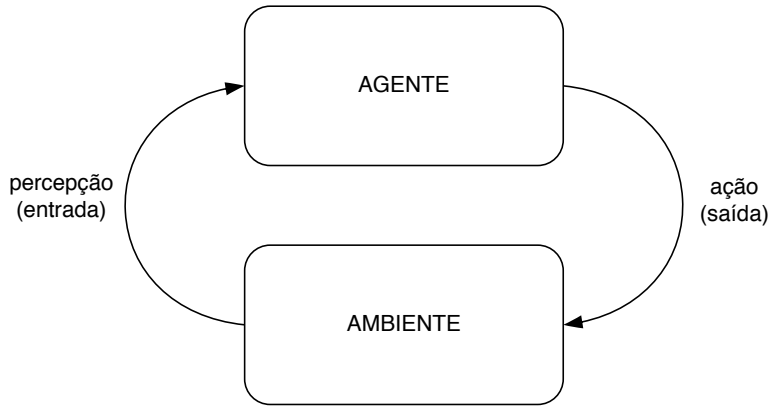

Figura 3.1 – Interação de um agente e seu ambiente.

O tipo mais simples de agente, ilustrado na Figura 3.2, escolhe ações com base naquilo que percebe do ambiente no momento da decisão. O programa faz um mapeamento de cada conjunto possível de entradas em uma ação a ser executada. Essa abordagem é eficaz em ambientes completamente observáveis, ou seja, quando o agente possui acesso à todas as informações necessárias para realizar uma decisão por meio de percepção. Entretanto, nem sempre é possível obter informações completas sobre o ambiente. Esse é o caso dos ambientes parcialmente observáveis. Supondo um terreno inexplorado, onde se encontra um agente cuja função é mapear esse espaço. O fato do terreno ser desconhecido elimina a hipótese de completa observabilidade do ambiente, afinal, se o agente possui conhecimento das entradas possíveis logo de início, não é necessário fazer a exploração. É semelhante ao caso em que o agente possui a habilidade de locomoção, mas não senso de direção: ele é capaz de se mover pelo ambiente, mas sem saber onde está e nem para onde está indo.

Uma maneira de aprimorar esse agente é adicionando a capacidade de armazenar um estado interno [7]. Essa mudança é ilustrada na Figura 3.3. A informação armazenada passa então a fazer parte do processo de escolha da ação de forma complementar às percepções.

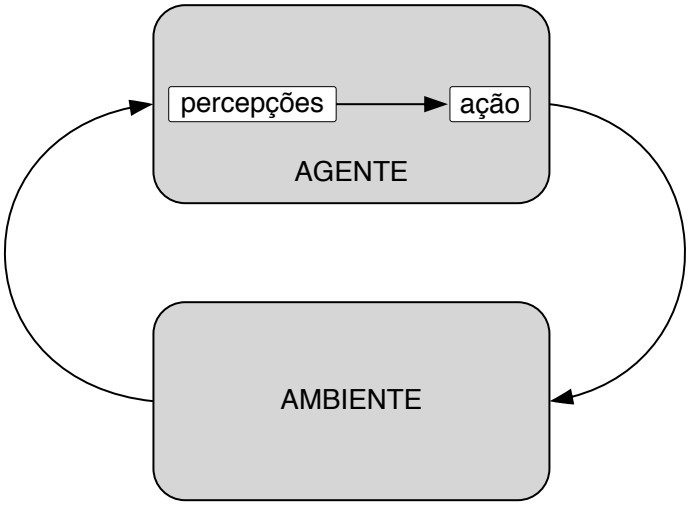

Figura 3.2 – Arquitetura simples de agente.

O estado interno do agente pode ser uma referência aos estados já visitados anteriormente ou alguma informação pertinente que não pode ser adquirida via percepção. No exemplo do terreno desconhecido visto anteriormente, o agente poderia manter um histórico das áreas do mapa que já foram exploradas e, portanto, evitaria revisitar essas porções do espaço desnecessariamente. Alternativamente, ele poderia manter uma referência para um local já visitado e, assim, se situar dentro do cenário.

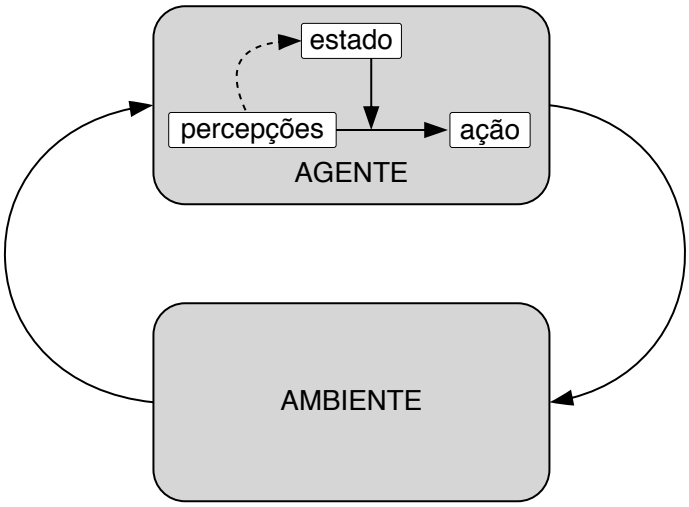

Figura 3.3 – Arquitetura de agente com estado interno.

Para que o comportamento dos agentes em um sistema possa ser avaliado, é utilizada uma medida de desempenho ajustada pelo ambiente de acordo com as ações executadas pelos agentes e, consequentemente, os estados pelos quais o ambiente passou. É possível estabelecer os melhores cursos de ação esperados para um determinado agente e, utilizando simulação, obter as medidas de desempenho dos agentes para concluir se estão agindo da forma desejada. Quando o comportamento do agente é coerente com a sua função e suas percepções, o que

resulta em ações consideradas ótimas, ele pode ser considerado racional. Há uma diferença, no entanto, entre ser racional e ser inteligente [7]. Um agente inteligente deve poder, além de reagir às mudanças do ambiente por meio de percepções, agir independentemente desses estímulos externos para atingir seus objetivos. Por fim, ele também deve conseguir interagir com outros agentes do sistema, seja de forma cooperativa ou competitiva.

3.1 Arquiteturas de Agente

Para interagir com o ambiente, os agentes precisam saber lidar com as informações que recebem, utilizando-as para decidir qual o melhor curso de ação a ser tomado. As definições para a utilização dessas informações variam de acordo com a arquitetura de cada agente. Da mesma forma que os tipos vistos anteriormente, algumas arquiteturas são mais simples que outras. O que pode mudar entre elas é a maneira como a informação é interpretada e traduzida para uma ação. Isso soluciona o problema das interfaces de ação e percepção, permitindo a heterogeneidade do sistema. Ou seja, não há necessidade de fazer a tradução das ações e percepções entre agentes de arquiteturas distintas, pois estas são transmitidas em formatos de alto nível, que podem ser interpretados por qualquer arquitetura de maneira transparente.

3.1.1 Arquitetura Reativa

Em uma arquitetura puramente reativa, um agente opera sempre sobre o último conjunto de informações recebido por seus sensores. O programa não mantém e nem se refere a um histórico de percepções. Isso torna a implementação desses agentes fácil, porém dificulta a geração de comportamentos complexos. O programa em questão é composto de uma série de estruturas condicionais `if-then-else` que traduzem cada conjunto de entradas, ou percepções, em uma ação resultante. Em contrapartida, sua implementação pode acabar se tornando extensa e opaca para o desenvolvedor se for pensado que, para cada nova situação com a qual o agente deve saber lidar, o programador precisa adicionar a respectiva cláusula condicional no programa. É possível concluir que para sistemas de proporções médias ou grandes essa abordagem não é escalável.

Em sistemas pequenos, a arquitetura reativa ainda pode se mostrar interessante. Supondo o seguinte cenário: em uma sala climatizada há um agente que é capaz de, por meio de um sensor, obter a temperatura atual do ambiente e, por meio de uma interface com o sistema climatizador, elevar ou diminuir a temperatura dele. Quando o sensor obtém uma leitura considerada muito fria para a sala, uma temperatura abaixo de 20°C, o agente envia ao climatizador o comando “elevar”. Alternativamente, se o sensor lê uma temperatura considerada muito quente, para este caso um valor acima de 24°C, o agente comunica ao climatizador o comando “diminuir”. Neste exemplo, não há necessidade de considerar o histórico de percepções

no momento de escolher uma ação. Basta que o agente consiga capturar a temperatura atual da sala para que seja aplicada a ação necessária sobre o controle de climatização.

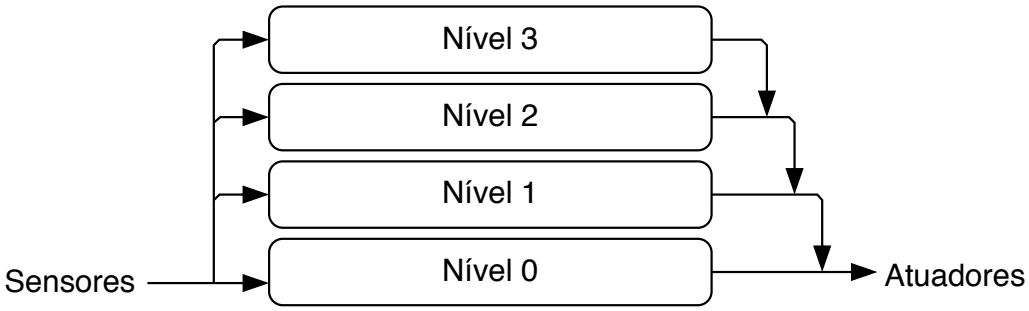

Figura 3.4 – Arquitetura de subsunção de Brooks.

Um outro exemplo de arquitetura reativa, utilizada na área de robótica, é a arquitetura de subsunção de Brooks [8]. É também chamada de arquitetura em camadas por fornecer um modelo de implementação de agentes reativos que trabalham com as percepções em diferentes níveis. Em cada nível há um canal de entrada de informações, que são utilizadas para decidir um curso de ação. As ações geradas pelas camadas de mais alto nível podem suprimir aquelas geradas pelas camadas inferiores, de forma que ações mais restritivas são preferencialmente implementadas nos níveis mais altos. A Figura 3.4 contém uma ilustração dessa arquitetura.

3.1.2 BDI

Uma das arquiteturas mais utilizadas no desenvolvimento de agentes inteligentes é a *Belief, Desire, Intention* (BDI) [9]. Essa arquitetura se baseia nos três conceitos de "crença", "desejo" e "intenção", em que o agente possui um conjunto de informações a respeito do ambiente sobre o qual suas decisões se baseiam, um ou mais estados objetivos que ele deseja atingir e um conjunto de planos de ações que são executados de forma a alcançar seus objetivos. De forma a expandir a capacidade de raciocínio e autonomia de um agente, e assim aproximá-lo da definição de um agente inteligente, esses três conceitos foram adaptados para o contexto de agentes.

As crenças são as informações que o agente possui a respeito do ambiente e que ele acredita serem verdadeiras. São as informações sobre as quais ele se baseia para raciocinar sobre o problema para o qual ele foi designado. Supondo novamente o exemplo da sala climatizada da Seção 3.1.1. As crenças do agente podem ser "a sala está quente" ou "a sala está fria". Essas crenças são derivadas das percepções que o agente tem sobre o ambiente. Por meio dessas percepções, o agente chega à conclusão de que a sala está quente ou fria, e com isso é possível trabalhar em uma lógica para definir qual é o próximo passo a ser tomado neste ambiente.

Um desejo, nesse contexto, é qualquer estado ao qual o agente deseja chegar. No ambiente da sala citado anteriormente, o desejo é que a sala se mantenha em temperatura

amena, entre 20°C e 24°C. Qualquer valor além ou aquém desse intervalo faz com que o agente se sinta desconfortável na sala e, portanto, precise ajustar o climatizador. Resta, então, definir o que é preciso fazer, ou o que o agente **pretende** fazer, para satisfazer esse desejo utilizando suas crenças e percepções.

Uma intenção representa o que o agente pretende fazer para atingir um determinado estado objetivo. Isso pode ser a execução de uma série de ações, a realização de algum desejo, ou uma combinação destes. No cenário que veio sendo alimentando, supondo que a sala está quente e, como consequência, que o agente se sente desconfortável nesta situação. Com o objetivo de restaurar o equilíbrio da temperatura do ambiente, o agente executa um plano de ações, que para esse cenário pequeno é composto apenas da ação "diminuir temperatura". É interessante destacar que, para cenários de porte maior, os planos podem ser bem mais complexos e até incluir uma hierarquia de planos e sub-planos.

3.2 Simuladores de Agentes

Existem algumas ferramentas que possibilitam a simulação de ambientes multiagentes por meio de especificação de modelos programáveis ou de bibliotecas para linguagens de programação. A ferramenta *NetLogo* [10], por exemplo, é utilizada para projetos pequenos ou educacionais, e trabalha com modelos de agente programáveis na linguagem de programação Logo extendida para programação de agentes. Uma versão distribuída dessa ferramenta, chamada *HubNet* [11], foi criada para que diversos agentes pudessem ser controlados de localizações distintas. O programa permite a visualização do ambiente e seus agentes, e as modificações pelas quais eles passam no decorrer da execução da simulação.

Outras abordagens de simulação mais profissionais incluem o *Multi-Agent Simulator Of Neighbourhoods* (MASON) [12] e *Repast* [13], sendo que este último possui ainda uma versão própria para computadores de alta capacidade. MASON é uma biblioteca de simulação feita para a linguagem de programação Java e inclui funções para modelagem de agentes e visualização da simulação. É possível também integrar os agentes modelados com outros *frameworks* Java ou aplicações. A ferramenta *Repast Simphony* trabalha com modelos programáveis de agente, como em *NetLogo*, porém oferece mais alternativas de linguagens para programar os agentes.

Um exemplo final de sistema de simulação de MAS é o *Multi-Agent Systems Simulator* (MASSim) [1], criado para incentivar o estudo de sistemas multiagentes e utilizado na competição *MAS Programming Contest* [2]. Essa, porém, é uma alternativa limitada quando é desejado trabalhar com cenários que não os fornecidos junto com o simulador. Apesar de ser possível programar agentes próprios na linguagem Java, isto não é encorajado pelos criadores do sistema.

4. PLANEJAMENTO

4.1 Planejamento Determinístico

Devido ao alto custo computacional requerido para gerar planos de ação determinísticos, é costume trabalhar com planos pré-prontos ou gerados pré-execução [14]. Entretanto, isso faz com que os agentes careçam de autonomia e flexibilidade, condicionando-os a planos específicos e imutáveis. Existem casos em que é interessante para o agente poder gerar novos planos de ação em tempo real, de acordo com sua necessidade de atingir objetivos que encontram-se inatingíveis, considerando os planos aos quais o agente está limitado. Para resolver esse problema, há pesquisas [14] que buscam a integração da arquitetura BDI com abordagens de planejamento determinístico que dão ao agente a capacidade de raciocinar sobre um determinado objetivo e gerar um plano de ação para atingí-lo. Duas dessas abordagens são descritas a seguir:

***Stanford Research Institute Problem Solver* (STRIPS)** [15] Define o plano como um conjunto de estados pelos quais o agente deve passar de forma a atingir seu objetivo. A transição entre os estados é feita por meio da execução de ações, e cada ação possui pré-condições e consequências. Para este tipo de planejamento, o ambiente é considerado completamente observável e determinístico.

***Hierarchical Task Network* (HTN)** [16] Enquanto STRIPS vê as ações como um meio de atingir os estados que compõem o plano, HTN as enxerga como os componentes principais usados para a construção destes. Como o nome sugere, os planos passam a ser um conjunto de tarefas, que podem ser divididas em tarefas não-primitivas e tarefas primitivas, que por sua vez são tarefas que não podem ser decompostas. No momento em que a tarefa do topo da hierarquia é concluída, o plano é considerado executado. Assim como em STRIPS, o ambiente é considerado completamente observável e determinístico.

Embora ambas as abordagens concedam ao agente a capacidade de raciocinar sobre o problema e gerar um novo plano de ação correspondente, o planejamento hierárquico oferece uma vantagem importante sobre o planejamento clássico: o HTN possui conhecimento de domínio. O programador não só especifica ao planejador quais as tarefas disponíveis para montar o plano, mas também como que essas tarefas poderão ser combinadas, com base nas restrições do ambiente. Isso permite que o planejador se ajuste à eventuais mudanças no ambiente, tornando-o uma opção formidável para sistemas adaptativos.

4.2 Planejamento Probabilístico

Um processo de decisão de Markov, ou *Markov Decision Process* (MDP) [5], é composto por um conjunto de estados, ações, probabilidades e recompensas que definem uma solução para um problema de decisão sequencial em um ambiente completamente observável, ou seja, um ambiente em que o agente está, em todos os momentos, ciente de onde se encontra. No entanto, as ações tomadas por agentes em MDPs não são determinísticas. Ao executar uma ação, não há garantias de que o agente obterá o resultado esperado. O ambiente é estocástico, e para modelar as interações entre agentes e o ambiente, é utilizado um modelo de transição, que define os resultados das ações tomadas em cada estado do ambiente de acordo com uma certa probabilidade da ação desejada ocorrer de fato. Esse processo é *markoviano*, portanto não possui memória ou um histórico de ações tomadas. As transições entre estados independem da sequência de estados visitados até então, fazendo com que uma ação só dependa do estado em que o agente se encontra atualmente.

Como visto no Capítulo 3, uma medida de desempenho é utilizada para avaliar o comportamento de um agente. Essa medida, que pode ser chamada também de utilidade, pode ser alterada durante a execução de um MDP com recompensas. Cada estado possui um valor de recompensa, que é utilizado para calcular o valor da utilidade do agente em cada etapa de uma sequência de ações. O cálculo do valor de utilidade para uma determinada sequência de estados pode ser visto na equação 4.1. A recompensa é dada pelo valor de R, enquanto γ representa a taxa de desconto, que deve ser um valor entre 0 e 1. A cada estado visitado, a recompensa sofre um determinado desconto, o que significa que, dependendo da taxa γ de desconto, os valores de recompensa mais ao final da sequência vão se tornando insignificantes, especialmente quando γ é um valor próximo de 0. Quando $\gamma = 1$, as recompensas são somadas sem nenhum desconto.

$$U([s_0, s_1, s_2, \dots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{4.1}$$

A sequência de estados que são visitados pelo agente depende da política adotada por ele e da estocasticidade do ambiente. Uma política define, para cada estado do ambiente, uma ação recomendada que deve ser executada pelo agente. Devido à probabilidade de uma ação não produzir o efeito esperado, a sequência de estados visitados produzida pela política não é necessariamente a mesma em execuções diferentes. Entretanto, dentre as várias políticas disponíveis para a solução de um MDP, uma delas é considerada **ótima** e garante a máxima utilidade esperada para o agente para qualquer número de execuções. Essa política é chamada *Maximum Expected Utility* (MEU).

Existem alguns algoritmos para encontrar a política MEU, e dentre eles dois são: **Iteração de valor** e **Iteração de política**. A iteração de valor calcula o valor de utilidade dos estados e seleciona as ações que conduzem aos estados de maior utilidade. O cálculo da

utilidade de um estado é definido pela equação 4.2, chamada de equação de Bellman. Para cada estado s do MDP, há uma equação de Bellman correspondente. O algoritmo de iteração de valor calcula os valores dos estados a partir de um valor inicial arbitrário, e a cada iteração a equação 4.3, chamada atualização de Bellman, é aplicada nos estados para que os valores de utilidade convirjam à um ponto fixo. A equação 4.4 mostra como é feita a extração da política ótima.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{4.2}$$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s') \tag{4.3}$$

$$\pi(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)U(s') \tag{4.4}$$

A convergência das utilidades pelo método da iteração de valor é a base para o algoritmo da iteração de política. Nesse algoritmo, é considerado que a precisão dos valores de utilidade não é importante, desde que a distinção entre uma ação ótima e as demais seja clara pela diferença de valores de utilidades obtidos pela execução delas. O algoritmo de iteração de política é dividido em duas etapas, uma que calcula a utilidade de cada estado de acordo com uma política π_i, começando com $i = 0$, e outra que calcula uma nova política π_{i+1} com mais ações ótimas que a anterior. Quando não há mais modificações a fazer na política, o algoritmo é terminado.

4.2.1 Aprendizado por Reforço

Em MDPs, cada estado visitado por um agente fornece uma recompensa que é contabilizada no cálculo do valor de utilidade do agente. As recompensas de cada estado são conhecidas e uma política ótima é calculada para que o agente obtenha a maior utilidade possível. Entretanto, é importante considerar o caso em que o valor da recompensa não está disponível para os agentes até que eles os visitem. Nesta situação, é preciso um mecanismo que permita que os agentes aprendam os valores dessas recompensas de forma a gerar novas políticas mais otimizadas para operar no ambiente.

O aprendizado por reforço [5] é uma área de estudo da IA que aborda algoritmos de aprendizado com aplicação em ambientes estocásticos, permitindo que os agentes aprimorem seu desempenho de maneira autônoma. Os conceitos de utilidade e recompensa são utilizados para que o agente seja capaz de perceber quais ações são consideradas positivas e quais ele deve evitar para que seu desempenho não seja prejudicado. O agente recebe essa informação por meio de *feedback* do ambiente, que é enviado como se fosse uma percepção em intervalos

constantes, como após a execução de um número n de ações, ou após atingir um certo estado objetivo.

A utilização de planejamento e aprendizado permite o desenvolvimento de agentes adaptativos, que utilizam dados do sistema para planejar suas próximas ações e otimizar seu desempenho. Apesar dessas técnicas não terem sido utilizadas na implementação final, é importante considerar a futura operação de tais agentes no ambiente e em como eles utilizam as informações percebidas em suas rotinas de planejamento. Portanto, este capítulo serve como referência para trabalhos futuros, onde agentes capazes de aprender com normas serão desenvolvidos.

5. JACAMO

A pesquisa na área de sistemas multiagentes levou ao estudo das opções de linguagens propícias para descrever agentes e integrá-los aos mais variados ambientes. Uma das estruturas que foram criadas com o objetivo de facilitar a especificação e programação de agentes, assim como suas interações e ambientes, foi a estrutura JaCaMo. Essa estrutura é composta por três *frameworks* que correspondem à abordagens diferentes para a implementação de MASs.

O primeiro, Jason [3], é um interpretador da linguagem de descrição de agentes *AgentSpeak* [17] escrito e integrado com a plataforma Java. Jason compõe a abordagem orientada a agentes e é tratado em mais detalhe na Seção 5.1. O segundo *framework* é o *Common Artifact Agent Open environment* (CArtAgO) [18] e é também integrado com Java, permitindo o uso de artefatos para a descrição de ambientes, que definem estruturas e operações que são acessadas pelos agentes. CArtAgO constitui a abordagem orientada a ambientes, que possui o objetivo de modelar ambientes reais para simulação e teste de agentes desenvolvidos em Jason. Maiores detalhes são abordados na Seção 5.2. O terceiro elemento é conhecido como Moise [19] e representa o desenvolvimento de MAS orientado a organizações. Esse tópico é tratado na Seção 5.3.

5.1 Jason

O interpretador Jason [3] fornece um ambiente para desenvolvimento de agentes utilizando a linguagem *AgentSpeak* e recursos da linguagem de programação orientada a objetos Java. Um agente Jason é construído sobre a arquitetura BDI [9] e, portanto, possui mecanismos embutidos que trabalham com os três conceitos revisados na Seção 3.1.2. Em *AgentSpeak*, no entanto, os três elementos que caracterizam o raciocínio de um agente são *beliefs*, *goals* e *goal plans*.

As crenças do agente, que fazem parte da *belief base*, são predicados escritos no formato `predicado(valor1,valor2,...)`, como por exemplo: `plays(jeff, guitar)` ou `likes(jimmy, jazz)`. Esse formato é adotado por linguagens de paradigma lógico, como Prolog, pela facilidade de expressão de idéias. Da mesma forma que em Prolog, as constantes iniciam com letra minúscula, enquanto variáveis iniciam com letra maiúscula ou com o caractere '_'. Assim, é possível utilizar um predicado no formato `predicado(valor,variável,...)`, como `plays(Who,guitar)` para fazer a pergunta "Quem toca guitarra?" e obter "jeff" como resposta.

Os *goals* são semelhantes aos desejos dos agentes por representarem os estados aos quais o agente deseja chegar. Em *AgentSpeak* há dois tipos de *goals*: *achievement goals* e *test goals*. O primeiro representa de fato algum estado em que o agente deseja chegar, sendo que este pode ser um objetivo final ou ainda um objetivo intermediário. O segundo representa uma

dúvida a respeito do estado do ambiente que o agente deseja sanar. Os dois tipos podem ser diferenciados pela presença de um caractere '!', que precede os objetivos, e um caractere '?', que precede os testes.

De forma a atingir seus objetivos, os agentes precisam decidir as ações que devem ser executadas. Para esse propósito os *goal plans* são utilizados. Assim como as intenções, os planos definem um conjunto de ações que devem ser executadas de forma que o agente possa chegar ao estado desejado. Essas ações podem ser também outros planos que precisam ser concluídos antes do agente prosseguir com o plano maior.

A linguagem permite a especificação de um conjunto de crenças iniciais para o agente. Além disso, é possível descrever planos de execução que são seguidos pelo agente de forma a atingir um determinado objetivo. Não há necessidade de programar o fluxo de execução BDI interno do agente, visto que este é o papel desempenhado pelo interpretador. A linguagem Java serve para a especificação de ações internas dos agentes, para o caso em que as funções inclusas não sejam suficientes. Isso permite a simulação de cenários de teste variados, uma vez que as ações dos agentes estão limitadas apenas pela capacidade do desenvolvedor de programá-las.

Utilizando Jason é possível criar simulações por meio da programação de um ambiente simulado, com o qual os agentes implementados interagem. Essa é uma forma útil e eficiente de testar agentes antes de aplicá-los no mundo real, mas também de programar simulações sociais e analisar interações entre agentes. O problema, no entanto, volta a ser cenários de grande porte

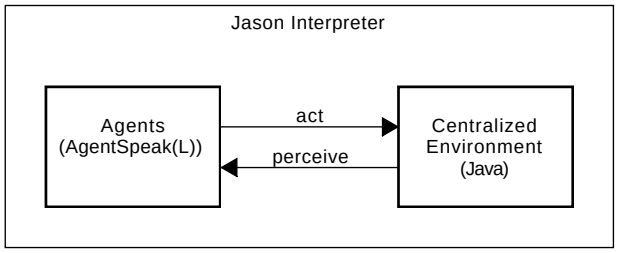

Figura 5.1 – Agentes Jason operando em ambiente centralizado.

ou grande complexidade. Os ambientes Jason são centralizados e não modulares, como ilustrado na Figura 5.1. A execução de ações pelos agentes é feita por uma única operação, que deve ser programada de forma a saber lidar com todas as entradas possíveis. É um problema semelhante ao estudado na Seção 3.1.1, de agentes puramente reativos. Felizmente, há uma alternativa para a implementação de ambientes simulados que fornece escalabilidade e modularidade. Essa alternativa é tratada na seção a seguir.

5.2 CArtAgO

Em conjunto com Jason, é possível utilizar também a tecnologia CArtAgO [18], que fornece uma base para o desenvolvimento de ambientes por meio de objetos chamados artefatos. Esses artefatos fornecem uma interface de operação, pela qual os agentes interagem com o ambiente. Neste caso, o ambiente passa a ser não uma entidade concreta e centralizada, mas sim uma abstração, e a partir desta abstração são criados os artefatos. Diferentemente do ambiente simulado Jason, a infraestrutura CArtAgO pode ser distribuída.

Os artefatos são contidos em *workspaces*, que podem estar fisicamente localizados em locais distintos. A vantagem disso é que o ambiente passa a ser modular, diferentes funcionalidades podem ser separadas em artefatos e agentes de diferentes sistemas podem se conectar de forma transparente, o que torna o desenvolvimento mais fluido e a estrutura mais escalável. De forma a interagir com o ambiente, os agentes precisam se conectar ao *workspace* onde os artefatos residem. Utilizando essa conexão, as operações descritas na implementação do artefato podem ser executadas. Dessa forma, é possível especificar à quais artefatos cada agente se conecta, não sendo necessário implementar todas as ações possíveis de agentes em um artefato só. Essa estrutura está ilustrada na Figura 5.2.

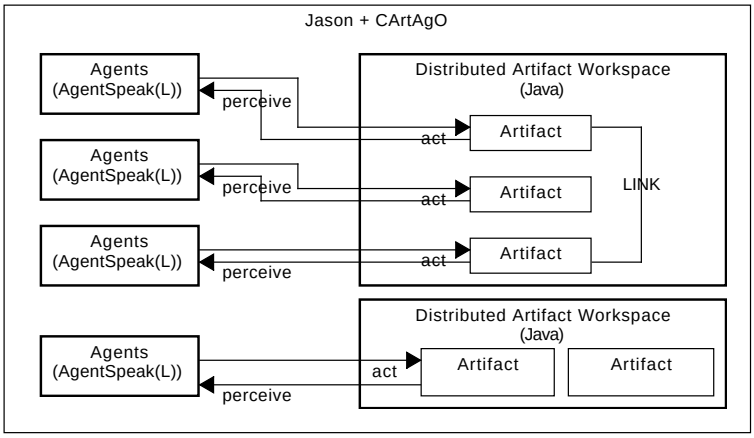

Figura 5.2 – Agentes Jason operando em ambiente distribuído.

Os artefatos não possuem um fluxo de execução autônomo, considerando que são os componentes que formam o ambiente. São reativos no sentido em que suas operações são executadas apenas quando um agente envia uma mensagem com a respectiva requisição. Entretanto, é possível que um artefato dispare uma operação em outro artefato utilizando *artifact linkings*. Isso pode ser útil para ativar rotinas de sistema que dependem da ação de agentes, mas que não são explícitas para eles, como um sistema de captura de ações, que podem ser armazenadas para análise posterior de desempenho dos agentes.

5.3 Moise

Moise [19] trata da adaptação de estruturas organizacionais para o desenvolvimento de sistemas multiagentes. Com essa abordagem, é possível modelar hierarquias e papéis que devem ser adotados por agentes no sistema. Esses papéis definem alguns limites de ação para quem os interpreta, ou regras, que formatam as interações entre agentes e entre agentes e seus ambientes, especificando o que estes podem ou não realizar dentro do espaço em que estão atuando.

O objetivo do Moise é mediar as interações dos agentes de forma a evitar que o comportamento de um agente interfira no de outro. Este também é o objetivo da implementação de um sistema normativo, assunto que é tratado no Capítulo 7. Na verdade, os fundamentos dessas duas abordagens são essencialmente os mesmos, porém a maneira com que lidam com as interações dos agentes são diferentes. Um sistema desenvolvido com Moise opera de forma regimentada: os agentes não podem executar operações consideradas ilegais pelo sistema organizacional. Se, eventualmente, um agente tenta realizar uma dessas operações, sua tentativa é descartada e o sistema continua a operar normalmente. Em um sistema normativo, o agente tem poder de decisão quanto a seguir ou não uma norma, porém não seguí-la torna-o suscetível a punições, que podem impactar seu desempenho geral.

Levando as diferenças dessas abordagens em consideração, os dois elementos que são usados neste trabalho são Jason e CArtAgO. A existência de Moise é, no entanto, um motivador para a criação de uma abordagem alternativa à organizacional. Considerando que é buscada máxima autonomia para os agentes, é de grande interesse que estes sejam capazes de raciocinar sobre as normas e decidir o melhor curso de ação por si mesmos.

6. CENÁRIO DE TESTE

Nos próximos capítulos são abordados os conteúdos relacionados ao raciocínio normativo dos agentes e ao sistema de monitoramento de normas. Para que seja possível mostrar claramente como são utilizados esses mecanismos, o cenário "Agentes de Imigração" foi preparado. O cenário inclui, então, elementos que permitem o fácil entendimento de como um sistema de normas pode ser aplicado em um ambiente qualquer e como ele pode ser simulado utilizando o *framework* proposto neste trabalho.

6.1 Agentes de Imigração

Em um determinado país de economia emergente, teve início um programa governamental de imigração visando a contratação de estrangeiros para acelerar o crescimento interno da nação. Além de imigrantes, turistas também são aceitos, pois o turismo nesse país contribui fortemente para sua economia. Na fronteira, alguns oficiais de imigração trabalham na vistoria de passaportes dos imigrantes, que só podem adentrar o país em questão caso seus registros estejam em ordem. Por ser um país em desenvolvimento, é interessante que haja o maior número de entradas possível, pois quanto mais trabalhadores houver disponíveis, mais acelerado será o seu crescimento. Entretanto, ao aprovar passaportes ilegais ou inconsistentes, convida-se também a desordem. Portanto, cada oficial de imigração possui a obrigação de aceitar todos os passaportes válidos, indiscriminadamente, ao mesmo tempo em que deve rejeitar todos aqueles que apresentarem inconsistências. Para cada passaporte aprovado, no entanto, o oficial executor ganhará um bônus de 5 créditos, e para os passaportes rejeitados, não há bônus.

No escritório da fronteira, um dos oficiais é responsável por monitorar a conduta dos oficiais de imigração e reportar as atividades para seu superior, o oficial de conduta, que irá identificar toda e qualquer atividade suspeita no ambiente de trabalho. No entanto, o oficial observador não é capaz de cuidar de todos os oficiais ao mesmo tempo, e existe uma certa chance de que uma eventual má conduta por parte de um dos oficiais de imigração não seja flagrada por ele. O oficial de conduta, ao receber os relatórios do oficial observador, os lê atentamente tentando identificar alguma infração no sistema de imigração. Ao detectar uma infração, o oficial de conduta aplica uma punição de acordo com a regra violada. O oficial que aprovar um passaporte que não deve ser aprovado recebe uma multa de 10 créditos, e além disso tem atividade suspendida por 10 segundos. Considerando que em 10 segundos em torno de 5 imigrantes podem ser atendidos, o potencial de perda para um agente em suspensão chega a 25 créditos, totalizando uma perda máxima de 35 créditos por infração detectada.

6.2 Modelagem do cenário em Jason

Os agentes de imigração são implementados em Jason como *officers* ou *corrupt officers*, sendo que este último apresenta tendências a burlar as normas do ambiente de trabalho. Sua rotina padrão como agentes Jason é executar o plano `!work`, que representa o ato da chegada do agente em um quiosque e a abertura do posto de verificação. Ao finalizar a abertura, um novo plano `!receivePassports` é instanciado para que seja iniciada a aceitação de passaportes. A recepção de passaportes ocorre de forma perpétua, ou seja, após aceitar ou rejeitar um passaporte, o agente recepcionará o próximo, até que a execução do sistema finalize. Para efetuar a decisão sobre o que fazer com um passaporte, é utilizado o plano `!checkPassport`, sendo que a seleção do plano ocorre pela presença do literal `valid` na *belief-base* do agente. Caso o literal `valid` esteja presente, então a ação de aceitar o passaporte é executada, caso contrário, o contexto é identificado pela ausência deste mesmo literal, descrito em *AgentSpeak* como `'not valid'`, e a ação executada é a de rejeição.

O agente corrupto possui o mesmo fluxo de execução, porém alguns detalhes são ajustados para que o papel de corrupto possa se concretizar. No plano `!checkPassport`, por exemplo, a presença ou ausência do literal '`valid`' não influencia na decisão do agente, que é a de sempre aceitar um passaporte, independente de sua condição, para maximizar seu ganho. Sendo um agente sujeito às normas, e especialmente por ter sido descrito especificamente para violá-las, é necessário também adicionar eventos ou planos para lidar com eventos normativos, como sanções. Um exemplo seria um evento Jason '`+sanction`' que atualiza a *Belief-base* do agente com as percepções consequentes de sua violação, que neste cenário é o decremento da quantidade de créditos bônus adquiridos pelo agente até o momento da detecção em uma dezena.

O código utilizado para os agentes *officer* e *corrupt officer* pode ser visto no Apêndice A. No próximo capítulo é abordado o que são normas e por que suas aplicações em um MAS são importantes. Uma vez definido o cenário, é possível pensar em normas de forma mais clara e objetiva. O cenário descrito será utilizado para auxiliar no processo de formalização das normas, traduzindo-as de um formato textual para uma estrutura de dados computável.

7. RACIOCÍNIO NORMATIVO

Em ambientes multiagente competitivos, o eventual conflito de interesses entre agentes é inevitável [20]. Por isso, é interessante que exista uma maneira de coordenar estes agentes de forma a otimizar suas utilidades e, ao mesmo tempo, reduzir a ocorrência de interferências entre agentes devido a políticas de ação conflitantes. Algumas ferramentas de simulação de ambientes multiagentes como *NetLogo* [10] e sua versão distribuída *HubNet* [11], criadas com propósitos educacionais, não oferecem suporte a um sistema de regras. Até mesmo em alternativas mais robustas como *Repast* [13] e *MASON* [12] o controle de ação dos agentes é feito por meio de modelos de interação.

De forma a agir coerentemente em sociedade, alguns mecanismos podem ser aplicados em um ambiente para controlar ou limitar as interações entre agentes. Uma das formas de regular as atividades desempenhadas por eles é utilizando estratégias de competição, estudadas na Teoria dos Jogos. O *framework* Moise [19] oferece uma plataforma de desenvolvimento orientado a organizações, que estipula regras que não podem ser quebradas pelos agentes. É interessante, no entanto, considerar a autonomia e independência dos agentes de deliberadamente decidir que ações executar. Para isso, o conceito de normas é introduzido como uma forma alternativa de controle de comportamento, por meio da definição de proibições, obrigações e permissões que devem ser seguidas pelos agentes. Para fazer tal adaptação, é necessário que os agentes possam analisar as normas do ambiente e modificar seus planos de ações para que comportamentos antigos não interfiram no equilíbrio do ambiente.

7.1 Normas

As normas são comumente divididas em obrigações e proibições [21] que se aplicam sobre um conjunto de ações ou estados. Obrigações definem comportamentos que devem ser seguidos sempre que o agente se encontra em determinadas condições. Como “seguir em frente” em uma via de tráfego de sentido único, presumindo que o agente já trafega no sentido correto, ou “estar com a lanterna ligada” quando dirigindo à noite. Proibições, por outro lado, são comportamentos que devem ser evitados por apresentarem risco para o equilíbrio do ambiente. Caso essas violações sejam detectadas, os agentes transgressores devem ser punidos, tendo suas utilidades penalizadas e suas ações inibidas temporariamente.

7.1.1 Representação

Para que seja possível compreender e raciocinar sobre normas, é preciso representá-las de forma estruturada, clara e não-ambígua. Com esse objetivo, a quantidade de informações que são armazenadas são limitadas ao mínimo exigido pelo contexto de simulação. Não é do

interesse dos agentes, por exemplo, saber dos detalhes de como uma certa norma foi concebida e qual o seu objetivo final. Contudo, é importante que ele saiba quais os estados em que uma norma se aplica, a condição de aplicação e qual a consequência para o caso de desobediência à regra. Seguindo essa especificação, chegamos à seguinte definição:

Definição 1. *Uma norma é representada pela tupla* $\mathcal{N} = \langle m, c, X, T, p \rangle$.

Em que:

- $m \in \{obligation, prohibition\}$ representa a modalidade da norma.
- $c \in \{state, action\}$ representa o tipo de condição encapsulada.
- X representa o conjunto de estados sobre os quais a norma deve ser verificada.
- T representa o conjunto de estados ou ação que devem ser verificados em uma obrigação ou evitados em uma proibição.
- p representa a ação de penalização que é aplicada sobre agentes transgressores.

Portanto, de acordo com o cenário visto no Capítulo 6, se há uma norma ativa no sistema com a especificação mostrada no Exemplo 1.

Exemplo 1.

$$\begin{aligned}\mathcal{N} = \langle & prohibition, \\ & action, \\ & [not\ valid(Passport)], \\ & accept(Passport), \\ & sanction(10) \rangle\end{aligned}$$

Então está definido que é proibida a aceitação de um passaporte inválido. Caso isso ocorra, a função de penalização "demérito" deve ser aplicada. Neste exemplo, a função de penalização recebe o valor 10, o que significa que o agente punido sofrerá penalidade de 10 créditos. A função de penalização varia de acordo com o contexto de simulação. A penalidade para esse caso poderia ser a adição de uma crença "suspenso" no agente transgressor, por exemplo, para que este não possa atuar momentaneamente. No Exemplo 2 é descrita a configuração de uma obrigação.

Exemplo 2.

$$\begin{aligned}\mathcal{N} = \langle & obligation, \\ & action, \\ & [valid(Passport)], \\ & accept(Passport), \\ & sanction(5)\rangle\end{aligned}$$

Neste exemplo, temos que a aceitação de um passaporte válido é imperativa. Caso a aceitação não aconteça, a devida penalização deverá ser aplicada. Os Exemplos 3 e 4 mostram como são definidas normas com estados como condição de aplicação.

Exemplo 3.

$$\begin{aligned}\mathcal{N} = \langle & obligation, \\ & state, \\ & [not\ suspended], \\ & [working], \\ & sanction(20)\rangle\end{aligned}$$

Exemplo 4.

$$\begin{aligned}\mathcal{N} = \langle & prohibition, \\ & state, \\ & [suspended], \\ & [working], \\ & sanction(10)\rangle\end{aligned}$$

No Exemplo 3, temos que se um agente não está suspenso, então ele deve estar trabalhando. Isso significa que, caso ele resolva fazer uma pausa, por exemplo, pode ser punido por estar violando a obrigação de trabalhar ininterruptamente. Já no Exemplo 4 a norma define que é proibido para um agente estar trabalhando enquanto está suspenso, sendo essa suspensão decorrente de uma violação prévia.

Esse formato de norma provê informações suficientes para realizar a detecção de violações da forma mais básica. Sob esse modelo é pressuposto que a norma está sempre ativa nos estados especificados no contexto de aplicação. Então, para o caso de uma norma ter perdido a sua validade, não é possível simplesmente desativá-la e mantê-la no banco de normas para eventual reativação. A única saída passa a ser apagá-la do sistema e, caso seja necessário uma reativação, adicioná-la novamente, e isso pode tornar o sistema inflexível.

Uma solução para esse problema é encontrada na representação vista em [21], em que normas podem ter condições de ativação e expiração, que definem o intervalo em que uma norma possui influência em seu ambiente. A inclusão desses elementos na representação expande o número de cenários possíveis para simulação e, portanto, é uma característica desejável em um sistema de simulação normativo.

Na abordagem vista em [22], é possível também especificar a classe de agente à qual a norma se aplica, o que representa uma extensão do contexto de aplicação. Para isso é necessária a introdução de conceitos organizacionais no sistema, tornando possível a definição formal de diferentes papéis que são associados aos agentes de acordo com as funções desempenhadas por cada um. Isso, no entanto, não é o foco deste trabalho, mas pode ser considerado como trabalho futuro.

7.1.2 Implementação

A maioria das arquiteturas de agentes tradicionais não possui capacidade de adaptação às normas do ambiente. É possível que os agentes passem a agir de forma ilegítima por estarem ignorantes quanto às mudanças de normas no ambiente. Esse problema pode ser resolvido por meio da implementação de um mecanismo [21] que reage a essas mudanças e gera novos planos de ação ajustados às novas normas do ambiente, inibindo os planos que estariam então violando essas normas.

Esse mecanismo pode ser implementado com a extensão de um interpretador de uma linguagem de agentes BDI para que seja possível computar as modificações de normas em tempo de execução, detectar possíveis violações de normas em planos de ação obsoletos e atualizá-los. Entretanto, é possível que o agente opte por não realizar essa atualização, o que caracterizaria a sua capacidade de raciocínio em relação às normas do ambiente, de forma a manter planos de ação que, apesar de possuírem risco de violação de normas, permitem o agente atingir um grau superior de utilidade.

7.2 Poder Normativo

Em um MAS normativo, é importante considerar não só a adaptabilidade dos agentes em relação às normas, mas também a das normas em relação ao ambiente [23]. As normas não são necessariamente estáticas, elas podem vir a existir no decorrer do tempo de execução de um sistema. Da mesma forma, elas podem ser alteradas, tendo suas influências expandidas ou retraídas, ou descartadas quando estas não são mais necessárias. Estes eventos possuem o mesmo estímulo: a necessidade de equilíbrio entre os agentes de uma sociedade competitiva.

Outro cenário a ser considerado é um ambiente em que os agentes são divididos em organizações. Em uma organização, um agente pode possuir um poder específico sobre um

determinado grupo de agentes. Um professor, por exemplo, pode possuir autoridade sobre seus alunos, estipulando regras que devem ser seguidas para o bom andamento de uma aula. Um diretor, por sua vez, possui autoridade sobre os professores e estipula, até certo ponto, as medidas educacionais que devem ser tomadas pelos professores durante seu período letivo. Enquanto um professor pode possuir liberdade de escolha entre diferentes métodos didáticos, ele não pode, por exemplo, se desfazer de alguma regra imposta pelo seu superior.

O poder normativo, então, é a capacidade de criar, alterar ou destruir normas de acordo com a necessidade de adaptação do sistema e de seus agentes. Para isso, a existência de uma autoridade reconhecida pelo MAS é necessária. Esse reconhecimento pode ser feito pela hierarquização dos agentes, em que quem está mais acima na hierarquia detém maior poder.

7.3 Normas em simulações

A utilização de normas em simulações de MAS, embora sirva para manter as interações dos agentes sob controle, representa também a liberdade de ação deles em relação ao ambiente, uma vez que as ações dos agentes estão limitadas apenas por suas respectivas definições, e não por regras ou modelos de interações que ora os impediriam de agir de forma ilegítima. Com as normas instaladas no sistema, resta acrescentar um mecanismo que garanta que essas normas estão sendo seguidas. Este mecanismo é responsável pela captura e análise das operações já executadas pelos agentes, para que então eventuais transgressões sejam identificadas e punidas. Este conteúdo é abordado no próximo capítulo.

8. MONITORAMENTO DE NORMAS

Como foi visto no capítulo anterior, é necessário estipular regras de comportamento para que os múltiplos agentes em um sistema possam cooperar entre si da forma menos destrutiva possível. Essas normas, no entanto, podem ou não ser seguidas, decisão que fica a critério de cada agente e da capacidade de raciocínio destes em consideração às suas utilidades. Caso o ambiente permita a possibilidade de violação dessas normas, é interessante considerar a introdução de dois mecanismos normativos no sistema: um de monitoramento de normas [20], reponsável por observar as ações dos agentes, e um de detecção de violações, responsável pela análise das ações observadas e aplicação das medidas de penalização de agentes transgressores.

Levando em consideração o fato de que há custo para manter um sistema de monitoramento de normas ativo, em que o custo requerido está diretamente ligado com a capacidade de detecção de violações do mecanismo, é inferido que o monitoramento do ambiente nem sempre é perfeito [22], ou seja, há brechas no sistema e é possível haver uma violação sem detecção. Embora em ambientes pequenos seja possível adotar a noção de que a entidade normativa detecta 100% dos casos de violação, é difícil acreditar nessa mesma idéia em ambientes de proporções mais realistas. Por isso é interessante pensar que os agentes atuando neste ambiente têm a capacidade de raciocinar sobre a eficiência do monitoramento e fazer uso dessa informação para decidir entre determinados planos de ação que possam trazer maior grau de utilidade, mas com risco de violação, ou menor grau e nenhum risco para eles.

É possível também que a entidade normativa do ambiente possa alterar a intensidade com a qual ela o monitora. Da mesma forma que os agentes podem se beneficiar com essa variação de intensidade, o sistema de normas pode elevar sua capacidade de detecção em determinados momentos para aumentar a expectativa que os agentes têm de serem descobertos e, por consequência, desencorajá-los a burlar normas, e então reduzir essa capacidade para também reduzir o custo de operação do sistema normativo. O raciocínio sobre normas por parte dos agentes leva à geração de novos planos de ação que levam em consideração a probabilidade de detecção e o *trade-off* de utilidade total obtida pelo agente ao final da execução desse plano. Dependendo da discrepância entre a probabilidade observada pelo agente e a probabilidade real de detecção, têm-se mais ou menos chance de gerar planos que incluam violações de normas.

8.1 Captura de ações

O monitoramento de ações dos agentes é fundamental para a realização das atividades de detecção de violações de normas. Para isso, no entanto, é preciso um mecanismo que permita o acesso à essa informação primeiro. Uma das maneiras de implementar esse mecanismo é modificando a implementação da arquitetura de agente para que seja feita a gravação de quaisquer ações que tenham sido executadas. Então, ao executar uma ação qualquer, a captura

é feita como consequência e a informação da atividade é gravada em uma estrutura de dados. Essa estrutura é chamada de "Histórico de Ações". Assim como com as normas, é preciso limitar as informações que compõem a representação de uma ação de acordo com o contexto da simulação.

Definição 2. *Um registro do Histórico de Ações é representado pela tupla:* $\mathcal{R} = \langle g, a, B \rangle$

Em que:

- g representa o agente que está executando a ação a.
- a representa a ação que está sendo executada pelo agente g.
- B representa o estado interno do agente g no momento da execução da ação a.

No Exemplo 5, é registrada a ação de aceitar um passaporte quando o agente percebe que o passaporte não é válido. Considerando a existência da norma no sistema com o formato especificado em 6, pode-se afirmar que ocorreu violação da norma. É interessante ressaltar que, em Jason, o operador `not` significa que o agente não acredita que o predicado seja verdadeiro, ou seja, essa crença não se encontra presente em sua *belief-base*. Um exemplo de ação que não violaria uma norma pode ser visto no Exemplo 7.

Exemplo 5.

$$\begin{aligned}\mathcal{R} = \langle & corrupt\_officer1, \\ & accept(Passport), \\ & [working, credits(X), goal(Y)]\rangle\end{aligned}$$

Exemplo 6.

$$\begin{aligned}\mathcal{N} = \langle & prohibition, \\ & [not\ valid(Passport), working], \\ & accept(Passport), \\ & sanction(10)\rangle\end{aligned}$$

Exemplo 7.

$$\begin{aligned}\mathcal{R} = \langle & corrupt\_officer1, \\ & accept(Passport), \\ & [valid(Passport), working, credits(X), goal(Y)]\rangle\end{aligned}$$

O armazenamento dessas informações ocorre apenas após o sucesso de execução da ação. Caso a ação falhe, não há necessidade de armazená-la. É interessante destacar que o

monitoramento não é feito em tempo real. O monitoramento e detecção em tempo real não são mandatórios pelo fato de que o ambiente não é regimentado, ou seja, ainda que comportamentos proibidos sejam passíveis de punição, eles não são de forma alguma impedidos de serem executados. A decisão ou não de executar uma ação considerada ilegal fica totalmente a cargo do agente. Ainda é possível que uma violação seja detectada logo após sua execução, dependendo da intensidade com a qual as ações executadas são monitoradas e com o quão rápido a detecção e sancionamento da violação são realizados. Com esses fatores em mente, a necessidade de uma forma de monitoramento tão minuciosa torna-se questionável.

8.2 Agente Monitor

O agente tratado nesta seção é referenciado como Agente Monitor e faz uso indireto do mecanismo de captura de ações descrito na seção 8.1 por meio de uma interface de acesso ao histórico de ações. Essa interface permite a percepção de novas entradas no histórico, que são lidas e então selecionadas para serem enviadas para análise pelo mecanismo de detecção.

Quando um novo conjunto de informações é adicionado no histórico, um evento é notificado para os agentes conectados à interface, notificando que novas entradas estão prontas para serem analisadas. A taxa com que essas informações são coletadas e a frequência com que os agentes monitores são notificados podem ser ajustados de acordo com a intensidade de monitoramento do ambiente. Como discutido anteriormente, uma intensidade maior de monitoramento exige maior custo computacional, mas pode produzir um ambiente mais equilibrado.

As informações armazenadas no histórico de ações são apagadas tão logo são lidas pelo Agente Monitor, o que torna essa estrutura semelhante a uma fila. Essa funcionalidade evita problemas de redundância, em que uma ação seria analisada mais de uma vez, desperdiçando tempo de computação e, para o caso de uma ação ilegal, gerando uma sanção extra desnecessária. Após lidas, o monitor faz uma seleção de quais registros serão de fato analisados, lembrando que há custo para fazer o monitoramento [22]. Tal seleção faz parte do mecanismo de controle de intensidade do monitoramento e pode obedecer a algumas regras pré-estabelecidas ou partir do próprio raciocínio do agente. Algumas dessas técnicas são descritas a seguir:

Seleção aleatória É configurada uma taxa com a qual as ações são mantidas para análise.

Seleção baseada em probabilidade Ações que não possuem uma norma associada são descartadas. As que possuem são enviadas com uma probabilidade configurável.

Seleção empírica Esta oferece maior complexidade por exigir implementação de aprendizado. As taxas de violação para as normas são aprendidas e a seleção é feita sobre as maiores probabilidades.

8.3 Agente *Enforcer*

Foi visto que o agente monitor compõe parte do mecanismo de monitoramento do ambiente, a outra parte sendo a captura de ações dos agentes. O segundo componente necessário em um ambiente normativo é o agente de fiscalização, ou agente *enforcer*, que faz a análise das ações capturadas de forma a detectar a ocorrência de violações. Este agente é responsável também por aplicar as funções de penalização definidas pelas normas.

As informações descritas na Seção 8.1 são analisadas pelo agente *enforcer* e comparadas com o contexto de aplicação das normas ativas no sistema. Caso o estado de aplicação da norma coincida com o estado do ambiente no momento da execução, a ação tomada pelo agente é analisada em seguida. Seja a norma uma obrigação, se a ação executada coincidir com a ação encapsulada, a norma foi obedecida e a análise cessa. Os Exemplos 7 e 8 ilustram essa situação. No entanto, se as condições não forem equivalentes, então uma violação ocorreu e a respectiva sanção deve ser aplicada, este é o caso representado pelos Exemplos 8 e 9. Da mesma forma, caso a norma seja uma proibição e a ação executada seja a mesma ação encapsulada, uma violação deve ser detectada e punida (Exemplos 5 e 6). Caso as ações difiram, não há violação e a análise cessa (Exemplos 6 e 7).

Exemplo 8.

$$\begin{aligned}\mathcal{N} = \langle & obligation, \\ & [valid(Passport), working], \\ & accept(Passport), \\ & sanction(10)\rangle\end{aligned}$$

Exemplo 9.

$$\begin{aligned}\mathcal{R} = \langle & corrupt\_officer1, \\ & reject(Passport), \\ & [valid(Passport), working, credits(X), goal(Y)]\rangle\end{aligned}$$

8.4 Módulo Normativo

O módulo normativo do sistema representa a camada responsável por fornecer aos agentes normativos as ferramentas necessárias para acesso às ações executadas no ambiente, às normas que se encontram ativas durante a execução da simulação e às funções de sancionamento de agentes. A percepção de normas e sanções por parte dos agentes comuns do sistema também fica a cargo deste módulo. Essas funções, quando mapeadas para um ambiente CArtAgO, compõem um artefato.

Para os agentes, este artefato serve como o elemento do ambiente pelo qual é possível perceber as mudanças efetuadas no sistema de normas. Para o *enforcer*, o artefato serve como uma interface de acesso para a base de dados normativa no ato de análise das ações dos agentes reportadas pelo agente monitor. Em caso de detecção de violação, essa interface também fornece as operações necessárias para penalizar os devidos agentes.

8.5 Percepção das normas

A introdução do módulo normativo no sistema não é suficiente para que os agentes estejam sujeitos às regras do ambiente. É preciso que eles tomem conhecimento das regras o quanto antes, para que tenham a chance de evitar comportamentos ilegítimos. O reconhecimento de normas por parte dos agentes acontece por meio de percepções, assim como acontece com qualquer outra informação gerada pelo ambiente.

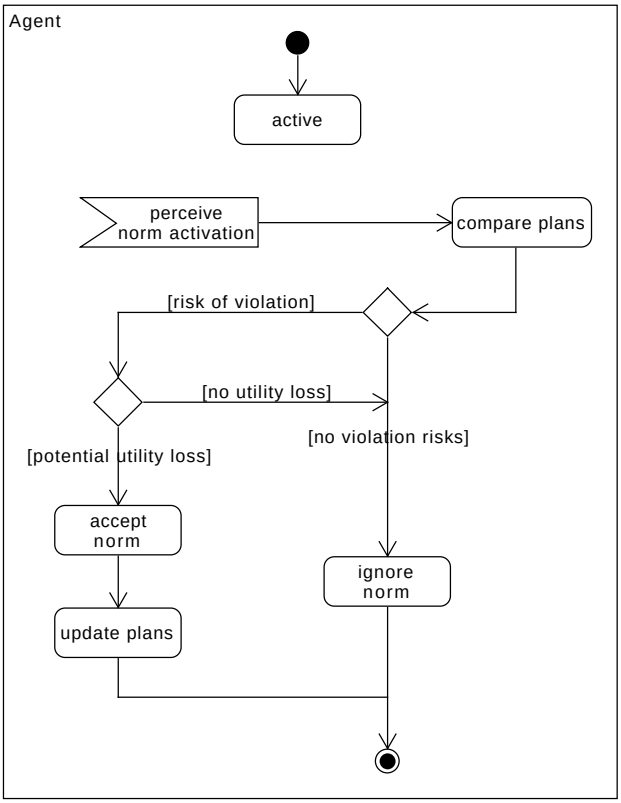

Figura 8.1 – Ao perceber a ativação de uma norma, o agente pode decidir se a aceita ou não.

Por meio de uma conexão com o artefato de normas, os agentes são capazes de perceber mudanças na camada normativa do sistema e então reagir a essas mudanças. Os seguintes eventos são perceptíveis aos agentes:

- Inclusão de novas normas no sistema.

- Ativação de normas.
- Desativação de normas.
- Alteração de normas.
- Destruição de normas.

Nas situações de inclusão de uma nova norma ou de ativação de uma já existente, o agente deve estudar a norma de forma a concluir se é interessante que ela seja aceita ou não. Se for aceita, é necessário verificar se há risco de violação em algum dos planos especificados para o agente e se há alguma outra norma que tenha sido aceita por ele que conflite com a norma mais recentemente adotada. Se houver, a inclusão de novos planos ou desativação de planos antigos deve ser considerada, assim como uma revisão do banco de normas do agente. O fluxo de aceitação ou rejeição de uma nova norma por parte de um agente pode ser visto na Figura 8.1, enquanto o fluxo para a destruição de uma norma está ilustrado na Figura 8.2.

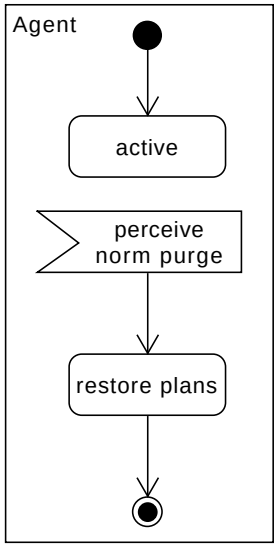

Figura 8.2 – Ao perceber a destruição de uma norma, o agente restaura seus planos originais.

Enquanto uma norma sempre deve ser aceita, a decisão final de aderir ou não é sempre do agente, com base no que ele acredita ser mais útil para si. No caso de haver de fato aceitação da norma, a norma percebida passa a fazer parte da *belief-base* do agente. Utilizando essa distinção entre percepção e crença [21], é possível identificar quando uma norma está sendo considerada nos planos de um agente ou não.

9. DIAGRAMAS E IMPLEMENTAÇÃO

Este capítulo é dedicado às descrições dos diagramas relativos a arquitetura geral do sistema e ao detalhamento da implementação realizada nas diferentes camadas do *framework*. Inicialmente os diagramas serão expostos e explicados para que então a implementação possa ser apresentada de forma mais clara.

Os diagramas ilustram os diferentes níveis de abstração adotados para a conceitualização e construção do *framework* NormMAS. Primeiro, é necessário entender como a estrutura geral do sistema funciona, quais os elementos envolvidos e seus papeís no contexto normativo de um sistema multiagente. Em seguida, considerando que o sistema deve ser programado, apresenta-se o modelo arquitetural de pacotes, onde cada pacote possui uma característica e as classes de implementação que dizem respeito a ela.

9.1 Arquitetura Geral do Sistema

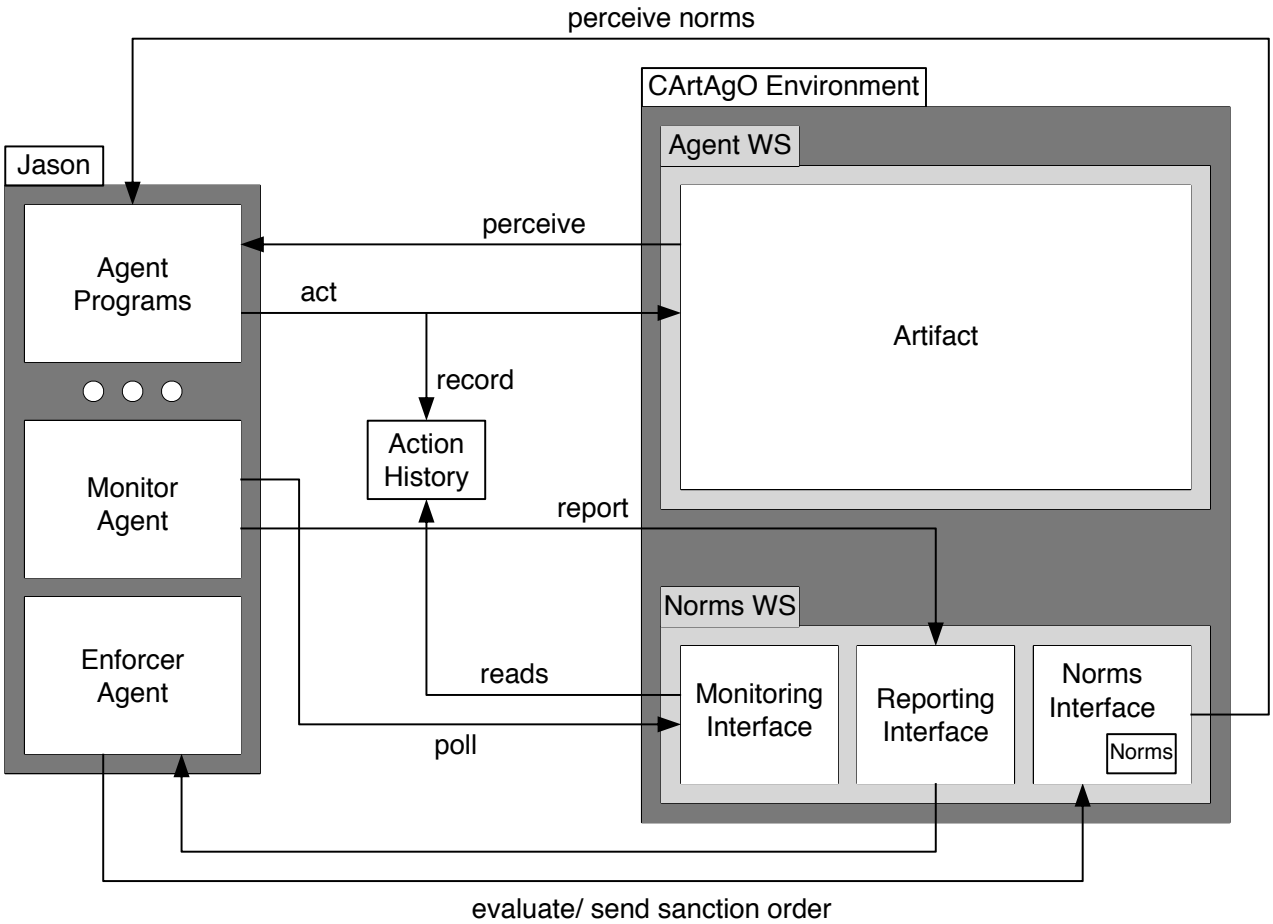

Figura 9.1 – Elementos do *framework* NormMAS

O diagrama representado na Figura 9.1 apresenta os componentes base do sistema. É possível observar também a relação estabelecida entre esses componentes por meio de ações e a direção que cada ação segue. Uma breve descrição de cada elemento é apresentada a seguir.

Agent Programs Programas de agentes que populam o MAS e interagem com o ambiente utilizando os artefatos do *Agent Workspace*. Devem ser configurados como observadores da *Norms Interface* para que possam perceber a influência das normas e a aplicação destas por meio de sanções. Ao interagir com artefatos, suas ações são, ao mesmo tempo, gravadas no *Action History* para futura leitura por um *Monitor Agent*.

Monitor Agent Agente responsável pelo monitoramento das ações dos demais agentes. Faz uso da *Monitoring Interface* para ler os registros armazenados no *Action History*. Ao observar uma ação de agente com sucesso, envia um relatório para o *Enforcer Agent* utilizando o artefato *Reporting Interface*.

Enforcer Agent Agente responsável por aplicar sanções em agentes cujas transgressões foram detectadas. A rotina de detecção é iniciada após a leitura bem sucedida de um relatório obtido da *Reporting Interface* e faz uso da *Norms Interface* para a análise em si.

Action History *Singleton* que armazena um banco de ações executadas pelos agentes.

Reporting Interface Artefato que realiza a passagem de relatórios do *Monitor Agent* para o *Enforcer Agent* seguindo o modelo Produtores e Consumidores.

Monitoring Interface Artefato que é utilizado por agentes monitores para ler os registros do *Action History*.

Norms Interface Artefato que é utilizado pelos *Enforcer Agents* para avaliar as normas e as ações reportadas pelo monitor durante a execução do sistema. Por meio dessa interface os demais agentes também recebem percepções de sanções quando estas forem efetuadas.

Esse diagrama assemelha-se àquele apresentado na primeira etapa do trabalho e sugere o modelo arquitetural de um sistema multiagente normativo implementado com as tecnologias Java, Jason e CArtAgO. Este modelo sofreu algumas modificações desde sua primeira versão e essas diferenças são abordadas nesta seção. A primeira diferença a ser reparada é no modo como as ações executadas pelos agentes são gravadas no Histórico de Ações. No modelo anterior um artefato tinha sido proposto para realizar tal função, através do que foi referido como *Artifact Linking*, função nativa do ambiente CArtAgO para disparar funções de um artefato por meio de outro artefato. Essa abordagem foi substituída por uma re-implementação da arquitetura de agente adaptada para agentes Jason atuarem em ambientes CArtAgO, em que, no ato da execução de uma ação, ela é automaticamente gravada no histórico. Essa nova maneira de persistir as ações dos agentes libera o desenvolvedor da tarefa monótona de programar uma conexão entre os distintos artefatos do ambiente e um artefato que grava as execuções realizadas.

Outra diferença que deve ser notada entre os diagramas é a maneira com quem o Agente Monitor lê novas ações gravadas no Histórico de Ações. O monitor (ou monitores) do sistema deve fazer consultas constantes à interface de monitoramento para tentar captar novas ações. Caso o monitor obtenha sucesso em ler uma ação, esta será enviada, por meio da interface de *reporting*, ao agente Enforcer, que ficará responsável por dar continuidade à rotina normativa do sistema. A adição de um artefato de comunicação entre monitor e enforcer é, também, uma das modificações feitas sobre o modelo que deve ser observada. Através de um modelo baseado no problema dos produtores e consumidores, os agentes monitores do ambiente utilizam essa interface para depositar relatórios contendo as ações dos agentes que foram possíveis capturar e devem ser analisadas para detecção de violações. Ao passo em que essas ações são colocadas em fila, os agentes Enforcer poderão coletar esses relatórios e analisá-los.

9.2 Diagrama de Pacotes do Sistema

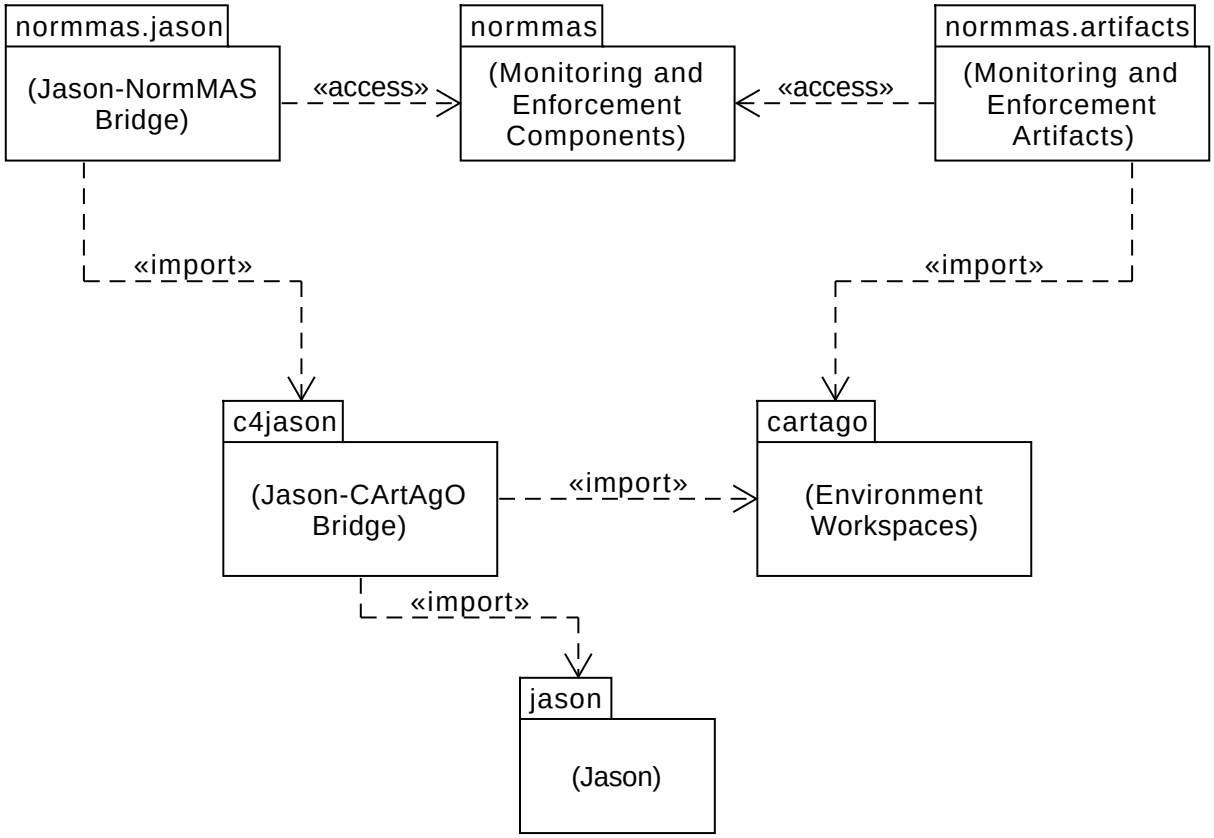

Figura 9.2 – Pacotes de desenvolvimento do *framework* NormMAS

O sistema NormMAS utiliza os pacotes padrão Jason e CArtAgO e extende algumas funcionalidades deste último. O pacote `normmas.artifacts`, por exemplo, fornece os artefatos necessários para mediar a interação de monitores, enforcers e seus ambientes. Neste pacote

encontram-se, então, os artefatos `MonitoringArtifact`, utilizado pelo Agente Monitor para ler ações do histórico, o `NormativeArtifact`, acessado pelo Agente Enforcer para detectar violações e aplicar sanções, e o `ReportingArtifact`, utilizado pelo Agente Monitor para enfileirar relatórios e pelo Agente Enforcer para lê-los.

O pacote `normmas.jason` contém os elementos necessários para adaptar o uso do *framework* NormMAS com Jason, como a arquitetura de agente utilizada para normatizar os agentes Jason atuando no ambiente CArtAgO. Os agentes Jason que devem ser controlados pelo sistema normativo devem ser implementados sob esta arquitetura. Por fim, o pacote `normmas` contém os elementos que são utilizados de acordo com os interesses normativos do sistema. Nele estão implementados os objetos de armazenamento de ações dos agentes, assim como a base de normas sobre a qual o sistema atua.

9.3 Agentes Normativos

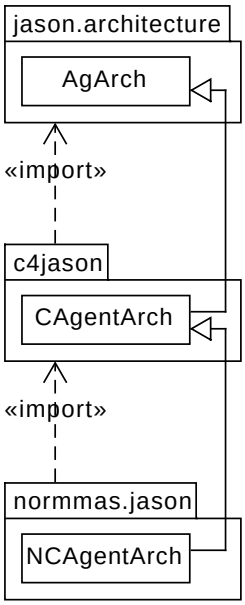

Figura 9.3 – Agentes normativos estendem o agente CArtAgO, que por sua vez estende a arquitetura Jason padrão.

Os agentes Jason atuando em ambientes CArtAgO são implementados sobre uma arquitetura que faz a ponte entre essas tecnologias. Essa arquitetura, referenciada na Figura 9.3 como `CAgentArch` é fornecida como parte do pacote *CArtAgO for Jason* e já implementa o mapeamento de percepções do ambiente e a tradução de ações em operações de artefatos. A classe `CAgentArch` é estendida para que, ao final da execução de uma ação, seja feita a captura da mesma, caso ela tenha sido bem sucedida. A classe resultante é referenciada como `NCAgentArch` (*NormMAS-CArtAgO Agent Architecture*) e é fornecida no pacote `normmas.jason`.

9.4 Sistema de Monitoramento

O sistema de monitoramento é dividido em duas partes: captura de ações e encaminhamento de relatórios. A captura de ações ocorre a cada execução de uma ação por parte de um agente. No contexto CArtAgO, isso significa que cada operação de artefato finalizada com sucesso é armazenada no histórico de ações do sistema. Se tratando de uma operação que ocorre sem intervenção do agente, a captura é implementada diretamente na sua arquitetura, e dependendo da arquitetura escolhida para a construção do agente, a solução de captura pode variar. Para este trabalho, foi utilizada uma extensão da arquitetura de agente Jason adaptada para ambientes CArtAgO, e a captura da ação foi inserida ao final de cada execução de ação, caso a ação não tenha falhado. Esse mecanismo está ilustrado na Figura 9.4.

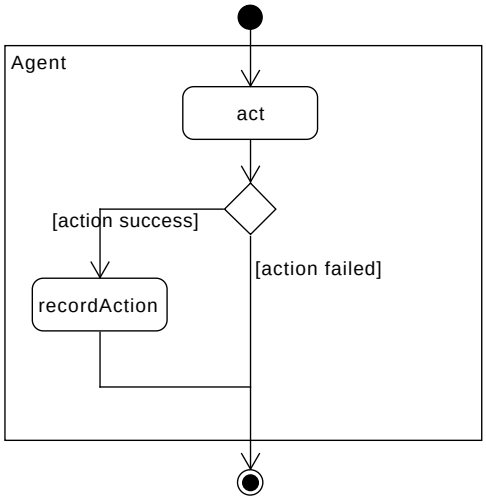

Figura 9.4 – Diagrama de atividades do processo de captura de ações.

Para que as ações gravadas pelo mecanismo descrito anteriormente possam ser analisadas, é preciso que um agente faça o encaminhamento dessas ações em forma de relatórios para outro agente. Para fazer isso, uma dinâmica semelhante à do problema dos produtores e consumidores foi utilizada, em que um agente deposita informações em um *buffer* continuamente, enquanto outro agente lê essas informações concorrentemente. O diagrama de sequência apresentado na Figura 9.5 contém o fluxo de operação do agente monitor, que representa o produtor de informações. Ao detectar uma nova ação pelo evento `+actionAvailable`, o monitor tenta realizar a leitura desta: dependendo da intensidade do monitoramento, esta leitura pode falhar, e caso isso aconteça então o monitor retornará ao seu estado inicial. No entanto, se a leitura for bem sucedida, o monitor deposita um novo relatório na *Reporting Interface*.

O monitoramento do sistema não está livre de custos [22], especialmente quando todas as ações são analisadas indiscriminadamente. De forma a possibilitar que o monitoramento opere de modo menos exaustivo, três estratégias estão disponíveis e podem ser configuradas

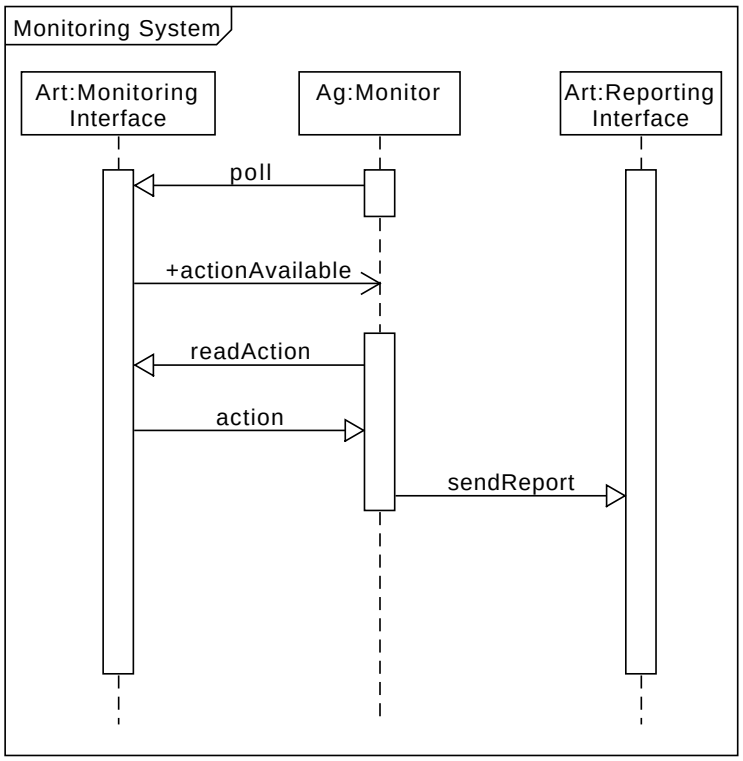

Figura 9.5 – Diagrama de sequência para o fluxo de monitoramento.

externamente por meio de um arquivo de configurações de monitoramento. Internamente, o monitor pode alterar suas configurações arbitrariamente, se necessário, utilizando operações do artefato de monitoramento. Essas estratégias estão descritas a seguir:

PROBABILISTIC As ações são lidas em ordem de gravação, porém há uma chance do monitor não ler a ação com sucesso. Essa chance está diretamente ligada à intensidade de monitoramento do ambiente. Essa intensidade pode ser configurada com um valor de 0 a 100 (intervalo fechado), sendo que o valor 100 representa o monitoramento exaustivo, ou seja, todas as ações são lidas, e o valor 0 representa a inibição do monitor (nenhuma ação é lida).

ENFORCED_ACTIONS_ONLY As ações são lidas em ordem de gravação, porém só são consideradas aquelas que estão associadas a alguma norma ativa no sistema.

ENFORCED_PROBABILISTIC Uma mistura das estratégias *PROBABILISTIC* e *ENFORCED_ACTIONS_ONLY*. As ações são lidas em ordem de gravação e são consideradas somente aquelas que estão associadas a alguma norma ativa no sistema, porém há uma chance, determinada pela intensidade de monitoramento, desse registro ser lido.

9.5 Sistema de Fiscalização (*Enforcement*)

O sistema de *enforcing* representa o consumidor de informações dentro do sistema normativo. Um agente *enforcer* se conecta à *Reporting Interface* e aguarda novos relatórios para análise. Quando um novo relatório está disponível, o evento `+newReport` é disparado e então sua leitura é realizada. Ao obter um registro para análise, o agente consulta a *Normative Interface* para obter o conjunto de normas ativas no sistema e verificar se alguma delas foi violada de acordo com o registro recém lido.

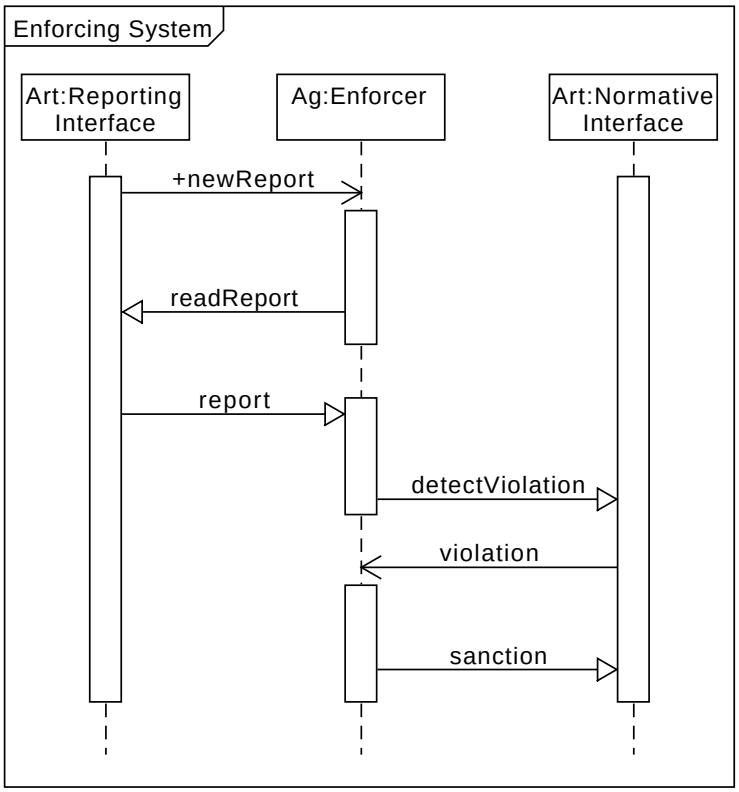

Figura 9.6 – Diagrama de sequência para o fluxo de fiscalização *enforcing*.

O agente *enforcer* percebe violações por intermédio do evento `+violation`, disparado cada vez que violação é identificada durante a rotina de detecção. Utilizando a mesma *Normative Interface*, o *enforcer* executa a função de punição, que sinaliza ao agente transgressor a punição encapsulada na norma violada. Os detalhes da rotina de detecção são abordados na seção à seguir.

Para que este fluxo funcione, no entanto, é necessário definir explicitamente quais agentes estão sujeitos às normas do ambiente. Embora a detecção das violações ocorra independentemente dessa definição, a sanção só poderá ser percebida pelos agentes caso estejam observando o artefato, assim como ocorre com quaisquer outros eventos mencionados. Isto pode ser configurado utilizando a operação `focus`, que é padrão do pacote CArtAgO. Da mesma

forma, os agentes que precisam ser mediados devem se submeter explicitamente à *Normative Interface* através da função `register`, para que o ambiente possa localizá-los e, assim, enviar os sinais de punição sempre que cabível.

9.6 Detecção de violações

Para fazer a detecção de violações, o Algoritmo 9.1 foi utilizado. A operação de detecção executa para cada relatório de ação R lido por um agente *enforcer*. Para cada ação lida, será verificado se alguma das normas ativas no sistema foi burlada. A operação inicia com uma lista de violações vazia (linha 2), e à medida em que violações forem detectadas para uma ação, serão adicionadas na lista para processamento posterior.

```
1:  function DetectViolation(⟨g, a, B⟩)
2:      V ← [ ]
3:      for each n = ⟨m, c, X, T, p⟩ ∈ ActiveNorms do
4:          isViolated ← False
5:          if ContextApplies(X, B) then
6:              if ConditionApplies(c, T, a, B) then
7:                  if m = prohibition then
8:                      isViolated ← True
9:              else
10:                 if m = obligation then
11:                     isViolated ← True
12:         if isViolated then
13:             V ← V ∪ {n}
14:     for each n ∈ V do
15:         SignalViolation(n, g)
```

Algoritmo 9.1 – Algoritmo para detecção de violações.

O processo de detecção em si ocorre em duas etapas: verificação de contexto e verificação de condição da aplicação. A variável $isViolated$ é inicializada com o valor $False$ a cada norma verificada pelo motivo de que, no primeiro momento, norma alguma está violada: apenas quando uma violação é detectada com sucesso o valor dessa variável deve ser alterado para $True$. A primeira etapa verifica se há, dentro da *Belief-base* do agente executor da ação, os predicados definidos no contexto X da norma n. O algoritmo específico desta etapa pode ser verificado na Figura 9.2. Isso significa que, para que seja considerado que a ação faz parte do contexto normativo, a condição $X \subseteq B$ deve ser satisfeita. Não sendo o caso, o valor de $isViolated$ permanece $False$ e a próxima norma pode ser analisada. Caso o contexto seja verificado, então o algoritmo segue para a verificação da condição de ativação, cujo processo está representado no Algoritmo 9.3 e descrito no parágrafo a seguir.

```
1: function ContextApplies(X = [l_1, ..., l_n], B = [l_1, ..., l_n])
2:     Require count(X) ≤ count(B)
3:     for each p ∈ X do
4:         negated ← p is of the form ¬φ
5:         if negated then
6:             p ← φ
7:         contains ← False
8:         for each l ∈ B do
9:             if l = p then
10:                contains ← True
11:        if (negated xor contains) then
12:            continue
13:        else
14:            return False
15:    return True
```

Algoritmo 9.2 – Algoritmo para comparação de contextos.

A condição de ativação de uma norma pode ser tanto uma ação executada por um agente quanto um estado ou conjunto de estados interno deste. Levando isso em consideração, a verificação de condição observa, primeiramente, qual o tipo de condição normativa que é desejado verificar. Caso a condição seja do tipo *action*, então basta verificar se a ação a executada pelo agente é a mesma definida na condição T. Caso a condição seja do tipo *state*, então é preciso verificar se o contexto definido por T está contido no estado interno do agente B, ou simplesmente se $T \subseteq B$. Ou seja, na verificação da condição de estados, o mesmo algoritmo de verificação de contextos, utilizado na primeira etapa da verificação da norma, é utilizado para verificar a condição de ativação de norma. Caso $T \neq a$ ou $T \not\subseteq B$, a função retorna com valor $False$ à função de detecção.

Quando as condições de contexto e condição normativa são satisfeitas, é considerado que o fluxo da norma foi seguido, restando apenas uma verificação final: se a norma trata de uma **proibição** ou **obrigação**. Se tratando de uma proibição, o fato do fluxo da norma ter sido seguido significa também que houve violação dessa norma, pois a norma de proibição define aquilo que não deve ser feito. Sendo assim, o valor de $isViolated$ é atualizado para $True$ para que, antes da análise da próxima norma iniciar, a norma violada seja adicionada à lista V. Caso a norma trate de uma obrigação, então o fluxo de execução ter sido seguido significa que a obrigação foi cumprida e, portanto, não houve violação. Seguindo esse raciocínio, caso a condição de ativação da norma não tenha sido verificada como verdadeira, mas a norma tratar de uma obrigação, então houve violação pois o processo obrigatório definido pela norma não foi concluído. Da mesma maneira que, neste caso, se a norma tratar de uma proibição, não terá havido violação.

Ao final da rotina de detecção de violações, é feito o processamento de todas as normas detectadas como violadas. Para cada norma violada, a função `signal`, nativa do ambiente CArtAgO, é chamada para que o agente transgressor perceba um evento de sanção descrito

```
1: function ConditionApplies(c, T, a, B)
2:     if c is action then
3:         if T = a then
4:             return True
5:     else
6:         if ContextApplies(T, B) then
7:             return True
8:     return False
```

Algoritmo 9.3 – Algoritmo para análise de condições de ativação de uma norma.

pelo elemento p da tupla n. Essa função pode ainda ser estendida para atualizar algum outro tipo de controle de violações, como um banco de dados central que controle as informações de cada agente, por exemplo.

10. EXPERIMENTOS E RESULTADOS

Ao finalizar a estrutura básica do *framework*, este é aplicado ao cenário de teste "Agentes de Imigração". As seguintes expectativas foram levantadas:

- Agentes corruptos terão utilidade superior para valores baixos de monitoramento.
- Agentes corretos manterão seus valores de utilidade durante todos os cenários.
- Em um determinado ponto, agentes corruptos apresentarão utilidade menor que os corretos, devido à intensidade de monitoramento crescente.

No gráfico representado na Figura 10.1 é possível ver a relação entre a intensidade de monitoramento do ambiente e a utilidade média dos agentes de acordo com o cenário de teste. A utilidade do agente, neste caso, é medida pela quantidade de créditos que cada agente consegue obter através de seu trabalho. Observando o comportamento geral do gráfico, é possível perceber que a quantidade de créditos dos agentes *corrupt officer 1* e *corrupt officer 2* decresce à medida que a intensidade de monitoramento aumenta. Isso é esperado, pois quanto mais ações são monitoradas, maior as chances de detecção de violação e, portanto, menos vantajoso é violar as normas do ambiente.

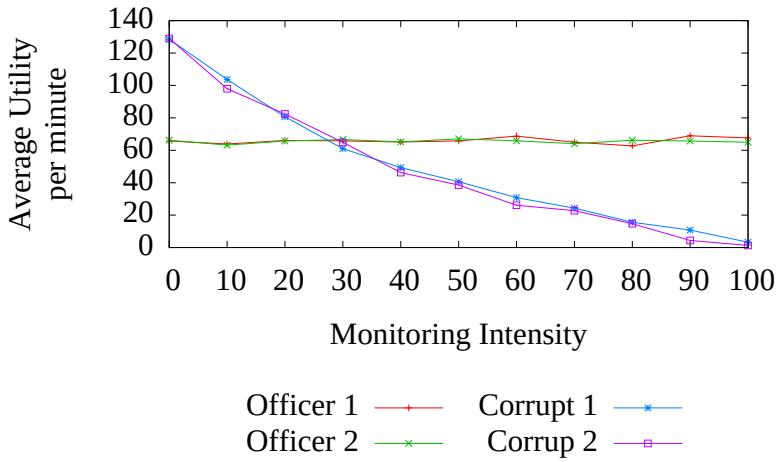

Figura 10.1 – Utilidade de agentes corruptos cai com o aumento da fiscalização.

Os dados contidos na Tabela 10.1 foram utilizados para montar o gráfico 10.1. Os valores de utilidade para cada agente levam em consideração 10 simulações de 5 minutos para 11 intensidades diferentes (de 0 a 10, intervalo fechado). A média é calculada em utilidade por minuto. A partir das estatísticas extraídas, é possível calcular também quantas violações aconteceram de fato, quantas foram detectadas e qual a expectativa de detecção baseado na intensidade de monitoramento. Esses dados estão representados na tabela 10.2 e representados no gráfico 10.2.

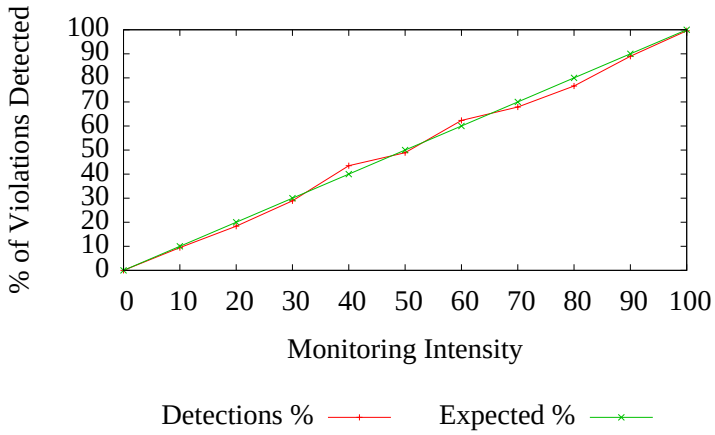

Figura 10.2 – Porcentagem de detecção e intensidade de monitoramento estão relacionados.

Tabela 10.1 – Recompensas por agente × Intensidade de Monitoramento.

Intensidade	*officer1*	*officer2*	*corrupt officer1*	*corrupt officer2*
0	65, 70	66, 20	128, 50	129, 00
10	63, 90	63, 20	103, 70	98, 00
20	66, 10	65, 70	80, 70	82, 40
30	65, 80	66, 60	61, 00	64, 90
40	65, 10	65, 20	49, 32	46, 29
50	65, 80	67, 00	40, 58	38, 52
60	68, 71	65, 84	30, 81	26, 07
70	64, 97	64, 11	24, 31	22, 72
80	62, 70	66, 2	15, 60	14, 70
90	68, 95	65, 72	10, 81	4, 39
100	67, 67	65, 01	3, 31	1, 28

À medida que a intensidade aumenta, os agentes corruptos se vêem em desvantagem em relação aos agentes corretos. Enquanto o valor de utilidade média para os agentes *officer* 1 e 2 se mantém dentro da mesma faixa em todos os cenários, o mesmo não acontece com os agentes *corrupt officer* 1 e 2, que têm suas utilidades gradativamente reduzidas. É possível notar também, ao observar o gráfico 10.3, que a quantidade de violações também diminui em função da intensificação do monitoramento. Este comportamento é esperado, pois ao serem sancionados, os agentes corruptos são suspensos de atividade por 10 segundos, reduzindo o número de ações totais e, consequentemente, o número de violações.

A proporção de atividades ilícitas detectadas acompanha o crescimento da intensidade de monitoramento do ambiente. Embora não sejam exatamente iguais, a quantidade de violações detectadas é, para todos os casos testados, muito próxima do valor esperado. Isso mostra que a intensidade de monitoramento do ambiente está diretamente ligada à probabilidade de detecção da violação, dada a utilização de uma estratégia de monitoramento puramente probabilística. Para analisar a tabela, é importante levar dois fatos em consideração:

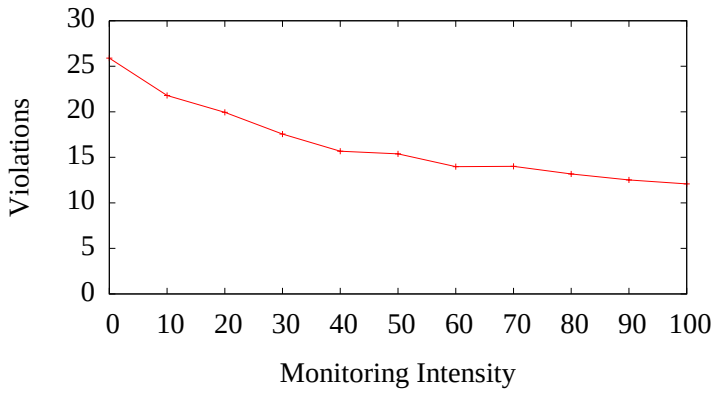

Figura 10.3 – Quantidade de violações decresce devido à suspensão de atividades dos agentes.

- a coleta de estatísticas é periódica; e
- o monitoramento não ocorre em tempo real.

Ou seja, é possível que haja uma violação e, em seguida, sua detecção, porém isto ocorrer entre uma coleta de estatística e outra. Portanto, alguns desvios de valores podem ocorrer, como é o caso do monitoramento de intensidade 100, em que a expectativa para 12.09 violações é 12.09 detecções, porém o valor real obtido foi de 12.04.

Tabela 10.2 – Intensidade × Detecções.

Intensidade	Violações	Detecções	Expectativa
0	25, 9	0, 0	0, 0
10	21, 8	2, 06	2, 18
20	19, 94	3, 66	3, 98
30	17, 56	5, 08	5, 26
40	15, 67	6, 82	6, 26
50	15, 39	7, 53	7, 69
60	13, 98	8, 71	8, 38
70	14, 01	9, 51	9, 80
80	13, 18	10, 1	10, 54
90	12, 52	11, 14	11, 26
100	12, 09	12, 04	12, 09

Os resultados obtidos com a simulação de teste foram de acordo com o esperado. Nos cenários onde a intensidade de monitoramento é baixa (menor que 30%), agentes corruptos possuem tanta utilidade quanto agentes corretos. Isso significa também que, para este cenário, não é necessário dedicar tantos recursos para o monitor para que a desvantagem de um comportamento ilegal se torne evidente. Caso um agente corrupto seja incluído no sistema com a capacidade de aprender, por exemplo, então com um monitoramento de 40% ele já será capaz

de perceber que sua performance está caindo e, então, alterar sua estratégia de operação. Com essa simulação teste, é possível concluir que o *framework* atinge seu objetivo de ser modular. Ao acoplar o *workspace* normativo desenvolvido no cenário Jason/CArtAgO "Agentes de Imigração", foi possível executar o MAS e extrair métricas de comportamento dos agentes baseado em um conjunto de normas definido previamente, sendo que essas métricas condizem com as expectativas levantadas inicialmente.

11. CONCLUSÃO

O *framework* NormMAS fornece uma estrutura para simular ambientes multiagente normativos de forma modular utilizando as tecnologias Java, Jason e CArtAgO. Com ele, é possível avaliar diferentes implementações de comportamento normativo. As estatísticas coletadas podem também ser configuradas para que os resultados das simulações possam ser comparados. Exemplos de abordagens que poderiam ser testadas incluem [21], [24], [25] e [26].

Ao desenvolver sistemas multiagente sobre ambientes CArtAgO, é possível mediar as interações entre agentes e seus ambientes por meio de artefatos. Com eles é possível também construir uma interface única de acesso a elementos externos ao sistema original, tornando possível a construção de artefatos que se comunicam com elementos normativos como o Histórico de Ações e a Base Normativa do sistema. Assim, é possível realizar a captura de ações dos agentes, o monitoramento de suas ações e o controle normativo do ambiente pelo acoplamento do módulo NormMAS em um MAS que opere em ambientes CArtAgO. Utilizando a tecnologia Java é possível integrar ambientes CArtAgO com agentes Jason, e por meio dessa interação fazer também a captura das ações dos agentes para armazenamento no Histórico de Ações.

A integração com Jason é uma das vantagens de desenvolver ambientes utilizando CArtAgO. Como mencionado anteriormente, Jason permite a programação de agentes de forma flexível e consistente, porém carece na programação de ambientes para simulação por oferecer apenas uma solução centralizada. Por permitir a distribuição de operações em artefatos, CArtAgO é uma tecnologia chave para fazer a separação de responsabilidades entre agentes e ações, ou seja, agentes têm acesso apenas às ações que são pertinentes a eles. Outra vantagem é o fato de ser possível integrar outras arquiteturas de agente em um ambiente CArtAgO, contanto que seja feita a devida adaptação, permitindo a criação de ambientes heterogêneos.

Embora a implementação tenha sido testada com sucesso, é possível levantar alguns pontos onde a estrutura pode ser aprimorada. No sistema normativo, por exemplo, embora as respectivas operações de ativação e desativação de normas estejam presentes no *framework*, não é possível especificar, de acordo com os formalismos de norma propostos em [22] ou [23], as condições de ativação e expiração de uma norma. Ou seja, uma vez adicionada a norma, o desenvolvedor precisa implementar explicitamente o momento de sua ativação e, se for o caso, desativação. Caso os formalismos citados fossem suportados, bastaria um mecanismo único responsável por monitorar o estado do sistema e atualizar o banco de normas com as normas aplicáveis àquele estado do sistema para que as normas fossem ativadas e desativadas sem necessidade de intervenção externa.

Outro ponto identificado é a ausência de controle de acesso à interface normativa. As operações de controle de normas, como adição, ativação, desativação e destruição estão acessíveis a quaisquer agentes que se conectem ao artefato respectivo. Como os agentes normativos precisam se conectar à essa interface para perceberem eventuais sanções, isso significa que eles também possuem acesso às funções anteriormente citadas. Uma solução é implementar uma in-

terface específica para a percepção de sanções, de onde não é possível acessar as demais funções de controle normativo. Entretanto, ainda seria possível conectar os agentes a outras interfaces programaticamente, e então o propósito de criar a interface extra seria vencido. O ideal é, então, implementar um sistema de hierarquia de agentes, em que agentes podem controlar normas em níveis específicos, assim como proposto em [23].

Sumarizando, dos objetivos identificados inicialmente, os seguintes foram atingidos:

- Implementar um workspace normativo, que contenha uma interface para monitoramento e uma para aplicação das normas. (fundamental)
- Construir um agente que monitore as ações dos demais agentes do sistema. (fundamental)
- Construir um agente que detecte e sancione violações de normas. (fundamental)
- Construir um cenário de simulação para validar a implementação realizada. (fundamental)
- Utilizar a linguagem de programação de agentes *Jason* e do *framework* de programação de ambientes CArtAgO. (fundamental)
- Implementar um módulo de monitoramento de intensidade ajustável. (desejável)

Enquanto os seguintes não foram:

1. Adicionar controle de acesso à camada normativa. (desejável)
2. Implementar processamento de normas com condições de ativação e expiração. (desejável)
3. Implementar processamento de normas com suporte à hierarquia de agentes. (opcional)
4. Implementar um mecanismo de aprendizado nos agentes para que estes possam aprender as normas do ambiente. (opcional)
5. Permitir a inclusão de agentes com poder normativo. (opcional)

Sendo que os objetivos 1 e 2 estão relacionados aos problemas citados anteriormente. Os objetivos 3, 4 e 5 representam extensões do *framework*, funcionalidades que não são fundamentais para seu funcionamento e, portanto, é necessário finalizar o desenvolvimento dos objetivos desejáveis antes que estes possam ser propriamente estudados e implementados.

A contribuição oferecida por este trabalho é a base para o sistema normativo de um ambiente multiagente heterogêneo. Uma vez fornecida esta base, é possível trabalhar sobre diferentes aspectos do desenvolvimento de agentes e ambientes para melhorar a qualidade dos

estudos realizados sobre simulações. Como trabalho futuro, o problema das condições de ativação e expiração das normas será estudado e uma solução será implementada. Serão desenvolvidas também arquiteturas de agente que raciocinam sobre as normas do ambiente, evitando violações completamente ou, no mínimo, mitigando a perda de desempenho ocasionada por violações constantes, caso estas sejam inevitáveis [21]. Tais agentes poderão usar também a informação da intensidade de monitoramento para seu benefício [22]. Por fim, a hierarquização dos agentes será introduzida no *framework* como uma maneira de lidar com normas específicas de organizações e de evitar que normas de uma organização sejam modificadas por um agente externo a ela.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] T. M. Behrens, M. Dastani, J. Dix, and P. Novák, "MASSim: Multi-Agent Systems Simulation Platform," *In: Begehung des Simulationswissenschaftlichen Zentrums*, 2008.

[2] M. Dastani, J. Dix, and P. Novák, "Multi-agent programming contest," 2005. Disponível em http://www.multiagentcontest.org.

[3] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[4] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

[6] Y. Shoham, "Agent-oriented programming," *Artif. Intell.*, vol. 60, pp. 51–92, Mar. 1993.

[7] M. Woolridge, *Intelligent Agents*. The MIT Press, 1999.

[8] R. A. Brooks, "A robust layered control system for a mobile robot," tech. rep., Cambridge, MA, USA, 1985.

[9] M. E. Bratman, *Intention, Plans and Practical Reason*. Harvard University Press, 1987.

[10] U. Wilensky, "NetLogo." Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL, 1999. http://ccl.northwestern.edu/netlogo/.

[11] U. Wilensky and W. Stroup, "HubNet." Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL., 1999. http://ccl.northwestern.edu/netlogo/hubnet.html.

[12] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multiagent Simulation Environment," *Simulation*, vol. 81, pp. 517–527, July 2005.

[13] M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, 2013.

[14] F. Meneguzzi and L. De Silva, "Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning," *The Knowledge Engineering Review*, vol. FirstView, pp. 1–44, 9 2013.

[15] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," in *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, (San Francisco, CA, USA), pp. 608–620, Morgan Kaufmann Publishers Inc., 1971.

[16] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, AAAI'94, (Menlo Park, CA, USA), pp. 1123–1128, American Association for Artificial Intelligence, 1994.

[17] A. S. Rao, "Agentspeak(l): Bdi agents speak out in a logical computable language," in *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-agent World : Agents Breaking Away: Agents Breaking Away*, MAAMAW '96, (Secaucus, NJ, USA), pp. 42–55, Springer-Verlag New York, Inc., 1996.

[18] A. Ricci, M. Viroli, and A. Omicini, "Cartago: A framework for prototyping artifact-based environments in mas," in *Proceedings of the 3rd International Conference on Environments for Multi-agent Systems III*, E4MAS'06, (Berlin, Heidelberg), pp. 67–86, Springer-Verlag, 2007.

[19] J. F. Hubner, J. S. Sichman, and O. Boissier, "Developing organised multiagent systems using the moise+ model: Programming issues at the system and agent levels," *Int. J. Agent-Oriented Softw. Eng.*, vol. 1, pp. 370–395, Dec. 2007.

[20] M. Fagundes, S. Ossowski, and F. Meneguzzi, "Analyzing the tradeoff between efficiency and cost of norm enforcement in stochastic environments populated with self-interested agents," in *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, 2014.

[21] F. Meneguzzi and M. Luck, "Norm-based behaviour modification in BDI agents," in *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pp. 177–184, 2009.

[22] F. Meneguzzi, B. Logan, and M. Silva Fagundes, "Norm monitoring with asymmetric information," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, (Richland, SC), pp. 1523–1524, International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[23] N. Oren, M. Luck, and S. Miles, "A model of normative power," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, (Richland, SC), pp. 815–822, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[24] J. Lee, J. Padget, B. Logan, D. Dybalova, and N. Alechina, "Run-time norm compliance in BDI agents," in *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pp. 1581–1582, 2014.

[25] W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman, "Normative conflict resolution in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 2, pp. 124–152, 2009.

[26] N. Criado, E. Argente, V. Botti, and P. Noriega, "Reasoning about norm compliance," in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, (Richland, SC), pp. 1191–1192, International Foundation for Autonomous Agents and Multiagent Systems, 2011.

APÊNDICE A – AGENTES JASON

A.1 Agente *Officer*

```
credits(0).
goal(50).
working.

!work.

+payment(S)
        <-      ?credits(C);
                Y = C+S;
                -+credits(Y);
                .my_name(Me);
                .concat("Credits for ", Me, Stat);
                updateStat(Stat, Y);
                .print("Received credits for work. Current total: ", (Y)).

+!work
        <-      .wait("+normsReady");
                !openBooth(Booth);
                focus(Booth);
                ?subject_to_norms(Id);
                focus(Id);
                register;
            !receivePassports.

+!openBooth(Booth)
        <-      .my_name(Me);
                makeArtifact(Me, "examples.java.ImmigrationBooth", [], Booth).

+?subject_to_norms(Id)
        <-      lookupArtifact("norms", Id).

-?subject_to_norms(Id)
        <-      .wait(100);
                ?subject_to_norms(Id).

+!receivePassports
        <-      .wait(2000);
                receivePassport(Passport);
                !checkPassport(Passport);
                !!receivePassports.

+!checkPassport(Passport)
        : valid
        <-      acceptPassport(Passport);
                .print("Passport approved.").

+!checkPassport(Passport)
        : not valid
        <-      rejectPassport(Passport);
                .print("Passport rejected.").
```

A.2 Agente *Corrupt Officer*

```
credits(0).
goal(50).
penalty(0).
working.

!work.

+payment(S)
        <-      ?credits(C);
                Y = C+S;
                -+credits(Y);
                .my_name(Me);
                .concat("Credits for ", Me, Stat);
                updateStat(Stat, Y);
                .print("Received credits for work. Current total: ", (Y)).

+!work: true
        <-      .wait("+normsReady");
                !openBooth(Booth);
                focus(Booth);
                ?subject_to_norms(Id);
                focus(Id);
                register;
                !receivePassports.

+!openBooth(Booth) : true
        <-      .my_name(Me);
                makeArtifact(Me, "examples.java.ImmigrationBooth", [], Booth).

+?subject_to_norms(Id)
        <-      lookupArtifact("norms", Id).

-?subject_to_norms(Id)
        <-      .wait(100);
                ?subject_to_norms(Id).

+!receivePassports: true & not suspended
        <-      .wait(2000);
                receivePassport(Passport);
            !checkPassport(Passport);
            !!receivePassports.

+!receivePassports : suspended
        <-      .wait("-suspended");
                !receivePassports.

+!checkPassport(Passport) : valid
        <-      acceptPassport(Passport);
                .print("Passport approved.").

+!checkPassport(Passport) : not valid
        <-      acceptPassport(Passport);
                incStat("Violations");
                .print("Passport should've been rejected, but was rather approved since I need th

+sanction(X)
        <-      ?credits(C);
```

```
                Y = C - X;
                -+credits(Y);
                ?penalty(P);
                T = P + X;
                -+penalty(T);
                .print("Oops! I've been discovered and lost ", X, " credits for my transgression.
                -working;
                +suspended.

+suspended
        <-      .print("suspended");
                .wait(10000);
                .print("not suspended");
                -suspended;
                +working.
```