3
JSAN

O JSAN fornece os mecanismos necessários para criar simulações de sistemas multiagentes normativos para compreender os impactos das normas sobre agentes capazes de adotar diferentes estratégias para lidar com normas. Para tanto, o *framework* possibilita a implementação de diferentes estratégias normativas, coleta de informações do ambiente simulado, visualização de informações acerca dos cenários simulados, suporte a implementação de diferentes estratégias de movimento dos agentes no ambiente e a geração de normas e objetivos dos agentes.

Neste capítulo inicialmente é apresentada a ideia geral do *framework* JSAN e seus respectivos módulos são apresentados, seguidos pela descrição do *framework* JSAN.

3.1.
Framework JSAN

O *framework* JSAN proposto neste trabalho contém uma parte imutável, chamada de pontos fixos, e a parte configurável, chamada de pontos flexíveis. Por utilizar o paradigma de orientação a objetos, é apresentado um diagrama de classes relativo à estrutura do *framework*. Nesse diagrama os pontos fixos e variáveis são identificados por estereótipos. Os pontos fixos são identificados utilizando o estereotipo *flozenSpot* e os pontos variáveis *hotSpots*.

O JSAN (i) fornece mecanismos que possibilitam a inserção de normas em um sistema multiagentes, (ii) possibilita a implementação de estratégias utilizadas pelos agentes com relação ao seu movimento no ambiente, (iii) permite a implementação de relatórios que apresentam informações acerca da simulação realizada, (iv) fornece uma estrutura de normas que suporta a definição de condições de ativação ou desativação de uma norma, estabelecimento do elemento regulado pela norma (ou seja, a ação ou estado regulado pela norma), e as sanções (ou seja, as recompensas e punições), (v) fornece mecanismos para

monitoramento do ambiente, (vi) possibilita que diferentes estratégias normativas sejam implementadas e analisadas e (vii) suporta a geração de normas e objetivos.

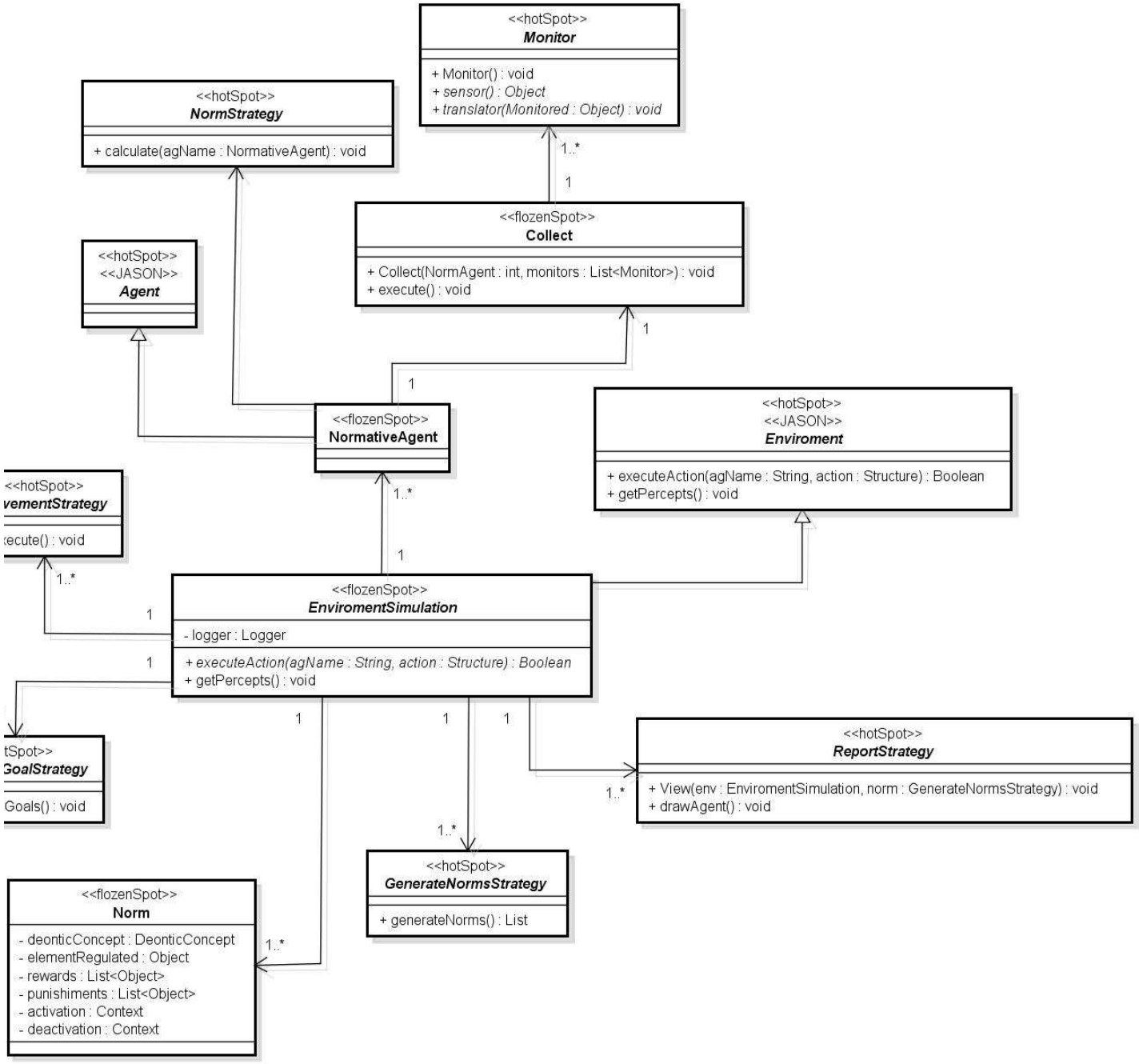

Figura 3.1 - Diagrama de Classes

3.2. Detalhes do JSAN

O diagrama de classes apresentado na Figura 3.1 apresenta as principais classes e métodos do *framework* JSAN. Os agentes são representados pela classe

NormativeAgent uma extensão da classe *Agent* do Jason (ver Seção 2.3). Além disto, o ambiente de simulação é representado pela classe *EnviromentSimulation* que estende a classe *Enviroment* do Jason e fornece suporte para desenvolver um de ambiente de simulação. O método *executeAction* foi estendido de *Enviroment* e grande parte do código do ambiente é escrito nele. Sempre que um agente tenta executar uma ação básica, seu identificador e sua ação escolhida são passados para este método. Portanto, o código do método *executeAction* deve verificar se a ação é válida e então realizar o que for necessário para que a ação seja de fato executada. Possivelmente a ação irá alterar as percepções dos agentes. Se este método retonar *true* significa que a ação executou com sucesso.

A classe *EnviromentSimulation* também é responsável por gerenciar a criação de normas e objetivos individuais dos agentes, isto através de instancias das classes *GenerateNormStrategy* e *GenerateGoalsStrategy*, respectivamente. Além disso, *EnviromentSimulation* é responsável por gerenciar as estratégias utilizadas pelos agentes para se movimentarem no ambiente conforme os eventos ocorridos durante a simulação, estendendo a classe *MovementStrategy*.

Cada norma do ambiente é representada no diagrama de classes apresentado na Figura 3.1 pela classe *Norm*. Como discutido na Seção 2.2 uma norma contém um conjunto de sanções, ou seja, recompensas e punições que são representadas através das listas *rewards* e *punishiments*, respectivamente. As condições de ativação e desativação de uma norma são representadas pelos atributos *activaction* e *deactivation*, e o elemento regulado pela norma é definido através do atributo *elementRegulated*. A classe *Norm* também contém um conceito deôntico, isto é, se a norma é de proibição, permissão ou obrigação, que é representado pelo atributo *deonticConcept*.

Como mencionado na seção 3.1, o JSAN, monitora o ambiente através da classe *Collect*. Esta gerencia um conjunto de monitores representados pela classe abstrata *Monitor*, que se baseia em dois conceitos para realizar a coleta, filtragem e estruturação dos dados coletados a partir do monitoramento do ambiente, são eles: *sensor* (representado pelo método *sensor*) e *translator* (representado pelo método *translator*), apresentados em detalhes na seção 3.2.5. No *sensor* devem ser definidos onde os dados devem ser coletados para que os agentes consigam lidar com as normas (ambiente de simulação). No *translator*, deve ser definido

como os dados coletados a partir do *sensor* devem ser estruturados de forma que os mesmos possam ser entendidos pelo agente.

Adicionalmente, o JSAN fornece um mecanismo para reportar o impacto das normas sobre os agentes normativos. Para utilizar este mecanismo é necessário estender a classe *ReportStrategy*, passando os seguintes parâmetros: (i) o ambiente em que a simulação está sendo realizada e (ii) uma implementação da classe *NormStrategy* (Ver Seção 3.2.6), a qual contém a estratégia utilizada pelo agente para lidar as normas.

Em suma, para construir uma simulação de sistemas multiagentes normativos utilizando JSAN é necessário estender algumas classes. A classe *GenerateNormsStrategy* deve ser estendida para gerar as normas que irão existir no ambiente de simulação. Já a classe *GenerateGoalStrategy* é preciso estende-la para que se possa gerar os objetivos dos agentes. Ao estender a classe *MovementStrategy* é possível gerar diferentes estratégias de movimento para os agentes no ambiente simulado. Na classe *NormStrategy* tem um método chamado *calculate* (Ver Classe *NormStrategy*) o qual deve ser implementado para descrever a estratégia que será utilizada para os agentes lidarem com as normas. Para que os agentes sejam capazes de monitorar informações do ambiente de simulação é necessário estender a classe *Monitor*. A classe *ReportStrategy* deve ser estendida para reportar informações específicas de cada simulação.

Para facilitar o entendimento da estrutura do *framework* JSAN, esta será explicada em mais detalhes nas seções seguintes junto com seus respectivos diagramas de classe.

3.2.1. GenerateNormsStrategy

No diagrama de classes apresentado na Figura 3.2, temos a classe *EnviromentSimulation*, responsável por gerenciar um conjunto de estratégias para geração de normas representadas pela classe abstrata *GenerateNormsStrategy*, a qual possui o método abstrato *generateNorms* onde deve ser definido a maneira como são geradas as normas no ambiente de simulação.

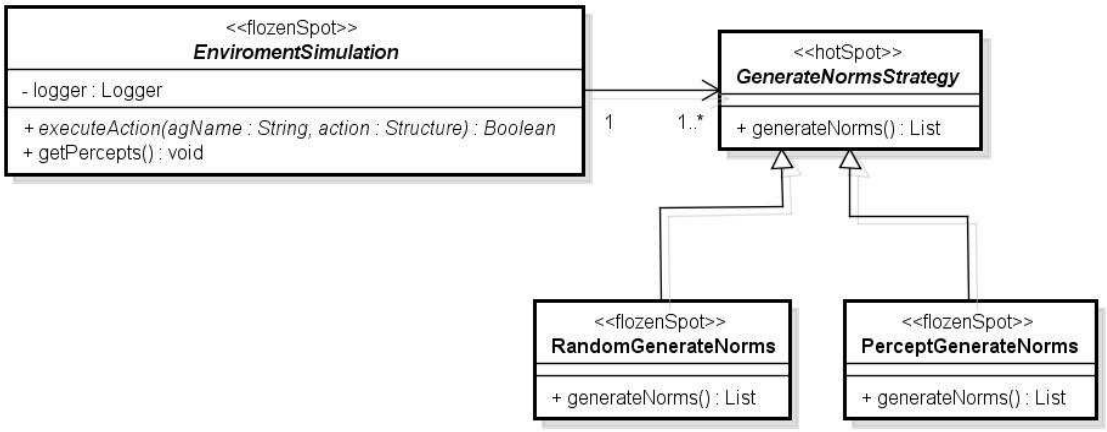

Figura 3.2 - Geração de Normas

Diversas implementações para gerar as normas podem ser definidas, o JSAN já fornece duas estratégias: (i) geração de normas aleatórias para um ambiente (Ver classe *RandomGenerateNorms*) e (ii) criação de normas no ambiente para um dado grupo de agentes (Ver classe *PerceptGenerateNorms*).

3.2.2. MovementStrategy

Na Figura 3.3 temos o ambiente de simulação, representado pela classe *EnviromentSimulation* que é responsável por gerenciar os movimentos dos agentes na simulação, representados pela classe *MovementStrategy*.

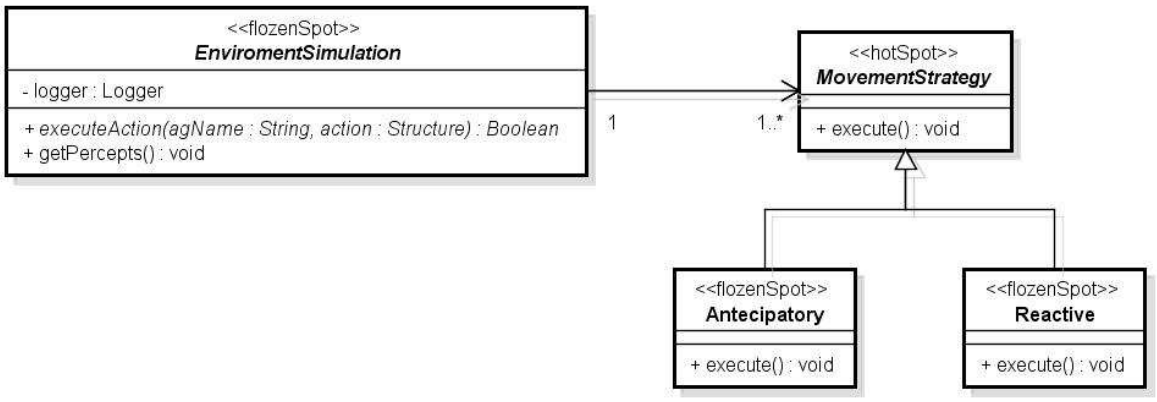

Figura 3.3 - Movimento dos Agentes

Diversas implementações de movimento dos agentes na simulação podem ser definidas, por hora, em JSAN foram disponibilizadas estratégias baseadas no trabalho de (BOSSE e GERRITSEN, 2010): (*reativa*) os agentes irão se

movimentar depois que um determinado estado seja atingido ou uma ação seja executada, e (*antecipatória*) onde os agentes irão predizer que um dado evento possa ocorrer e procurar se movimentar antes que ele aconteça.

3.2.3. ReportStrategy

Na Figura 3.4 temos o ambiente de simulação, representado pela classe *EnviromentSimulation*. Esta é responsável por gerenciar a visualização das características particulares de cada cenário de uso, através da classe *ReportStrategy*.

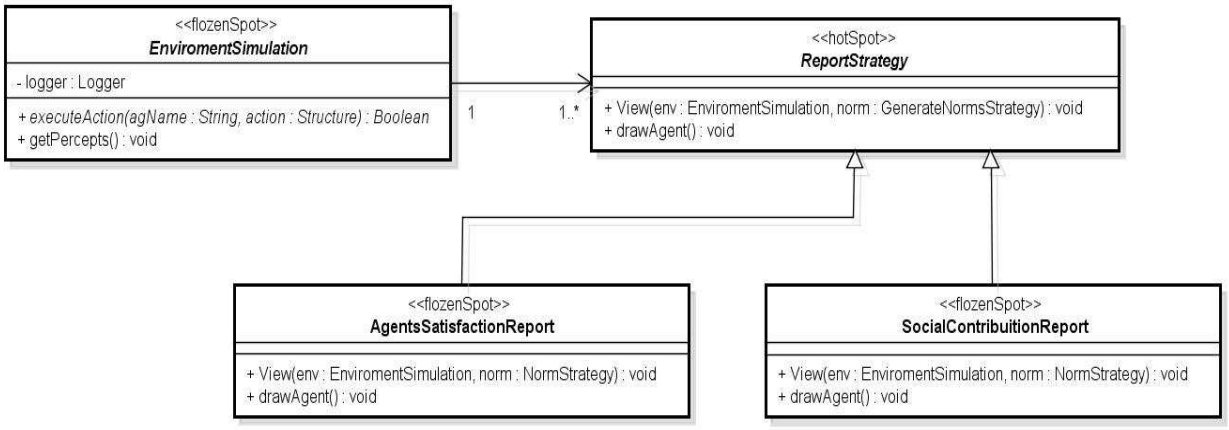

Figura 3.4 - Mecanismo para gerar Relatório

Diversas implementações da forma como pode ser visto os resultados da simulação podem ser definidos, por hora, em JSAN foram disponibilizadas as seguintes funcionalidades: (i) mecanismos capazes de mostrar informações sobre a contribuição social (por contribuição social entenda o quanto o agente contribui para o estabelecimento de uma ordem social desejável) do agente (Ver Classe *SocialContribuitionReport*) e (ii) sua satisfação pessoal (por satisfação individual entenda o quanto foi realizado dos desejos individuais de cada agente) (Ver Classe *AgentsSatisfactionReport*).

3.2.4. Collect

O classe *collect* é responsável por gerenciar a execução do conjunto de monitores representados pela classe abstrata *Monitor*, a qual possui os métodos abstratos *sensor* e *translator* representando duas importantes tarefas no que diz

respeito à monitoramento.O parâmetro *NormAgent* é importante para poder atribuir quais agentes receberão as informações coletadas no ambiente.

No *sensor* deve ser especificado o que monitorar e quais dados devem ser coletados durante o monitoramento. Neste caso, o JSAN oferece: (i) um mecanismo (representado pela classe *MessageMonitor*) que coleta a última mensagem enviada no ambiente e (ii) um mecanismo para coletar informações de outros agentes (representado pela classe *EnvironmentMonitor*).

No método *translator* devem ser implementados mecanismos que recebem os dados coletados pelo *sensor* atual (representado pelo parâmetro *Monitored* do tipo *Object*) e o estruture de forma que o mesmo possa ser entendido pelos agentes.

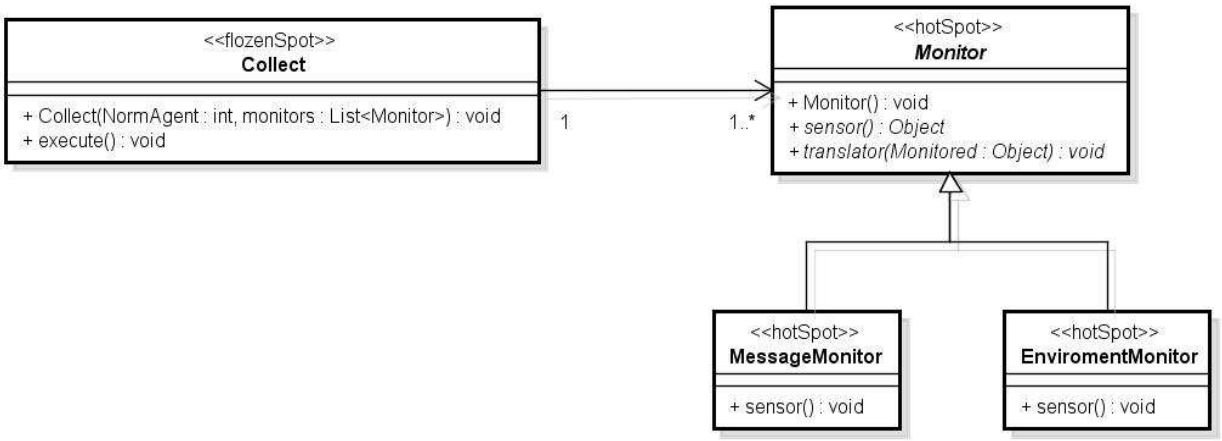

Figura 3.5 - Atividade *Collect*

3.2.5. NormStrategy

Na Figura 3.7, os agentes estão representados pela classe *NormativeAgent* que é responsável por gerenciar um conjunto de estratégias para lidar com normas, representado pela classe *NormStrategy*. O *framework* JSAN permite que diversas estratégias para lidar com normas sejam implementadas, algumas destas já estão disponíveis: (i) uma estratégia *Social* a qual o agente foca em cumprir com as

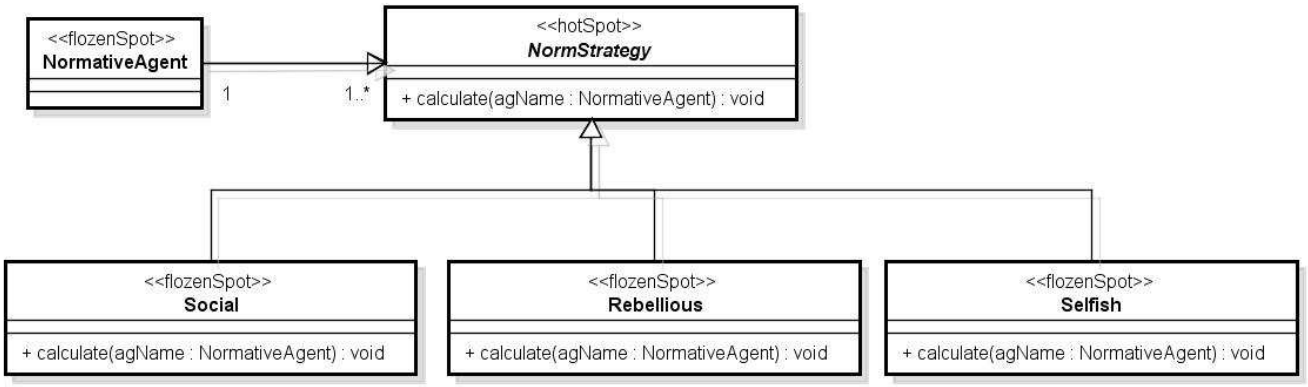

Figura 3.6 - Estratégia para lidar com normas

normas sem se preocupar com seus objetivos individuais (Ver classe *Social* na Figura 3.7) (LÓPEZ, LUCK e D'INVERNO, 2002), (ii) uma estratégia *Rebellious* para os agentes que se preocupam somente em alcançar seus objetivos individuais, sem se importar com as punições atreladas a violação daquela norma (Ver classe *Rebellious* na Figura 3.7) (LÓPEZ, LUCK e D'INVERNO, 2002) e (iii) uma estratégia *Selfish*, na qual o agente analisa a situação aonde o cumprimento de normas irá contribuir com a realização de pelo menos um de seus objetivos individuais (Ver classe *Selfish* na Figura 3.7) (LÓPEZ, LUCK e D'INVERNO, 2002).

3.2.6. GenerateGoalsStrategy

No diagrama de classes apresentado na Figura 3.8, temos a classe *EnviromentSimulation* (uma extensão da classe *Enviroment* do JASON), responsável por gerenciar um conjunto de estratégias para geração de objetivos representado pela classe abstrata *GenerateGoalStrategy*, a qual possui o método abstrato *generateGoals* onde deve ser definido a maneira que serão gerados os objetivos dos agentes.

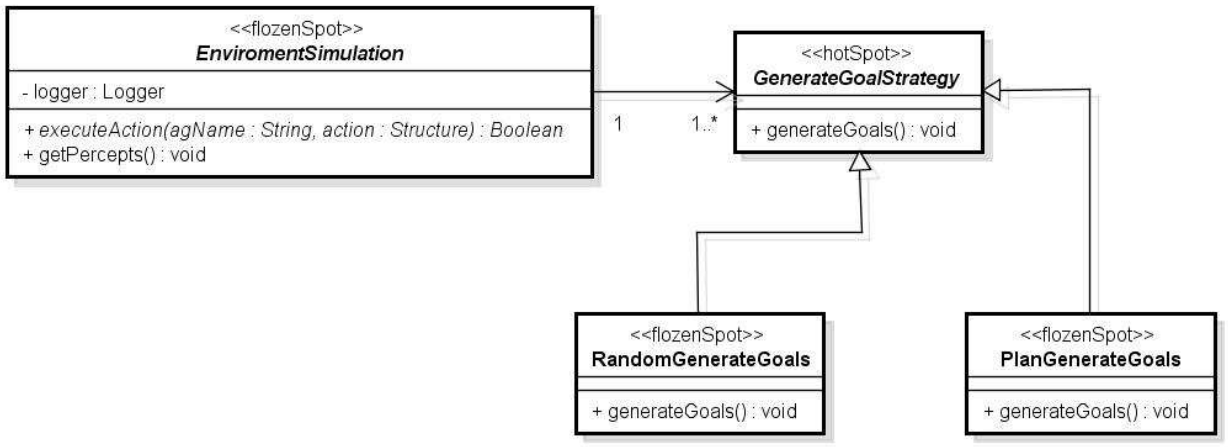

Figura 3.7 - Gerar *Goals*

Foi implementado no método *generateGoals* da classe *RandomGenerateGoals* a geração aleatória de objetivos individuais dos agentes dado um conjunto de objetivos pré-definidos para cada ambiente de simulação. Já a criação de objetivos através do método *generateGoals* da classe *PlanGenerateGoals* é feito com base nas motivações que o agente tem para alcançar um determinado estado do ambiente.

3.3. Pontos Fixos e Pontos Flexíveis do JSAN

Sabendo que JSAN estende o JASON, o núcleo do JASON é também o núcleo do JSAN e os pontos flexíveis do JASON são os pontos flexíveis do JSAN.

Os pontos fixos especificamente providos pelo JSAN são:

– Um mecanismo para criar agentes normativos: (Ver classe *NormativeAgent*): através da classe *NormativeAgent* é possível criar agentes capazes de lidar com normas.

– Um mecanismo para criar o ambiente de simulação (Ver classe *EnviromentSimulation*): através da classe *EnviromentSimulation* é possível criar o ambiente onde os agentes irão perceber e agir.

– Dois mecanismos de geração de normas (Seção 3.2.1): (i) geração de normas aleatórias dado um conjunto de normas previamente conhecido e (ii) criação

de normas no ambiente de simulação para regular apenas um grupo de agentes.

– Dois mecanismos para os agentes movimentarem na simulação (classe abstrata *MovementStrategy* na Figura 3.3): (i) reativa, os agentes irão se movimentar depois que um determinado estado comece ou uma ação se realize, e (ii) antecipatória, onde os agentes irão predizer que um dado evento possa ocorrer e procurar se movimentar antes que ele aconteça.

– Dois mecanismos para serem utilizados na visualização dos resultados das simulações (classe abstrata *ReportStrategy* na Seção 3.2.3): (i) irá verificar a contribuição social de uma norma caso seja cumprida ou violada pelos agentes e (ii) a satisfação pessoal do agente conforme o tempo decorrido da simulação.

– A atividade *Collect* (Seção 3.2.5) é responsável por gerenciar a execução do conjunto de monitores representados pela classe abstrata *Monitor*.

– Três mecanismos para o agente lidar com as normas (Seção 3.2.6): (i) *Social*, o agente foca em cumprir com as normas sem se preocupar com seus *goals* individuais, (ii) *Rebellious*, o agente que se preocupa somente em alcançar seus *goals* individuais, sem se importar com as punições atreladas a violação daquela norma e (iii) *Selfish*, o agente analisa a situação aonde o cumprimento de normas irá contribuir com a realização de pelo menos um de seus *goals* individuais, isto é, cumprir com normas é uma forma de conseguir os benefícios obtidos pelas recompensas.

– Dois mecanismos para geração dos *goals* dos agentes (Seção 3.2.7): (i) geração de *goals* aleatório dado um conjunto de *goals* previamente conhecido e (ii). geração de *goals* com base nas crenças e intenções dos agentes.

Os pontos flexíveis especificamente providos pelo JSAN são*:*

– Geração de normas (classe abstrata *GenerateNormsStrategy* Figura 3.2): permite acoplar outros algoritmos para gerar normas estendendo a classe abstrata *GenerateNormsStrategy*.

– Mecanismos para movimentação dos agentes (interface *MovementStrategy* na Figura 3.3): diferentes movimentos podem ser incorporados a partir da implementação da classer abstratra *MovementStrategy*.

– Visualização do ambiente de simulação (classe abstrata *ReportStrategy* na Figura 3.4): diferentes visualizações das características particulares de cada cenário de uso podem ser incorporados a partir da implementação de *ReportStrategy*.

– Monitoramento (classe abstrata *Monitor* na Figura 3.6): mecanismos para monitoramento podem ser adicionados estendendo a classe abstrata *Monitor*.

– Estratégias para lidar com normas (classe abstrata *NormStrategy* na Figura 3.7): técnicas para lidar com normas podem ser incorporadas a partir da implementação da classe abstrata *NormStrategy*.

– Geração de objetivos (classe abstrata *GenerateGoalStrategy* na Figura 3.8): é possível definir novas técnicas para gerar os objetivos dos agentes através da implementação da classe abstrata *GenerateGoalStrategy*.