Pontifícia Universidade Católica
do Rio de Janeiro

Marx Leles Viana

Design e Implementação de Agentes de Software Adaptativos Normativos

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do grau de Doutorado pelo Programa de Pós-graduação em Informática do Departamento de Informática da PUC-Rio.

Orientador: Prof. Carlos José Pereira de Lucena

Rio de Janeiro
Dezembro de 2016

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Marx Leles Viana

Design e Implementação de Agentes de Software Adaptativos Normativos

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Informática do Departamento de Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Carlos José Pereira de Lucena
Orientador
Departamento de Informática – PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Presidente
Departamento de Informática - PUC-Rio

Prof. Julio Cesar Sampaio do Prado Leite
Departamento de Informática - PUC-Rio

Prof. Simone Diniz Junqueira Barbosa
Departamento de Informática - PUC-Rio

Prof. Elder José Reioli Cirilo
Departamento de Informática – UFSJ

Prof. Paulo Sérgio Conceição Alencar
Computer Science Department – University of Waterloo

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro Técnico Científico – PUC-Rio

Rio de Janeiro, 05 de dezembro de 2016

Marx Leles Viana

Obteve seu Mestrado na PUC-Rio em 2012. Atualmente atua na área de Desenvolvimento de Software Orientado a Agentes no Laboratório de Engenharia de Software (LES), da PUC-Rio.

Ficha Catalográfica

Viana, Marx Leles

Design e Implementação de Agentes de Software Adaptativos Normativos / Marx Leles Viana; orientador: Carlos José Pereira de Lucena. Rio de Janeiro: PUC-Rio, Departamento de Informática, 2016.

v., 157 f.; il. ; 29,7 cm

1. Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2016.

Incluí referências bibliográficas.

1. Informática – Tese. 2. Sistemas Multiagentes. 3 Modelagem de Software. 4 Adaptação de Software. 5 Sistemas Normativos. I. Lucena, Carlos José Pereira de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Aos meus pais, Antonio Roberto e Rita, e aos meus irmãos,
Roberto e Viviane, pelo carinho e amizade.

Agradecimentos

A melhor parte de uma tese é poder agradecer as pessoas, que de alguma maneira contribuíram para tornar esse sonho realidade. Pois chega um momento da sua vida, que você sabe quem é imprescindível para você, quem nunca foi, quem não é mais, e quem sempre será!

Aos meus pais, Antonio Roberto e Rita pelo apoio e conselhos durante toda minha vida. Agradeço a eles em especial por sempre me mostrar que não existem barreiras quando buscamos nossos sonhos. Sem eles nada disso teria acontecido!

Aos meus irmãos e grandes amigos Roberto e Viviane, que apesar da distância nesses anos sempre estiveram ao meu lado, me incentivando a nunca desistir e que todas as dificuldades enfrentadas valeriam a pena.

A minha namorada Marina, por toda parceria nesses anos de luta, sempre me dando suporte para enfrentar todas as dificuldades nessa fase de nossa vida.

Ao meu orientador e amigo Professor Carlos José de Pereira Lucena, pela oportunidade de trabalharmos juntos e todo incentivo que recebi para a realização deste trabalho.

Ao Professor Paulo Alencar por toda ajuda e orientação, sem ele este trabalho não teria sido possível.

Aos membros da banca que contribuíram de forma contundente para o aperfeiçoamento deste trabalho.

Aos meus amigos que fizeram meus dias mais felizes e foram de grande ajuda para que este trabalho fosse realizado, muito obrigado a todos!

Ao pessoal do LES pela ajuda de todos os dias e dedicação ao nosso trabalho, em especial a Vera Menezes, pela cumplicidade e conselhos.

Ao CNPq e à PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Resumo

Viana, Marx Leles; Lucena, Carlos José Pereira de (Orientador). **Design e Implementação de Agentes de Software Adaptativos Normativos**. Rio de Janeiro, 2016. 157p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Sistemas multiagentes foram introduzidos como um novo paradigma para a conceituação, concepção e implementação de sistemas de software que estão se tornando cada vez mais complexos, abertos, distribuídos, dinâmicos, autônomos e altamente interativos. No entanto, a engenharia de software orientada a agentes não tem sido amplamente adotada, principalmente devido à falta de linguagens de modelagem que não conseguem ser expressivas e abrangentes o suficiente para representar abstrações relacionadas aos agentes de software e apoiar o refinamento dos modelos de projeto em código. A maioria das linguagens de modelagem não define como essas abstrações devem interagir em tempo de execução, mas muitas aplicações de software precisam adaptar o seu comportamento, reagir à mudanças em seus ambientes de forma dinâmica, e alinhar-se com algum tipo de comportamento individual ou coletivo de aplicações normativas (por exemplo, obrigações, proibições e permissões). Neste trabalho, foi proposta uma abordagem de metamodelo e uma arquitetura para o desenvolvimento de agentes adaptativos normativos. Acredita-se que a abordagem proposta vai avançar o estado da arte em sistemas de agentes de modo que tecnologias de software para aplicações dinâmicas, adaptáveis e baseadas em normas possam ser projetadas e implementadas.

Palavras-chave

Sistemas Multiagentes; Modelagem de Software; Adaptação de Software; Sistemas Normativos.

Abstract

Viana, Marx Leles; Lucena, Carlos José Pereira de (Advisor). **Design and Implementation of Adaptive Normative Software Agents**. Rio de Janeiro, 2016. 157p. Doctoral Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Multi-agent systems have been introduced as a new paradigm for conceptualizing, designing and implementing software systems that are becoming increasingly complex, open, distributed, dynamic, autonomous and highly interactive. However, agent-oriented software engineering has not been widely adopted, mainly due to lack of modeling languages that are expressive and comprehensive enough to represent relevant agent-related abstractions and support the refinement of design models into code. Most modeling languages do not define how these abstractions interact at runtime, but many software applications need to adapt their behavior, react to changes in their environments dynamically, and align with some form of individual or collective normative application behavior (e.g., obligations, prohibitions and permissions). In this paper, we propose a metamodel and an architecture approach to developing adaptive normative agents. We believe the proposed approach will advance the state of the art in agent systems so that software technologies for dynamic, adaptive, norm-based applications can be designed and implemented.

Keywords

Multi-agent Systems; Software Modeling; Software Adaptation; Normative Systems.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Engenharia de software baseada em agentes (sigla AOSE, que significa "Agent-Oriented Software Engineering" em inglês) emergiu como uma nova tecnologia para construção de sistemas complexos. Estes sistemas são caracterizados por serem distribuídos e compostos de entidades autônomas que interagem entre si (Wooldridge, 2011). Sistemas Multiagentes (SMAs) são sociedades nas quais entidades autônomas, heterogêneas e projetadas individualmente trabalham em função de objetivos que podem ser comuns ou diferentes (López, 2003). Assim, a utilização de agentes para a construção de sistemas complexos é considerada uma abordagem promissora (Zambonelli et al., 2003). No entanto, SMAs não têm sido amplamente adotados, principalmente devido à falta de linguagens de modelagem que exploram o uso de abstrações relacionadas a agentes e promovam o refinamento dos modelos de design para código (Gonçalves et al., 2015). Além disso, a maioria das linguagens de modelagem não define como estas abstrações interagem em tempo de execução (Beydoun et al., 2009), mas muitas aplicações de software precisam mudar o seu comportamento e reagir a mudanças nos seus ambientes dinamicamente.

Em geral, as linguagens de modelagem devem representar os aspectos dinâmicos e estruturais dos agentes de software, expressando as características essenciais das entidades. Os aspectos estruturais incorporam a definição das entidades, suas propriedades e seus relacionamentos. Vários autores reconhecem a importância de modelar os agentes em seus ambientes tanto no tempo de design como no tempo de execução (Van Riemsdijk et al., 2015a; Cernuzzi et al., 2014; Dam & Winikoff, 2013; Beydoun et al., 2009; Silva, 2004; Bresciani et. al., 2004). Tais autores também observaram que a maioria das linguagens de modelagem não representam alguns conceitos importantes presentes em SMAs como, por exemplo, adaptação e normas (Beydoun et al., 2009; Hollander, 2011). Sem a adoção dessas abstrações, não seria possível modelar certas classes de problemas relevantes e naturalmente solucionáveis por SMAs, a exemplo de serviços de emergência e

desastres naturais (e.g., alagamentos, incêndios, deslizamentos de terra, terremotos) (Beamon et al., 2006; Cerqueira, 2009; Janssen, 2010), casas e cidades inteligentes (Cook, 2012; Atzori et al., 2011; Gubbi et al., 2013), construção de patrimônios virtuais (e.g., Second Life (Bogdanovych et al., 2011)) e saúde (e.g., doenças ou patologias envolvendo o sistema imunológico, tais como HIV (Laroum, 2011)).

1.1 Motivação

Há alguns anos, o Laboratório de Engenharia de Software (LES) da PUC-Rio começou a trabalhar de forma intensa no desenvolvimento de sistemas de software para predição de deslizamento de terra (Cerqueira, 2009) e simulação de evacuação de pessoas em áreas de risco (Viana et al., 2015b, c) nas regiões de encosta do Estado do Rio de Janeiro. No entanto, esses sistemas eram compostos por agentes de softwares normativos (Viana et al., 2015a, d, e), capazes de raciocinar sobre normas com o uso de diferentes tipos de estratégias pré-fixadas.

Após uma extensa revisão de literatura, percebeu-se a oportunidade de desenvolver um modelo de raciocínio interno ao agente baseado na proposta de autoadaptação apresentada pela IBM (IBM, 2003) para criar agentes adaptativos normativos, ou seja, criar agentes capazes de lidar dinamicamente com restrições (normas) de ambiente. A partir dessa análise percebeu-se também que os trabalhos até então tinham sido desenvolvidos de forma *ad hoc*, isto é, apenas para uma dada finalidade, impedindo seu reuso em diferentes domínios de aplicação. Devido à ausência de uma abordagem genérica, decidiu-se investigar quais informações poderiam ser utilizadas pelos agentes para que pudessem lidar com normas em diferentes domínios.

Para auxiliar no entendimento de criação e manutenção de agentes que se adaptam para lidar com normas, a geração de documentação torna-se necessária. Com isso, decidiu-se realizar uma investigação para documentar as informações usadas pelos agentes para lidar com normas a partir do uso de uma nova linguagem de modelagem.

Com o objetivo de exemplificar situações de agentes capazes de se adaptarem a normas no ambiente, será exemplificado com dois cenários de uso. No primeiro cenário se situa no domínio de evacuação de pessoas em áreas de risco. O resultado

ideal para os bombeiros seria resgatar e prestar o maior número de atendimentos aos feridos. Entretanto, os bombeiros têm um grupo limitado de recursos, que são regulados pelo comandante dos bombeiros. Esta regulação é feita através de normas, que restringem o comportamento dos bombeiros para que o resgate ocorra de forma coordenada e com o melhor aproveitamento de recursos. Por exemplo, estes recursos podem ser veículos aéreos, veículos terrestres e equipamentos de escavação. Dado que os recursos são escassos, o comandante dos bombeiros é responsável por gerenciar a assistência de recursos aos grupos de salvamento. Para gerenciar os recursos, o comandante precisa adaptar o seu comportamento com relação as normas do sistema, com o objetivo de assistir a todos os salvamentos, priorizando a forma como os recursos devem ser distribuídos.

Já no segundo cenário de uso, foi considerado um mercado virtual, onde um agente vendedor precisa adaptar seu comportamento às normas do ambiente para que ele consiga realizar uma venda para um agente comprador. Os agentes vendedores devem seguir as normas que existem na loja, entretanto podem existir situações onde normas sejam conflitantes. Por exemplo, um agente vendedor só pode remarcar o preço dos seus produtos antes da loja abrir, mas é obrigado a remarcar o preço das mercadorias caso seja anunciada uma promoção. Com isso, o agente precisará decidir qual das normas cumprir e qual delas será violada. A detecção e resolução de conflito entre normas será realizada através da autoadaptação do agente ao lidar com as normas vigentes no ambiente e endereçadas ao seu papel.

1.2 Problema

Apesar de algumas linguagens de modelagem envolverem abstrações de adaptação e/ou normas (Gowri, 2014; Gómez-Rodríguez, 2014; Bresciani et al., 2004; Van Riemsdijk et al., 2015a; Mefteh, 2015; Cernuzzi et al., 2014; Gonçalves et al., 2015; Da Silva Figueiredo et al., 2011; Beydoun et al., 2009), estas abordagens não mostram explicitamente como fatores adaptativos e normativos influenciam um SMA, tais como:

(F1) as arquiteturas tradicionais de agentes e suas linguagens associadas não fornecem um mecanismo para um agente adaptar às normas que restringem seu comportamento (Meneguzzi & Luck, 2009);

(F2) devido a essa limitação, essas linguagens não podem ser usadas para modelar muitos aspectos essenciais na representação de agentes, tais como aspectos envolvendo adaptação, normas e as interações entre ambas (Barbier, 2015; Beydoun et al., 2009);

(F3) tanto no tempo de design quanto no de execução, as mudanças de comportamento dos agentes e suas reações à mudanças no ambiente precisam ser caracterizadas (Beydoun et al., 2009);

(F4) como agentes podem adaptar seus comportamentos para cumprir com as novas normas adotadas (Van Riemsdijk, 2015a).

Diante do apresentado, nota-se a necessidade de mecanismos que permitam o desenvolvimento de sistemas multiagentes capazes de: (F1) entender e adaptar as normas endereçadas a eles no ambiente; (F2) uma linguagem que permita representar os conceitos de normas e adaptação explicitamente; (F3) verificar como estas abstrações se relacionam; e (F4) representar mecanismos que auxiliem na construção de agentes capazes de se adaptarem para lidar com normas.

1.3 Limitações dos Abordagens Atuais

Embora seja possível encontrar soluções na literatura que possibilitam o projeto e/ou implementação de agentes capazes de lidar com normas ou realizar alguma forma de adaptação (Santos Neto, 2012b; Da Silva Figueiredo et al., 2011; Silva et al., 2008; Boissier et al., 2011; Beydoun, 2009; Cervenka & Trencansky, 2007; Gowri, 2014; Mefteh, 2015; Bernon, 2003), nenhuma delas apresenta uma solução para o desenvolvimento de uma linguagem de modelagem para descrever conceitos relacionados à adaptação e normas que: (i) suporte esses conceitos como abstrações de primeira classe; (ii) baseie estes conceitos em uma descrição explícita de um metamodelo de SMAs; (iii) modele os aspectos dinâmicos e estruturais frequentemente descritos em SMAs para estes conceitos; (iv) e promova o refinamento destes modelos de design para código. Além disso, nenhuma das abordagens estudadas propôs o uso de autoadaptação para o agente deliberar sobre normas, as quais restringem o seu comportamento no ambiente onde este se encontra.

Ao contrário do que é proposto neste trabalho, algumas abordagens (Van Riemsdijk et al., 2015a; Mefteh, 2015; Meneguzzi & Luck, 2009) adotam uma perspectiva que fornece mecanismos para a construção de agentes dirigidos por normas cujo principal objetivo é o cumprimento de normas, em oposição à realização dos interesses dos agentes.

1.4 Questões de Pesquisa

Tomando como base os problemas e limitações apresentadas anteriormente, podemos definir as seguintes questões de pesquisa:

1. Como alterar o comportamento dos agentes de software para suportar normas?
2. Como realizar a combinação entre modelos adaptativos, modelos normativos e modelos de agentes?
   2.1. Quais fatores inerentes à adaptação um agente adaptativo normativo deve ser capaz de lidar?
3. Quais mecanismos adaptativos são necessários para modelar um agente de software?
   3.1. Com quais mecanismos são necessários para que um agente seja capaz de se adaptar para lidar com tais fatores normativos?
   3.2. Como tais mecanismos podem ser introduzidos no processo de raciocínio de agentes?
   3.3. Como projetar agentes dotados de raciocínio adaptativo normativo?
   3.4. Como avaliar a eficácia do raciocínio adaptativo normativo adotado por um agente capaz de se adaptar a normas?

A abordagem proposta neste trabalho lida com tais questões.

1.5 Solução Proposta

Este trabalho apresenta uma abordagem para o desenvolvimento de agentes adaptativos normativos. A abordagem fornece um modelo conceitual que estende a arquitetura *Belief-Desire-Intention* (BDI) (Rao & Georgeff, 1995; Weiss, 1999) incluindo funções para apoiar o raciocínio normativo. A abordagem também fornece uma linguagem de modelagem para projetar agentes capazes de se

adaptarem para lidar com normas. O modelo conceitual proposto é baseado no metamodelo TAO (Silva, 2002) e no metamodelo FAML (Beydoun, 2009). A fim de possibilitar a modelagem de agentes projetados a partir do metamodelo proposto, a abordagem fornece uma extensão da linguagem MAS-ML (Silva, 2004) e uma extensão do *framework* JSAN (Viana, 2015b), para implementação de agentes adaptativos normativos. Em resumo, a partir da abordagem proposta será possível realizar as seguintes tarefas:

- Modelar os conceitos de adaptação e/ou normas em Sistemas Multiagentes;
- Criar agentes capazes de:
    - Perceber normas endereçadas a eles no ambiente e adaptar seu comportamento para lidar com elas;
    - Selecionar quais normas devem ser cumpridas ou violadas;
    - Aplicar raciocínio adaptativo para tomada de decisão com relação às normas que restringem o comportamento de agentes;
    - Detectar e solucionar conflitos entre normas;
    - Selecionar planos para alcançar seus objetivos dadas as restrições de ambiente.

1.6 Contribuições Esperadas

A seguir, são apresentadas as principais contribuições da tese:

- **Metamodelo**. Ele define os aspectos estruturais e comportamentais dos SMAs, ou seja, descreve as entidades que são normalmente apresentadas nos SMAs, além das abstrações de adaptação e normas, suas propriedades e um conjunto de relacionamentos que podem ser usados para combinar essas entidades. As abstrações foram descritas tanto em tempo de design como em tempo de execução;
- **Linguagem Adaptativa Normativa**. Uma linguagem de modelagem para projetar ANA (*Adaptive Normative Agent*). Essa linguagem irá possibilitar visualizar como os componentes de ANA estão estruturados e como se relacionam entre eles e com entidades do ambiente;

- **JSAN 2.0**. Uma extensão do *framework* JSAN, o qual fornece suporte para a construção de agentes adaptativos normativos, além de oferecer flexibilidade para implementação de diferentes *control-loops* de adaptação;
- **Mecanismos Normativos**. Com o intuito dos agentes lidarem com as normas, um conjunto de mecanismos normativos serão fornecidos para: entender uma norma, cumprir ou violar com a norma, detectar e resolver conflito entre normas, avaliar os efeitos de cumprir ou não com a norma, e selecionar desejos e planos depois de escolher cumprir ou violar a norma;
- **Mecanismos Adaptativos**. Dado que o agente entenda a norma e raciocine sobre ela, mecanismos adaptativos serão disponibilizados para modificar o seu comportamento;
- **Cenários**. A aplicação do *framework* JSAN 2.0 em dois cenários de uso;
- **Avaliação Empírica**. A realização de experimentos que demonstram a importância do uso de uma linguagem de modelagem para o modelo conceitual ANA.

1.7 Organização da Tese

Esta proposta de tese é estruturada como segue:

- O Capítulo 2 apresenta alguns conceitos e tecnologias utilizadas na abordagem;
- O Capítulo 3 apresenta alguns trabalhos relacionados;
- O Capítulo 4 apresenta o modelo conceitual para lidar com ANA;
- O Capítulo 5 apresenta a linguagem de modelagem ANA-ML do metamodelo apresentado no Capítulo 4;
- O Capítulo 6 apresenta uma avaliação empírica da linguagem ANA-ML;
- O Capítulo 7 apresenta a arquitetura e o *framework* desenvolvido para implementar agentes adaptativos normativos;
- O Capítulo 8 apresenta dois cenários de uso com a linguagem de modelagem e a implementação usando o *framework* JSAN 2.0;
- O Capítulo 9 apresenta as conclusões e trabalhos futuros;
- O Capítulo 10 apresenta as Referências Bibliográficas;

- Por fim, no Apêndice A é mostrado o questionário aplicado no estudo da tese.

2
Fundamentação Teórica

Esta seção descreve as principais características dos conceitos de sistemas multiagentes, adaptação e normas. Em primeiro lugar serão abordados os conceitos de SMAs, depois discute o que é adaptação, como ela ocorre e como um sistema de software se adapta. Além disso, será discutido o que são normas, como elas são entendidas pelos agentes e como ocorrem conflitos entre normas.

2.1
Sistemas Multiagentes

Sistemas multiagentes surgiram como uma subárea da Inteligência Artificial Distribuída dedicada à pesquisa e desenvolvimento de soluções para problemas complexos, que não são de fácil resolução, através de algoritmos clássicos da programação (Russell & Norvig, 2009). Com o objetivo de melhor gerenciar a complexidade inerente à aplicação do conceito de sistemas autônomos em ambientes de computação móvel, a engenharia de sistemas multiagentes mostrou-se mais apropriada, dada a sua habilidade em agregar "inteligência" ao sistema (Lucena, 1987; Garcia et al., 2003; Sardinha et al., 2006; Silva & Lucena, 2007).

Um SMA envolve um número de agentes que interagem uns com os outros. No caso mais geral, os agentes atuarão em nome de usuários com diferentes objetivos e motivações. Para interagirem com sucesso, serão necessárias as habilidades de cooperação, coordenação e negociação entre eles baseados nas habilidades humanas (Wooldridge, 2011).

O conceito chave utilizado nos sistemas multiagentes é uma abstração chamada agente. Um agente é uma entidade autônoma virtual (ou física) capaz de compreender o ambiente no qual está inserido. Um agente possui a habilidade de se comunicar com outros agentes do sistema para atingir um objetivo comum, que um agente sozinho poderia não ser capaz de alcançar.

Existem duas principais características distintivas para agentes. Em primeiro lugar, tarefas relativamente de alto nível podem ser delegadas aos agentes que irão

realizá-las de forma autônoma. Em segundo lugar, os agentes estão situados em um ambiente que pode dinamicamente afetar seu comportamento e estratégia na resolução do problema (Jennings & Wooldridge, 1996).

As propriedades fundamentais que caracterizam um agente são (Jennings & Wooldridge, 2000; Wooldridge, 2011):

- *Autonomia* refere-se à capacidade de realizar a maior parte de suas atividades sem a interferência direta de um humano e possuem certo nível de controle sobre suas ações e estados;
- *Habilidade Social* compreende a capacidade de interagir com outros agentes e seres humanos a fim de atingir seus objetivos ou ajudar outros com suas atividades;
- *Reatividade* considera a capacidade de perceber e responder a estímulos do ambiente e a qualquer mudança que nele ocorra;
- *Proatividade* expressa a capacidade de responder aos estímulos do ambiente de forma oportuna, orientando-se por seus objetivos e tomando a iniciativa quando apropriado.

Por ser uma entidade com certo grau de complexidade, em 1996 foi criada uma organização responsável por produzir padrões que deveriam ser utilizados na manipulação dos agentes. A organização criada chama-se *Foundation for Intelligent Physical Agents* (FIPA), que definiu uma máquina de estados (FIPA, 2016) com cinco estados, um início e fim para melhor explicar o ciclo de vida de um agente.

As especificações FIPA (FIPA, 2016) representam um conjunto de normas que se destinam a promover a interação dos agentes e os serviços a que eles podem se comunicar. A seguir temos a descrição dos estados dos agentes FIPA.

A Figura 1 mostra como funciona a máquina de estados definida para um agente que segue as especificações FIPA. O estado inicial de todo agente é o *Initiated*. Neste estado o agente é construído, mas ainda não está registrado no sistema, portanto não possui nome, um identificador de agente (IAD) e não pode se comunicar com outros agentes. A partir deste estado o agente pode realizar transições para quatro outros estados que são: *Active* - o agente está registrado no sistema e ativo. Possui um nome e um IAD e poderá utilizar todos os serviços da plataforma; *Suspended* - o agente está inativo, ou seja, não esta executando nenhum comportamento; *Waiting* - o agente está bloqueado esperando que alguma condição

seja satisfeita para voltar a executar seu comportamento; e *Transit* - o agente entra neste estado quando está migrando de plataforma. Toda mensagem recebida pelo agente neste estado será redirecionada para o novo endereço do agente. O último estado do agente é o *Deleted*, e neste estado o agente está literalmente desativado e não possui mais registro no Sistema de Gerenciamento de Agentes (AMS).

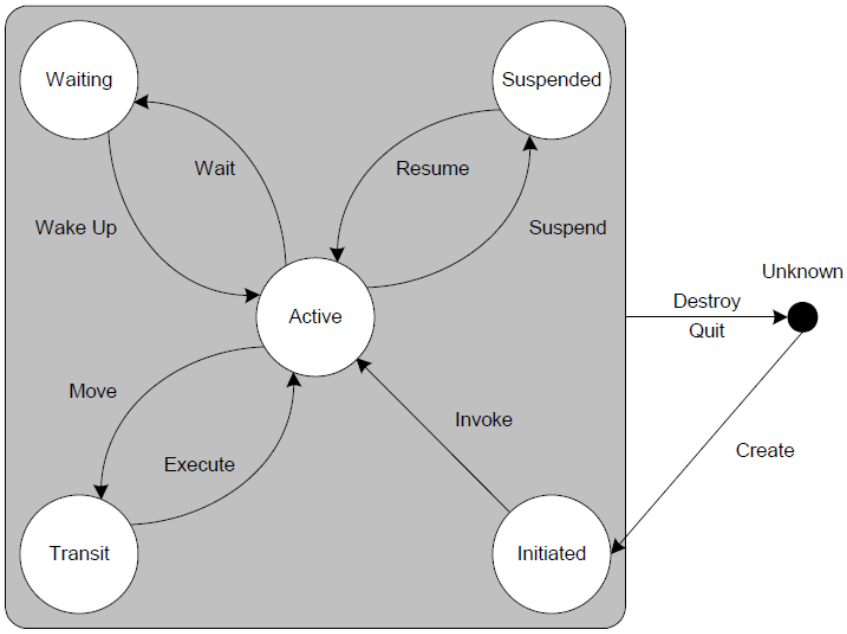

Figura 1 - Máquina de estados dos agentes FIPA (FIPA, 2002)

2.2 Arquitetura BDI

O modelo BDI (*Belief-Desire-Intention*) originalmente proposto em (Bratman, 1987) como uma teoria filosófica do raciocínio prático, explicando o comportamento humano com as seguintes atitudes: crenças, desejos e intenções. A suposição básica do modelo BDI é que ações são derivadas a partir de um processo chamado raciocínio prático, o qual é constituído de dois passos. No primeiro passo, deliberação (de objetivos), faz-se a seleção de um conjunto de desejos que devem ser alcançados, de acordo com a situação atual das crenças do agente. O segundo passo é responsável pela determinação de como esses desejos concretos produzidos como resultado do passo anterior podem ser atingidos através do uso dos meios disponíveis ao agente (Wooldridge & Ciancarini, 2001). As três atitudes mentais que compõem o modelo BDI são melhor detalhadas a seguir:

- ***Beliefs*** – Representam as características do ambiente, as quais são atualizadas apropriadamente após a percepção de cada ação. Essas crenças representam o conhecimento sobre o mundo;
- ***Desires*** – Contêm informação sobre os objetivos a serem atingidos, bem como as prioridades e os custos associados com os vários objetivos. Podem ser pensados como a representação do estado motivacional do sistema.
- ***Intentions*** – Representam o atual plano de ação escolhido. Capturam o componente deliberativo do sistema.

Rao e Georgeff (Rao & Georgeff, 1995) adotaram o modelo BDI para agentes de software apresentando uma teoria formal e um interpretador BDI abstrato que é a base para praticamente todos os sistemas BDI históricos e atuais. O interpretador opera sobre as crenças, objetivos e planos do agente, os quais representam os conceitos das noções mentais ligeiramente modificados. A diferença mais significativa é que os objetivos são um conjunto consistente de desejos concretos que podem ser atingidos todos juntos, evitando assim a necessidade de uma complexa fase de deliberação de objetivos. A principal tarefa do interpretador é a realização do processo meios-fim através da seleção e execução de planos para um dado objetivo ou evento.

O processo de raciocínio prático em um agente BDI é apresentado na Figura 2. Existem sete componentes principais em um agente BDI:

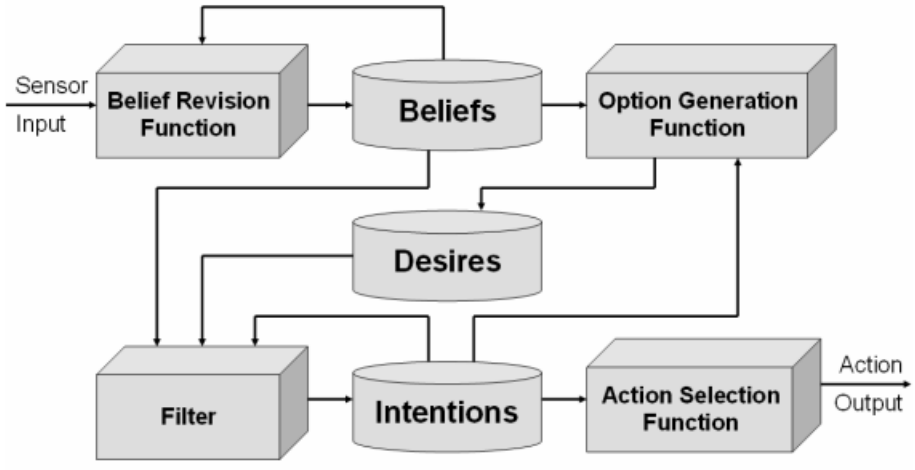

Figura 2 - Arquitetura BDI genérica (Wooldridge et al., 1999)

- um conjunto de crenças (*beliefs*) atual, representando a informação que o agente tem sobre seu ambiente;

- uma função de revisão de crenças (*belief revision function*), a qual determina um novo conjunto de crenças a partir da percepção da entrada e das crenças do agente;
- uma função de geração de opções (*option generation function*), a qual determina as opções disponíveis ao agentes (seus desejos), com base nas suas crenças sobre seu ambiente e nas suas intenções;
- um conjunto de opções (*desires*) corrente que representa os possíveis planos de ações disponíveis ao agente;
- uma função de filtro (*filter*), a qual representa o processo de deliberação do agente, que determina as intenções do agente com base nas suas crenças, desejos e intenções atuais;
- um conjunto de intenções (*intentions*) atual, que representa o foco atual do agente, isto é, aqueles estados que o agente está determinado a alcançar;
- uma função de seleção de ação (*action selection function*), a qual determina uma ação a ser executada com base nas suas intenções atuais.

2.3 Adaptação de Software

A palavra adaptação significa "qualquer alteração na estrutura ou funcionamento de um organismo que faz com que seja mais adequado ao seu ambiente" (Oxford Dictionary of Science, 2015). Portanto, de acordo com esta definição, a adaptação de software tem dois níveis de ação: mudar a estrutura e alterar o comportamento. Estas alterações visam tornar a aplicação mais adequada ao seu contexto evolutivo (Da et al., 2011).

Há uma forte sinergia entre os conceitos de evolução e adaptação em engenharia de software: a adaptação de software refere-se a ambos, o software atual que está sendo adaptado e ao processo de evolução que conduz ao novo software adaptado (Kalareh, 2012). Mudanças de evolução para efeitos de adaptação são normalmente feitas em desenvolvimento ou em tempo de compilação, e são destinados a lidar com situações previsíveis na forma de solicitações de mudança de software. Por outro lado, o software pode também mudar e adaptar-se com base nas alterações no seu ambiente. Tais mudanças adaptativas são geralmente dinâmicas, e são adequadas para lidar com mudanças imprevisíveis ou temporárias

no ambiente operacional do software. Uma solução promissora para a adaptação de software é desenvolver sistemas de software autoadaptativos que podem gerenciar mudanças dinamicamente durante a execução de forma rápida e confiável (Huber, 2014). Para Nallur (Nallur, 2013) autoadaptação em sistemas de software, são softwares que monitoram eles mesmos e seu ambiente operacional, e tomam as medidas adequadas quando as circunstâncias mudam. Assim, uma das principais vantagens do software autoadaptativo é a sua capacidade para gerir a complexidade decorrente de ambientes operacionais altamente dinâmico e não-determinísticos.

Em Kalareh (2012) o autor fala que para um sistema de software seja considerado autoadaptável, ele deve ter as seguintes características: (i) a capacidade de observar as mudanças no seu ambiente operacional; (ii) a capacidade de detectar e diagnosticar mudanças do ambiente operacional e avaliar o seu próprio comportamento; (iii) a capacidade de alterar o seu próprio comportamento, a fim de adaptar-se às mudanças; e (iv) suporte a um comportamento dinâmico, ou seja, seu comportamento interno/externo deve ser capaz de ser alterado intencionalmente e automaticamente.

Quando se fala de sistemas autoadaptáveis, é preciso voltar um pouco e relembrar o termo Computação Autonômica proposto pela IBM (IBM, 2003). Este tem como objetivo descrever sistemas computacionais considerados auto-gerenciáveis. Tais sistemas possuem quatro propriedades: self-configuration, self-optimization, self-healing and self-protecting. Detalhes de cada self-* são apresentados em (IBM, 2003).

A Figura 3 ilustra um processo padrão de autoadaptação (também chamado de *control-loop*) proposto pela IBM (IBM, 2003). Tal processo é similar a um modelo de agente genérico proposto por (Russel & Norvig, 2003), em que um agente inteligente acompanha, a partir de sensores, o que acontece em seu ambiente, usando-os para definir quais ações executar.

O ciclo geral de adaptação de software mostrado na Figura 3 inclui observação do ambiente, seleção de adaptações e a sua execução. Este ciclo é explicado em quatro diferentes fases: Monitorar, Analisar, Decidir e Executar. O monitor, através dos sensores recolhe as primeiras informações a partir do monitoramento do ambiente. A seguir, a atividade de análise avalia os dados de contexto e produz um plano de adaptação, podendo produzir uma lista de planos ou apenas um plano. Na atividade de decisão é analisado o risco de cada plano e se a

adaptação ocorre dada uma base prévia de conhecimento. Finalmente, é executada a adaptação conforme o que foi descrito no plano escolhido através dos efetuadores. O núcleo da adaptação são as atividades de análise e decisão.

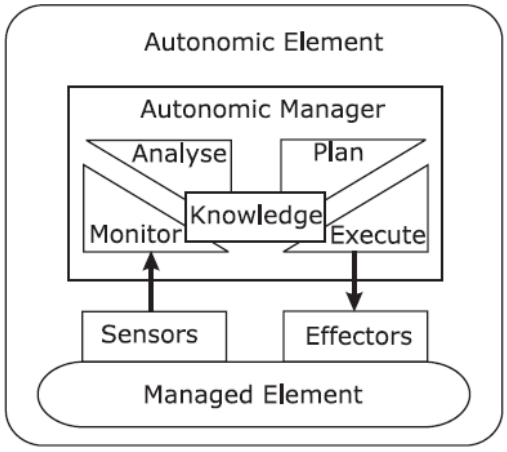

Figura 3 - Processo de autoadaptação proposto pela IBM (IBM, 2003)

Apesar de diversas abordagens (Huber, 2014; Nallur, 2013; Kalareh, 2012) descreverem como sistemas podem realizar autoadaptações, elas não implementam agentes de software. Assim, propriedades relacionadas aos agentes e consideradas importantes para sistemas autoadaptativos não são contempladas, como, por exemplo, autonomia, aprendizado, raciocínio e proatividade (Wooldridge et al., 1999; Jennings & Wooldridge, 2000).

2.4 Normas

Em relação ao conceito de normas, o dicionário online Webster (Merriam-Webster, 2015) oferece três definições: (i) um padrão autoritário; (ii) um princípio de ação certa vinculada aos membros de um grupo e que serve para orientar, controlar ou regular o comportamento adequado e aceitável; e (iii) uma ponderação. Uma norma é tratada como: (a) um conjunto padrão de desenvolvimento ou realização, geralmente derivado do desempenho médio ou a mediana de um grande grupo; (b) um padrão ou ser entendido como o comportamento típico de um grupo social; e (c) uma prática generalizada ou de costume. Essas definições são significados do termo norma, usados nas diversas áreas de pesquisa normativa, incluindo a lógica deôntica, teoria do direito, sociologia, psicologia social e filosofia social, teoria da decisão e teoria dos jogos (Verhagen, 2000).

Quando aplicada a sistemas multiagentes, entende-se normas como sendo mecanismos capazes de regular o comportamento dos agentes, representando a maneira com que agentes entendem as responsabilidades de outros agentes (López, 2003). Assim, os agentes trabalham com a crença de que os outros irão se comportar de acordo com a prescrição das normas, mas, sobretudo, as normas são mecanismos para capacitar os agentes a exigir que os outros agentes se comportem de uma determinada maneira (Santos Neto, 2010a). Desta forma, o uso de normas é uma necessidade em sistemas multiagentes em que os membros são autônomos, mas não autossuficientes, e, portanto, a cooperação é necessária, mas não pode ser assegurada sem a introdução de mecanismos específicos para isso.

Sabendo que normas são mecanismos que a sociedade tem a fim de influenciar o comportamento dos agentes (Bogdanovych, 2011), normas podem ser utilizadas para alcançar diferentes finalidades, que incluem desde a construção de um simples acordo entre agentes até casos mais complexos envolvendo o sistema jurídico (López et al., 2002). Uma norma define uma permissão, obrigação ou proibição sobre o comportamento do agente. Normas podem persistir durante diferentes períodos de tempo, por exemplo, enquanto o agente permanece na sociedade, ou apenas por um curto período de tempo até que o objetivo social tenha sido realizado (López et al., 2002). Com isso, diferentes aspectos podem ser usados para caracterizar normas. Em primeiro, normas são sempre criadas para serem cumpridas por um determinado conjunto de agentes a fim de alcançar objetivos sociais. Em segundo, normas não são sempre aplicadas, e a sua ativação depende de condições do contexto em que os agentes estão situados. Além disso, é preciso saber qual o tipo de elemento que a norma regula, se é uma ação do agente, ou um estado do ambiente. Finalmente, em alguns casos, normas podem estabelecer um conjunto de sanções para serem impostas quando os agentes cumprirem ou violarem certas normas (López et al., 2002; López, 2003; Santos Neto, 2010b).

A definição de normas utilizada neste trabalho (Santos Neto, 2012b) é composta de várias propriedades. Essas propriedades incluem: *Addressees*, *Condition* (por exemplo, *Activation*, *Expiration*), *Motivation* (por exemplo, *Rewards*, *Punishments*), *Deontic Concept* e *States*. A descrição de cada propriedade é fornecida abaixo: (i) *Addressee* é usado para especificar os agentes ou papéis responsáveis pelo cumprimento da norma; (ii) *Activation* é a condição de ativação para a norma se tornar ativa, (iii) *Expiration* é a condição de validade para que a

norma se torne inativa; (iv) *Rewards* é usada para representar o conjunto de recompensas para ser dado ao agente pelo cumprimento de uma norma; (v) *Punishments* é o conjunto de punições para ser dado ao agente por violar uma norma; (vi) *Deontic Concept* é usado para indicar se a norma estabelece uma obrigação, uma permissão ou uma proibição; e (vii) *State* é usado para descrever o conjunto de estados ou ações que estão sendo regulados.

2.4.1 Conflito entre Normas

Considerando duas normas que regulam o comportamento do mesmo agente e falam sobre a mesma ação, no mesmo intervalo de tempo, pode-se ter os seguintes conflitos entre essas normas: (i) uma norma ter uma influência deôntica de obrigação e a outra norma de proibição; (ii) uma norma ter a influência deôntica de proibição e a outra de permissão; e (iii) uma norma ser de obrigação e a outra de permissão (Vasconcelos, 2007). Para esclarecer esses tipos de conflito, serão apresentados alguns exemplos.

Na Figura 4 tem-se o seguinte exemplo: uma norma impõe que o vendedor não pode sair do balcão de vendas enquanto a loja estiver aberta e a outra norma assegura que o vendedor deve buscar mercadoria no estoque para o cliente.

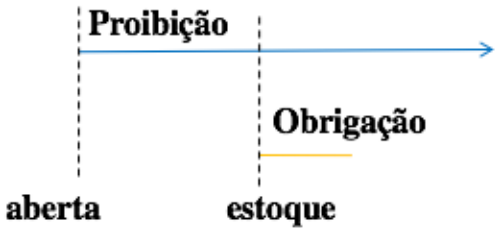

Figura 4 - Conflito entre normas de proibição e obrigação

Na Figura 5, tem-se normas que não entram em conflito, pois não regulam estado / ação no mesmo intervalo de tempo. A primeira norma age sobre o vendedor, onde este é obrigado a entregar o item ao comprador depois de receber o dinheiro da venda. Já a segunda norma age proibindo o vendedor de entregar o item ao comprador antes de receber o dinheiro da venda.

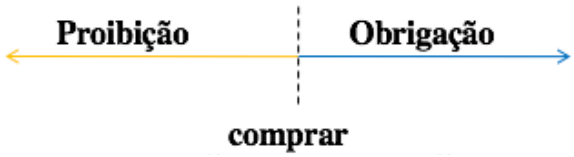

Figura 5 - Normas SEM conflito entre proibição e obrigação

A Figura 6 tem um conflito entre uma norma de obrigação e uma de permissão. No cenário do mercado virtual, tem-se as seguintes normas: (i) o vendedor só pode remarcar preço antes da loja abrir; e (ii) o vendedor tem que remarcar preço quando for anunciada uma promoção.

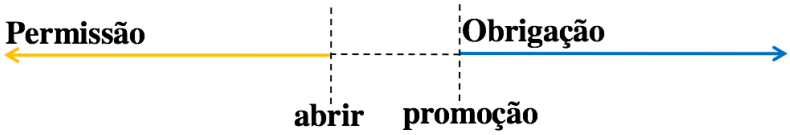

Figura 6 - Conflito entre normas de permissão e obrigação

A Figura 7 mostra um conflito entre normas proibitivas e permissivas. Imagine o seguinte: uma norma que age sobre o comprador proibindo-o de devolver o produto comprado, mas outra norma age sobre o comprador permitindo-o devolver o produto comprado antes de ser utilizado.

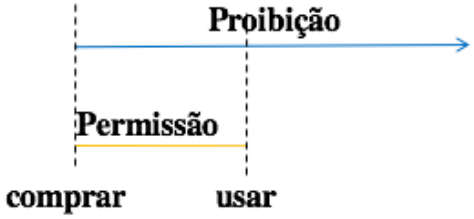

Figura 7 – Conflito entre normas de proibição e permissão

A Figura 8 e Figura 9 ilustram duas situações distintas que só podem ser verificadas em tempo de execução. Segue um exemplo para melhor visualização do problema. Imagine uma norma que obriga o agente a executar uma função qualquer **A** depois de **X**, mas existe uma outra norma que permite ao agente executar a função **A** depois de **Y**. Assim, pode-se verificar que na Figura 8 não ocorre um conflito entre as normas, entretanto, na Figura 9 existe um o conflito.

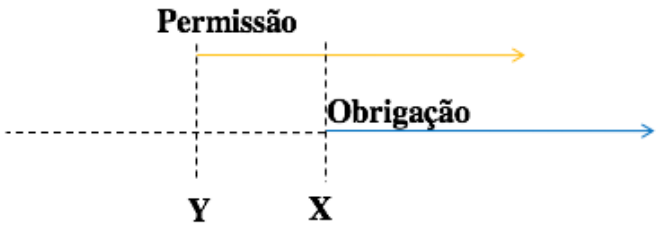

Figura 8 - Normas SEM conflito entre permissão e obrigação

Figura 9 – Conflito entre normas de permissão e obrigação

2.5 Interpretador *Jason*

A plataforma Jason possibilita o desenvolvimento e execução de agentes BDI (*Belief, Desire and Intention*). O interpretador disponibilizado pela plataforma Jason para a execução de agentes BDI é apresentado em detalhe na Figura 10 (reproduzida de (Bordini et al., 2007)). O conjunto de crenças, representadas por *Belief Base*, eventos representados por *Events*, planos representados por *Plan Library* e intenções representadas por *Intentions*, todos são representados por retângulos. Losangos representam a seleção de um elemento de um conjunto. Círculos representam alguns dos processos envolvidos pelo interpretador.

A função *Belief Review* (BRF) revisa a base de crenças com um literal para ser adicionado ou removido, e a intenção que requisitou a mudança da crença. Assume-se que as crenças são atualizadas pela percepção e que sempre que houver mudanças na base de crenças do agente, isso implica a inserção de um evento no conjunto de eventos.

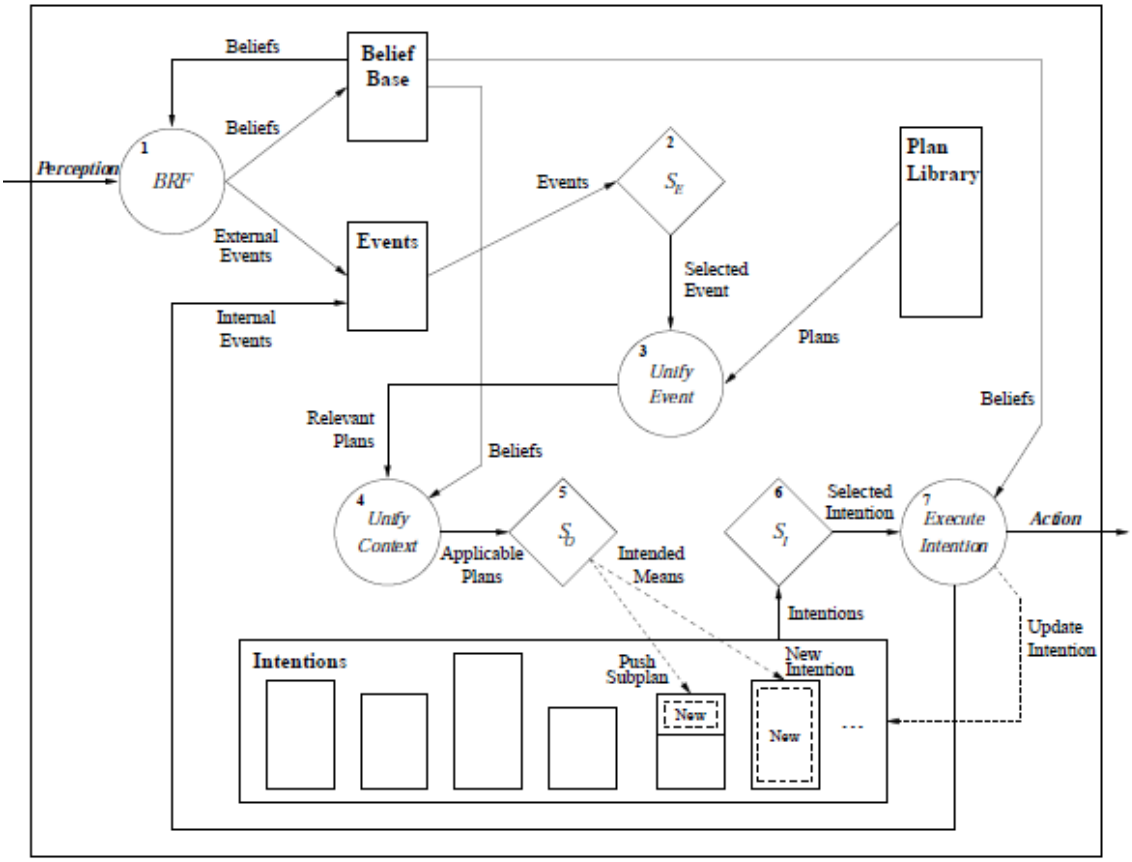

Figura 10 - Interpretador Jason

Depois da função *Selecting Event* (SE) selecionar um único evento, o interpretador tem que unificar este evento com as condições de invocação dos planos armazenados na biblioteca de planos. Isso gera um conjunto de todos os planos relevantes para o evento escolhido. A partir da verificação dos contextos dos planos relevantes, o interpretador Jason determina o conjunto de planos aplicáveis (planos que podem ser usados, na situação presente, para tratar o evento selecionado naquele ciclo). Depois, a função *Selecting Option* (SO) escolhe entre os planos aplicáveis aquele que será utilizado para lidar com o evento selecionado, e adiciona o plano para o conjunto de intenções do agente. Nesse ponto, vale ressaltar que uma intenção é uma sequência de planos.

A função *Selecting Intention* (SI), seleciona uma das intenções do agente para ser executada pelo processo *Execute Intention*. Quando todas as fórmulas no corpo de um plano são executadas, o plano é removido da lista de uma intenção, e se ainda existe planos na intenção o próximo plano é selecionado para ser executado; caso contrário a intenção é removida do conjunto de intenções, e assim o processo inicia novamente.

2.6 Considerações Finais

Este Capítulo definiu na Seção 2.1 o conceito de sistemas multiagentes. A Seção 2.2 definiu a arquitetura BDI. Na Seção 2.3 foi apresentado o conceito de adaptação de software, tal conceito serviu como base para a elaboração do raciocínio do agente proposto neste trabalho. Na Seção 2.4 apresentou-se o conceito de normas adotado neste trabalho e como acontece os conflitos entre normas. Na Seção 2.5 apresentou-se o interpretador Jason, o qual serviu como base para o fornecimento de mecanismos que possibilitassem a implementação de agentes a partir do modelo proposto neste trabalho.

3
Trabalhos Relacionados

Este capítulo apresenta os trabalhos existentes na literatura no que diz respeito à abordagens para design ou implementação de agentes adaptativos e/ou normativos e suas respectivas verificações.

Os trabalhos relacionados serão descritos em três grupos: Agentes Normativos, neste grupo serão apresentados os trabalhos que propõem uma abordagem para design ou implementação de agentes capazes de lidar com normas; Agentes Adaptativos, neste grupo serão apresentados abordagens que permitem o desenvolvimento de agentes capazes de realizar algum tipo de adaptação e/ou lidar com normas; e por último o grupo de Modelagem de Agentes, neste grupo serão apresentados abordagens que permitem o design de SMAs para facilitar a construção de sistemas baseados em agentes de software.

3.1
Agentes Normativos

A arquitetura n-BDI proposta por Criado et al. (2010) apresenta um modelo para projetar agentes capazes de operar em ambientes regidos por normas, as quais são estáticas e não sofrem qualquer alteração no ambiente. A arquitetura n-BDI lida com normas simplificadas e considera que a seleção de objetivos deve ser realizada tomando como base a prioridade associada a cada objetivo, onde a prioridade de um objetivo é determinada levando em consideração a prioridade das normas que o regulam. No entanto, além de não deixar claro como os componentes de uma norma realmente podem ser avaliados a fim de determinar as propriedades da norma, também não são apresentadas estratégias dinâmicas para lidar com as normas.

Em (Van Riemsdijk et al., 2015b) foi proposta a construção de um ajudante eletrônico social, capaz de entender as normas existentes no ambiente. Os autores partem das seguintes hipóteses: (i) na tecnologia de suporte social está inerentemente ligada ao contexto social do usuário (por exemplo, famílias e cuidadores); e (ii) a tecnologia existente não é flexível quando se trata desta

natureza social. Essa rigidez pode levar à violação das normas sem suporte e à falta de flexibilidade para lidar com as normas suportadas pelo sistema. Ao decorrer do artigo os autores argumentam sobre a necessidade da tecnologia dar suporte às diferentes normas e situações imprevistas, surgindo assim a necessidade se de adaptar. Van Riemsdijk et al. (2015b) fala sobre a necessidade de técnicas que abrangem as áreas de agentes normativos, interação entre seres humanos e agentes, e ética em Inteligência Artificial. Entretanto, em nenhum momento criaram uma arquitetura ou metamodelo onde seria possível definir agentes capazes de se adaptarem para lidar com as normas ativas no ambiente. Com isso, os autores neste trabalho perceberam a necessidade de entender e adaptar às normas do sistema, mas não lidaram com o problema.

Beheshit et al. (2015) propôs uma arquitetura cognitiva para sistemas multiagentes fazendo uma ponte entre a arquitetura n-BDI (*Norms-Belief-Desire-Intention*) (Criado et al., 2010) e um modelo apenas de aprendizado social (Sen & Airiau, 2007). Nesta arquitetura as normas emergem através de mecanismos de aprendizado social, enquanto usam n-BDI raciocínio para lidar com adoção e cumprimento de normas. Os autores, através de cenários de uso, procuraram demonstrar a importância de agentes normativos para representar a influência de grupos no comportamento da social. Apesar disso, a arquitetura de Beheshit et al. (2015) não trata questões normativas que ficaram pendentes no trabalho de Criado et al. (2010), além de não mencionar nada sobre a necessidade de agentes se adaptarem para lidar com as normas que restringem seus comportamentos. Os trabalhos anteriores continuam utilizando estratégias pré-fixadas para lidar com as normas.

A arquitetura proposta em (Santos Neto, 2012b) fornece um modelo arquitetural para apoiar um agente no raciocínio sobre as normas. Tal modelo estende o modelo *Belief-Desire-Intention* adicionando um conjunto de funções que auxiliam o agente na adoção de novas normas. Além da verificação da ativação, desativação, cumprimento e violação das normas, na seleção de quais normas devem ser cumpridas ou violadas, na detecção e resolução de conflitos entre normas, na geração de novos objetivos e na seleção de objetivos, planos e intenções, levando em consideração as normas do sistema. Além disto, a abordagem provê uma arquitetura para construção de agentes normativos capazes de raciocinar sobre normas. Entretanto, não fornece os mecanismos necessários para os agentes se

adaptarem caso a norma tenha o conceito deôntico de permissão. No trabalho o autor lidou apenas com normas de obrigação e proibição, não inserindo no raciocínio do agente a capacidade de lidar com a lógica deôntica de permissão. Além disso, o trabalho não desenvolveu nenhuma estrutura onde fosse possível implementar agentes capazes de utilizar o processo de autoadaptação (IBM, 2003; Da et al., 2011) para lidar com questões normativas de forma dinâmica.

O modelo BOID (*Belief-Obligation-Intention-Desire*) proposto em (Broersen et al., 2001) e operacionalizado em (Dastani & Van Der Torre, 2004) é uma extensão do modelo BDI que considera a influência das crenças, obrigações, intenções e desejos do agente sobre a geração de objetivos. O modelo BOID aplica a noção de tipos de agentes para guiar a geração de objetivos. Se os desejos do agente sobrepõem suas obrigações, então, o agente é do tipo egoísta; e se suas obrigações sobrepõem seus desejos, então, o agente é do tipo social. Portanto, este trabalho adota uma abordagem estática para lidar com normas baseando-se no tipo de agentes, desta forma, não fornecendo a flexibilidade necessária para lidar com a dinamicidade inerente a sistemas normativos onde a cada instante novas normas podem ser adotadas, ativadas, desativadas, cumpridas ou violadas.

O *framework* de programação JaCaMo descrito em (Boissier et al., 2011) combina o Jason, Cartago (Ricci et al., 2009) e o Moise+ (Hübner et al., 2006) a fim de fornecer uma estrutura para programar agentes capazes de lidar com artefatos normativos (isto é, normas) do ambiente. Entretanto, tais trabalhos não apresentam soluções envolvendo o raciocínio dos agentes da sociedade.

Em Lopez & Marquez (2004) propõem um modelo formal, utilizando a linguagem de especificação formal Z para modelar de agentes capazes de atingir seus objetivos levando em consideração as normas do sistema. Segundo os autores, um agente criado a partir de tal modelo é capaz de: (i) checar se ele é um dos responsáveis por cumprir uma norma; (ii) verificar a ativação e desativação de uma norma levando em consideração as crenças do agente; (iii) avaliar e decidir por cumprir ou violar cada norma do sistema; e (iv) efetivar a decisão por cumprir ou violar uma norma, removendo ou adicionando objetivos do agente. Além de não elucidar como a avaliação de uma norma é realizada, não é apresentada nenhuma preocupação com relação à identificação e resolução de conflitos entre normas, à verificação de normas cumpridas ou violadas, ou ainda sobre a influência das normas sobre o processo de seleção de planos e intenções do agente.

Em López et al. (2002) é apresentada uma especificação de agentes dotados de diferentes estratégias para decidir por cumprir ou violar uma norma levando em consideração os seus interesses; em particular, os seguintes agentes são apresentados: (*Pressured*) Estes agentes decidem por cumprir uma norma se a violação da mesma gera alguma perda para o agente; (*Opportunistic*) Estes agentes decidem por cumprir uma norma se o seu cumprimento resulta em ganhos para o agente; (*Selfish*) tais tipos de agentes é resultante da combinação da estratégia *Pressured* e *Opportunistic*, onde normas são selecionadas para serem cumpridas se o agente pode ter ganhos ao cumprir com a norma e perdas se a norma é violada. Embora os autores apenas apresentem uma formalização da estratégia adotada por tais agentes, o trabalho apresenta resultados interessantes de experimentos que demonstram que o uso de agentes *selfish* pode levar a um equilíbrio entre a satisfação individual do agente e o estabelecimento da ordem social do sistema. Entretanto, os autores não levaram em consideração a capacidade do agente mudar de estratégia para cumprir ou violar com uma norma no tempo de execução do sistema, ou seja, utilizam apenas estratégias fixas.

Em (Castelfranchi et al., 1999) é apresentado um modelo arquitetural que permite o projeto de agentes orientados a objetivos capazes de lidar com questões normativas, tais como: (i) verificar se o agente é responsável por cumprir a norma; (ii) em quais situações a norma está ativada ou desativada levando em consideração as crenças do agente; e, (iii) a necessidade de avaliar a influência da norma sobre a geração de objetivos, e seleção de objetivos e planos. Entretanto, a proposta apresentada em (Castelfranchi et al., 1999) é incompleta dado que ela somente apresenta diretrizes em alto nível sobre como a influência das normas sobre o raciocínio do agente ocorre, além de não discutir sobre como o cumprimento ou violação de normas pode ser verificado, sobre a detecção e resolução de conflitos entre normas, ou ainda sobre como realizar a seleção de normas não conflitantes para serem cumpridas ou violadas.

Os autores em (Felicíssimo et al., 2011) propuseram uma metodologia chamada de DynaCROM, para apoiar o desenvolvedor do sistema nas tarefas de implementação, gestão e evolução das normas em um SMAs. A metodologia inclui as fases de: (i) contextualização, através de uma classificação *top-down* para normas contextuais; (ii) concretização e representação, através de uma ontologia normativa contextual; e, finalmente, (iii) composição de normas, através de um

processo de composição normativa. A classificação *top-down* para as normas contextuais propostas pelo DynaCROM facilita as tarefas de elicitação (Leite, 2008), organização e gestão de normas. A ontologia normativa contextual DynaCROM suporta agentes heterogêneos com um entendimento comum sobre as normas do sistema. O processo de composição de normas definido pelo DynaCROM facilita a atualização da regulação do sistema por ambas as normas em desenvolvimento em um único recurso (uma ontologia) e / ou personalizando regras específicas para diferentes composições de normas contextuais. Embora os agentes de aplicação possam ser informados sobre suas normas atuais (contextuais), usando o comportamento DynaCROM, os desenvolvedores de agentes implementam seus agentes independentemente dessas informações. Neste caso, os agentes precisam de uma solução que os informe continuamente sobre os dados do sistema, de acordo com seus contextos atuais, a fim de lidar com as normas aplicáveis de cada ação realizada por eles. Apesar de construir uma solução de coleta de normas e ver a sua evolução, o DynaCROM, não auxilia os desenvolvedores com modelos para construção de normas e não sugere soluções para resolver conflitos entre normas no sistema.

3.2 Agentes Adaptativos

O trabalho apresentado em (Shan & Zhu, 2015) propõe uma abordagem (CAMLE) baseada em agentes para modelar e simular sistemas adaptativos distribuídos. Apesar de os autores mostrarem o funcionamento da ferramenta e os conceitos utilizados pela linguagem CAMLE, o modelo conceitual não deixa claro como os agentes irão se comportar para realizarem autoadaptação. Entretanto os autores não se preocupam com a necessidade de mecanismos internos aos agentes para autocontrole, para dinamicamente escolher possíveis ações, tais como negociação, violação e monitoramento. Além disso, os agentes são obrigados a seguirem as normas pré-definidas na simulação, assim, os objetivos do agente não importam, apenas os objetivos sociais são levados em consideração. Caso as normas sofram alterações ao longo do tempo, isso geraria inconsistência no design do modelo (Shan & Zhu, 2015) e em tempo de execução não seria possível detectar e

resolver conflitos, pois esses mecanismos de resolução de conflitos não foram implementados.

Em Van Riemsdijk et al. (2015[a]) propuseram um *framework* conceitual para especificar mecanismos de execução para agentes sociais adaptativos. O *framework* foi expresso em Lógica Temporal Linear. Embora o trabalho proponha tratar a adaptação do agente ao lidar com a norma, os autores consideram uma forma forte de cumprimento das normas, a fim de evitar que os agentes as violem. Assim, não são levados em consideração os efeitos de cumprimento das normas com relação aos objetivos individuais dos agentes. E caso o agente viole as normas, nenhuma punição é endereçada a este agente, ou seja, a satisfação individual do agente não é levada em consideração, retirando assim alguma autonomia do agente.

Ohmura et al. (2009) apresentam agentes sociais adaptativos como sendo agentes que obtêm comportamento através de experimentos com pessoas. Embora os agentes aprendam com as ações humanas, eles são sempre obrigados a cumprir as normas do sistema. Essa abordagem pode levar o agente a um baixo grau de satisfação, dado que nem todos os seus objetivos individuais podem estar de acordo com os objetivos do sistema.

Em Meneguzzi & Luck (2009) é apresentada uma extensão da linguagem *AgentSpeak*(L) (Machado & Bordini, 2001) que permite o desenvolvimento de agentes capazes de modificar seu comportamento em resposta às normas recém adotadas, particularmente, novos planos são criados para cumprir com obrigações e planos que violam normas de proibição são suprimidos. Em outras palavras, a abordagem proposta por Meneguzzi & Luck (2009) simplesmente cria, seleciona ou suprime planos em resposta a obrigações e proibições. Como consequência, é possível que planos que o agente tenha alto interesse em realizar não sejam executados. Além disto, questões normativas importantes não são consideradas pelos autores, tais como conflitos entre normas.

3.3 Modelagem de Agentes

Cernuzzi et al. (2014) fazem uma descrição completa da metodologia GAIA (Wooldridge et al., 1999), a qual inicialmente foi desenvolvida para modelar em pequena escala sistemas multiagentes fechados. Após sofrer diversas modificações

(Zambonelli et al., 2003), os autores procuram mostrar as vantagens de se utilizar GAIA para criar modelos tanto no nível social, quanto interno aos agentes. Embora GAIA tenha sofrido diversas modificações para lidar com as regras (normas) sociais das organizações, ela não modela como o agente entende essas regras e como ele irá lidar com elas. Os autores se preocuparam apenas em relacionar as regras com os papéis dos agentes na organização. Além disso, a metodologia não considera possíveis adaptações do agente no ambiente para lidar com normas.

Em Silva et al. (2008) é apresentada a linguagem de modelagem MAS-ML, que estende UML e foi proposta baseada no *framework* conceitual (metamodelo) chamado TAO (Silva & Lucena, 2004). A grande diferença entre esta abordagem e as demais presentes na literatura é a clara definição e representação de abstrações e comportamentos que compõem um SMA. Entretanto, a abordagem de (Silva et al., 2008) é incompleta dado que abstrações importantes como adaptação e normas não são descritas no TAO nem na linguagem de modelagem MAS-ML.

Em Gonçalves et al. (2015) é proposta uma extensão da linguagem de modelagem MAS-ML (Silva et al., 2008), trazendo como principal novidade não somente mecanismos que suportem a modelagem de agentes proativos, mas também a capacidade de modelar diferentes arquiteturas internas dos agentes, como as apresentadas em (Russel & Norving 2003). Estas diferentes arquiteturas lidam apenas com o comportamento dos agentes, sendo baseadas nos fundamentos reativos e proativos. Embora os autores tragam avanços para lidar com os diferentes tipos de comportamentos encontrados na literatura, não se preocuparam em lidar com abstrações que estarão presentes nos SMAs onde esses agentes irão existir.

Em Da Silva Figueiredo et al. (2011) é apresentada a linguagem de modelagem NormML (Figueiredo, 2011) baseada em UML para especificação das normas para restringir o comportamento das entidades em SMAs. O metamodelo da NormML fornece uma linguagem para a modelagem de funções, permissões, ações, recursos e restrições de autorização juntamente com as relações entre permissões e papéis, ações e permissões, recursos e ações, e restrições e permissões. Em outras palavras, esta abordagem simplesmente permite a modelagem dos aspectos estáticos das normas. Não é possível definir aspectos dinâmicos de uma norma, nem definir as normas em uma interação do contexto. Não é possível identificar as normas que estão ativas e nem as que foram violadas. Além disso, as

normas descritas nesta modelagem não são capazes de sofrer alterações ao longo do seu tempo de ativação no ambiente.

O *framework* FAML (Beydoun et al., 2009) contêm a unificação de diversos conceitos encontrados em outros metamodelos. Este fornece a promessa de produzir um metamodelo geral, capaz de suportar a padronização de conceitos em toda a comunidade de agentes. Entretanto, conceitos como adaptação e normas não são mencionados na estrutura do metamodelo, tanto em tempo de design como em execução. Com isso não é possível raciocinar sobre as normas e como os agentes poderão se adaptar as mudanças do ambiente. Além disso, os autores chegam a falar da necessidade dos conceitos de adaptação e normas nos modelos de sistemas multiagentes, porém decidem não abordar em sua pesquisa devido à complexidade dessas abstrações.

O framework i* (i-estrela), originalmente proposto por (Yu, 1999), trata da modelagem de contextos organizacionais tomando como base os relacionamentos de dependência entre os atores participantes. O principal objetivo no i* é representar, através de modelos, os atores participantes e as dependências entre eles, para que suas metas próprias sejam alcançadas, recursos sejam fornecidos, tarefas sejam realizadas e metas flexíveis sejam "satisfeitas a contento" ou "razoavelmente satisfeitas".

Em Gowri (2014) é apresentado um *framework* usando a metodologia Tropos (Bresciani et. al, 2004), o qual proporciona desenvolvimento sistemático incluindo no início da implantação a análise de requisitos. Tropos foi construído para sistemas baseados em agentes utilizando a modelagem proposta de i* (Yu, 1999). No trabalho de Gowri, o *feedback loop* é feito explicitamente e como cidadão de primeira classe, usando requisitos adaptativos e requisitos evolutivos. Embora os autores falem que a abordagem deles é genérica e pode ser aplicada a qualquer tipo de sistema, não apresentam uma solução para a modelagem de agentes que se adaptem para lidar com as normas do sistema endereçadas a eles.

Em Gómez-Rodríguez (2014) é apresentada uma definição de um processo para a metodologia INGENIAS (Pavón & Gómez-Sanz, 2003), a qual é uma adaptação do *Unified Software Development Process* – UDP (Jacobson, 1999) para desenvolver sistemas multiagentes. Em (Pavón & Gómez-Sanz, 2003) foi apresentada uma linguagem de modelagem (INGENIAS) e um conjunto de ferramentas para o desenvolvimento de sistemas multiagentes. INGENIAS foi

construída baseada na metodologia MESSAGE (Caire et al., 2001), cuja principal contribuição foi a definição de metamodelos para especificação dos elementos que podem ser usados para descrever cada um dos aspectos que constituem um SMA. Embora tenha sido considerada uma abordagem promissora, INGENIAS não se concentra no aspecto dinâmico do ambiente de software e nem sobre a capacidade de adaptação dos agentes.

Em Bernon (2003) é apresentada a metodologia ADELFE capaz de guiar e ajudar projetistas a construírem sistemas multiagentes adaptativos. Um importante ponto é que os agentes considerados nesta teoria não são adaptativos, somente todo o sistema. O SMA desenvolvido de acordo com ADELFE fornece uma função global emergente. O que se chama de função global é a função do sistema, ao passo que função local é fornecida por um agente. A função global é qualificada como emergente porque não está codificada dentro do agente. Os agentes não estão cientes desta função global. A função global deste sistema resulta do comportamento coletivo dos diferentes agentes que o compõem. Apesar da adaptação do sistema ser realizado globalmente, a metodologia ADELFE não trata as normas vigentes no sistema. Além disso, como o agente individualmente não se adapta, não é possível modelar agentes adaptativos capazes de suportar normas, podendo decidir por cumprir ou violar com as normas do sistema.

ADELFE 3.0 (Mefteh, 2015) foi proposta para projetar situações não colaborativas que um agente pode encontrar ou criar. Para cada uma dessas situações, estas devem deixar as ações a serem executadas para garantir que o agente volte e fique em um estado de cooperação com os outros e com ele mesmo. Os autores neste artigo integraram uma abordagem de design baseada em simulação a fim de auxiliar o design na detecção e a correção dessas situações não cooperativas. Apesar dos autores proporem a capacidade de agentes se auto projetarem, em nenhum momento foi mostrado como as características atômicas de um agente foram conectadas aos conceitos de adaptação. Além disso, os autores não se preocuparam como os agentes iriam lidar com normas que restringem seus comportamentos. E por fim, criaram um mecanismo para encontrar situações não colaborativas entre os agentes e os obriguem a voltar a se comportar de forma cooperativa, retirando assim a autonomia dos agentes.

3.4 Considerações Finais

Este capítulo apresentou os principais benefícios e lacunas dos trabalhos relacionados à abordagem proposta nesta tese, tomando como base questões que devem ser levadas em consideração para o agente se adaptar ao lidar com as normas do sistema.

A fim de discutir os tipos de agentes adaptativos, normativos, modelagem de agentes e verificação de modelos existentes na literatura, foram organizados os trabalhos relacionados em diferentes grupos: Agentes Normativos, Agentes Adaptativos e Modelamento de Agentes.

Por fim, notou-se que embora exista uma variedade de abordagens para o design e/ou implementação de agentes adaptativos normativos, nenhuma delas fornece uma solução para desenvolver agentes capazes de lidar com as várias questões de adaptação ao raciocinar sobre normas.

4
Modelo Conceitual ANA

Neste capítulo, apresenta-se o modelo conceitual ANA (*Adaptive Normative Agent*) para sistemas multiagentes. Este metamodelo possibilita a adaptação estrutural e comportamental de agentes de software capazes de lidar com normas. Usou-se a expressão modelo conceitual de forma intercambiável com o termo metamodelo usado pela OMG (UML, 2016). Além disso, a semântica do metamodelo é apresentada usando a linguagem natural no estilo do metamodelo da UML (UML, 2016). Com base da separação utilizada em (Beydoun, 2009) com relação ao tempo de design e ao tempo de execução, os metamodelos a seguir têm como objetivo mostrar os relacionamentos e entidades externas e internas ao agente de software.

O metamodelo ANA foi proposto com as características baseadas em agentes de software em relação aos conceitos não encontrados nos modelos conceituais TAO (Silva, 2002) e FAML (Beydoun, 2009). Embora estes sejam trabalhos significativos, não abordaram em sua totalidade os conceitos de adaptação e normas. Com isso, a principal função do metamodelo ANA é oferecer conceitos para entender as abstrações de adaptação e normas e os seus relacionamentos. Os novos conceitos foram introduzidos e marcados por um retângulo azul. Já os que foram alterados para se relacionarem com os conceitos inseridos estão envoltos de retângulos da cor verde.

A fim de oferecer suporte para desenvolvimento de SMAs, o metamodelo proposto visou trazer e conectar abstrações consolidadas, como adaptação e normas, inseridas na engenharia de software baseadas em agentes dada a crescente complexidade de se fazer software. Pode-se verificar a discussão recente destes conceitos em (Gowri, 2014; Mefteh, 2015; Santos Neto, 2012b; Da Silva Figueiredo et al., 2011; Silva et al., 2008; Boissier et al., 2011; Beydoun, 2009; Cervenka & Trencansky, 2007).

4.1 Descrição do Metamodelo em Tempo de Design Externo ao Agente

Para que se crie um sistema multiagentes é necessário que os agentes, as organizações e suas normas estejam dentro de um *ambiente*, sendo que os agentes não podem residir em mais de um ambiente ao mesmo tempo (d'Inverno & Luck, 2001; Silva, 2004; Beydoun 2009). A Figura 11 representa o tempo de design externo ao agente, ou seja, como um agente de software se relaciona com as entidades do ambiente que ele reside. Este metamodelo foi adaptado de TAO (Silva, 2002).

Sabe-se que *Agentes* de software são entidades autônomas capazes de realizar tarefas sem a interferência humana (Woodridgde, 2011). Agentes transformam o próprio ambiente, portanto é necessário considerar os estados do ambiente e os "*estados mentais*" do agente para atender o seu funcionamento. Além disso, um agente pode se comunicar com outros agentes para realizarem objetivos em comum, que ele sozinho não conseguiria.

O comportamento de um agente é caracterizado por meio de seus planos, ações e as normas do ambiente, que vai ao encontro de características do agente, interação, autonomia e adaptação (Santos Neto, 2012b; Silva, 2008; Jennings, 2000). Além disso, os agentes podem interagir enviando e recebendo mensagens no ambiente com outros agentes. A autonomia do agente é dada através da sua capacidade de pró-atividade, ou seja, não precisam de estímulos externos. Por fim, os agentes são entidades adaptativas, podendo adaptar seu estado e comportamento com relação às alterações e restrições do ambiente.

O *estado mental* de um agente é o estado (atributos do agente) e o comportamento em um dado momento, ou seja, são seus objetivos, crenças, desejos e intenções atrelados aos seus planos e ações. Ao executar ações, os agentes podem modificar seu estado mental, tendo novas percepções do ambiente, enviando e recebendo mensagens de outros agentes. Estes conceitos foram incluídos no modelo conceitual para que seja possível utilizar a arquitetura BDI no raciocínio do agente.

Um *papel* do agente orienta e restringe o comportamento de um agente descrevendo seus objetivos ao exercer o papel, definindo as ações que devem executar e as ações que podem executar ao exercer o papel (Yu , 1999). Caso um agente aceite a desempenhar um determinado papel, o agente adicionará os objetivos e crenças daquele papel à sua base de conhecimento. Estes novos objetivos e crenças

podem descrever restrições no seu comportamento. Além disso, nada garante que um agente, sendo uma entidade autônoma, sempre execute todas as ações daquele papel, uma vez que estas podem ser contrárias aos seus objetivos individuais.

Na abordagem de TAO (Silva, 2004) foram utilizados os seguintes conceitos: deveres e direitos, dados pelo papel ao agente que o está exercendo. Entretanto, não é suficiente para mostrar ao agente a necessidade de cumprir com os objetivos do papel, pois o agente não tem nenhuma vantagem caso resolva cumprir ou violar os deveres daquele papel. Caso o agente não veja nenhuma vantagem em realizar os objetivos do papel, ele poderá apenas cumprir seus objetivos individuais. Por isso, foi inserido o conceito de *normas* (Ver Seção 2.4) no metamodelo ANA, com o objetivo de restringir o comportamento também do papel do agente. Como propriedades de a uma norma, existem recompensas e punições, deixando para o agente raciocinar se vale a pena ou não cumprir com os objetivos daquele papel.

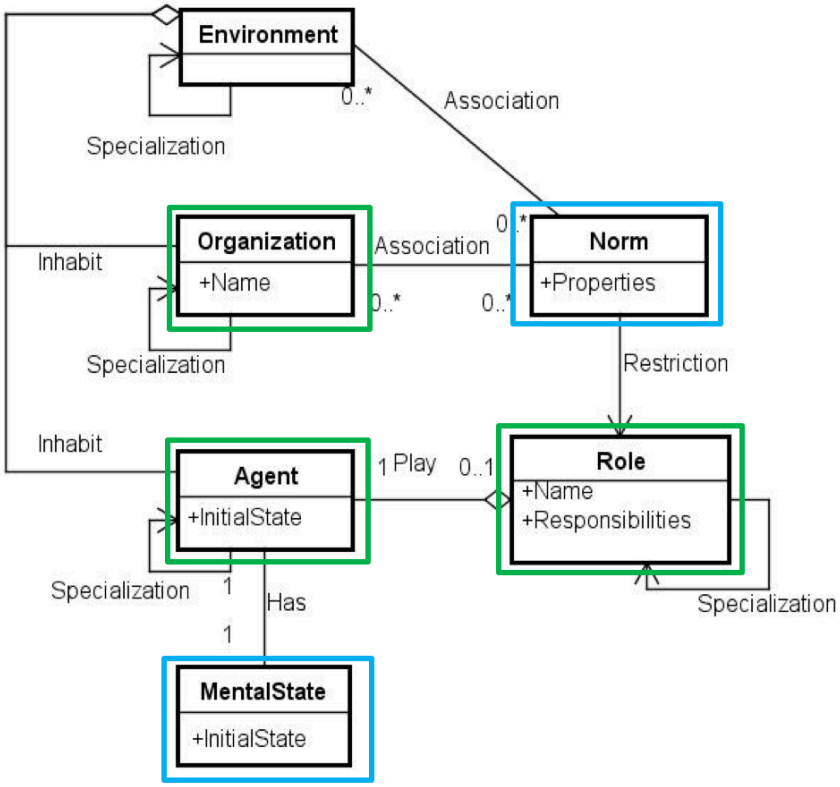

Figura 11 - Modelo em tempo de design externo ao agente

Uma *organização* por sua vez agrupa os agentes de um sistema multiagentes em *grupos* (agentes que exercem o mesmo papel) e *papéis*, ambos definindo a estrutura dos diferentes grupos de agentes e subgrupos dentro da organização (Jacamo, 2011). Na organização são definidos os planejamentos sociais, seus

objetivos e finalmente as especificações normativas. Estas especificações normativas são definidas através de normas (Ver Seção 2.4) que ligam papéis aos objetivos da organização, restringindo o comportamento do agente quando este está realizando um determinado papel.

Em TAO foi definido o conceito de *axioma*, no qual agentes e sub-organizações devem obedecer. Já no FAML, foi definido o conceito de *convenção*, um acordo entre a organização e o agente. Entretanto, os conceitos utilizados por TAO e FAML não passam a ideia precisa de como regular o comportamento dos agentes sem retirar sua autonomia e, ainda sim, incentivá-los a cumprir com os objetivos atrelados ao papel que desempenham.

Quanto aos relacionamentos em tempo de design externo, temos *specialization* e *association*. O relacionamento *specialization* define que a sub-entidade que especializa a super-entidade herda as propriedades e os relacionamentos definidos na super-entidade. As propriedades herdadas também podem ser redefinidas pela sub-entidade (Silva, 2004). Já o relacionamento *association* especifica um relacionamento de semântica que pode ocorrer entre instâncias tipificadas. Assim, se uma entidade estiver associada à outra entidade, ela saberá da existência da outra entidade e, então, poderá interagir com ela (Silva, 2004).

Além desses relacionamentos percebeu-se a necessidade de mais um relacionamento, entre a norma e o papel do agente. Assim, criou-se o relacionamento *restriction*, o qual define um relacionamento de restrição dado um papel a ser desempenhado.

4.2 Descrição do Metamodelo em Tempo de Design Interno ao Agente

A Figura 12 representa o tempo de design interno do agente, ou seja, quais são suas propriedades internas e como se relacionam. Assim, para que se possa expressar o estado mental de um agente é necessário explicitar seus componentes mentais como crenças, objetivos, intenções, planos e ações (Silva, 2004).

A entidade *AgentDefinition* tem funções genéricas no sistema, que serve para inicializar todos os agentes do sistema e uma função específica do papel, quando este agente se propõe a desempenhar tal papel.

A entidade *MentalState* é composta por componentes como crenças, objetivos, intenções, desejos planos e ações. Este é formado pelas crenças que o agente tem sobre si ou sobre o ambiente, suas intenções a serem realizadas no ambiente, seus objetivos individuais. Além disso, o *MentalState* é responsável por verificar como uma norma pode afetar as crenças e desejos de um agente. A entidade *MentalState* foi acrescentada para guardar o estado e comportamento do agente em um dado momento.

Um *plano* é composto por ações e está relacionado a um conjunto de objetivos que o agente pode alcançar ao executá-lo. Ao executar uma *ação*, o agente pode mudar seu estado mental, pode enviar uma *mensagem*, realizar passos de uma *adaptação*, por exemplo. Um agente pode ser capaz de escolher um plano com base nos seus objetivos e nas *normas* que restringem o seu comportamento.

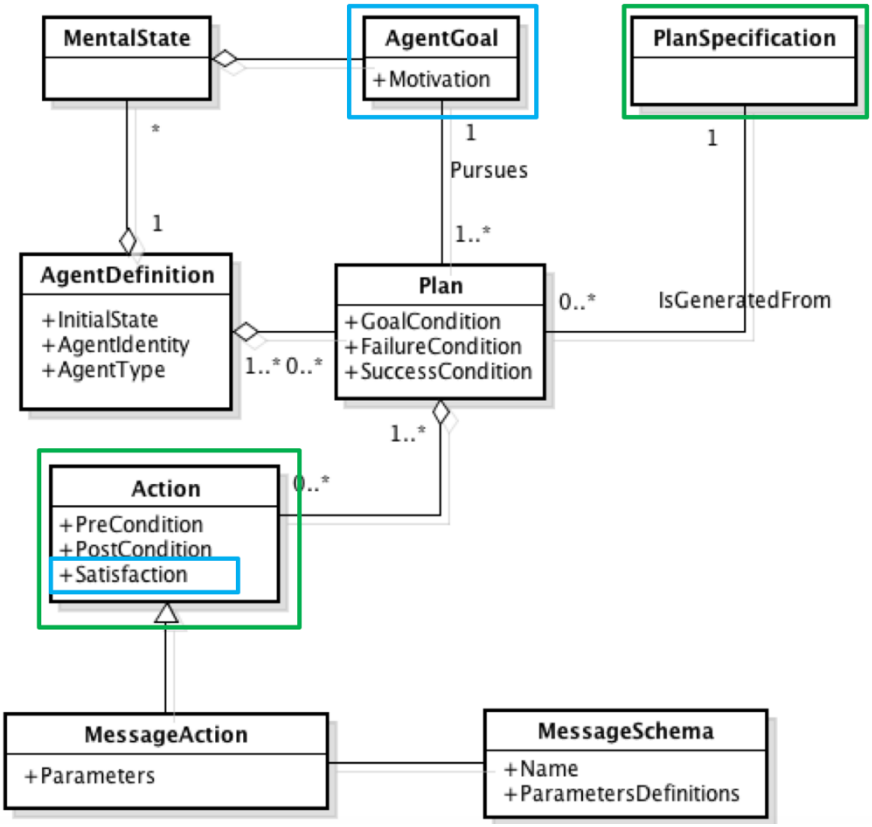

Figura 12 - Modelo em tempo de design interno do agente

O atributo *GoalCondition* em *Plan* caracteriza os objetivos pelos quais os planos podem ser aplicados. O atributo *FailureCondition* tem como objetivo expressar quando um plano não será capaz de atingir o objetivo pretendido, pode-se abortar o plano e tentar outro. Já o atributo *SucessCondition* descreve quando um plano pode ser considerado como tendo alcançado o objetivo com êxito.

Para a entidade *action* foram criados os seguintes atributos: (i) *PreCondition* são as condições (eventos do sistema) que devem ser realizadas antes da ação; (ii) *PosCondition* são as condições (eventos do sistema) que devem ser realizadas depois da ação; e (iii) o atributo *Satisfaction*, para descrever qual é o nível de satisfação que o agente terá em executar aquela dada ação.

Para que fosse possível ao agente realizar planos adaptativos, foi aproveitado a entidade *PlanSpecification* do metamodelo FAML, que permitirá definir planos de como monitorar, analisar, decidir e efetuar alterações internas no agente para que este possa lidar com as normas.

A mensagem enviada e recebida por um agente nunca está conectada às ações executadas pelo agente que recebe a mensagem. Um agente não pode chamar a execução de ações de outro agente que está enviando uma mensagem a ele.

Os objetivos do agente consistem em estados futuros ou desejos que ele pretende realizar. É importante salientar que estes objetivos estão associados a pelo menos um plano que o agente executa para realizá-lo. O atributo *Motivation* descreve a motivação que o agente tem para realizar aquele objetivo.

Os relacionamentos descrevem como as entidades do agente estão conectadas. A conexão dessas entidades define o estado mental de um agente, com suas crenças, objetivos, intenções, ações e planos. Cada instância do agente pode alterar seu estado mental inicial adicionando, modificando ou removendo as crenças, objetivos, ações e planos iniciais.

4.3 Descrição do Metamodelo em Tempo de Execução Externo ao Agente

A Figura 13 apresenta as classes relacionadas com o *ambiente* em que os agentes "vivem", em tempo de execução. Estas classes coexistem com instâncias do nível de design, por instância, como mostrado na Figura 11 e na Figura 12. As classes relacionadas a *Environment* em tempo de execução estão preocupadas apenas com as características que existem apenas em tempo de execução no ambiente.

As abstrações suportadas pelo metamodelo em tempo de execução são: (i) o histórico do *ambiente* ordenado de forma cronológica, eventos, e registros de mensagens; (ii) *Event* dá suporte para eventos de diferentes tipos; (iii) pontos de

acessos ao sistema e relacionamento com eventos, recursos, organizações e suas normas e; (iv) além do relacionamento entre agentes e os conceitos apresentados

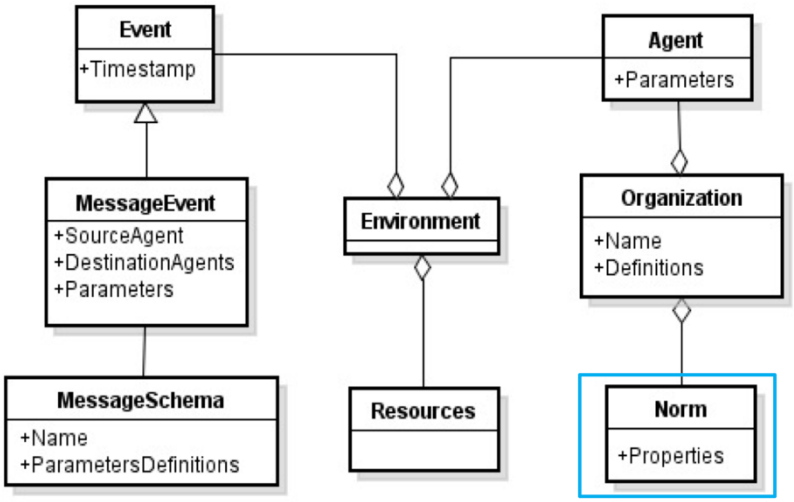

Figura 13 – Modelo em tempo de execução externo ao agente

4.4 Descrição do Metamodelo em Tempo de Execução Interno ao Agente

Finalmente, a Figura 14 mostra as classes relacionadas à parte interna do agente em tempo de execução. Essas classes só podem ser percebidas quando consideramos em tempo de execução a parte interna ao agente: (i) planos e ações; (ii) relacionamentos entre ações, mensagens; (iii) comunicação e a relação entre mensagens e protocolos; (iv) estados mentais e o relacionamento com a arquitetura BDI; (v) as relações entre as abstrações anteriores e os estados do ambiente; e (vi) o raciocínio adaptativo utilizado pelo agente para se adaptar às normas vigentes no sistema.

Neste metamodelo foram introduzidas as mudanças importantes em relações ao TAO (Silva, 2002) a possibilidade de utilizar agentes com raciocínio cognitivo baseado na arquitetura BDI, representado pelas classes *MentalState* e *AgentGoal*, uma classe mais genérica para comunicação. Os atributos *FailureCondition* e *SucecessCondition* em tempo de execução foram introduzidos para *Plan*. O primeiro descreve quando o plano não pode atingir o objetivo que está buscando, já o segundo, descreve quando pode ter considerado que o objetivo foi alcançado. Em relação ao metamodelo FAML (Beydoun, 2009) pode-se verificar a inserção do raciocínio adaptativo que o agente utilizará para se adaptar as mudanças e como às normas do ambiente são entendidas internamente pelo agente.

A abstração de adaptação é representada internamente ao agente pelas classes: (i) *Collect*, tem o atributo *sensor*, o qual tem como objetivo perceber o ambiente;(ii) *Analyze*, na qual procurando representar as diferentes *estratégias* para lidar com as mudanças no ambiente, foi criado o tipo *Strategy*, além disso, criou-se as diferentes *táticas* que poderiam ser utilizadas por diferentes estratégias; (iii) *Decision*, é responsável por tomar as decisões de adaptação através da escolha do melhor conjunto de planos dado uma situação; e (iv) *Effector*, através do atributo *actuator* é responsável por aplicar suas ações no ambiente.

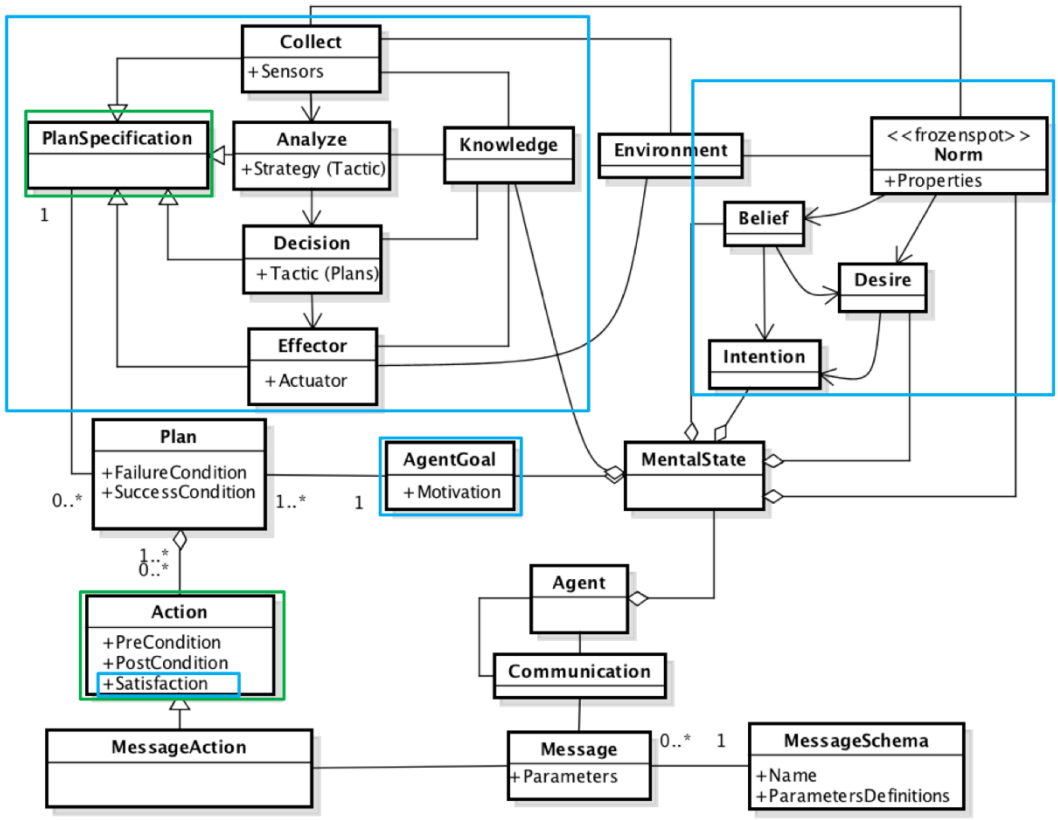

Figura 14 – Modelo em tempo de execução interno ao agente

O metamodelo possibilitou o raciocínio cognitivo do agente para entender as normas vigentes no ambiente. Isso foi realizado devido às relações definidas entre as normas, crenças, desejos e intenções. Os desejos do agente podem ser determinados independentes de suas crenças, mas também podem ser atualizados enquanto suas crenças mudam. As intenções são derivadas de um conjunto de desejos, baseados no que o agente sabe do ambiente (crenças). Assim, os agentes conseguem aprender com suas experiências de lidar com as normas e guardam esse conhecimento na entidade *Knowledge*. Assim, o agente consegue aprender, depois para realizar seu

comportamento ele cria planos adaptativos para interagir com as normas do ambiente.

4.5 Considerações Finais

Neste capítulo foram apresentados os modelos conceituais responsáveis por ilustrar como um agente entende uma norma e como ele irá proceder tanto externa quanto internamente para se adaptar a essa restrição. A partir desse metamodelo, foi possível responder algumas perguntas sobre agentes adaptativos normativos, como, por exemplo, quais são as propriedades internas ao agente, como ele se adapta e como ele entende as normas.

No próximo capítulo é apresentada uma nova linguagem de modelagem que usa como base o modelo conceitual proposto. O foco central da linguagem é permitir a modelagem de informações que ajudem no desenvolvimento de agentes capazes de se adaptar as normas que regulam seus comportamentos.

5
Linguagem de Modelagem ANA-ML

Neste capítulo apresenta-se a linguagem de modelagem ANA-ML (*Adaptive Normative Agent – Modeling Language*) criada a partir do modelo conceitual apresentado no Capítulo 4 e que foca em modelar informações úteis para criar agentes capazes de adaptar o seu comportamento para lidar com normas. A ANA-ML é uma extensão da linguagem MAS-ML (Silva, 2004) que, por sua vez, é uma extensão de UML, padrão OMG (UML, 2016) que foca no diagrama de classes estático.

Este capítulo está organizado da seguinte maneira. Na seção 5.1 são apresentados os mecanismos adotados para realizar a extensão na UML e na MAS-ML original. Na seção 5.2 são apresentadas as propriedades de um Agente Adaptativo Normativo. Na seção 5.3, o papel do agente. Na seção 5.4, a inserção do conceito de normas no modelo ANA-ML. Na seção 5.5 são apresentados seus diagramas estruturais e na Seção 5.6 são apresentados os diagramas comportamentais. Por último, na Seção 5.7, algumas considerações finais.

5.1
O Metamodelo de ANA-ML

Visando modelar as informações apresentadas nos modelos conceituais do Capítulo 4, o metamodelo MAS-ML foi estendido. O trabalho apresenta uma extensão da linguagem de modelagem MAS-ML (Silva, 2004). Para inserir nos novos conceitos mostrados no capítulo 4 algumas metaclasses sofreram alterações e outras foram criadas. Os mecanismos de extensão adotados foram a criação de um perfil que foca na definição de novas metaclasses, restrições e estereótipos de MAS-ML.

Para que se crie novas metaclasses da MAS-ML, que é uma extensão de UML, foi necessário recorrer à sua definição, onde metaclasses são definidas como o mecanismo de extensão de primeira classe porque irão modificar os metamodelos existentes (UML, 2016). As restrições, por sua vez, podem ser acopladas a qualquer

elemento do modelo com o intuito de refinar a semântica da linguagem de modelagem. O estereótipo é o elemento que define valores adicionais, restrições, ou até mesmo uma nova representação no modelo.

Na Figura 15 são ilustrados os estereótipos definidos pela ANA-ML. Alguns estereótipos apresentados são totalmente novos, enquanto outros já são oferecidos pela MAS-ML original. No entanto, estes estereótipos tiveram a adição de restrições e/ou propriedades para representar informações providas nos modelos conceituais proposto no capítulo 4.

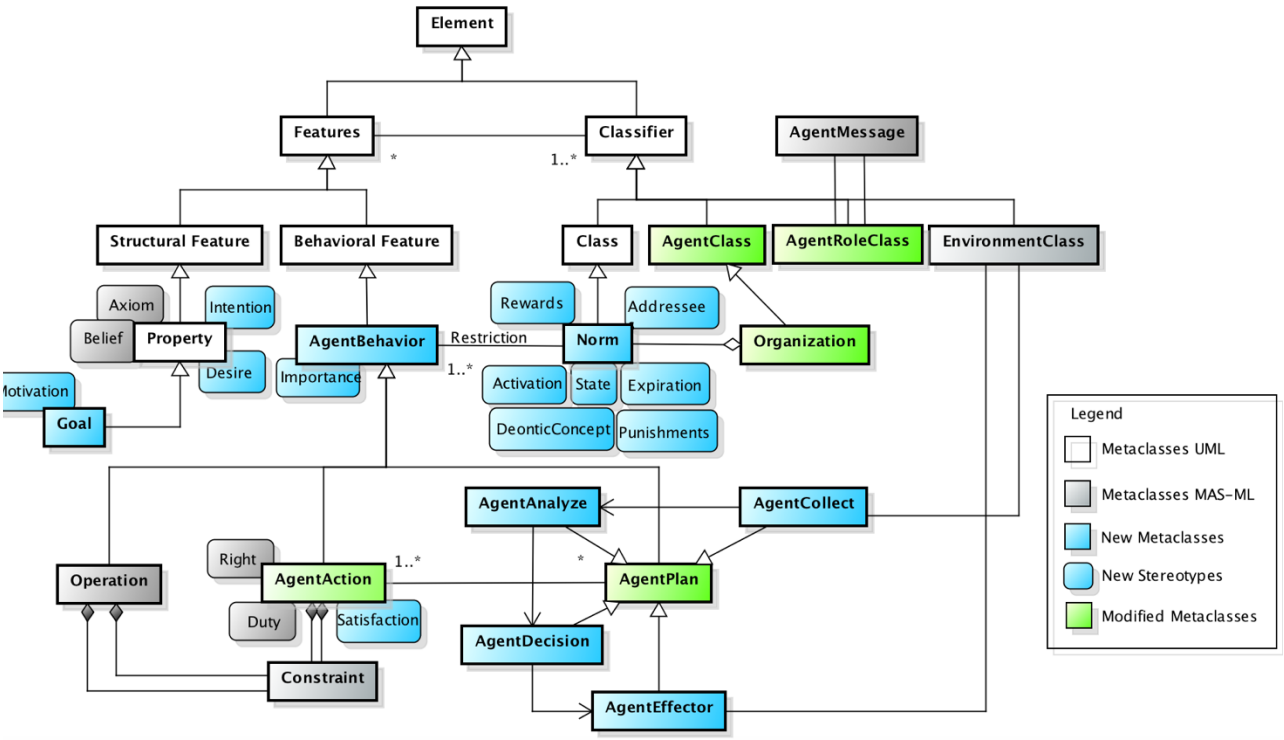

Figura 15 - O metamodelo MAS-ML estendido para incorporar as abstrações de adaptação e normas

A metaclasse *Element* é superclasse da metaclasse *Classifier*, que é a superclasse, por exemplo, da metaclasse *Class* (usada em diagramas de classe). Assim, as metaclasses *Norm* e *EnviromentClass* podem ser utilizadas em qualquer submetaclasse de *Classifier*, enquanto as metaclasses *AgentCollect*, *AgentAnalyze*, *AgentDecision*, *AgentEffector* estão relacionadas à metaclasse *AgentBehavior*, uma submetaclasse de *BehavioralFeature*. Esta última permite a modelagem em entidades comportamentais.

A metaclasse *Classifier* na Figura 15 está relacionada às metaclasses *StructuralFeature* e *BehavioralFeature*. Uma característica estrutural é uma

característica de um classificador que especifica a estrutura das suas instâncias. Uma característica comportamental é uma característica de um classificador que especifica um aspecto de comportamento de suas instâncias. Assim, a metaclasse *StructuralFeature* é uma generalização da metaclasse *Property* (atributos de uma classe são representados como instâncias de *Property*), e a metaclasse *BehavioralFeature* é uma generalização das metaclasse *Operation*, de acordo com a definição de UML (UML, 2016).

Os agentes foram descritos através da metaclasse *AgentClass* de MAS-ML, que estende a classe *Classifier* de UML. A metaclasse *AgentClass* foi modificada para que fosse possível ao agente raciocinar através da arquitetura BDI. Além disso, foram criados estereótipos tanto para que o agente entendesse as responsabilidades endereçadas a eles pelas normas quanto para modificar seu comportamento para lidar com as restrições impostas por elas ao seu comportamento.

Para representar crenças, foi aproveitado o estereótipo <<*belief*>> definido em MAS-ML. Foram adicionados novos estereótipos para criar o estado mental do agente: (i) <<*desire*>> para representar o conjunto de desejos e (ii) <<*intention*>> para definir o conjunto de intenções. Todos eles foram definidos com base na metaclasse *Property*. Uma propriedade de uma classe é um atributo (UML, 2016).

As definições de ações e planos do agente são diferentes das definidas por UML. Com isso, MAS-ML criou as metaclasses *AgentAction* e *AgentPlan*. No ANA-ML, foi criada a metaclasse *AgentBehavior* antes do relacionamento de *AgentAction* e *AgentPlan* com *BehavioralFeature*, isto para descrever o comportamento do agente como um todo. A metaclasse *AgentBehavior* possui um estereótipo <<*Importance*>> o qual define a motivação de realizar um objetivo ou a satisfação em executar uma ação (Lopez & Marquez, 2004).

Para inserir normas no metamodelo, criou-se uma nova metaclasse chamada *Norm*. A *Norm* estende a metaclasse UML *Class* e, portanto, uma norma pode ser representada como um objeto. Uma norma está relacionada com a metaclasse *AgentBehavior* através do relacionamento *Restriction*. E se relaciona também com *OrganizationClass* através do relacionamento de *Aggregation*.

As organizações estendem a noção de agentes, porém existem outras propriedades e relacionamentos. A alteração feita na metaclasse *OrganizationClass* da MAS-ML foi a incorporação do conceito de normas, o qual é mais preciso e restrito do que *axiomas*, que se caracterizam pelas restrições globais da organização

(Silva, 2004). Já as normas foram incorporadas para mostrar aos agentes comportamentos atrelados ao seu papel em uma organização que são obrigatórios, permissivos ou proibitivos (Viana, 2015c).

O papel do agente por sua vez continua a ser definido pela metaclasse *AgentRoleClass*, entretanto foi incorporado a ele um conjunto de normas com as quais o agente decidirá como lidar . Note que intenções e desejos, além de planos, são propriedades internas ao agente e não a seu papel.

Para que fosse possível criar o comportamento adaptativo no agente foram criadas as seguintes metaclasses: (i) *AgentCollect* para perceber as normas no ambiente; (ii) *AgentAnalyze* analisar os possíveis planos para lidar com as normas endereçadas ao agente (iii) *AgentDecision* para decidir através de diferentes estratégias e; (iv) *AgentEffector* para efetuar as ações e planos no ambiente para que o agente se adapte a norma.

Nas próximas subseções são apresentadas em detalhes as extensões propostas para ANA-ML. Serão descritas as metaclasses e estereótipos criados para representar as entidades, as propriedades e os relacionamentos definidos nos metamodelos do capítulo 4. Para estereótipos já propostos pela MAS-ML são apresentadas, exclusivamente, as informações incluídas nas entidades modeladas que possuem tais estereótipos.

5.2 Propriedades do Agente

O agente utilizado no modelo ANA é uma extensão do descrito em MAS-ML, estendendo a classe UML *Classifier* e, portanto está associada às metaclasses *StructuralFeature* e *BehavioralFeature*. As características estruturais definidas por um agente em MAS-ML são objetivo e crença, e suas características comportamentais são ação e plano. A fim de descrever como a classe *AgentClass* irá lidar com as normas através de adaptação, serão definidas novas características como desejos e intenções e as características comportamentais de percepção do ambiente através da metaclasse *AgentCollect*, a análise desses dados por *AgentAnalyze*, as decisões a serem tomadas por *AgentDecision* e um comportamento específico para efetuar as ações do agente no ambiente, representado por *AgentEffector*.

O foco do metamodelo será na construção de planos adaptativos através de novas características inseridas no metamodelo para que os agentes sejam capazes de lidar com normas. Além disso é necessário explicar por que algumas características foram modificadas, como por exemplo *AgentClass*, *AgentRoleClass*, *OrganizationClass* e *AgentPlan*.

5.2.1 Objetivos, Crenças, Desejos e Intenções

Quando se pensa no estado mental de um agente é preciso considerar características como objetivos, crenças, desejos e intenções. Os objetivos representam o que o agente quer realizar no seu ambiente; as crenças representam o conhecimento sobre o mundo que ele tem; os desejos representam o que eles esperam que realizem e suas intenções, representam os estados que o agente selecionou para serem alcançados.

Para representar objetivos na linguagem MAS-ML, Silva (2004) criou a metaclasse *Goal*, a qual descreve uma *tag* que está ligada aos planos e uma restrição. Entretanto, em ANA-ML viu-se a necessidade de incorporar ao conceito de objetivo uma motivação para que este fosse cumprido. A motivação para alcançar um objetivo é definida pelo estereótipo <<*motivation*>>, que mapeia um objetivo do tipo *Goal* para um inteiro, que representa a intensidade do desejo do agente para alcançar tal objetivo. Esse estereótipo é inspirado na *motivation* apresentada em (Wooldridge, 2011). Essa motivação leva em conta apenas um valor que é dado ao objetivo do agente. No entanto, caso objetivos tenham a mesma prioridade, poderia ser levado em consideração outros fatores, tais como os traços de personalidade – atitudes e emoções, apresentado no trabalho (Barbosa et al., 2014) que poderiam resolver o problema de qual objetivo priorizar dado que esses têm o mesmo valor de motivação.

Já na representação de crenças, foi mantido o estereótipo <<*belief*>>, não especificando qualquer tipo de restrição, apenas identificando os atributos que são crenças. Em ANA-ML por sua vez, foram definidos novos conceitos observando a entidade *mentalState* no modelo conceitual do Capítulo 4 para lidar com agentes BDI. Assim, foram criados novos estereótipos <<*desire*>> e <<*intention*>>. O estereótipo <<*desire*>> descreve os desejos que o agente espera alcançar. Já o

estereótipo <<*intention*>> descreve os estados que o agente deu prioridade para serem alcançados.

5.2.2 Ação

As ações são características comportamentais de agentes. As ações associadas a agentes nunca são chamadas por outro agente, mas apenas executadas sob o controle do próprio agente. Assim, criou-se uma nova metaclasse chamada *AgentBehavior* que estende *BehavioralFeature* a fim de representar o comportamento do agente. A metaclasse *AgentAction*, da MAS-ML, agora é uma extensão de *AgentBehavior*, e continua com a mesma finalidade: a fim de representar as ações executadas por agentes. Associou-se a metaclasse *Constraint* a *AgentAction*, com o objetivo de definir as precondições e pós-condições que devem ser verdadeiras quando a ação é chamada. Essas precondições são pressupostas por uma implementação dessa ação. As pós-condições para uma ação definem as condições que serão verdadeiras quando a execução da ação é concluída com sucesso, pressupondo que as precondições tenham sido satisfeitas. Essas pós-condições devem ser satisfeitas por qualquer implementação da ação.

Uma ação definida em ANA-ML, além de ter as características descritas acima, recebeu um estereótipo de satisfação <<*satisfaction*>>. Essa satisfação do agente é definida com base nos ganhos e perdas ao executar uma ação. Satisfação é definida pela função *satisfaction*, que mapeia uma ação do tipo *Action* para um inteiro que representa a intensidade de satisfação do agente para executar tal ação. Considerou que tal função é definida em tempo de projeto e é inspirada em (Dam & Winikoff, 2011).

5.2.3 Plano

As características comportamentais de agentes também são compostas por seus planos. A classe *AgentPlan* da MAS-ML foi estendida para que seja possível criar planos adaptativos para lidar com normas. Isto sabendo que um plano está associado a um objetivo e é representado por uma sequência de ações executadas por um

agente para alcançar seus objetivos. Para que a criação de planos adaptativos fosse possível, criou-se diferentes estereótipos que serão melhor explicados abaixo.

Em ANA-ML *AgentPlan* é uma especialização da metaclasse *AgentBehavior* que é uma especialização de *BehavioralFeature*. Foi preciso colocá-la como especialização de *AgentBehavior* para deixar clara a relação de restrição entre as metaclasses *Norm* e *AgentBehavior*.

5.2.4 O Comportamento do Agente

Como se sabe, o comportamento de um agente é expresso por meio dos seus planos e ações, que se baseiam em suas características, interação, autonomia e adaptação. Os agentes podem interagir com outras entidades enviando e recebendo mensagens. A característica de autonomia refere-se à capacidade proativa de um agente. Além disso, são entidades adaptativas, uma vez que podem adaptar seu estado e comportamento respondendo a mensagens enviadas pelo ambiente ou outros agentes.

Dado que este trabalho está focado na característica de agentes adaptativos normativos, eles devem decidir qual comportamento deve ser realizado. Para que isso aconteça, deve ser levada em conta a *importância* de um comportamento antes de tomar uma decisão (Wooldridge, 2011). A importância de realizar um comportamento do tipo *AgentBehavior* é definida pelo estereótipo *Importance* que mapeia um comportamento do tipo *AgentBehavior* para um inteiro representando quão importante é a realização de tal comportamento para o agente. A importância é avaliada levando em consideração a motivação (Ver Seção 5.2.1) no caso do comportamento ser um objetivo, ou a satisfação do agente em executar uma ação (Ver Seção 5.2.2) no caso do comportamento ser uma ação.

Para lidar com normas e para representar estes comportamentos, foram criadas as novas metaclasses: *AgentCollect*, *AgentAnalyze*, *AgentDecision*, *AgentEffector*. Além da metaclasse que leva as propriedades de uma norma, representada por *Norm*.

A metaclasse *AgentCollect* foi criada para representar a percepção do agente sobre o ambiente, ou seja, o que ele monitora enquanto está ativo. Diferentemente do trabalho (Felicíssimo et al., 2011), que utiliza o raciocínio e dedução na atividade

de coleta, aqui, a coleta serve apenas para perceber se a norma é endereçada a um determinado agente. Para cada norma identificada no ambiente pelo agente, as seguintes verificações são realizadas: (i) se a norma ainda não existe na base de normas adotadas; e (ii) se a norma é endereçável ao agente.

As normas são representadas pela metaclasse *Norm*, responsável por restringir o *comportamento* dos agentes e por sua vez, são percebidas através de sensores (estereótipo <<sensor>>) da metaclasse *AgentCollect*.

A avaliação do agente para decidir sobre quando se adaptar à norma é realizada na metaclasse *AgentAnalyze*. Esta etapa se baseia nas informações coletadas pelos sensores de *AgentCollect* e utiliza um conjunto de mecanismos de raciocínio para detecção de problemas e prover a melhor estratégia de comportamento em relação a norma.

Já a metaclasse *AgentDecision* é responsável pela tomada de decisão, ou seja, nosso ponto de adaptação, se baseia na execução de um conjunto de mecanismos de seleção para escolher, dentre os vários planos disponibilizados, os melhores planos para o agente decidir cumprir ou violar a norma que está sendo deliberada. O estereótipo *AgentEffector* é responsável por receber a solução sugerida pela atividade *Decision*, e executá-la através dos seus atuadores.

5.3 Papel do Agente

Um papel de agente está interessado no comportamento do agente para realizar os objetivos de uma organização e, ao mesmo tempo, restringir o comportamento destes agentes para que eles possam realizar suas tarefas com maior eficiência. Existe aqui uma diferença intrigante entre MAS-ML e ANA-ML, porque em MAS-ML foram descritos os seguintes mecanismos para restringir os comportamentos dos agentes: (i) deveres, ou seja, ações que o agente deve realizar; (ii) direitos, ações que os agentes têm permissões para realizar; e (iii) protocolos, os quais o agente deve obedecer e através das quais interagir com outras entidades do sistema. Já em ANA-ML foi inserido o conceito de normas, para restringir o comportamento dos agentes. O grande diferencial aqui é que uma norma, além de dar direitos e deveres, incentiva o agente a cumpri-la, pois, atrelado a essa norma vem um conjunto de recompensas, caso seja cumprida, e punições, caso a norma seja violada.

5.4 Normas em ANA-ML

As propriedades internas de uma norma (Ver Seção 2.4) são muito particulares e por isso foi necessário defini-las através dos estereótipos da metaclasse *Norm*: (i) o estereótipo<<*addressees*>> identifica os agentes, grupos ou papéis responsáveis pelo cumprimento da norma; (ii) o estereótipo <<*deonticConcept*>> é usado para indicar se a norma estabelece uma obrigação, uma permissão ou uma proibição; (iii) o estereótipo <<*activation*>> identifica uma condição de ativação da norma; (iv) o estereótipo <<*expiration*>> define a condição para que uma norma se torne inativa; (v) o estereótipo <<*state*>> é usado para indicar um conjunto de estados ou ações que estão sendo regulados pela norma; (vi) o estereótipo <<*rewards*>> identifica um conjunto de recompensas para ser dado ao agente caso ele cumpra com a norma; e (vii) o estereótipo <<*punishments*>> identifica um conjunto de punições para ser dado ao agente caso ele viole com a norma.

5.5 Elementos de Diagramas Estruturais

A fim de introduzir as novas abstrações no metamodelo ANA-ML, precisou-se criar novos elementos de diagrama para representar as novas entidades e seus atributos. Tem-se como objetivo focar na criação dos diagramas de classes e no diagrama de atividades. Esses dois diagramas foram escolhidos porque permitem a incorporação de aspectos estruturais e dinâmicos das informações úteis para coordenação da adaptação do agente em um ambiente normativo apresentado no modelo conceitual proposto.

Nas próximas subseções são apresentadas em detalhes as restrições e propriedades definidas em cada entidade e estereótipo usado pela ANA-ML. Para estereótipos já propostos pela linguagem MAS-ML original (Silva, 2005) são apresentadas, exclusivamente, as informações incluídas nas entidades modeladas que possuem tais estereótipos. Para ajudar no entendimento da ANA-ML, a seguinte estrutura é adotada para explicação: (i) descrição do estereótipo, (ii) notação a ser usada, (iii) restrições, e (iv) um simples exemplo ilustrando como

representá-lo em um modelo. Tal estrutura é baseada nos documentos oferecidos pela OMG (UML, 2016) que apresentam perfis oferecidos para a UML.

5.5.1 AgentClass

A Figura 16, mostra a classe do agente. O estereótipo <<*sensor*>> foi adicionado à metaclasse *AgentClass* para representar a percepção do agente em relação ao ambiente. Além disso, foram incorporados ao modelo os estereótipos <<*desire*>> e <<*intention*>> para que fosse possível construir agentes baseados na arquitetura BDI, a qual se baseia nas noções de crenças, desejos e intenções como atitudes mentais que orientam a realização de comportamentos.

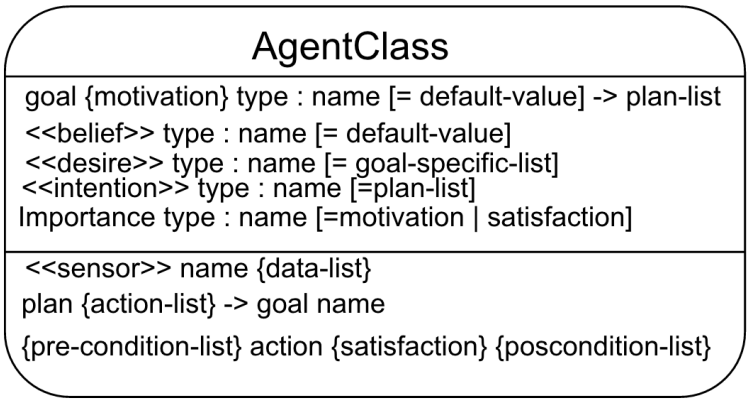

Figura 16 - A classe do Agente

5.5.2 OrganizationClass

A Figura 17 mostra a classe da organização modificada. *OrganizationClass* agrupa os agentes de um sistema multiagentes em grupos e papéis, ambos definindo a estrutura dos diferentes grupos de agentes e subgrupos dentro da organização.

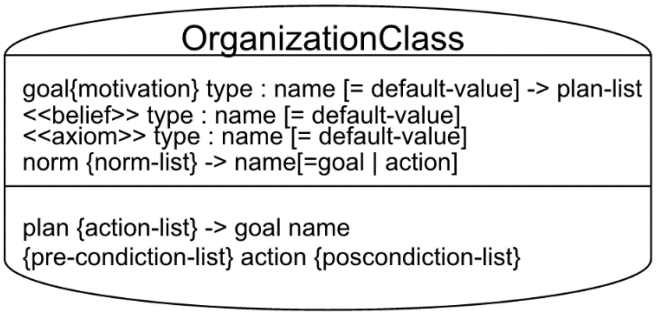

Figura 17 - A classe da organização

A modificação feita em relação ao conceito de MAS-ML foi a extensão de uma organização para inserir a entidade norma, que será explicada em detalhes em seguida. Essa é uma característica estrutural, mas que modificará o comportamento dos agentes.

5.5.3 Norm

A Figura 18 mostra uma norma em ANA-ML como um objeto em UML, entretanto, com dois compartimentos separados por linhas horizontais, além das bordas esquerda superior e direita inferior serem arredondadas. O compartimento superior contém o nome da norma e deve ser único. No compartimento inferior serão colocadas suas características estruturais, sendo elas os estereótipos descritos abaixo.

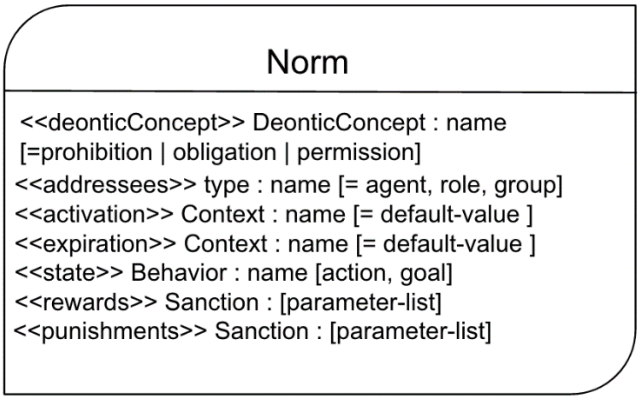

Figura 18 - A classe Norma

Restrições: Uma norma deve conter os seguintes atributos:

1. *deonticConcept* é usado para indicar se a norma estabelece: (i) se o agente é obrigado a cumpri-la; (ii) se o agente tem permissão para realizar uma dada ação por ela regulada; ou (iii) se a norma proíbe que ele realize a ação que ela regula;
2. *addressees* identifica os responsáveis pelo cumprimento da norma, ou seja, uma lista de agentes, representada por nomes, uma lista de papéis desempenhados pelos agentes e uma lista de grupos;
3. *activation* identifica uma condição de ativação da norma;
4. *expiration* define a condição para que uma norma se torne inativa;

5. *state* é usado para indicar um estado ou ação que está sendo regulada pela norma;
6. *rewards* identifica um conjunto de recompensas para ser dado ao agente caso ele cumpra com a norma e;
7. *punishments* identifica um conjunto de punições para ser dado ao agente caso ele viole com a norma.

5.6 Diagramas Dinâmicos da ANA-ML

Os diagramas propostos em (Gonçalves et al., 2015) e utilizados para diagramas de atividades foram reusados. Assim, cada atividade é representada por um retângulo com os cantos arredondados. As características do agente são representadas com quadrados com a identificação de qual estereótipo seria do agente.

O objetivo dos diagramas de atividade é representar o comportamento do agente quando ele entende que uma norma foi endereçada a ele. O comportamento do agente adaptativo normativo é representado na Figura 19 e detalhado na Figura 20. A realização de raciocínio adaptativo normativo é possível através de modificações realizadas no processo de aplicação normativa (Viana et al., 2015c) (Ver Seção 7.1).

A Figura 19 mostra o comportamento do fluxo dinâmico em ANA-ML, que começa quando ele identifica normas no ambiente e realiza a consciência normativa, realizada pela atividade de *NormAwareness*. O agente de software identifica que existem normas ativas no ambiente endereçadas a ele. Em seguida, a adoção de normas é feita pela atividade *NormDeliberation*, os agentes reconhecem suas responsabilidades através de outros agentes por internalizar as normas onde a responsabilidade deles é especificada. Já na atividade *NormCompliance*, a deliberação de normas é feita através do raciocínio do agente, vendo quais seriam os planos que ele poderia realizar ou não ao aceitar cumprir com uma dada norma. Na última atividade, chamada de *NormImpact*, o agente irá executar suas intenções e planos para lidar com a norma, e seus objetivos serão atualizados. Em seguida, o ciclo continua e o agente começa a identificar outras normas que devem ser adotadas.

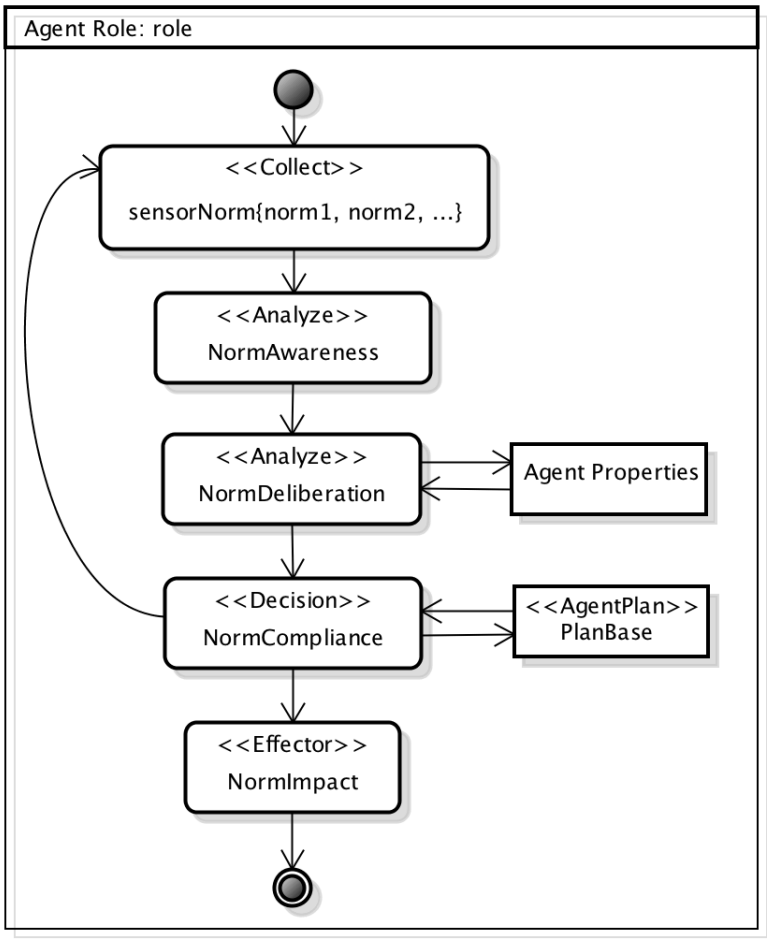

Figura 19 - Diagrama de atividades: agente adaptativo normativo

Na Figura 20 fica claro que após verificar quais as normas o agente selecionou para serem adotadas ou violadas, começa a serem escolhidos os planos que ele traçará para realizar com essas normas. Os planos estão ligados a pelo menos um objetivo e quais seriam as ações necessárias para cumprir com os objetivos e ações que a norma está regulamentando.

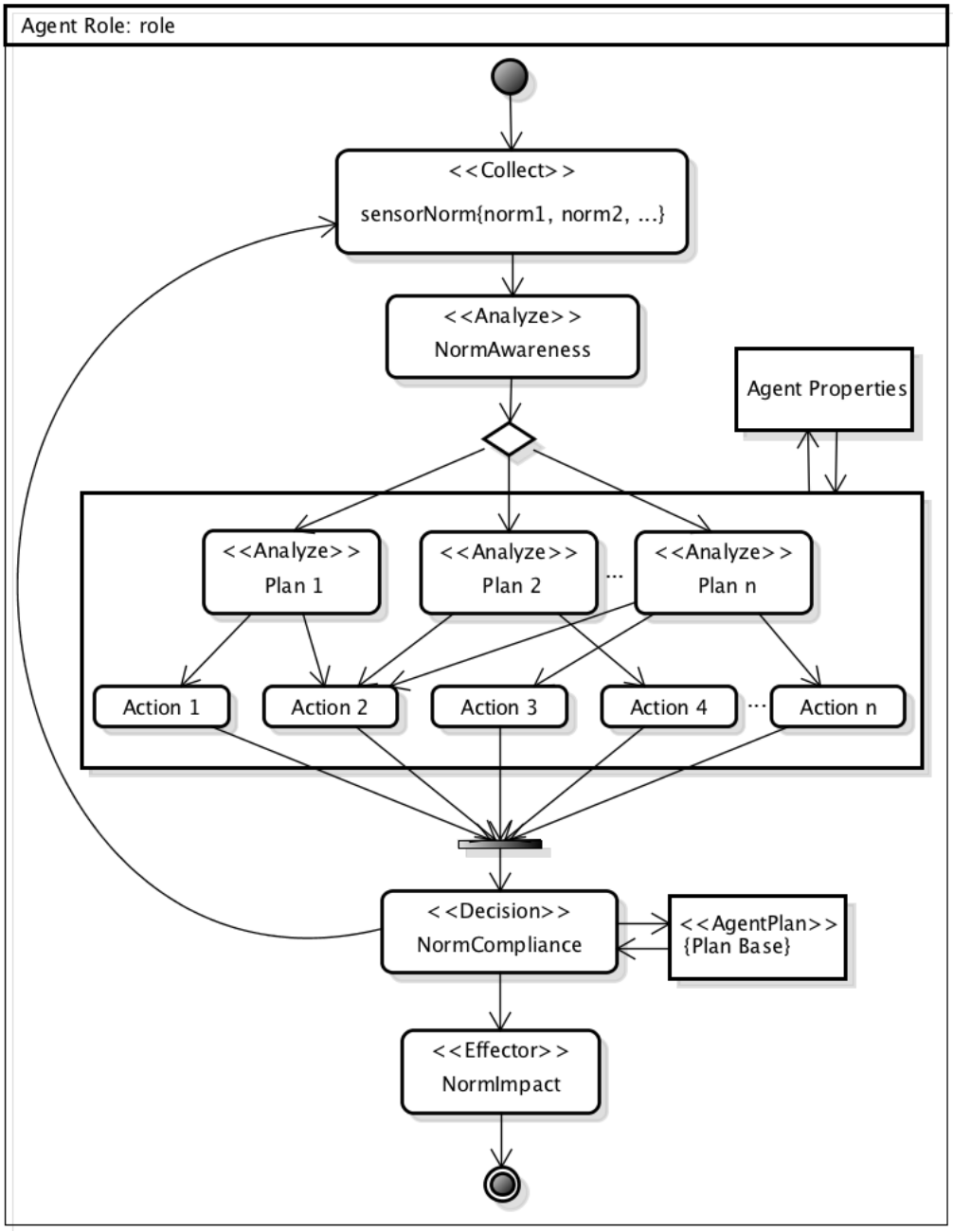

Figura 20 - Diagrama de atividades ANA-ML: detalhamento da atividade de deliberação de normas

5.7 Considerações Finais

Neste capítulo foi apresentada a linguagem ANA-ML voltada para modelar informações úteis para identificar as adaptações que o agente poderá sofrer para lidar com as normas. Para explica-la foi apresentado: (i) os mecanismos adotados para realizar a extensão de MAS-ML; (ii) seu metamodelo acompanhado com exemplos simples; (iii) o mapeamento das informações consideradas no modelo

conceitual, proposto no capítulo 4, em relação a ANA-ML e (iv) avaliações da linguagem de modelagem ANA-ML envolvendo pessoas com diferentes níveis de conhecimento na área de modelos de sistemas multiagentes.

No próximo capítulo será realizado um experimento para validar a importância da criação da linguagem de modelagem ANA-ML.

6 Avaliações da ANA-ML

A proposta deste capítulo é apresentar a avaliação da linguagem de modelagem ANA-ML. Em particular, a avaliação foca em três pontos distintos: erros, tempo e dificuldade dos participantes durante a realização do experimento.

Este capítulo está organizado da seguinte maneira. Na seção 6.1 estão descritas as hipóteses que este estudo espera ratificar. Na seção 6.2 são analisados os resultados do estudo e na seção 6.3, as possíveis ameaças ao estudo e como foram contornados.

6.1 Avaliação para Criação e /ou Manutenção dos Modelos

Visando analisar a linguagem ANA-ML, avaliações empíricas (Kithenham, 2002; Wohlin et al., 1999; Costa, 2012; Baia, 2016) foram realizadas. O foco dessas avaliações foi investigar as seguintes hipóteses:

H0: Duas amostras são idênticas para cada métrica.

H1: ANA-ML reduz o erro na criação e/ou manutenção de modelos de agentes com características adaptativas normativas comparados ao uso de agentes com técnicas da MAS-ML.

H2: ANA-ML reduz o tempo na criação e/ou manutenção de modelos de agentes com características adaptativas normativas comparados ao uso de agentes com técnicas da MAS-ML original.

H3: ANA-ML reduz a dificuldade na criação e/ou manutenção de modelos de agentes com características adaptativas normativos comparados ao uso de agentes com técnicas da MAS-ML.

Para analisar as hipóteses ***H0***, ***H1***, ***H2*** e ***H3*** foi realizado uma avaliação empírica com 14 participantes com diferentes conhecimentos e experiências na área de sistemas multiagentes. Os participantes envolvidos tiveram formações em diferentes instituições superiores e diferença nos anos de experiência, entretanto, todos realizaram a disciplina de sistemas multiagentes de pós-graduação.

6.1.1 Questionário e Aplicações do Questionário

Os questionários aplicados solicitaram a criação e manutenção de modelos baseados na MAS-ML e na ANA-ML. O tema considerado foi referente ao domínio de Prevenção de crimes. Como alternativa, ao uso de padrões de deslocamentos antigos, criminologistas têm unido forças com pesquisadores da Ciência da Computação e Inteligência Artificial para explorar os benefícios de simulações sociais (Bosse e Gerritsen, 2010), com o uso de agentes, para investigar o deslocamento de crimes. Assim, a perspectiva desta abordagem é usar os ambientes de simulação para predizer a dinâmica do deslocamento de crimes no futuro, melhor que analisar as dinâmicas do passado.

A simulação de prevenção de crimes foi modelada como uma organização que habita o ambiente de uma cidade e define para os agentes pessoa os seguintes papéis: civil, bandido e guarda. Os civis se deslocam pelas diferentes localidades da cidade para ir a shoppings, museus, grandes eventos, ou seja, áreas com grande concentração de pessoas. Já os agentes no papel de bandido procuram por esses locais para cometer o maior número de assaltos possível. E os agentes guardas se deslocam pelas diferentes áreas da cidade com os objetivos de evitar o maior número de crimes e prender esses bandidos.

Na Tabela 1 está detalhado o objetivo e quais conceitos de agentes adaptativos normativos estavam envolvidos em cada questão dos questionários aplicados.

Para aplicação do questionário, dividiu-se os participantes em dois grupos, A e B. Na primeira etapa do experimento, os participantes do grupo A realizaram as duas atividades utilizando apenas as informações disponíveis pelo treinamento na linguagem de modelagem MAS-ML. Na segunda etapa do experimento, os participantes do grupo A utilizaram a linguagem de modelagem ANA-ML para auxiliar a realizar as duas atividades restantes.

Tabela 1 - Detalhamento dos questionários aplicados

Questões	Principais informações envolvidas	Objetivo principal
Questão 1	• Criação de normas • Manutenção do diagrama de ambiente; • Relacionamento entre normas e organizações	Criar novas normas em um ambiente regido por uma organização principal.
Questão 2	• Criação de agentes • Identificação de normas • Criação de planos	Criar diagramas de agentes com características e comportamentos capazes de lidar com normas.
Questão 3	• Manutenção de diagramas normativos • Criação de novas normas	Manter diagramas de classes que representa normas.
Questão 4	• Diagrama de atividades • Uso das abstrações de normas e adaptação • Raciocínio do agente sobre a norma	Criar diagramas de atividades capazes de reproduzir o processo de autoadaptação do agente sobre uma norma.

De forma inversa, na primeira etapa do questionário, os participantes do grupo B realizaram as duas atividades utilizando apenas as informações passadas durante o treinamento da linguagem de modelagem ANA-ML. E na segunda etapa utilizam apenas as informações do treinamento na linguagem de modelagem MAS-ML para realizar as duas atividades restantes. Com isso, foi projetado o *Quadrado Latino* (Winer, 1962.), podendo aplicar métricas para avaliar a relevância do uso da ANA-ML por grupos, etapas e por atividades (Para maiores detalhes sobre o questionário consultar o apêndice A).

Aplicou-se os questionários para alunos de pós-graduação, todos já cursaram ou estão cursando a disciplina de sistemas multiagentes. Essa aplicação aconteceu no mesmo dia, porém em horários diferentes. Foi acordado entre os participantes que nenhuma informação poderia ser passada a diante para não prejudicar ou induzir os resultados. Assim, os grupos A e B realizaram os questionários em diferentes ambientes. A Tabela 2 ilustra como ficou a configuração do *Quadrado Latino*.

Tabela 2 – Quadrado Latino projetado para o estudo

	A	B
Primeira Etapa	MAS-ML	ANA-ML
Segunda Etapa	ANA-ML	MAS-ML

A avaliação de criação e/ou manutenção de modelos adotou seis fases. O grupo A recebeu treinamento na primeira fase para que pudessem criar modelos a partir do uso da MAS-ML original e de técnicas livres da UML (ex: uso de estereótipos e comentários) para complementar a modelagem da MAS-ML. Já o grupo B recebeu o treinamento para ANA-ML, seguindo os mesmos passos anteriores. Os treinamentos duraram de 30 a 60 minutos.

Na segunda fase (Primeira etapa do *Quadrado Latino*) foi aplicado o questionário do apêndice A para os dois grupos. A primeira parte do questionário coleta informações sobre cada participante. Para isso, aplica-se perguntas referentes a formação, conhecimentos específicos e finalmente sobre tempo de experiência em modelagem de software para sistemas multiagentes. A segunda parte do questionário se divide em duas sub etapas, cada uma com duas atividades de modelagem, solicitando a criação de diagramas baseados na MAS-ML e na ANA-ML, para os diferentes grupos. Os grupos A B realizaram as atividades 1 e 2.

Já a terceira etapa focou no treinamento inverso dos grupos, ou seja, o grupo A recebeu treinamento de ANA-ML e o grupo B de MAS-ML. O treinamento durou de 30 a 60 minutos.

A quarta fase (Segunda etapa do *Quadrado Latino*) aplicou outro questionário aos participantes solicitando a criação de modelos baseados na ANA-ML para o grupo A e criação e/ ou manutenção de modelos baseados em MAS-ML para o grupo B. Tanto a estrutura como a complexidade desse questionário foi similar ao questionário aplicado na segunda fase. Para evitar qualquer influência nas questões dos questionários três especialistas em modelagem e engenharia de software experimental realizaram a validação destes. Os grupos A B realizaram as atividades 3 e 4.

Já a quinta fase realizou entrevistas com cada participante para identificar quais facilidades e dificuldades encontradas enquanto criavam os diagramas solicitados. Por último, na sexta fase, os dados providos pelos participantes foram recuperados

e analisados. Detalhes da quinta e sexta etapa são apresentados na próxima subseção.

6.1.2 Métricas

Com o intuito de analisar as hipóteses apresentadas acima, as variáveis independentes consideradas foram as seguintes: (i) participantes envolvidos, (ii) treinamentos aplicados para modelar com UML, MAS-ML e ANA-ML, e (iii) atividades solicitadas aos participantes para a criação de modelos. Já as variáveis dependentes foram: (i) o tempo (esforço) gasto; (ii) a quantidade de erros modelados pelos participantes (qualidade dos modelos) a partir das atividades solicitadas; e (iii) dificuldade encontrada.

Para calcular a métrica do tempo, em minutos, o participante registra a hora e minuto no início e no final de cada atividade realizada. Desta forma foi registrado o tempo total em minutos para cada atividade específica. Além disso, ao final de cada atividade, o participante também registra o nível de dificuldade percebida, em uma escala de 1 a 5. O 1 representa uma menor percepção de dificuldade e 5 representa uma maior percepção de dificuldade.

A fim de avaliar a taxa de erros nas atividades, calculou os erros inseridos nas atividades pelos participantes de acordo com a Tabela 10. Os detalhes dos cálculos para a taxa de erro em cada atividade estão na subseção 6.2.2 em que foram analisados os dados coletados e apresentou-se os resultados.

Para testar as hipóteses deste estudo, primeiro verificou se a distribuição da amostra é normal (Shapiro-Wilk) e se a variância é igual (Levene). Se esses testes forem aprovados, será utilizado o teste ANOVA para avaliar as hipóteses, ou seja, se há alguma evidência de que as médias das populações são diferentes. Caso contrário, será aplicado o teste não paramétrico de Kruskal-Wallis. Se esses testes levam à conclusão de que há evidências de que os meios do grupo são diferentes, então estamos interessados em investigar quais meios são diferentes. Para a correção e variável de tempo, mantemos um nível de confiança típico de 95 por cento (alfa = 0,05).

6.2 Resultado das Avaliações Aplicadas

Como mencionado anteriormente os participantes foram divididos em dois grupos, A e B. Para estes grupos, foram aplicados o mesmo questionário que possui duas etapas, cada etapa possui duas atividades. Para maiores informações sobre os questionários, consultar apêndice A.

A análise de dados e apresentação dos resultados foram divididas em duas subseções. Na primeira subseção será analisado o perfil dos participantes. Na segunda subseção, analisado o *Quadrado Latino* de forma horizontal, por etapas, e de forma vertical, por grupo. Para isso, será seguida uma abordagem quantitativa e qualitativa.

6.2.1 Perfil dos Participantes

O gráfico da Figura 21 ilustra a distribuição em anos de experiência por participantes. Os perfis dos participantes foram definidos na primeira parte do estudo, comum em ambos.

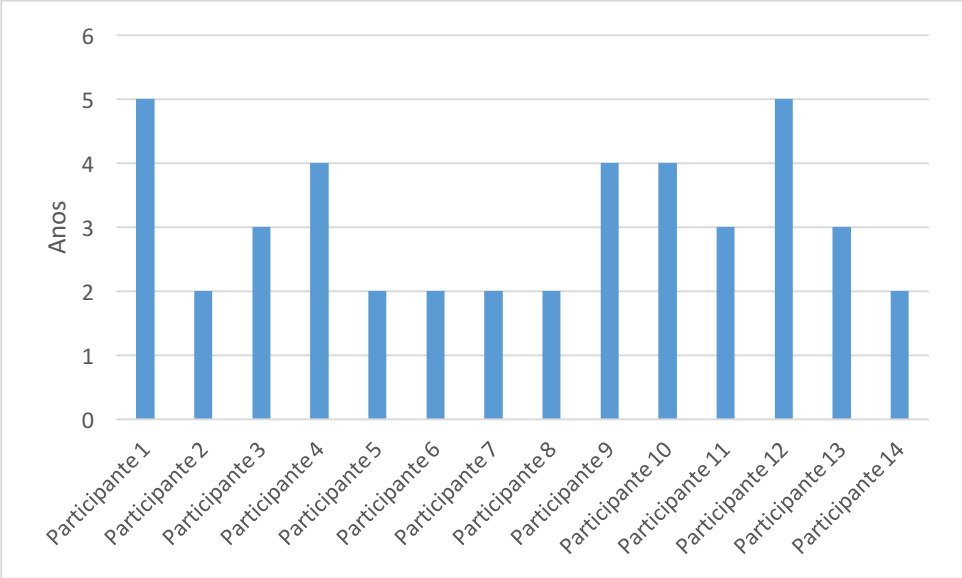

Figura 21 - Anos de experiência dos participantes em modelagem

O estudo contou com a participação de 14 alunos, onde 1 tinha pós-doutorado, 6 estão cursando o doutorado e 7 estão no mestrado. Estes 14 alunos foram divididos nos grupos A e B.

O principal objetivo da primeira fase foi determinar a experiência dos participantes em qualquer linguagem de modelagem. Em relação a experiência em

modelagem todos possuem pelo menos dois anos. Especificamente, apenas 2 participantes com 5 anos de experiência (alunos de doutorado), 3 participantes com 4 anos (um aluno de doutorado e 3 de mestrado), 3 participantes (um aluno de doutorado e um de mestrado) com 3 anos e 6 participantes (alunos de mestrado) com 2 anos de experiência.

A Figura 22 mostra os participantes com relação ao conhecimento: (i) na linguagem de modelagem UML; (ii) em sistemas multiagentes; e (iii) na linguagem de modelagem MAS-ML. Com relação a UML, foram 4 participantes com alto conhecimento, 6 participantes com médio conhecimento e 4 participantes com baixo conhecimento, não teve participante sem nenhum conhecimento. Sobre o conhecimento em sistemas multiagentes, a maioria tem pelo menos conhecimento médio, mais especificamente 8 participantes e com baixo conhecimento foram 6 participantes. Já para o conhecimento sobre a linguagem de modelagem MAS-ML, nenhum participante tinha alto conhecimento, 5 participantes tinham médio, 6 participantes com baixo e 3 com nenhum conhecimento.

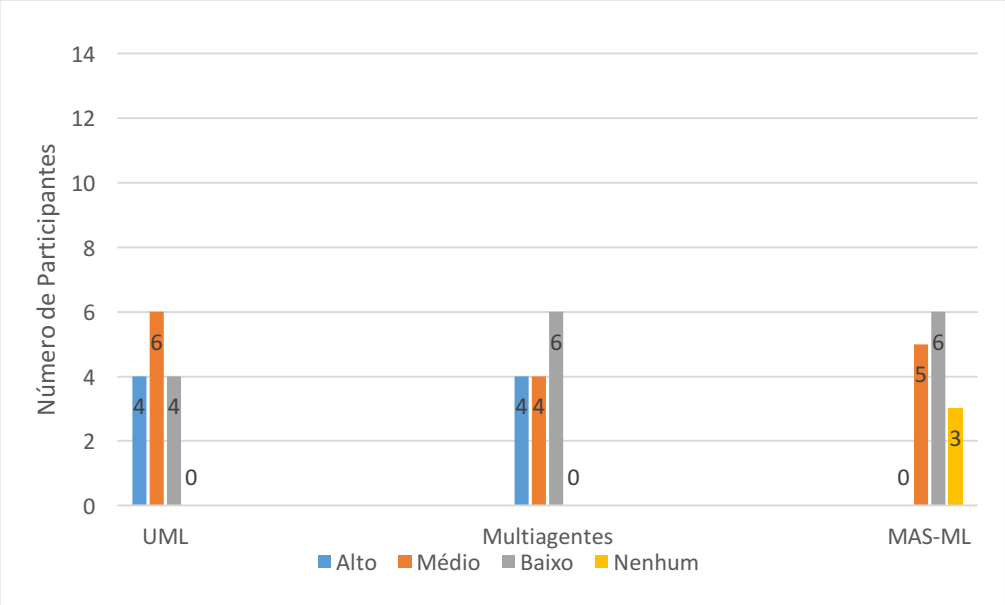

Figura 22 - Perfil dos participantes

6.2.2 Análise dos Dados e Resultados das Atividades

Como mencionado anteriormente, os participantes foram divididos em dois grupos, A e B. Para esses grupos foram aplicados o mesmo questionário que possui

quatro atividades. Porém, quando um grupo fazia uma etapa com MAS-ML, o outro grupo fazia a mesma etapa com ANA-ML. Desta forma, proporcionando a comparação dos resultados da mesma etapa quando é realizada com diferentes linguagens de modelagem, ou seja, uma comparação horizontal dos grupos, A e B, na primeira etapa e outra na segunda etapa, com o objetivo de mitigar o desbalanceamento dos grupos no resultado. Assim, como comparação de etapas diferentes para cada um dos grupos, ou seja, uma comparação vertical das etapas, primeira e segunda, no Grupo A e outra no Grupo B, a qual tem objetivo de mitigar a influencia da complexidade das etapas no resultado. Desta forma, utilizando o *Quadrado Latino* apresentado anteriormente para mitigar o viés da utilização da ANA-ML tanto para o grupo quanto para as etapas.

Neste contexto, para auxiliar na avaliação estatística utilizou a linguagem de programação R. Com isso, calculou inicialmente as médias das métricas por métrica, ou seja, para métrica de taxa de erro, tempo e dificuldade, ilustrada na Tabela 3. Em seguida, procurou verificar qual o teste de hipótese aplicar, com este calculou o *p-value* para a hipótese. O intervalo de valor de confiança de 95% foi definido, como geralmente é utilizado.

Tabela 3 - Média das Métricas

	A			B		
	Erro	Tempo	Dificuldade	Erro	Tempo	Dificuldade
Primeira Etapa	48,75%	31,685	4,435	23%	28,165	2,745
Segunda Etapa	27%	26,21	2,785	42,45%	30,16	4,08

Análise Horizontal – Primeira Etapa

A análise horizontal da primeira etapa é realizada entre os grupos A e B na primeira etapa. O resultado de teste estatístico de Shapiro-Wilk para taxa de erro, retornou o *p-value* = 0,1083. Com isso, refutando a hipótese ***H0***, de que A e B são iguais em relação a taxa de erro. Por conseguinte, foi feita a comparação das médias das taxas de erro dos grupos. Dessa forma, obteve-se indícios para as suposições: (i) o grupo A tem uma taxa de erro maior do que o grupo B; (ii) a linguagem de

modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Para o tempo de execução, o resultado do teste estatístico de Shapiro-Wilk retornou o *p-value* = 0.8421. Com isso, refutando a hipótese ***H0***, de que A e B são iguais em relação ao tempo. Por conseguinte, foi realizada uma comparação da média gasta entre os grupos A e B, assim obteve as seguintes suposições: (i) o grupo A executa em um tempo maior do que o grupo B; (ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Já para o nível de dificuldade de realizar as tarefas, o resultado do teste estatístico de Shapiro-Wilk retornou o *p-value* de 0,0009605, o que significa que ANOVA não poderia ser utilizado para o teste da hipótese nula. Por conseguinte, foi utilizado um teste não paramétrico de Kruskal-Wallis. Como a *chi-squared* = 17.266, *degree-of-fredom* = 1 e *p-value* = 3.25e-05, o teste Kruskal-Wallis indicou que existe uma significante diferença entre as técnicas investigadas com relação aos termos de corretude, podendo assim, rejeitar a hipótese nula, ***H0***, de que os grupos A e B são iguais em relação ao nível de dificuldade percebida. Por conseguinte, a comparação das médias obteve as seguintes suposições: (i) o grupo A teve um nível de dificuldade maior do que o grupo B; e (ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

Análise Horizontal – Segunda Etapa

Para a análise horizontal da segunda etapa, também foi realizada entre os grupos A e B. Comparando as médias das taxas de acerto dos grupos. Dessa forma, obteve-se indícios para as suposições: (i) o grupo A tem uma taxa de erro menor do que o grupo B; (ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Com relação ao tempo de realização das tarefas, foi comparado a média gasto entre os grupos A e B, assim obteve as seguintes suposições: (i) o grupo A executa em um tempo menor do que o grupo B; (ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Já para o nível de dificuldade de realizar as tarefas, ao comparar as médias obteve as seguintes suposições: (i) o grupo A teve um nível de dificuldade menor

do que o grupo B; e (ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

Análise Vertical – Grupo A

Quando a análise é feita de forma vertical entre a primeira e segunda etapas no grupo A, comparou-se a média da taxa de erros entre a primeira e segunda etapas, obteve-se as seguintes suposições: (i) os participantes possuem uma taxa de erro maior na primeira etapa do que na segunda; e (ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Com relação ao tempo de realização das tarefas, foi comparado a média gasto entre a primeira e segunda etapa apresentados na Tabela 3. Assim, obteve as seguintes suposições: (i) os participantes executam em um tempo maior a primeira etapa em relação a segunda etapa; e (ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Já para o nível de dificuldade de realizar as tarefas, ao comparar as médias entre a primeira e segunda etapa, obteve as seguintes suposições: (i) os participantes percebem um nível de dificuldade maior na primeira etapa em relação a segunda etapa; e (ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

Análise Vertical – Grupo B

Quando a análise é feita de forma vertical entre a primeira e segunda etapas no grupo B, comparou-se a média da taxa de erros entre a primeira e segunda etapas, obteve-se as seguintes suposições: (i) os participantes possuem uma taxa de erro menor na primeira etapa do que na segunda; e (ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Com relação ao tempo de realização das tarefas, foi comparado a média gasta entre a primeira e segunda etapa apresentados na Tabela 3. Assim, obteve as seguintes suposições: (i) os participantes executam em um tempo menor a primeira

etapa em relação a segunda etapa; e (ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Já para o nível de dificuldade de realizar as tarefas, ao comparar as médias entre a primeira e segunda etapa, obteve as seguintes suposições: (i) os participantes percebem um nível de dificuldade menor na primeira etapa em relação a segunda etapa; e (ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

Análise dos Resultados do Quadrado Latino

A partir das análises horizontal e vertical do *Quadrado Latino* obteve-se indícios para algumas suposições. A Tabela 4 mostra as suposições para a métrica de taxa de erros na análise horizontal. Consegue-se observar que: (i) a suposição na linha 1 é refutada pela suposição da linha 3; e (ii) as suposições das linhas 2 e 4 são equivalentes, ou seja, a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Tabela 4 - Suposição da taxa de erro horizontal

Primeira Etapa	1	(i) o grupo A executa com uma taxa de erro maior do que o grupo B;
	2	(ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.
Segunda Etapa	3	(i) o grupo A executa com uma taxa de erro menor do que o grupo B;
	4	(ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

A Tabela 5 mostra as suposições para a métrica de tempo na análise horizontal. Pode-se observar que (i) a suposição da linha 1 é refutada pela suposição da linha 3; e (ii) as suposições das linhas 2 e 4 são equivalentes, ou seja, o uso da linguagem

de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Tabela 5 - Suposições de tempo horizontal

Primeira Etapa	1	(i) o grupo A executa em um tempo maior do que o grupo B;
	2	(ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.
Segunda Etapa	3	(i) o grupo A executa em um tempo menor do que o grupo B;
	4	(ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

A Tabela 6 mostra as suposições para a métrica de nível de dificuldade da análise horizontal. Assim, é possível observar que: (i) a suposição da linha 1 é refutada pela suposição da linha 3; e (ii) que as suposições das linhas 2 e 4 são equivalentes, isto é, o uso da linguagem de modelagem ANA-ML diminuiu o nível de dificuldade em realizar as tarefas.

Foi analisado até o momento de forma horizontal as suposições restantes, mitigando o viés dos grupos, taxa de erro, tempo de execução e nível de dificuldade percebido pelos participantes. Agora será analisado as suposições geradas de forma vertical no *Quadrado Latino*.

Tabela 6 - Suposições de nível de dificuldade horizontal

Primeira Etapa	1	(i) o grupo A teve um nível de dificuldade maior do que o grupo B;
	2	(ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.
Segunda Etapa	3	(i) o grupo A teve um nível de dificuldade menor do que o grupo B;
	4	(ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

A Tabela 7 mostra as suposições para a métrica de taxa de erros na análise vertical. Consegue-se observar que: (i) a suposição na linha 1 é refutada pela suposição da linha 3; e (ii) as suposições das linhas 2 e 4 são equivalentes, ou seja,

a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Tabela 7 - Suposição da taxa de erro vertical

Grupo A	1	(i) os participantes possuem uma taxa de erro maior na primeira etapa do que na segunda;
	2	(ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.
Grupo B	3	(i) os participantes possuem uma taxa de erro menor na primeira etapa do que na segunda;
	4	(ii) a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro.

Já a Tabela 8 mostra as suposições para a métrica de tempo na análise vertical. Pode-se observar que (i) a suposição da linha 1 é refutada pela suposição da linha 3; e (ii) as suposições das linhas 2 e 4 são equivalentes, ou seja, o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

Tabela 8 - Suposições de tempo vertical

Grupo A	1	(i) os participantes executam em um tempo maior a primeira etapa em relação a segunda etapa;
	2	(ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.
Grupo B	3	(i) os participantes executam em um tempo menor a primeira etapa em relação a segunda etapa;
	4	(ii) o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor.

A Tabela 9 por sua vez, mostra as suposições para a métrica de nível de dificuldade da análise vertical. Assim, é possível observar que: (i) a suposição da linha 1 é refutada pela suposição da linha 3; e (ii) que as suposições das linhas 2 e

4 são equivalentes, isto é, o uso da linguagem de modelagem ANA-ML diminuiu o nível de dificuldade em realizar as tarefas.

Tabela 9 - Suposições de nível de dificuldade vertical

Grupo A	1	(i) os participantes percebem um nível de dificuldade maior na primeira etapa em relação a segunda etapa;
	2	(ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.
Grupo B	3	(i) os participantes percebem um nível de dificuldade menor na primeira etapa em relação a segunda etapa;
	4	(ii) o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas.

Após a análise do *Quadrado Latino* de forma horizontal e vertical, serão analisadas as suposições restantes. Com relação a métrica de taxa de erros, restaram na Tabela 4 as suposições das linhas 2 e 4 e na Tabela 7 as suposições da linha 2 e 4. Pode-se observar que são equivalentes, logo a primeira conclusão é que a linguagem de modelagem ANA-ML auxilia na construção de agentes com características adaptativas e normativas com uma menor taxa de erro. Para a métrica de tempo, restaram na Tabela 5 as suposições da linha 2 e 4, e na Tabela 8 as suposições da linha 2 e 4. Pode-se observar que são equivalentes, logo a segunda conclusão é que o uso da linguagem de modelagem ANA-ML ajuda os participantes a executar a tarefa em um tempo menor. E por fim, para a métrica de percepção de dificuldade, restaram na Tabela 6 as suposições da linha 2 e 4 e na Tabela 9 as suposições da linha 2 e 4. Pode-se observar que são equivalentes, logo a terceira conclusão é que o uso da linguagem de modelagem ANA-ML diminui o nível de dificuldade em realizar as tarefas. Com isso pode-se dizer que as hipóteses apresentadas no início da seção 5.7 são verdadeiras, pois a taxa de erro, tempo de execução e dificuldades das tarefas foram menores em relação a linguagem de modelagem MAS-ML.

Na Tabela 10 foram descritos os erros encontrados nas respostas dos participantes do estudo. Os erros mais comuns no na utilização de MAS-ML foi a falta de abstrações que permitissem a modelagem de normas, além de como essa normas eram entendidas internamente aos agentes. Já nos modelos de ANA-ML, os

erros mais comuns foram a falta de alguma característica dos conceitos de normas e / ou adaptação.

Tabela 10 - Tipos de erros identificados avaliações aplicadas

(Uso da MAS-ML e técnicas UML)	(Uso da ANA-ML)
Problema do uso da sintaxe de UML-ML.	Problema do uso da sintaxe de UML.
Não modelou algum ponto solicitado.	Não modelou algum ponto solicitado.
Uso de comentários que não são claros para prover algum tipo de informação nos diagramas.	Erro no uso da sintaxe de ANA-ML.
Não seguiu um padrão nos modelos produzidos, gerando diferentes resultados para situações similares.	Modelou de forma errada alguma informação, como por exemplo, a norma ou características de adaptação.
Erro de uso na sintaxe de MAS-ML	
Modelou de forma errada alguma informação, como por exemplo, a norma ou características de adaptação.	

Os participantes consideraram mais fácil o uso da ANA-ML do que a MAS-ML original, ambas atreladas a UML. Uma das principais razões foi que MAS-ML não manipula diversas informações de adaptação e normas solicitadas para modelar. Além disso, alguns participantes com maior entendimento do conceito de adaptação e normas considerou a abordagem ANA-ML mais fácil e intuitiva de trabalhar. A seguir há alguns exemplos de frases mencionadas pelos participantes durante a entrevista após a aplicação dos questionários:

"*Com ANA-ML foi possível criar normas e entender o processo de adaptação, apesar de ficar com algumas dúvidas nas características dessas abstrações*."

"*Percebi que para algumas solicitações ficam difíceis de serem realizadas apenas com os recursos disponíveis, ou seja, recursos demostrados na apresentação do MAS-ML*."

"*Os problemas que tive na primeira etapa da modelagem com MAS-ML foram resolvidos com o uso dos modelos disponíveis pela ANA-ML para modelar normas e o processo adaptativo do agente*."

6.3 Possíveis Ameaças

Esta seção discute as restrições ao estudo. Para cada categoria, foi listado as possíveis ameaças e o que foi feito para contorná-las.

Conclusão da validação. O maior risco nesse experimento foi o engajamento dos participantes para realização do experimento, devido a resolução dos questionários (quase duas horas para cada participante). No entanto, houve uma rotação da ordem de abordagens, dado que tinha sido adotado o quadrado latino. Outra ameaça foi a heterogeneidade dos participantes. Não foi tomado nenhum cuidado especial para selecionar os participantes e assim eles podem representar escolhas aleatórias. Embora a heterogeneidade dos sujeitos possa também ser considerada uma ameaça à validade da conclusão, ela ajuda a promover a validade externa do estudo. Finalmente, a qualidade dos instrumentos investigados é também um risco para a validade da conclusão. No entanto, não se observou erros que dificultaram a compreensão das especificações ou forçaram os participantes a gastar mais tempo respondendo a uma pergunta.

Validade da construção. Foram identificadas as seguintes ameaças à validade de construção: perguntas confusas e sessão de treinamento insuficiente. Para minimizar esses problemas, respondemos às perguntas dos participantes à medida em que estavam emergindo. Para evitar a polarização dos resultados da experiência, limitou-se as explicações sobre as funcionalidades da linguagem ao que foi demonstrado durante a sessão de treinamento e sobre as perguntas para quais esclarecimentos eram absolutamente necessários.

Validação Interna. As ameaças internas residem em como foi especificado o conhecimento das linguagens de modelagem com as diferentes técnicas. Assegurou-se que as linguagens comparadas no experimento tenham sido especificadas seguindo os mesmos padrões. Para isso, aplicou-se previamente o questionário para um grupo de três pessoas com conhecimento de engenharia de software baseada em sistemas multiagentes. Estas pessoas não fizeram parte dos grupos que participaram do estudo. Com isso, verificou-se os possíveis imprevistos,

sendo uma forma de testar o nosso estudo. De fato, o tamanho e a complexidade, do cenário exposto no questionário (Ver Apêndice A) foram fatores que influenciaram os resultados.

Validação Externa. O maior risco externo aqui está relacionado com as linguagens de modelagem. As linguagens de modelagem selecionadas podem não representar todas as abstrações de SMAs. Para reduzir esse risco, selecionou duas linguagens de modelagem que são fortemente baseadas nos conceitos e abstrações discutidos nessa tese. No entanto, são necessárias repetições adicionais para outros cenários, para determinar se os achados deste capítulo podem ser generalizados para outros domínios.

6.1 Considerações Finais

Neste capítulo foi apresentada a avaliação da linguagem ANA-ML, que foi baseada na execução de atividades de modelagem. Cada uma das atividades possuía um objetivo especifico. Além disso, essas atividades foram divididas em duas etapas. Foi configurado o *Quadrado Latino* para mitigar a tendenciosidade das etapas e dos grupos. O estudo foi aplicado a alunos da pós-graduação em informática e todos já tinham ou estavam cursando a disciplina de sistemas multiagentes. Com a avaliação realizada pode-se observar que a linguagem de modelagem ANA-ML auxilia na criação de modelos para sistemas multiagentes onde adaptação e normas são necessários. Com isso, foi possível concluir que ANA-ML auxiliou os participantes em três aspectos: (i) alcançar um menor número de erros; (ii) executar em menor tempo; e (iii) reduzir o nível de dificuldade.

No próximo capítulo será apresentado o *framework* JSAN 2.0, qual foi implementado usando os conceitos do metamodelo conceitual ANA (ver capítulo 4).

7
JSAN 2.0: Um *Framework* para Agentes Adaptativos Normativos

O JSAN 2.0 é um *framework* que se baseia na experiência adquirida durante o desenvolvimento do seu antecessor (JSAN 1.0) a fim de fornecer suporte para criação de agentes de software capazes de se adaptar para lidar com normas. A principal diferença entre o JSAN 2.0 e o JSAN está na adição de um conjunto de mecanismos para monitoramento, análise, planejamento, tomada de decisão e efetuação para o processo deliberativo do agente sobre as normas do ambiente no qual o agente reside.

Neste capítulo será apresentada inicialmente a ideia geral do *framework* JSAN e seus respectivos módulos, em seguida a descrição do *framework* JSAN 2.0.

7.1 JSAN 1.0

O JSAN 1.0 (Viana et al., 2015b) é uma extensão do Jason (Bordini et al., 2007) e tem como objetivo oferecer mecanismos que possibilitam o desenvolvimento de agentes capazes de raciocinar sobre normas guiadas por estratégias normativas. A realização de raciocínio normativo no JSAN 1.0 é possibilitada através da implementação do processo de aplicação normativa (Viana et al., 2015c), o qual é composto de 4 atividades. Figura 23 apresenta a sequência de execução das atividades:

Consciência normativa. O agente de software identifica que existem normas ativas no ambiente e, portanto, estas normas estão endereçadas para agentes específicos.

Adoção de normas. Os agentes reconhecem suas responsabilidades através de outros agentes por internalizar as normas onde a responsabilidade deles é especificada.

Deliberação de normas. Em ordem para executar uma norma específica, um agente deve acessar diferentes informações: (i) os objetivos que devem ser

restringidos, satisfazendo os objetivos normativos; e (ii) os objetivos e ações que podem beneficiar dado as recompensas associadas.

Impacto de normas. Após o agente executar uma estratégia para lidar com uma norma, os objetivos dos agentes são atualizados. Em seguida, o ciclo continua e o agente começa a identificar outras normas que devem ser abordadas.

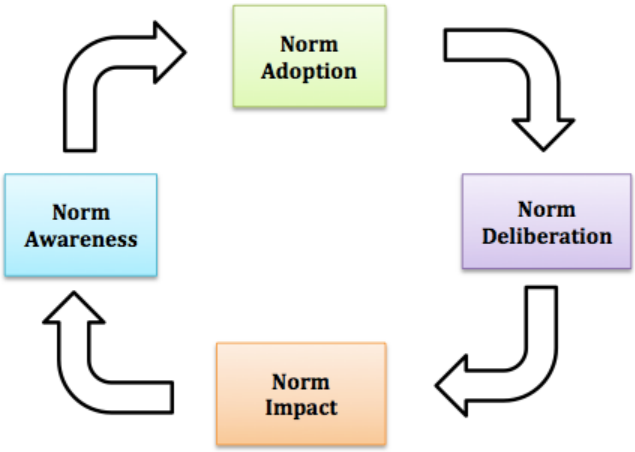

Figura 23 - O processo de estratégia normativa provido pelo *framework* JSAN 1.0

O JSAN 1.0 (i) fornece mecanismos que possibilitam a inserção de normas em um sistema multiagente, (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.

7.1.1 Detalhes do JSAN 1.0

O diagrama de classes apresentado na Figura 24 apresenta as principais classes e métodos do *framework* JSAN 1.0. Os agentes são representados pela classe *NormativeAgent*, a qual é uma extensão da classe *Agent* do Jason (ver Seção 2.5). Além disso, o ambiente de simulação é representado pela classe *EnvironmentSimulation*, que estende a classe *Environment* do Jason e fornece

suporte para desenvolver um ambiente de simulação. O método *executeAction* foi estendido de *Environment* 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 retornar *true* significa que a ação executou com sucesso.

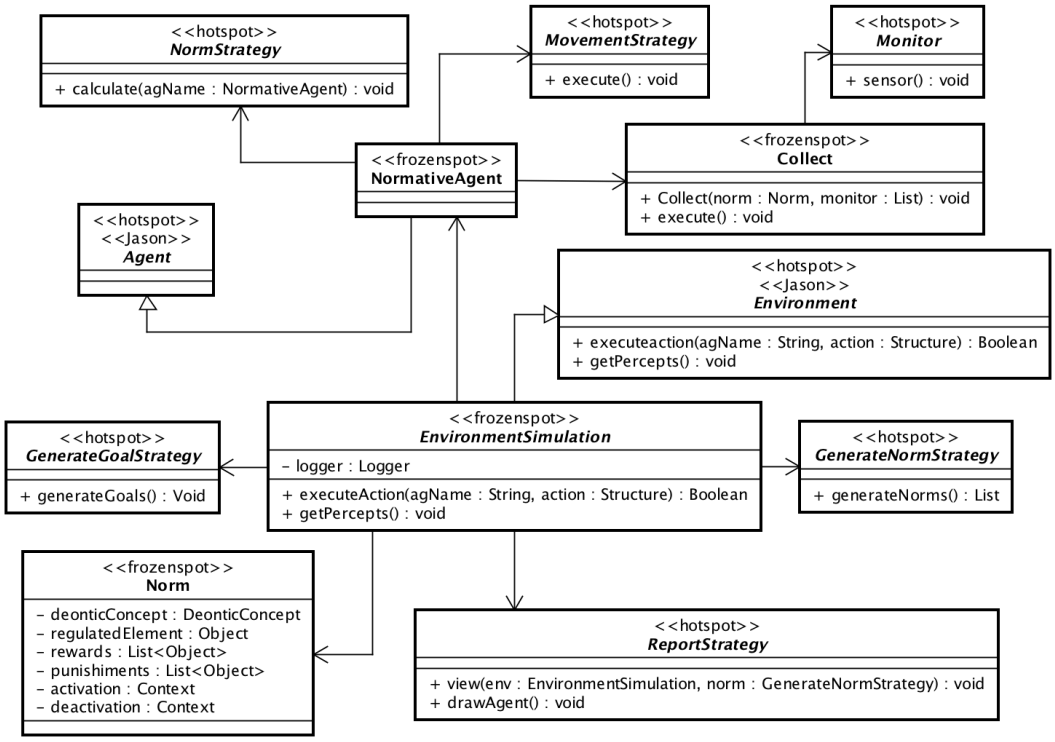

Figura 24 - Classes do JSAN 1.0

A classe *EnvironmentSimulation* também é responsável por gerenciar a criação de normas e objetivos individuais dos agentes, isto através de instâncias das classes *GenerateNormStrategy* e *GenerateGoalsStrategy*, respectivamente. A classe *NormativeAgent* 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 24 pela classe *Norm*. Como discutido na Seção 2.4, uma norma contém um conjunto de sanções, ou seja, recompensas e punições que são representadas através das listas *rewards* e *punishments*, respectivamente. As condições de

ativação e desativação de uma norma são representadas pelos atributos *activation* e *deactivation* (no JSAN 2.0 o nome da variável foi modificado para *expiration*), e o elemento regulado pela norma é definido através do atributo *regulatedElement* (no JSAN 2.0 o nome da variável foi modificado para *state*). 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*.

Adicionalmente, o JSAN 1.0 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*, a qual contém a estratégia utilizada pelo agente para lidar as normas.

Em resumo, para construir uma simulação de sistemas multiagentes normativos utilizando JSAN 1.0 é necessário estender algumas classes. A classe *GenerateNormStrategy* deve ser estendida para gerar as normas que irão existir no ambiente de simulação. Já a classe *GenerateGoalStrategy* é preciso estendê-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. A 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.

7.1.2 Pontos Fixos e Pontos Flexíveis do JSAN 1.0

Sabendo que JSAN 1.0 estende o Jason, o núcleo do Jason é também o núcleo do JSAN 1.0 e os pontos flexíveis do Jason são os pontos flexíveis do JSAN 1.0.

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 *EnvironmentSimulation*): através da classe *EnvironmentSimulation* é possível criar o ambiente onde os agentes irão perceber e agir.

– Dois mecanismos de geração de normas: (i) geração de normas aleatórias dada uma base de dados previamente conhecidas 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*): (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*): (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* é responsável por gerenciar a execução do conjunto de monitores representados pela classe abstrata *Monitor*.

– Três mecanismos para o agente lidar com as normas: (i) *Social*, o agente foca em cumprir 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 à 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: (i) geração de *goals* aleatórios 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: permite acoplar outros algoritmos para gerar normas estendendo a classe abstrata *GenerateNormStrategy*.

– Mecanismos para movimentação dos agentes: diferentes movimentos podem ser incorporados a partir da implementação da classe abstrata *MovementStrategy*.

– Visualização do ambiente de simulação: diferentes visualizações das características particulares de cada cenário de uso podem ser incorporadas a partir da implementação de *ReportStrategy*.

– Monitoramento: mecanismos para monitoramento podem ser adicionados estendendo a classe abstrata *Monitor*.

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

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

7.2 JSAN 2.0

O JSAN 2.0 é uma evolução do *framework* JSAN 1.0 e tem como objetivo fornecer os mecanismos que possibilitam a implementação de agentes adaptativos normativos, isto é, agentes passíveis de adaptação para lidar com as normas vigentes no ambiente em que este reside. Para tanto, JSAN 2.0 possibilita a implementação de agentes capazes de realizar as seguintes tarefas relacionadas a um raciocínio adaptativo normativo: (i) monitoramento; (ii) criação de normas no ambiente de simulação e suas propriedades; (iii) implementação de estratégias para movimento no ambiente; (iv) criação de diferentes *control-loops* normativos; e (vi) seleção de melhor plano dentre as disponíveis.

Apesar do JSAN 1.0 ter um processo de estratégia normativa (Figura 23), em nada este se compromete com a realização de adaptação para lidar com normas. Uma vez escolhida a estratégia, essa será utilizada sempre pelo agente. Além disso, não são realizados a detecção e resolução de conflitos nessa primeira versão do *framework* JSAN.

A grande novidade do JSAN 2.0 em frente ao seu antecessor (JSAN 1.0) é a possibilidade de se utilizar uma implementação conceitual do *control-loop* de autoadaptação proposto pela IBM (IBM, 2003). Com isso, torna-se possível atuar sobre normas como descrito abaixo (Figura 25 apresenta a sequência de execução das atividades do JSAN 2.0 onde é importante notar que agora tem-se as atividades de adaptação do agente atuando sobre um ambiente normativo):

Collect: Utiliza tecnologias para monitoramento do ambiente com o objetivo de extrair informações do ambiente que sirvam de base para detecção de mudanças e autoadaptação. Além disso, estrutura tais informações de forma que elas possam ser entendidas pelas atividades seguinte;

Analyze: Esta fase se baseia nas informações coletadas na atividade *Collect* e utiliza um conjunto de mecanismos de raciocínio para detecção de problemas e raciocínio sobre qual será sua contribuição caso viole ou cumpra uma norma;

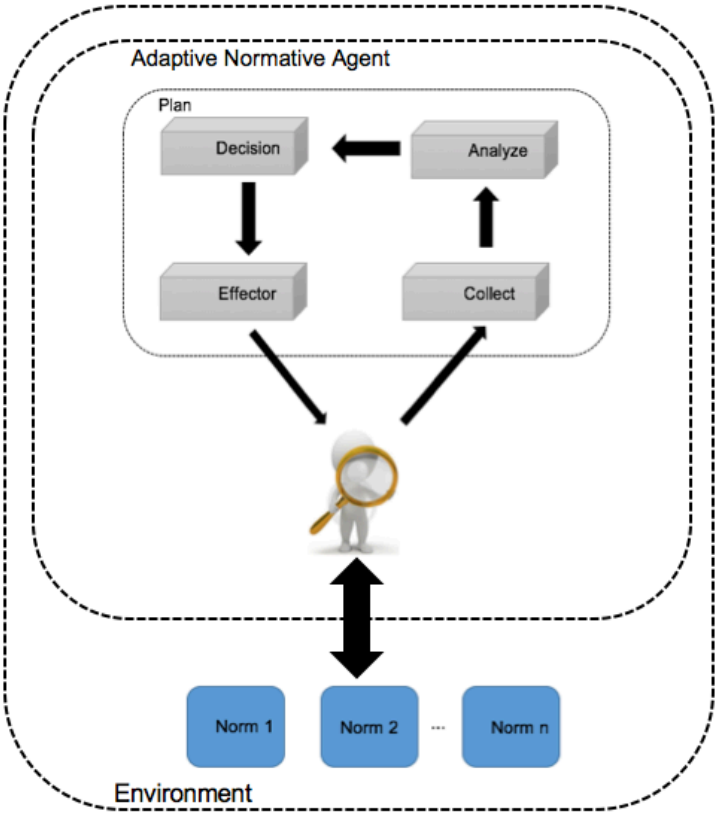

Figura 25 - *Control-loop* JSAN 2.0

Decision: Esta atividade baseia-se na execução de um conjunto de mecanismos de seleção para escolher, dentre os vários planos disponibilizadas pela atividade *Analyze*, a estratégia que tem os melhores planos para o agente decidir cumprir ou violar a norma que está sendo deliberada;

Effector: Realiza as notificações e configurações necessárias para disponibilização da estratégia selecionada pela atividade *Decision*.

A seguir, foi proposta uma arquitetura mostrando o raciocínio de um agente ANA, onde é possível verificar a relação entre os conceitos de adaptação, normas e raciocínio prático (BDI).

7.2.1 Diagrama da Arquitetura ANA-BDI

A arquitetura ANA-BDI proposta neste trabalho estende o modelo BDI (ver Seção 2.2). A Figura 26 mostra o funcionamento interno de um agente adaptativo normativo gerado através desta arquitetura. Observe que as normas e os agentes estão no ambiente, e a autoadaptação acontece interna ao agente.

A arquitetura foi dividida em dois grandes componentes. O primeiro, *Normative Component*, mostra como as normas influenciam o raciocínio cognitivo do agente (BDI). Já o segundo componente, *Adaptive Component*, mostra como o processo de autoadaptação entende as normas e como o processo de deliberação do agente será realizado para a tomada de decisão do agente em relação às normas.

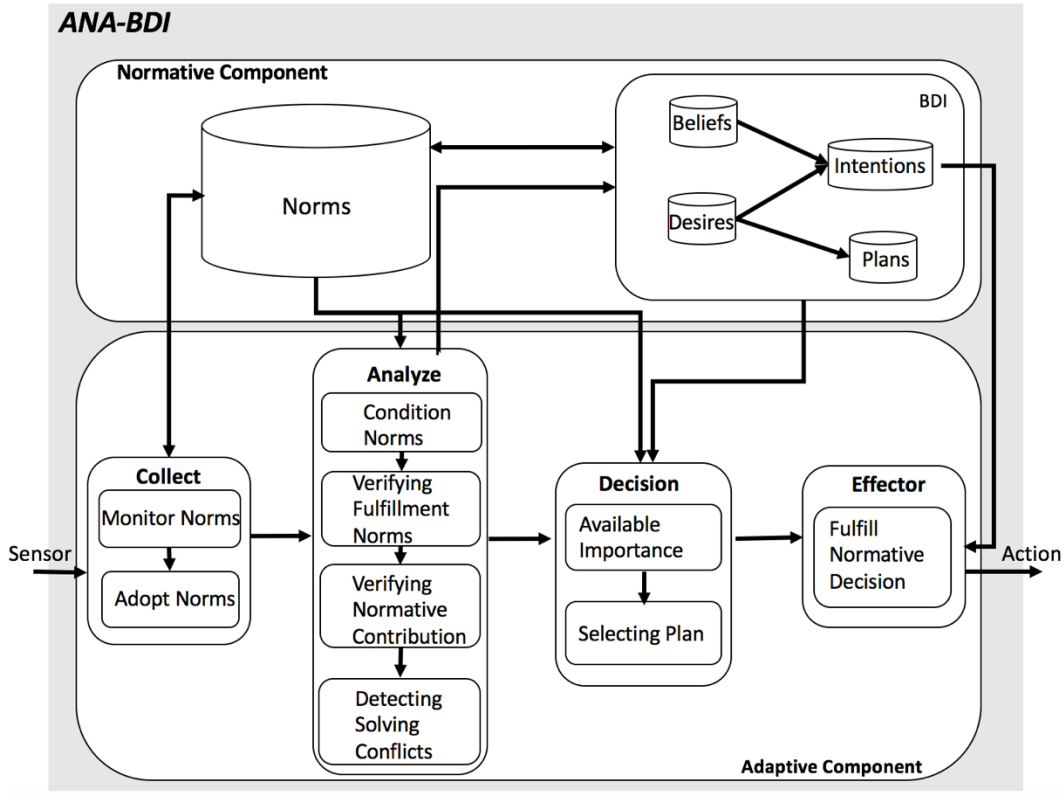

Figura 26 – Arquitetura ANA-BDI

Na fase *Collect*, o agente percebe as normas no ambiente através de seus sensores, os quais são controlados por *Monitor Norms*. Em seguida, o agente

verifica (*Adopt Norms*) se a norma é dirigida ao seu papel e armazena a nova norma em sua base *Norms* (pelo algoritmo 1 da Seção 7.2.1.1) caso ela ainda não exista. Essa tarefa funciona como a função *Belief Revision Function* descrita no modelo BDI (ver Seção 2.2). Após adicionar as normas na base *Norms*, suas crenças e desejos são atualizados em relação a essas normas.

Em seguida, começa a fase *Analyze*, onde é verificado (*Condition Norms*) se as normas estão ativas ou se já expiraram (algoritmo 2 da Seção 7.2.1.2), considerando que algumas se tornem ativas ou expirem dado o cumprimento ou violação de outras normas; posteriormente é verificado se essas normas já foram cumpridas ou violadas (*Verifying Fulfillment Norms*) dado a diferentes operações realizadas pelo agente (algoritmo 3 da Seção 7.2.1.2); é verificada a contribuição normativa de cada norma (*Verifying Normative Contribution* (algoritmo 4 da Seção 7.2.1.2)) considerando os benefícios e punições atrelados à norma, além do seu conceito deôntico; e a detecção e resolução de conflitos (*Detecting Solving Conflicts* (algoritmo 5 da Seção 7.2.1.2))). Após filtrar o conjunto de normas que o agente tem a intenção de cumprir, serão gerados novos desejos do agente a partir de suas crenças. Essa tarefa funciona como a função *Option Generation Function* descrita no modelo BDI (ver Seção 2.2).

Já na fase *Decision*, *Available Importance* fica responsável por calcular a importância de um agente realizar um determinado objetivo ou ação regulada pela norma (algoritmo 6 da Seção 7.2.1.3); e posteriormente, *Selecting Plan*, por fazer a seleção dos planos (*Plan*) do agente para cumprir ou violar com a norma (algoritmo 7 da Seção 7.2.1.3). Essa tarefa funciona como a função *Filter* descrita no modelo BDI (ver Seção 2.2).

Por fim, na fase *Effector*, *Fulfill Normative Decision* é responsável por executar as decisões normativas. Isso acontece dado o conjunto de intenções do agente (*Intentions*) que foram filtradas do conjunto de desejos (*Desires*) dadas as crenças do agente (*Beliefs*), ambos influenciados pelas normas (algoritmo 8, da Seção 7.2.1.4). Essa tarefa funciona como a função *Action Selection Function* descrita no modelo BDI (ver Seção 2.2).

7.2.2 Detalhes do JSAN 2.0

O diagrama de classes apresentado na Figura 27 apresenta as principais classes e métodos do *framework* JSAN 2.0. Esta ferramenta foi desenvolvida baseada na arquitetura ANA-BDI (ver Seção 7.2.1). No JSAN 2.0 os agentes adaptativos normativos são representados pela classe *AdaptiveNormativeAgent* e a grande novidade, o *control-loop* de adaptação pela classe *AdaptationControlLoop*.

Como mencionado acima, o JSAN 2.0 também oferece o *control-loop* de autoadaptação baseado no proposto pela IBM, o qual é representado pela classe *ControlLoop* onde está definida a ordem de execução das atividades: *Collect*, *Analyze*, *Decision* e *Effector*. Para utilizar o *control-loop* citado acima é necessário instanciar a classe *ControlLoop*, passando como parâmetros as instâncias das classes que representam as atividades que o compõem.

Em suma, para implementar um agente capaz de se adaptar para lidar com normas utilizando o JSAN 2.0 é necessário estender a classe *AdaptiveNormativeAgent* passando os seguintes parâmetros: (i) normas endereçadas a ele; (ii) dados coletados a partir do monitoramento de eventos (iii) implementar o método *createSequence* (Ver classe *AdaptiveNormativeAgent*) o qual retorna os comportamentos que o agente deve executar, incluindo o *control-loop* de *autoadaptação*.

Para facilitar o entendimento das atividades do *control-loop*, estas são explicadas em mais detalhes nas seções seguintes, junto com seus respectivos diagramas de classes.

Além disso, serão detalhados todos os algoritmos utilizados, os quais foram propostos na arquitetura ANA-BDI.

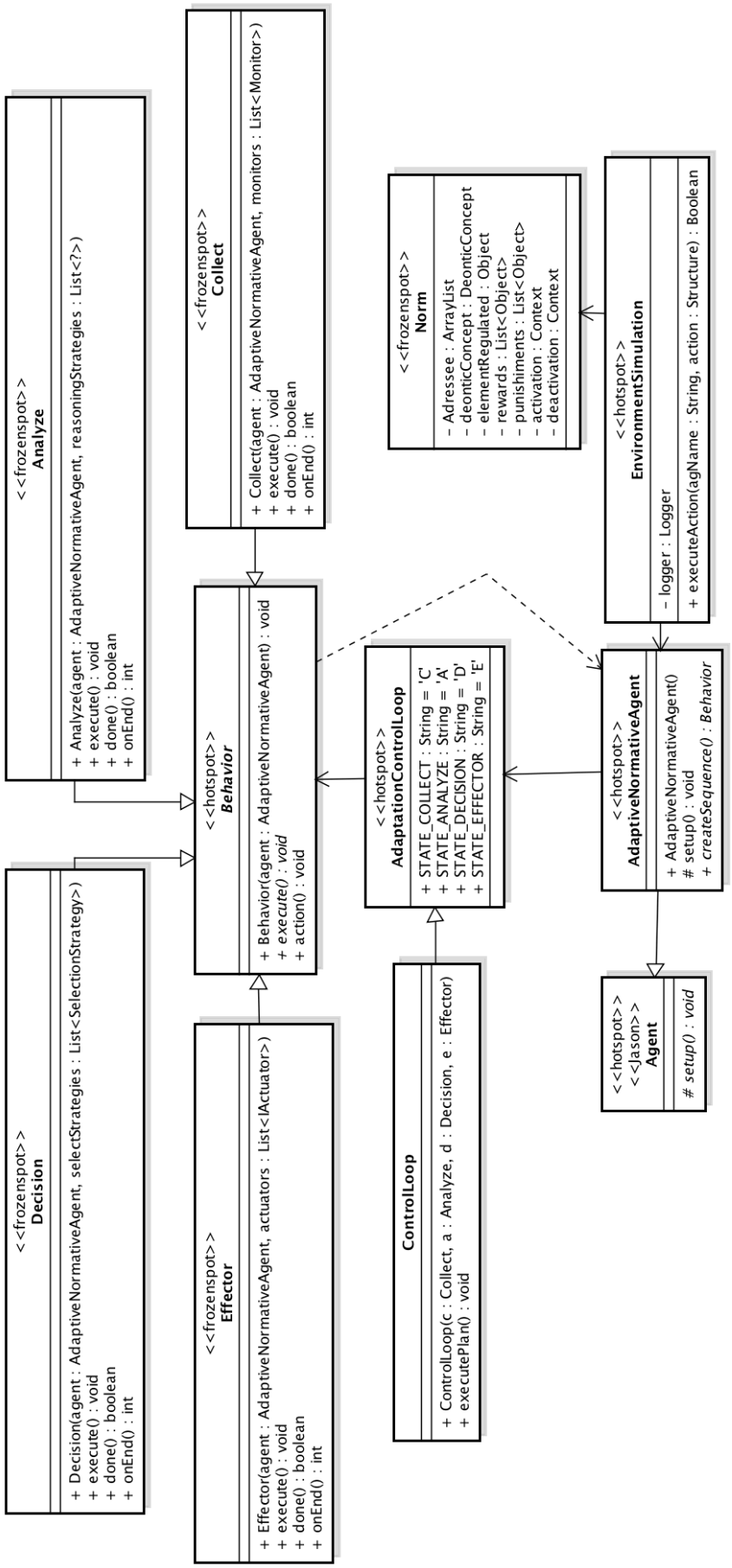

Figura 27 – Classes do JSAN 2.0

7.2.2.1 Collect

Este é o primeiro passo executado pelo *control-loop* provido pelo JSAN 2.0. Ele é responsável por monitorar o ambiente em que o agente reside através de sensores e, por fim, filtrar e estruturar os dados coletados a partir de tais tarefas de forma que eles possam ser entendidos pelas atividades seguintes.

A Figura 28 apresenta o diagrama de classes da atividade *Collect* onde tem-se a classe *Collect* responsável por gerenciar a execução do conjunto de monitores representados pela classe abstrata *Monitor*, a qual possui métodos abstratos *sensor* representando uma importante tarefa no que diz respeito ao monitoramento:

sensor – No *sensor* deve ser especificado o que monitorar e quais dados devem ser coletados durante o monitoramento. Neste caso, o JSAN 2.0 oferece: um mecanismo (classe *Monitor*) que coleta as normas vigentes no ambiente.

O algoritmo 1, de adoção de normas está dentro da classe *NormMonitor*, desenvolvido na função *sensor*, ele é responsável por identificar as normas endereçadas a cada agente, papel ou grupo de agentes, as quais especificam suas responsabilidades.

Algoritmo 1 - Adoção de Normas
1: **Requisito**: Normas identificadas no ambiente, representadas por *sensednorms* 2: **Requisito**: Normas adotadas, representadas por *adoptednorms* 3: **Requisito**: Nome do Agente, representado por *name* 4: **Requisito**: Papéis assumidos pelo Agente, representados por *roles* 5: **Requisito**: Grupos de que o Agente faz parte, representados por *groups* 6: **PARA TODA** Norma em *sensednorms* **FAÇA** 7: **SE** não existem normas em *sensednorms* 8: **SE** está endereçada (*name*, *roles*, *groups*, *norma.addressees*) **ENTÃO** 9: Norma é adicionada em *adoptednorms* 10: **FIM de SE** 11: **FIM de SE** 12: **FIM de PARA TODA**

A operação tem início a partir do conjunto de normas identificadas pelo agente. Para cada norma identificada, as seguintes verificações são realizadas: (i) se a norma ainda não existe na base de normas adotadas; e (ii) se a norma é endereçável ao agente. Se tais condições são satisfeitas, a norma é adicionada ao conjunto de normas adotadas.

Após os monitores finalizarem sua execução, a atividade *Analyze* inicia sua execução, visando detectar quais normas serão cumpridas e/ou violadas pelo agente e qual a melhor forma de realizar essa tarefa.

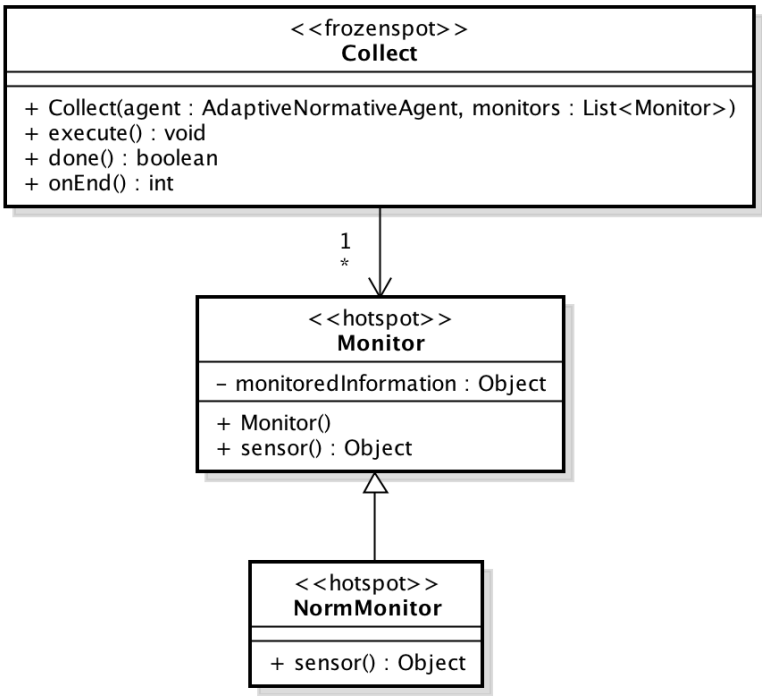

Figura 28 – Atividade Collect

7.2.2.2 Analyze

A atividade *Analyze*, representada pelo diagrama de classes da Figura 29, utiliza métodos de raciocínio a fim de analisar os dados coletados, detectar problemas e sugerir soluções para a atividade *Decision*.

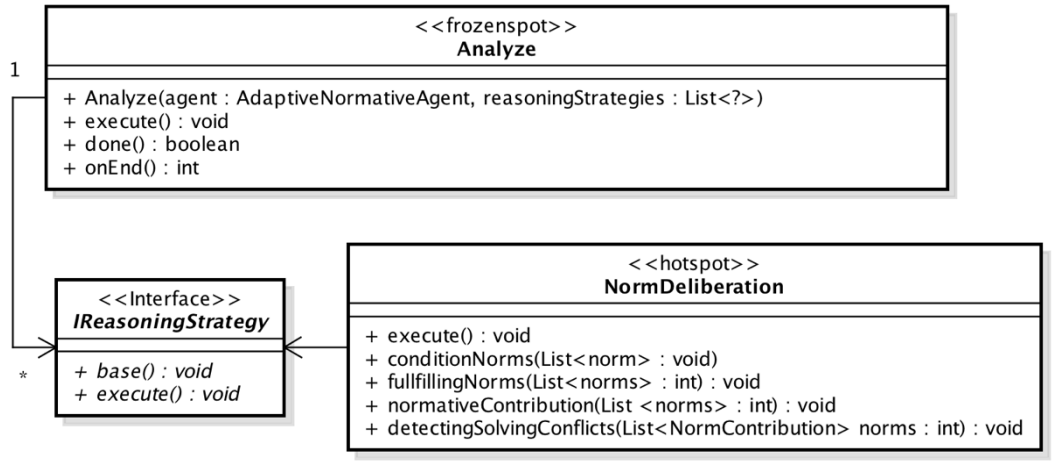

Figura 29 – Atividade Analyze

A classe *Analyze* é responsável por gerenciar a execução do conjunto de raciocínios apresentado pela interface *IReasoningStrategy*, a qual possui o método base, onde deve ser definida a base de conhecimento, a qual foi implementada apenas para lidar com normas, e que servirá como base para o seu respectivo raciocínio, o método execute, onde a lógica de execução de cada raciocínio deve ser especificada.

Algoritmo 2 - Verificar se a norma está ativa ou expirou
1: **Requisito**: Base de Crenças do agente, representada por *beliefs* 2: **Requisito**: Normas Adotadas, representadas por *adoptednorms* 3: **Requisito**: Normas Ativadas, representadas por *actnorms* 4: **Requisito**: Normas Expiradas, representadas por *expnorms* 5: **PARA TODA** norma *normX* em *adoptednorms* faça 6: **SE** verificar contexto (normX.ativa, *beliefs, actnorms, expnorms*) então 7: Adiciona *normX* em *actnorms* 8: **FIM de SE** 9: **FIM de PARA TODA** 10: **PARA TODA** norma *normY* em *actnorms* faça 11: **SE** verificar contexto (*normY.expirou*, *beliefs, actnorms, expnorms*) então 12: Remove *normY* de actnorms 13: Adiciona *normY* em *expnorms* 14: **FIM de SE** 15: **FIM de PARA TODA**

A função *conditionNorms* (Ver Figura 29), representada no algoritmo 2, após uma norma ser coletada por um monitor da atividade *Collect*, deve-se checar se a norma ainda está ativa (linhas 5 a 9) ou se expirou (linhas 10 a 15). Como pode-se ver no algoritmo abaixo, é verificado se o contexto de ativação é satisfeito, então a norma é adicionada ao conjunto de normas ativas. Porém, se o contexto de que a norma expirou é satisfeito, a norma é removida do conjunto de normas ativas.

Após essa etapa é necessário checar e atualizar o conjunto das normas que já foram cumpridas ou violadas. Para isso, foi desenvolvida a função *fullfillingNorms* (Ver Figura 29), que será explicada em detalhes no algoritmo 3, abaixo:

Caso a norma esteja ativa e tenha tido seu comportamento realizado, se ela é de obrigação, foi cumprida, seu status de cumprimento é atualizado para cumprida (linhas 6 a 8). Se a norma é de permissão, foi cumprida, seu status de cumprimento é atualizado para cumprida (linhas 10 a 12).

Algoritmo 3 - Verificar o cumprimento de normas

1: **Requisito**: Normas Ativadas *actnorms*
2: **Requisito**: Normas Expiradas, representadas por *expnorms*
3: **Requisito**: Se a norma foi cumprida ou violada,
4: representada pelo conjunto *normfulfillment*
5: **PARA TODA** norma *normX* em *actnorms* faça
6: **SE** comportamento regulado por *normX* foi realizado então
7: **SE** *deonticConcept* é de *Obligation* então
8: *normX* é adicionada em *normfulfillment* e seu status muda para cumprida
9: **FIM de SE**
10: **SE** *deonticConcept* é de *Permission* então
11: *normX* é adicionada em *normfulfillment* e seu status muda para cumprida
12: **FIM de SE**
13: **SE** *deonticConcept* é de *Prohibition* então
14: *normX* é adicionada em *normfulfillment* e seu status muda para violada
15: **FIM de SE**
16: **FIM de SE**
17: **FIM de PARA TODA**
18: **PARA TODA** norma *normY* em *expnorms* faça
19: **SE** comportamento regulado por *normY* NÃO foi realizado então
20: **SE** *deonticConcept* é de *Obligation* então
21: *normY* é adicionada em *normfulfillment* e seu status muda para violada
22: **FIM de SE**
23: **SE** *deonticConcept* é de *Permission* então
24: *normX* é adicionada em *normfulfillment* e seu status muda para violada
25: **FIM de SE**
26: **SE** *deonticConcept* é de *Prohibition* então
27: *normY* é adicionada em *normfulfillment* e seu status muda para cumprida
28: **FIM de SE**
29: **FIM de SE**
30: **FIM de PARA TODA**

Caso a norma seja de proibição, e tenha sido violada, seu status muda para violada (linhas 13 a 15). Caso a norma esteja expirada e seu comportamento e tenha sido realizado, se ela é de obrigação, foi violada, seu status muda para violada (linhas 20 a 22). Se é uma norma de permissão, foi violada, seu status muda para violada (linhas 23 a 25). E caso a norma seja de proibição, foi cumprida, seu status muda para cumprida (linhas 26 a 28).

Após verificar se a norma está ativa ou expirada no ambiente, será atualizado se esta foi cumprida ou violada, o agente precisa avaliar se é de interesse dele cumprir com as normas endereçadas a ele e que ainda estão ativas. Com esse objetivo foi desenvolvido a função *normativeContribution* (Ver Figura 29), mostrada em detalhes pelo algoritmo 4, para verificar a contribuição normativa, ou

seja, o quanto o agente tem a ganhar por decidir cumprir ou violar uma norma dado o estado

Algoritmo 4 – Avaliação da Contribuição Normativa

1: **Requisito**: Normas Ativas em *actnorms*
2: **Requisito**: Contribuição normativa, representada por *normativeContribution*
3: **Requisito**: *Goals / states* do agente que são restringidos por normas
4: obrigatórias, representado por *existGoalStateObligation*
5: **Requisito**: *Goals / states* do agente que são restringidos por normas
6: proibitivas, representado por *existGoalStatePunishment*
7: **Requisito**: *Goals / states* do agente que são restringidos por normas
8: permissivas, representado por *existGoalStatePermission*
9: **PARA TODAS** normas em *actnorms*
10: **SE** *deonticConcept* é *Obligation*
11: **SE** *existGoalStateObligation*
12: *normativeContribution*:= *normativeContribution* +1
13: **FIM** de **SE**
14: **SENÃO**
15: **SE** *existGoalStatePermission*
16: *normativeContribution*:= *normativeContribution* +1
17: **FIM de SE**
18: **SENAO SE** *existGoalStatePunishment*
19: *normativeContribution*:= *normativeContribution* -1
20: **FIM de SENÃO**
21: **FIM de SENÃO**
22: **FIM de SE**
23: **SE** *deonticConcept* é *Prohibition*
24: **SE** *existGoalStateObligation*
25: *normativeContribution*:= *normativeContribution* -1
26: **FIM de SE**
27: **SE** *existGoalStatePermission*
28: *normativeContribution*:= *normativeContribution* -1
29: **FIM de SE**
30: **FIM de SE**
31: **SE** *Rewards* é igual a *goal* do agente
32: *normativeContribution*:= *normativeContribution* +1
33: **FIM de SE**
34: **SE** *deonticConcept* é *Obligation*
35: **SE** *existGoalPunishiment*
36: *normativeContribution*:= *normativeContribution* – 1
37: **FIM de SE**
38: **FIM de SE**
39: **SE** *deonticConcept* é *Prohibition*
40: **SE** *existGoalPunishiment*
41: *normativeContribution*:= *normativeContribution* – 1
42: **FIM de SE**
43: **FIM de SE**
44: **SE** *deonticConcept* é *Permission*
45: **SE** *existGoalPunishiment*
46: *normativeContribution*:= *normativeContribution* – 1
47: FIM de SE
48: **FIM de SE**

```
49: FIM DE PARA TODOS
50: RETORNAR normativeContribution
```

ou ação restringidos por ela, o seu conceito deôntico e sanções atreladas a essa norma. Uma das grandes diferenças em relação aos trabalhos relacionados é o suporte para a autoadaptação do agente em relação às normas e o cálculo do conceito de permissão na contribuição normativa.

Algoritmo 5 – Detecção e Resolução de Conflitos entre normas

```
1: Requisito: Normas Ativas em actnorms e calculo da normativeContribution
2: PARA TODA normX em actnorms
3:   PARA TODA normY em actnorms
4:   SE (normX é obligation == normY é prohibition) AND
5:       (normX activation <= normY expiration) THEN
6:    SE normativeContribution(normX)>= normativeContribution(normY)
7:       normX será cumprida e normY será violada
8:     FIM de SE
9:     SENÃO
10:         normY será cumprida e normX será violada
11:    FIM de SENÃO
12:  FIM de SE
13:  SE (normX é obligation == normY é permission) AND
14:    (((normX activation < normY activation) AND
15:          (normX expiration >= normY expiration) ) OR
16:      ((normX activation > normY expiration) OR
17:        (normX expiration < normY activation))
18:   ) THEN
19:    SE normativeContribution(normX)>= normativeContribution(normY)
20:       normX será cumprida e normY será violada
21:    FIM de SE
22:    SENÃO
23:       normY será cumprida e normX será violada
24:    FIM de SENÃO
25:  FIM de SE
26:  SE (normX é prohibition == normY é permission) AND
27:     (((normX activation < normY activation) AND
28:          (normX expiration >= normY expiration)) OR
29:      ((normX activation < normY expiration) OR
30:        (normX expiration <normY expiration))
31:   ) THEN
32:    SE normativeContribution(normX)>= normativeContribution(normY)
33:      normX será cumprida e normY será violada
34:    FIM de SE
35:    ELSE
36:      normY será cumprida e normX será violada
37:    FIM de ELSE
38:  FIM de SE
39:  FIM de PARA TODA
40: FIM de PARA TODA
```

Essa função *normativeContribution* ajudará na tomada de decisão do agente posteriormente. Na fase de decisão ele analisará quais são a contribuição normativa de cada norma, a satisfação individual dele de realizar uma determinada ação e/ou a motivação que ele tem para realizar seus objetivos individuais.

Já a função *detectingSolvingConflicts*, detalhada pelo algoritmo 5, é responsável por detectar e resolver conflitos entre normas com conceitos deônticos diferentes e que restringem o mesmo estado ou ação do agente. Caso o conceito deôntico da primeira norma dada a sua contribuição normativa seja maior ou igual a contribuição normativa da segunda norma, a primeira norma é selecionada para ser cumprida e a segunda para ser violada. Caso contrário, a primeira norma é selecionada para ser violada, e a segunda norma para ser cumprida.

7.2.2.3 Decision

A atividade *Decision*, representada pelo diagrama de classes apresentado na Figura 30, irá decidir qual o plano de maior contribuição normativa e importância para ser executado, o qual será enviado para a atividade *Effector*.

Algoritmo 6 – Avaliação da Importância de Cumprir ou Violar a Norma

```
1: Requisito: Lista de planos do agente, representado por plans
2: Requisito: Uma norma Ativa, representadas por activenorm
3: Requisito: A importância de realizar um comportamento, representado
4: por Importance
5: Requisito: A motivação de realizar um objetivo, representado por Motivation
6: Requisito: A satisfação de realizar uma ação, representado por Satisfaction
7: Requisito: Contribuição normativa, representada por normativeContribution
8: PARA TODOS os plan de plans faça
9:  SE action / state restringido por activenorm é igual a GOAL do agente THEN
10:     Importance do comportamento é o valor de importância somado a
11:     Motivation do agente realizar aquele GOAL + normativeContribution
12:  FIM de SE
13  SE action / state restringido por activenorm é igual a ACTION
14: do agente THEN
15:     Importance do comportamento é o valor de importância somado a
16:     Satisfaction do agente realizar aquele GOAL + normativeContribution
17:  FIM de SE
18: RETORNAR Importance
```

A classe *Decision*, responsável por gerenciar a execução do conjunto de mecanismos de seleção representados pela classe abstrata *SelectionStrategy*.

Esta declara o método abstrato *selectSolutions* que recebe o plano selecionado pelo mecanismo de seleção anteriormente aplicado e adiciona a base de eventos do agente para serem executados. A classe *BehaviorImportance* tem os métodos *availableImportance* e *selectSolutions* implementados para avaliar a importância de cumprir ou violar com uma norma (algoritmo 6) e para selecionar o plano a ser realizado (algoritmo 7), respectivamente.

Algoritmo 7 - Selecionar o plano a ser realizado

```
1: Requisito: Lista de planos, representado por plans
2: Requisito: Uma norma Ativa, representadas por activenorm
3: Requisito: A importância de realizar um comportamento,
4:    representado por Importance
5: PARA TODOS os plan em plans i =1 até plans.size faça
6:   SE Importance de plans(i) para activenorm é maior que Importance
7:     de plan(i) para activenorm THEN
8:     plan := plans(i)
9:   FIM de SE
10: FIM de PARA TODOS
11: RETORNAR plan
```

Na tomada de decisão, é necessário avaliar a importância de atingir os objetivos e ações que compõem um plano. Assim, o algoritmo 6 ficou responsável por avaliar a importância de cumprir ou violar uma dada norma, ou seja, se a importância é uma motivação que o agente tem em atingir um determinado objetivo, ou a satisfação que o agente terá em executar uma determinada ação.

Finalmente, no algoritmo 7, o plano com a maior importância será escolhido e adicionado à base de eventos do agente para, posteriormente ser executado na fase de Efetuação.

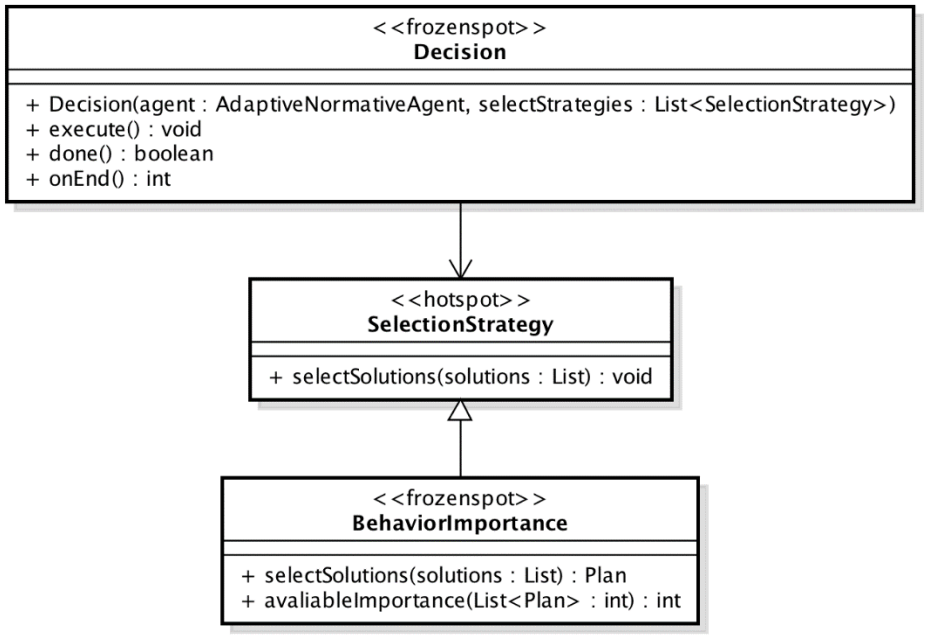

Figura 30 – Atividade Decision

7.2.2.4 Effector

Esta atividade é responsável por efetivar as decisões normativas tomadas, adicionando novos eventos à base de eventos do agente. Como discutido na Seção 2.5, agentes implementados utilizando Jason reagem a eventos cujo *trigger* podem estar representando a adição de objetivos. Desta forma, novos eventos são gerados para representar o desejo do agente em realizar comportamentos regulados por normas de obrigação selecionadas para serem cumpridas, normas de proibição selecionadas para serem violadas e normas de permissão selecionadas para serem cumpridas ou violadas.

Algoritmo 8 - Cumprir as decisões normativas
1: **Requisito**: Normas ativas, representadas por *actnorms* 2: **PARA TODA** norma *normX* em *actnorms* faça 3: **SE** não existe um comportamento que efetive a decisão normativa 4: Adicionar comportamento na base de eventos do agente, 5: sendo estes para cumprir ou violar com *normX* 6: **FIM de SE** 7: **FIM de PARA TODA**

No algoritmo 8, é verificado se já não existe um comportamento que efetive a decisão normativa, se não existe e uma norma de obrigação foi selecionada para ser

cumprida, uma norma de proibição foi selecionada para ser violada e uma norma de permissão foi selecionada para ser cumprida ou violada. Então, um novo evento que representa a adição do comportamento regulado pela norma é gerado e adicionado à base de eventos.

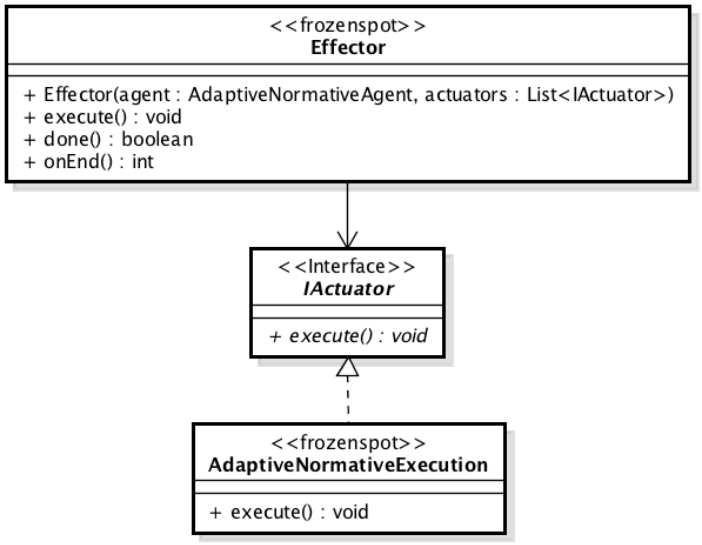

Figura 31 - Atividade Effector

Na Figura 31 tem-se a atividade *Effector* representada pela classe *Effector* que é responsável por gerenciar os atuadores representados pela interface *IActuator*, o método *execute* do atuador *AdaptiveNormativeExecution* foi implementado utilizando o algoritmo 8, com o objetivo de cumprir as decisões normativas escolhidas pelo agente.

7.2.3 Pontos Fixos e Flexíveis

Considerando que o JSAN 2.0 estende o JSAN 1.0, o *kernel* do JSAN 1.0 e do Jason também são o *kernel* do JSAN 2.0 e os hot-spots (ou pontos flexíveis) dos dois são os hot-spots do JSAN 2.0.

Os *frozen-spots* especificamente providos pelo JSAN 2.0 são:

- Um *control-loop* de autoadaptação seguindo a sequência de execução apresentada na Figura 25 e representado pela classe *ControlLoop* (Ver Seção 7.2 para maiores detalhes como instanciá-lo). Tal classe utiliza quatro atividades definidas: *Collect*, *Analyze*, *Decision* e *Effector*.

- Quatro atividades: *Collect* (Seção 7.2.1.1), *Analyze* (Seção 7.2.1.2), *Decision* (Seção 7.2.1.3) e *Effector* (Seção 7.2.1.4).

- Um mecanismo relacionado a atividade *Effector*: (i) realiza a decisão normativa do agente.

Os *hot-spots* especificamente providos pelo JSAN 2.0 são:

- Planos de autoadaptação (classe abstrata *AdaptationControlLoop* na Figura 27): é possível definir novos *control-loops* e a sequência para executar as atividades dos *control-loops*.

- Atividades (classe abstrata *Behavior* na Figura 27): é possível definir novas atividades.

- Monitoramento (classe abstrata *Monitor* na Figura 28): mecanismos para monitoramento podem ser adicionados estendendo a classe abstrata *Monitor*.

- Raciocínios (interface *IReasoningStrategy* na Figura 29): permite acoplar outros algoritmos de raciocínio a partir da implementação da interface *IReasoningStrategy*.

- Técnicas para seleção de estratégias para lidar com normas (classe abstrata *SelectionStrategy* na Figura 30): técnicas de seleção podem ser incorporadas a partir da implementação da classe abstrata *SelectionStrategy*.

- Mecanismos para efetivação da solução (interface *IActuator* na Figura 31): diferentes efetuadores podem ser incorporados a partir da implementação da interface *IActuator*.

7.3 Considerações Finais

Neste capítulo foi apresentado o *framework* JSAN 1.0 e o seu sucessor, o JSAN 2.0. Este último, responsável por permitir a criação de agentes de software autoadaptáveis capazes de lidar com normas. Para que esses agentes pudessem ser criados, o *framework* permite: (i) a criação de diferentes processos de autoadaptação para lidar com normas; (ii) a criação de diferentes atividades, que compõem os processos de autoadaptação; (iii) diferentes estratégias para lidar com normas através do *control-loop*. O JSAN 2.0 procurou implementar a relação entre as informações propostas no modelo conceitual do Capítulo 4 e a arquitetura ANA-

BDI (ver Seção 7.2.1) para permitir aos agentes lidarem com as normas dirigidas a eles.

No próximo Capítulo será apresentado diferentes cenários de uso onde serviu para testar a validade do *framework* JSAN 2.0. Além disso, foi utilizado a linguagem de modelagem ANA-ML para construção dos modelos que representam os cenários.

8
Cenários de Uso

Neste capítulo será apresentada a aplicabilidade da linguagem ANA-ML e do *framework* JSAN 2.0. Esta será feita através de dois cenários de uso. O primeiro cenário está relacionado a missões de resgate de civis que estão em áreas de risco (Viana, 2015b). Já o segundo cenário será aplicado no contexto de mercados virtuais (He et al., 2003; Jennings & Wooldridge, 1999; Lomuscio et al., 2003).

Com o intuito de auxiliar os usuários da ANA-ML e do *framework* JSAN 2.0, definiu-se uma abordagem (Seção 7.1) para facilitar o uso conjunto de tais propostas. Além disso, este capítulo apresenta em detalhe as descrições dos sistemas multiagentes modelados e desenvolvidos em (Seção 7. 2 e Seção 7.3). Por fim, uma discussão é apresentada sobre os exemplos desenvolvidos (Seção 7.5).

8.1
Abordagem Adotada

A abordagem tem como objetivo ajudar o desenvolvedor a trabalhar de forma conjunta com a ANA-ML e o JSAN 2.0. Essa abordagem descreve o que é necessário ser especificado a fim de representar completamente um design por meio de diagramas ANA-ML e criar agentes autoadaptativos capazes de lidar com normas usando o JSAN 2.0.

8.1.1
Diagramas Estruturais

Usando ANA-ML, o desenvolvimento de um sistema deve começar com a identificação das classes de ambiente e da organização principal. A instância da organização principal residirá na instância da classe de ambiente. A descrição da classe da organização principal inclui a definição de seus objetivos, crenças, planos, ações e normas. A descrição da classe de ambiente deverá ser modelada como um agente com objetivos, crenças, planos e ações.

Depois da identificação das classes de ambiente e organização principal, os papéis exercidos na organização principal podem ser identificados. Ao identificar as classes de papel do agente, os objetivos, crenças, deveres, direitos de cada uma dessas classes devem ser definidos. A definição de suas propriedades sofre a influência das normas das classes de organização que definem o papel.

As entidades que exercem os papéis precisam ser identificadas. As classes de papel associadas a uma classe de suborganização ou agente podem influenciar sua definição. Os objetivos das classes de papel estão relacionados aos objetivos das classes de suborganização ou agente. As ações e os planos do agente ou suborganização estão diretamente associados aos deveres, direitos e definidos nas classes de papel. Além do mais, as normas associadas à classe de suborganização estão relacionadas as normas descritas em sua classe de superorganização. As normas de uma classe de suborganização são definidas com base nas normas da classe de superorganização.

Quando um papel do agente é exercido por uma suborganização, deve ser criado um novo diagrama de organização a fim de descrever a organização, os papéis definidos, as entidades que exercem esses papéis e os ambientes em que residem suas instâncias. Para cada classe de suborganização definida no sistema, deve ser criado um diagrama de organização. Durante a definição de um diagrama de organização, só é importante definir as propriedades das entidades e os papéis que exercem. Os relacionamentos entre essas entidades são modelados no diagrama de classes.

Nem todas as classes são modeladas em diagramas de papel e organização. As classes que não exercem papéis e não estão relacionadas a papéis não são modeladas em diagramas de papel e organização. As classes que não são modeladas em diagramas de papel e organização devem ser modeladas em diagramas de classes.

Todas as classes de entidade modeladas em um diagrama de organização estão relacionadas à mesma classe de ambiente. Todas as entidades que exercem papéis em uma organização residem na mesma instância de ambiente dela. Entidades que residem em outro ambiente não podem exercer papéis nessas organizações. Na verdade, não é possível garantir durante o processo de modelagem que todas as instâncias de suborganização e agente residirão na mesma instância de

ambiente em que residem as organizações em que estão exercendo papéis. Essa restrição deve ser garantida no momento da execução.

8.1.2 Diagramas Dinâmicos

O foco principal de um diagrama de atividades é modelar o aspecto comportamental de processos. Neste diagrama, uma atividade é modelada como uma sequência estruturada de ações, controladas potencialmente por nós de decisão e sincronismo.

A fim de modelar os fluxos de execução internas ao agente, os diagramas de atividades devem ser utilizados. Um diagrama de atividades modela a ordem de execução do raciocínio autoadaptativo do agente com relação às normas. Um diagrama de atividades modela as possíveis ações a serem realizadas entre (i) agentes que estão exercendo papéis e monitorando o ambiente, (ii) normas ativas no ambiente endereçadas estes papéis, (iii) tomadas de decisão por um agente para raciocinar sobre uma norma no ambiente. Um diagrama de atividades também modela os planos dos agentes, organizações e ambientes.

8.1.3 Processo de Adaptação no JSAN 2.0

Toda vez que um agente autoadaptativo capaz de raciocinar sobre normas for criado no JSAN 2.0, deve ser definido o processo de autoadaptação que ele deverá executar. Atualmente, o JSAN 2.0 oferece um processo padrão que pode ser utilizado por diferentes agentes de software. No entanto, isso não impede que novos processos de autoadaptação possam ser criados.

A partir do momento que algum agente autoadaptativo deva realizar o raciocínio normativo, o processo a ser utilizado deve conter as atividades Coleta, Análise, Decisão e Efetuação. Tais atividades ajudam, respectivamente, a selecionar e executar o raciocínio normativo do agente e tomar suas decisões.

8.2 Cenário de uso: Resgate de Pessoas em Áreas de Risco

A fim de exemplificar a utilização da linguagem ANA-ML e do *framework* JSAN 2.0 é apresentado um cenário no domínio de evacuação de pessoas em áreas de risco. O resultado ideal para os bombeiros seria resgatar e prestar o maior número de atendimentos aos feridos. Para isso, os bombeiros têm um grupo limitado de recursos, que são regulados pelo comandante dos bombeiros. Esta regulação é feita através de normas, que restringem o comportamento dos bombeiros para que o resgate ocorra de forma coordenada e com o melhor aproveitamento de recursos.
A fim de cumprir suas tarefas, os grupos supracitados têm disponível um conjunto de recursos limitados, por exemplo, estes recursos podem ser veículos aéreos, veículos terrestres e equipamentos de escavação. Sabendo que os recursos são escassos, o comandante dos bombeiros é responsável por gerenciar a assistência de recursos aos grupos de salvamento. Para gerenciar os recursos, o comandante precisa adaptar o seu comportamento com relação às normas do sistema, com o objetivo de assistir a todos os salvamentos, priorizando a forma como os recursos devem ser distribuídos. Desta forma, é objetivo de cada grupo obter mais recursos. Por fim, assume-se que cada grupo tem associada uma reputação; logo, é objetivo de cada grupo aumentar ou manter sua reputação.

Neste cenário aborda-se como o comportamento dos bombeiros pode ser influenciado a partir da definição de um conjunto de normas e como eles irão se adaptar a elas. A *importância* de realizar um comportamento é avaliada levando em consideração a *motivação* que o agente tem para atingir um objetivo, ou a *satisfação* que ele tem de realizar uma ação.

As seguintes normas foram consideradas neste cenário:

Norma N1: Se bombeiros estão indo resgatar civis em áreas de risco, Comandantes são obrigados a oferecer recursos.

Recompensas: Se recurso é provido:

Recompensa 1: A reputação dos Comandantes é aumentada.

Recompensa 2: Mais bombeiros são disponibilizados para ajudar pessoas que estão em uma determinada área de risco.

Punição 1: Se recursos não são fornecidos, a reputação do Comandante é diminuída.

Norma N2: Se Corporação do Bombeiros não é capaz de fornecer os recursos necessários para garantir a segurança mínima para seus bombeiros, estes não poderão realizar salvamento.

Punição 1: Caso o salvamento seja realizado sem as mínimas condições de segurança, a reputação da Corporação do Bombeiros é diminuída.

Norma N3: Se durante o fornecimento de salvamento o número de pessoas em área de risco é maior do que o de bombeiros, a Corporação do Bombeiros é obrigada a assistir na prestação do serviço com veículos aéreos.

Recompensa 1: Se assistência é fornecida, poderá pedir ajuda de tropas.

Recompensa 2: Se assistência é fornecida, poderá pedir mais helicópteros.

Recompensa 3: Se assistência é fornecida, a reputação da Corporação dos Bombeiros é aumentada.

Punição 1: Se assistência não é fornecida, a reputação da Corporação dos Bombeiros é diminuída.

O objetivo principal dos comandantes de bombeiros é prestar auxílio ao grupo de bombeiros que irá prestar o resgate, sua *motivação* será 10. Embora os recursos tenham um custo associado à sua utilização, eles são fundamentais para os bombeiros realizarem o salvamento. Portanto, será assumida uma *satisfação* de valor 3 para os comandantes de bombeiros utilizar veículos aéreos, satisfação 5 para enviar veículos terrestre e satisfação 7 para enviar mais equipamentos de escavação.

8.2.1 Modelagem a partir da ANA-ML

Nesta subseção são apresentados os diagramas ANA-ML criados seguindo a abordagem apresentada na seção 7.1. Inicialmente são apresentados os diagramas de classes estáticos, seguido pelo diagrama de atividade dinâmico definido para o sistema.

8.2.1.1 Diagramas Estruturais

Na Figura 32 é possível verificar o ambiente e a organização do cenário de uso. Esta visão tem como objetivo mostrar as classes da organização principal e do

ambiente, juntamente com os elementos e papéis. Essa representação simplificada não mostra os compartimentos inferior e intermediário dos elementos do diagrama.

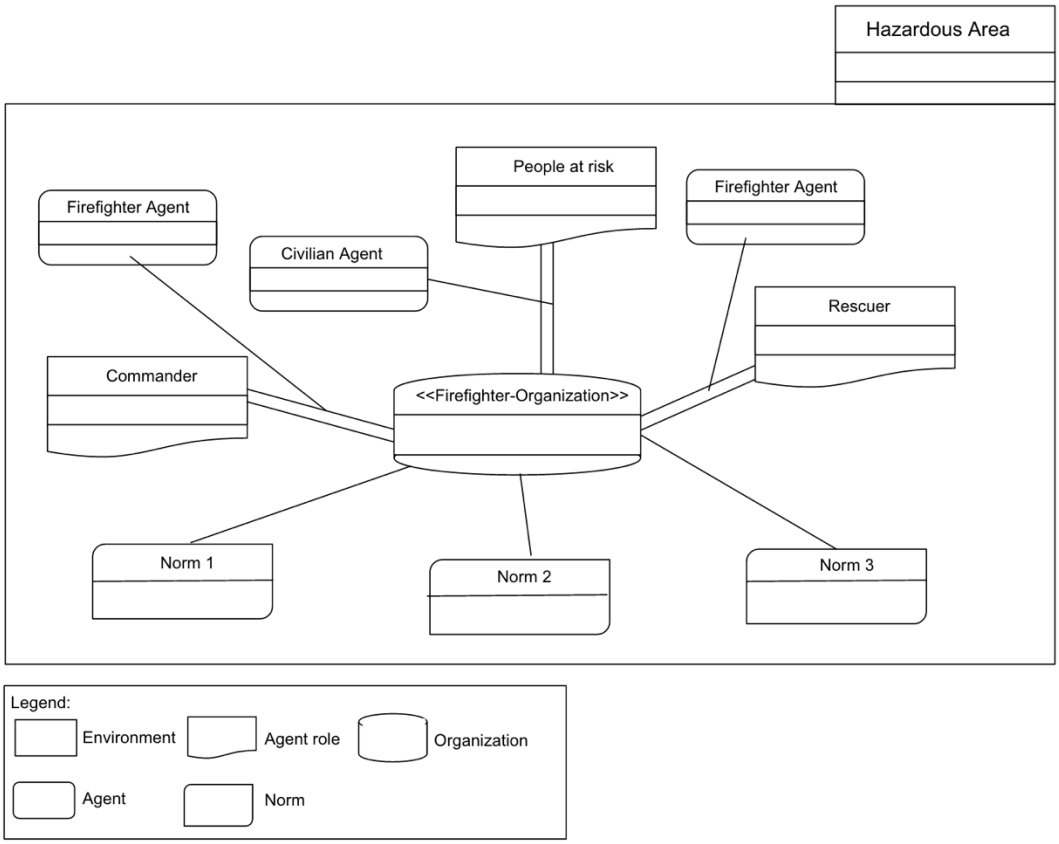

Figura 32 - O diagrama de organização dos bombeiros

O diagrama apresenta a classe de ambiente *hazardous area*. Além disso, foram modelados nesse sistema dois tipos de agentes: *civilian agent* e *firefighter agent*. O diagrama também ilustra as classes de papel *person at risk*, *rescuer* e *commander firefighter* definidas pela organização *firefighter organization*. O papel *person at risk* é exercido por *civilian agent*, já os papéis *rescuer* e *commander firefighter* são exercidos por agentes do tipo *firefighter agent*. Foram definidos normas pela organização principal, as normas ***N1***, ***N2*** e ***N3*** (Seção 7.2.1), com o objetivo de restringir o comportamento do papel realizado pelos bombeiros em missões de resgate.

8.2.1.2 Descrição Parcial da Organização e Ambiente

Ao analisar a descrição do problema identificou-se a organização *firefighter organization* que está dentro do ambiente *harzadous area*. O ambiente é modelado

como um ambiente ativo, onde informações são adicionadas ao longo do tempo para que atributos como condições meteorológicas e de deslizamento de terra possam ser introduzidas no sistema. Ele implementa os *métodos*: (i) *addPercept* novas características do ambiente, (ii) *executeAction* quais as ações o ambiente irá executar e (iii) *stop* para finalizar o SMA. Como pode ser visto parcialmente na Figura 33.

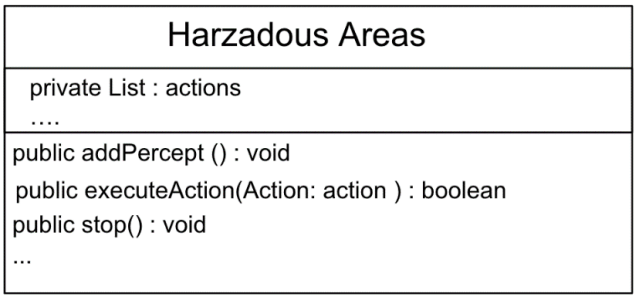

Figura 33 - A classe do ambiente Hardazous Areas (parcial)

A organização principal *Firefighter Organization* (parcialmente ilustrada na Figura 34), é a organização principal do sistema, não exercendo nenhum papel e apenas uma instância dessa classe pode ser criada por ambiente.

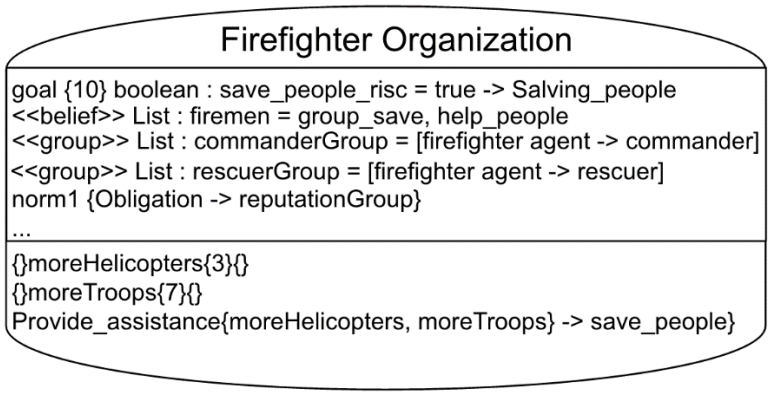

Figura 34 - A classe da organização do Corporação de Bombeiros (parcial)

O objetivo da organização principal é o salvamento de todas as pessoas em áreas de risco, por isso sua motivação é máxima, recebendo valor 10. Para alcançá-lo, a organização principal define normas (descritas na Seção 7.2.1.3) e planos para (i) resgatar pessoas; (ii) gerenciar equipamento; (iii) atualização do ambiente a fim de informar que uma pessoa foi salva; (iv) avaliar a missão. Os agentes dentro desta organização podem realizar diferentes papéis, de *Commander*, *Rescuer* e *People Risk*. Esses papéis serão detalhados na Seção 7.2.1.4. Os objetivos da organização principal são o gerenciamento dos recursos escassos. Para alcançá-los, a

organização principal define planos para (i) salvar pessoas e (ii) normas para gerenciar o uso recursos. As crenças da organização principal estão relacionadas com informações relativas aos agentes bombeiros. Além disso, os agentes bombeiros são separados em diferentes grupos, dependendo do seu papel, ou seja, para o papel de *rescuer* tem-se o grupo *rescuerGroup*, e para o papel de *commander*, tem-se o grupo *commanderGroup*. Lembrando que cada grupo tem uma reputação associada a ele, o objetivo é aumentar ou manter sua reputação.

8.2.1.3 Descrição das Normas

A seguir, na Figura 35, foram modeladas normas ativas no cenário de uso (Ver Seção 7.2).

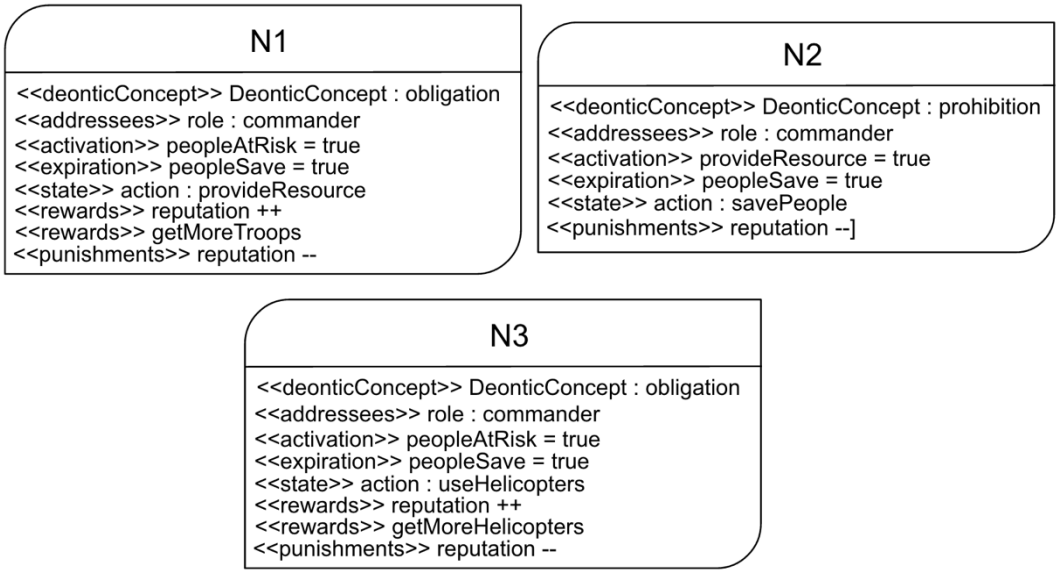

Figura 35 – Normas vigentes no cenário de evacuação de pessoas em áreas de risco

8.2.1.4 Descrição dos Papéis Exercidos por Agentes na Organização Principal

Conforme já visto, a organização define os papéis de *person at risk, firefighter e command firefighter* (Figura 32) cujos respectivos *objetivos* são: ficar a salvo, salvar pessoas e gerenciar recursos escassos, dada sua reputação. Para alcançar seus objetivos, os bombeiros interagem com as normas que restringem o comportamento do papel que eles se dispuseram a realizar. O dever de um bombeiro é salvar vidas.

As normas que gerenciam o uso de recursos escassos e o comportamento do agente devem ser entendidas pelo agente bombeiro.

Na Figura 36 foi descrito o papel de uma pessoa em risco, onde seu principal *objetivo* é estar a salvo, tendo sua motivação com valor máximo. Além disso, existe sua crença de estar em perigo e seu desejo de conseguir ajuda.

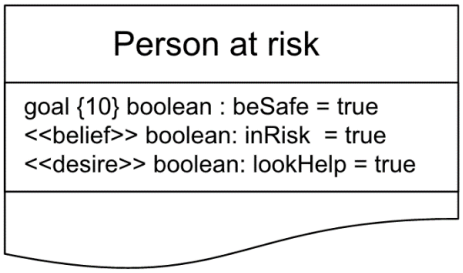

Figura 36 - A classe do papel Person at risk (parcial)

Já na Figura 37 o papel do agente que irá realizar o salvamento tem como *objetivo* principal salvar pessoas, sua motivação é 10. Os *deveres* relacionados com este papel são o de se manter a salvo, ou seja, não colocar a sua vida em risco. Já como *direitos* eles podem pedir por *recursos* que serão utilizados durante o resgate de pessoas em áreas de risco.

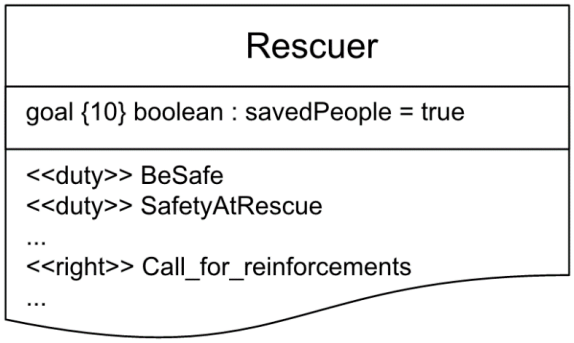

Figura 37 - A classe do papel Rescuer (parcial)

Já na Figura 38 o papel do agente comandante, cujo objetivo principal é gerenciar recursos no salvamento de pessoas, sua motivação é máxima, valor 10. Seguido da prestação de assistência para os bombeiros, com motivação um pouco menor, com valor de 8. Os *deveres* relacionados com este papel são o de se manter a salvo e o de criar normas na organização, ou seja, não colocar as suas vidas em risco e restringir o comportamento de seus subordinados. Já como *direitos*, o papel de comandante permite que ele gerencie os recursos escassos da corporação.

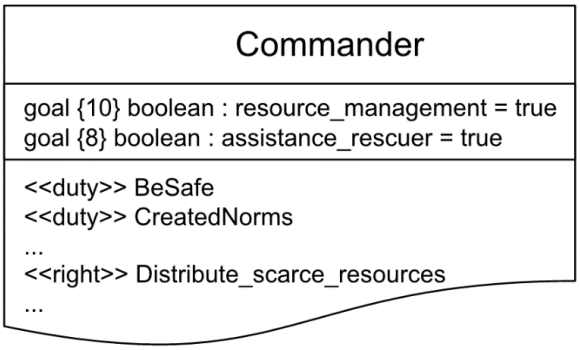

Figura 38 - A classe do papel Commander (parcial)

8.2.1.5 Descrição dos Agentes

A Figura 39 ilustra a classe *Firefighter Agent* descrevendo (i) os *objetivos* "*salvedPeople*" e "*increasedReputation*", (ii) suas crenças de como lidar com os problemas do dia a dia nas operações de salvamento, (iii) o desejo de aumentar sua reputação e (iv) sua lista de intenções a serem realizadas.

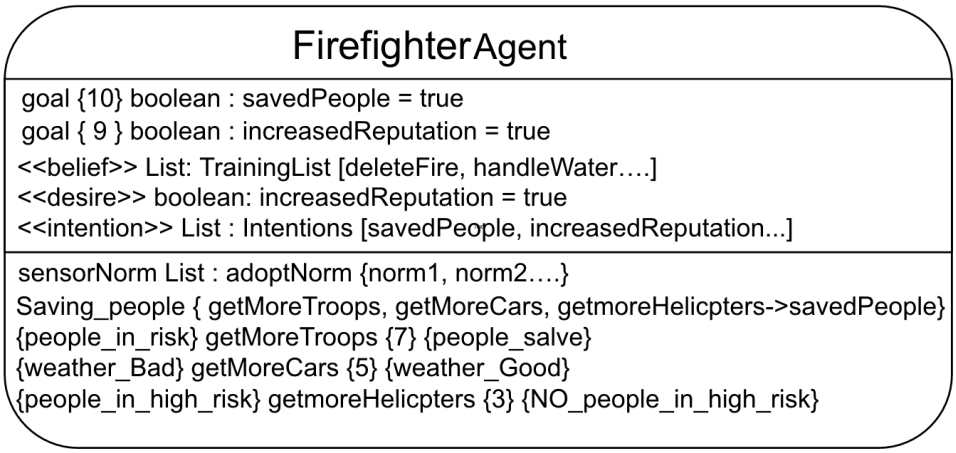

Figura 39 - A classe Firefighter Agent (parcial).

Já a parte inferior contêm as normas que foram identificadas pelos monitores do agente e foram entendidas pelo agente que são para o papel que ele realiza na organização. Além disso, tem o planejamento que o agente deseja realizar para salvar pessoas, e as ações e suas satisfações atreladas que o *Firefighter Agent* desempenhará para salvar pessoas. Por exemplo, se o tempo estiver ruim, o pedido de mais veículos terrestre, tem uma *satisfação* representada pelo inteiro 5, sendo mediana conforme definida no cenário de uso.

Um *Civilian Agent* (Ver Figura 40) representa as pessoas em risco do sistema. Essa classe descreve um único objetivo e com motivação máxima, ou seja, ficar a salvo. *Civilian Agent* quando percebe que *está em risco* tem essa crença adicionada à sua base. O seu desejo de *ficar a salvo* torna-se prioridade e logo ele começa a criar sua lista de intenções, que tem como objetivo principal *ficar a salvo* e *chamar por ajuda dos bombeiros*.

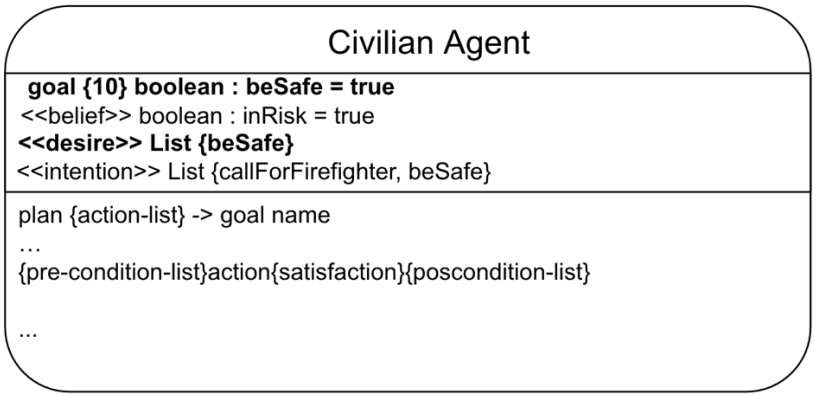

Figura 40 - A classe Civilian Agent (parcial).

8.2.2 Diagramas Dinâmicos e a Autoadaptação Realizada pelo Agente Bombeiro

Os diagramas de atividades propostos por Gonçalves et al. (2015) foram adaptados para inserir os elementos necessários para o agente bombeiro, que está fazendo o papel de *Commander* lide com às normas ativaas no sistema endereçadas a ele e como ele irá se comportar para realizar um determinado salvamento. Assim, cada atividade é representada por um retângulo arredondado. As características do agente são representadas com quadrados com a identificação de qual estereótipo seria do agente.

Quando um agente bombeiro no papel de *Commander* não consegue realizar o salvamento de pessoas em áreas de risco, por exemplo dada a norma ***N3***, a qual diz que se o número de pessoas em risco for maior do que o de bombeiros (*Rescuer*), o comandante deve assistir ao salvamento enviando veículos aéreos. O processo de autoadaptação oferecido pelo JSAN 2.0 em relação à norma é executado pelo agente no papel de *Commander*. Esse processo procura por outras formas de fazer o resgate de modo que consiga salvar essas pessoas sem o uso de veículos aéreos, com os

dados que ele tem da situação. Esse processo é composto pelas seguintes atividades: Coleta, Análise, Decisão e Efetuação. Como pode ser visto na Figura 41, abaixo.

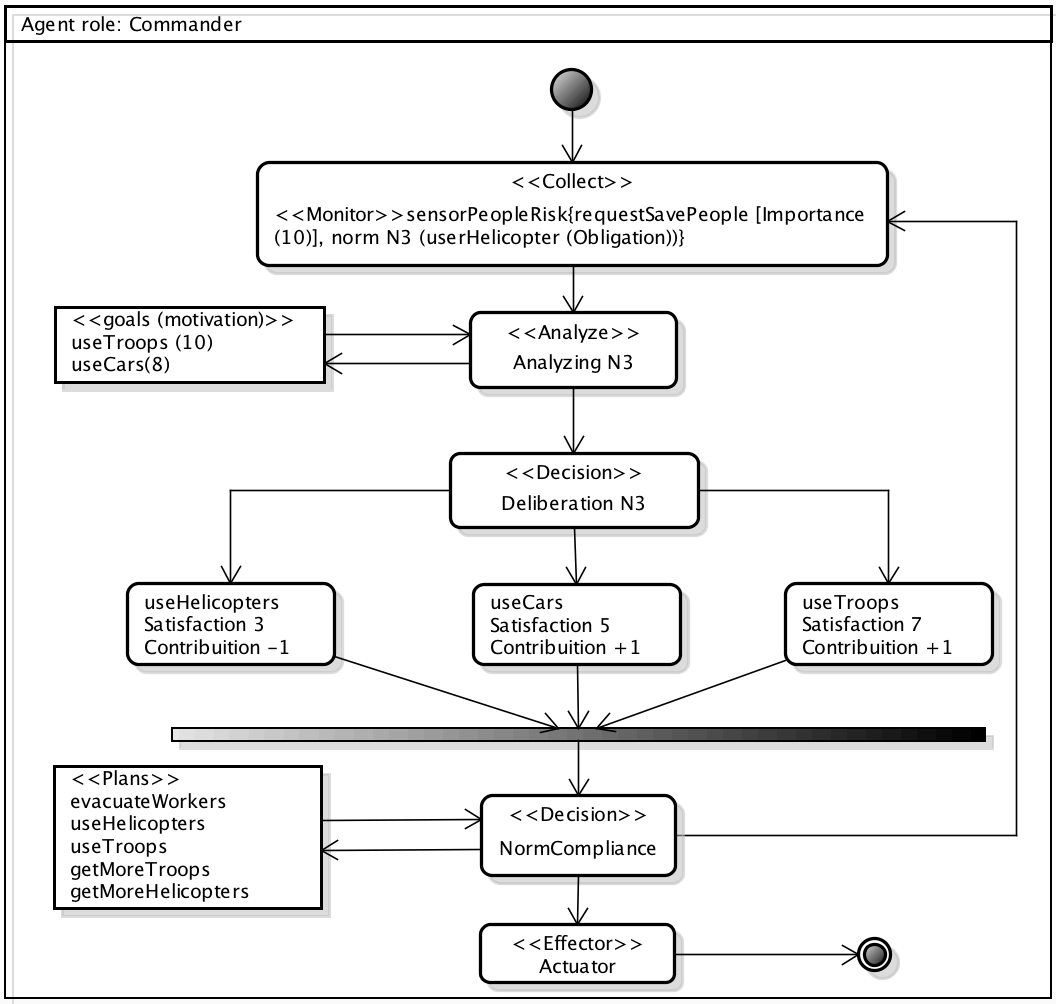

Figura 41 – Diagrama de atividades do firefighter agent no papel de Rescuer

A adição de normas no ambiente é feita através de funções de *addPercept* descrita na Figura 33. A Figura 42 mostra as normas sendo adicionadas no ambiente. São criadas as normas ***N1***, ***N2*** e ***N3***. Pegando como exemplo a ***N1***, tem-se na linha 17 a 25 os seguintes componentes de uma norma: (linha 17) a quem a norma foi endereçada; (linha 18) quando a norma será ativada; (linha 19) quando ela será desativada; (linha 20) o conceito deôntico de obrigatório o seu cumprimento; (linha 21) as recompensas providas caso o agente cumpra com a norma, nesse caso poder chamar reforços e ter sua reputação aumentada; (linha 22 a 24) é a punição que o agente receberá caso a viole; e (linha 25) o nome da norma.

```
8  public class HarzadousArea extends Environment {
9
10     private Logger logger = Logger.getLogger("EvacuationScenario."+HarzadousArea.class.getName());
11
12     /** Called before the MAS execution with the args informed in .mas2j */
13     @Override
14     public void init(String[] args) {
15         super.init(args);
16
17         addPercept("firemen",Literal.parseLiteral("normspecification(CommanderFireman, " +
18                 "peopleAtRisk, " +
19                 "peopleSave, " +
20                 "obligation, " +
21                 "provideResource, " +
22                 "[getMoreTroops, reputation ++], " +
23                 "[" +
24                 "[reputation --]" +
25                 "], 1)."));
26         addPercept("firemen",Literal.parseLiteral("normspecification(CommanderFireman, " +
27                 "provideResource, " +
28                 "peopleSave, " +
29                 "prohibition, " +
30                 "savePeople, " +
31                 "[], " +
32                 "[" +
33                 "[reputation --]," +
34                 "], 2)."));
35         addPercept("firemen",Literal.parseLiteral("normspecification(CommanderFireman, " +
36                 "peopleAtRisk, " +
37                 "peopleSave, " +
38                 "prohibition, " +
39                 "useHelicopters, " +
40                 "[getMoreHelicopters, reputation ++], " +
41                 "[" +
42                 "[reputation --]" +
43                 "], 3)."));
44     }
```

Figura 42 - Código das normas vigentes no cenário

O próximo passo é a atividade de Coleta que é executada todo o tempo durante o período da simulação. Assim que o *Comander agent* é entende que existe a norma **N3** que o ***obriga*** a enviar ajuda de veículos aéreos (Essa identificação de responsabilidade do agente é feita na Figura 43, a qual mostra parte do algoritmo 1, da Seção 7.2.1.1). Entretanto, esse agente tem a crença de que este é um recurso muito caro (com baixa satisfação) e que deve procurar alternativas na assistência desse resgate. Assim, são coletadas informações sobre a área de risco e normas endereçadas a ele nessa situação.

```
82  private List<Literal> sensor(List<Literal> norms) {
83
84
85      for (Literal instantiatedNorm : norms) {
86          System.out.println("\n Instantiated Norm: ");
87
88          Term deonticConcept = instantiatedNorm.getTerm(3);
89          System.out.println("Deontic Concept: "+ deonticConcept.toString());
90
91          Term state = instantiatedNorm.getTerm(4);
92          System.out.println("State: "+ state.toString());
93
94          List<Term> rewradTerms = new ArrayList<Term>();
95          Term rewards = instantiatedNorm.getTerm(5);
96
97          if (!rewards.toString().trim().equalsIgnoreCase("[]")) {
98              for (Term term : getRewards(rewards.toString())) {
99                  System.out.println("Rewards: "+ term.toString());
100                 rewradTerms.add(term);
101             }
102         }else
103             System.out.println("Rewards Not Exist.");
104
105         List<Punishment> punishmentList = new ArrayList<Punishment>();
106         Term punishments = instantiatedNorm.getTerm(6);
107         punishmentList = getPunishments(punishments.toString());
108         System.out.println("Punishments");
109         for (Punishment punishment : punishmentList) {
110             System.out.println("Deontic Concepts: "+punishment.getDeonticConcept());
111             System.out.println("State: "+punishment.getState());
112         }
```

Figura 43 - Agente entendendo a norma

Em seguida, na Figura 44, uma parte do código da atividade de Análise (algoritmo 4, ver Seção 7.2.1.2) é executada a fim de encontrar uma solução com o custo menor para realizar o salvamento dessas pessoas com uma menor despesa e sem comprometer os recursos escassos. Essa análise é feita a partir das informações coletadas das normas vigentes na simulação. Exemplos possíveis de causas que podem ter impedido o resgate podem ser as seguintes: (i) número de bombeiros menor que o de pessoas em riscos, (ii) baixa reputação do bombeiro que está realizando o resgate, (iii) falta de equipamentos para resgate. A partir desse diagnóstico, o *Commander agent* tem sua reputação decrementada em -1 caso tenha mais de uma dessas causas identificadas. O processo de análise inicia após a norma ser entendida pelo agente e este começa seu raciocínio deliberativo. Como a norma é de obrigação, mas não tem nenhum objetivo pessoal do agente que gostaria de realizar a ação de assistir o salvamento enviando veículos aéreos, a *contribuição* de cumprir com a norma é entendida como negativa e receberá o valor somado de -1 (linha 155). Caso a ação ou estado que a norma regula fosse igual ao objetivo do agente, a *contribuição* da norma seria somada de +1 (linha 150).

```
boolean existGoalStateObligation = false;
if (deonticConcept.toString().trim().equalsIgnoreCase("obligation")) {
    for (Term goal : goals) {
        if (unifier.unifies(state, goal)) {
            System.out.println("State Obligation Norma: Os termos: "+state.toString()+" e "
            +goal.toString()+" foram unificados.");
            existGoalStateObligation = true;
        }
    }
    if (existGoalStateObligation) {
        normContribution.setFullfillContribution(normContribution.getFullfillContribution()+1);
        System.out.println("Obligation Norm Contribution For Fulfilling +1: "+normContribution.getFullfillContribution());
    }else{
        normContribution.setFullfillContribution(normContribution.getFullfillContribution()-1);
        System.out.println("Obligation Norm Contribution For Fulfilling -1: "+normContribution.getFullfillContribution());
    }
}



boolean existGoalStateProhibition = false;
if (deonticConcept.toString().trim().equalsIgnoreCase("prohibition")) {
    for (Term goal : goals) {
        if (unifier.unifies(state, goal)) {
            System.out.println("State Prohibition Norma: Os termos: "+state.toString()+" e "
            +goal.toString()+" foram unificados.");
            existGoalStateProhibition = true;
        }
    }
    if (existGoalStateProhibition) {
        normContribution.setFullfillContribution(normContribution.getFullfillContribution()-1);
        System.out.println("Prohibition Norm Contribution For Fulfilling -1: "
        +normContribution.getFullfillContribution());
    }
}
```

Figura 44 - Contribuição normativa

Com isso, o *Commander agent* verifica que, se ele enviar helicópteros durante o resgate, não irá realizar nenhum dos seus objetivos individuais e ainda terá uma satisfação baixa (valor 3) em realizar tal ação regulada pela norma. Assim o agente comandante verifica outros planos, por exemplo, o plano de enviar tropas, como a satisfação de realizar está ação é de 7 e ainda tem uma *contribuição* normativa somada de +1, ou seja, a norma ter o conceito deôntico de obrigação incentiva ele a realizar o objetivo de gerenciar os recursos com o menor custo possível. No plano de utilizar carros, o agente tem a satisfação de realizar a ação de 5 e *contribuição* da norma ser realizada somada de -1.

Logo depois a atividade de Decisão é executada para escolher o melhor plano para atender a solicitação de ajuda, dado as seguintes variáveis: (i) a satisfação de realizar uma ação ou (ii) a motivação de realizar um objetivo individual, e (iii) a contribuição de realizar uma dada norma (Ver algoritmo 5 e 5, Seção 7.2.1.3). Assim, *Commander agent* verifica o que foi escolhido pelo seu raciocínio normativo para verificar se cumprir com a norma ***N3*** é a melhor solução para resolver o resgate.

Em seguida, a atividade de Efetuação é realizada dado o plano escolhido na atividade de Decisão. Caso contrário, a atividade Decisão é executada novamente para escolher outro plano candidato. Esse ciclo de execução continua até que o

plano seja considerado apto para realizar o salvamento dessas pessoas em áreas de risco.

8.2.3 Simulação de Evacuação de Pessoas de Áreas de Risco

Para ilustrar a simulação feita utilizando o JSAN 2.0 no cenário de Evacuação de Civis em Áreas de Risco, foi gerada uma visualização que será descrita em detalhes abaixo. A Figura 45 mostra o agente Comandante dos Bombeiros recebendo uma mensagem sobre as condições do tempo, falando que o tempo está ruim e logo em seguida uma mensagem sobre a existência de civis em uma localidade de risco. Após receber as mensagens o agente Comandante dos Bombeiros envia um alerta para os agentes bombeiros que recebem as informações da existência de civis em áreas de risco.

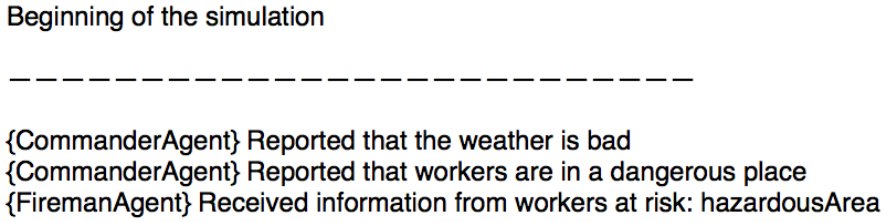

Figura 45- Início da Simulação

Já na Figura 46, tem-se o planejamento de poder utilizar veículos aéreos, veículos terrestres, e outros planos caso eles precisem chamar reforços de veículos aéreos ou terrestres e pedir por mais equipamentos e tropas.

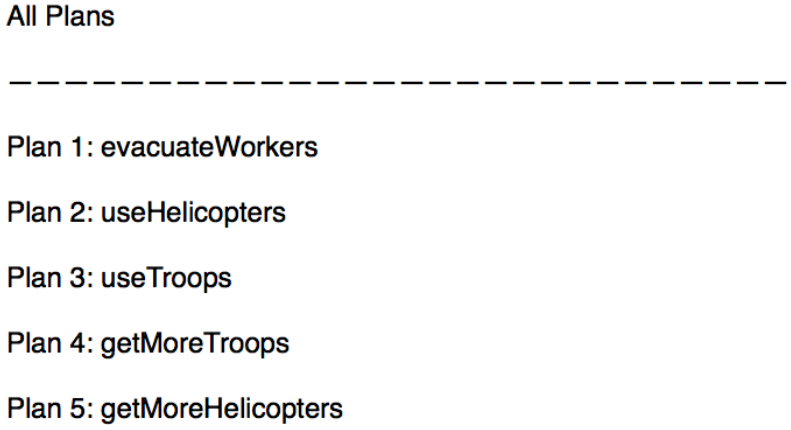

Figura 46 - Planos de Resgate

Na Figura 47 tem-se a norma ***N3*** discutida na Seção 7.2.2, “Restringe o uso de helicópteros”. Seu conceito deôntico é de obrigação, (ii) uma recompensa caso a norma seja cumprida é o agente obter apoio aéreo ou apoio terrestre, caso seja violada o agente não receberá apoio terrestre para o resgate, (iii) a norma é ativada caso o número de bombeiros seja menor do que o número de pessoas em e a norma é desativada quando todos civis estão a salvo, e (v) o elemento que a norma regula é a ação de usar veículos aéreos. *Norm Reward* são as recompensas associadas à norma e *Contribution* é a contribuição normativa, a qual mostra o quanto o agente irá ganhar caso cumpra com a norma.

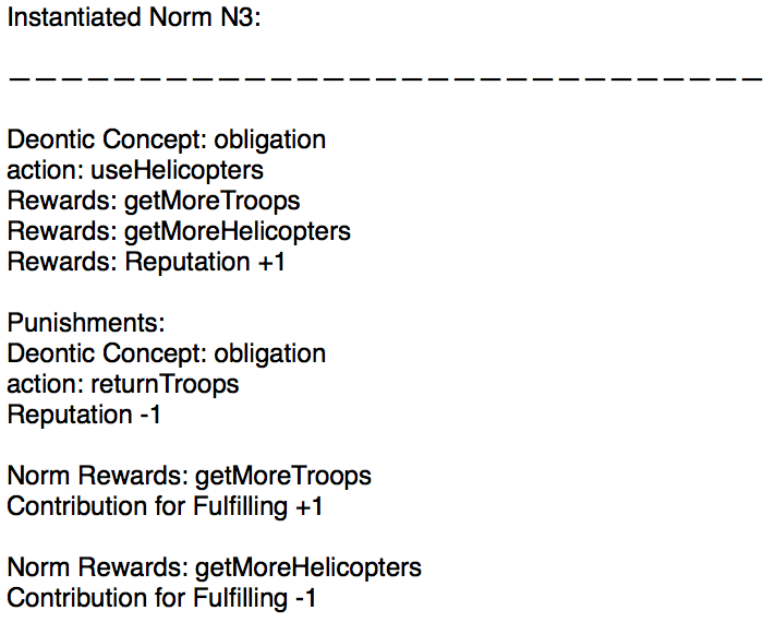

Figura 47 – Uma das Normas Criadas na Simulação

Por fim, na Figura 48, pode ser visualizado qual dos planos os agentes bombeiros decidiram utilizar depois que a norma foi ativada devido à existência de civis em áreas de risco.

Se ele optar pelo uso de mais veículos aéreos isto irá ter uma contribuição normativa negativa, já se ele optar pelo uso envio de mais bombeiros para colaborar com o resgate, sua contribuição será positiva. Após verificar a importância de realizar o resgate utilizando diferentes ações, o *Commander agent* decidiu evacuar os civis que estavam nesta área de risco através do envio de mais contingente.

```
Plans in Select Option
————————————————————————————————

Activation: evacuateWorkers:

        If (useHelicopters)
              Importance: satisfaction (3) + Contribution (-1)
        if (useTroops)
              Importance: satisfaction (7) + Contribution (+1)
        if (useCars)
              Importance: satisfaction (5) + Contribution (+1)


Result
————————————————————————————————

{FiremanAgent} FiremanAgent evacuate using Troops
{FiremanAgent} FiremanAgent uses troops
```

Figura 48 - Decisão tomada pelos agentes bombeiros

8.3 Cenário de uso: Mercado Virtual

Com o objetivo de mostrar a aplicabilidade da linguagem ANA-ML e do *framework* JSAN 2.0 é apresentado um outro cenário de uso, agora no domínio de mercados virtuais. Estes mercados virtuais representam mercados na Web, onde usuários podem comprar produtos. Na instância desenvolvida, cada usuário é representado por um agente comprador, que solicita compra de um determinado livro a um agente vendedor. Foi escolhido este exemplo de comércio eletrônico, porque há referências na literatura (He et al., 2003; Jennings & Wooldridg, 1999; Lomuscio et al., 2003) como sendo um *benchmark* de SMA. Foram descritas todas as entidades que compõem o sistema de mercado virtual, todas as propriedades associadas às entidades e todos os relacionamentos entre as entidades. Ao ilustrar as entidades de SMAs usando os elementos de diagrama definidos na Seção 5.5, omitiu-se algumas propriedades. Além disso, foram modelados diagramas de atividades, a fim de descrever diferentes aspectos dinâmicos.

Quando o usuário comprador realiza seu cadastro no sistema, ele deve informar qual seu mercado preferido, já que a partir dessa informação, o agente *comprador* procura realizar a compra com o agente *vendedor* representante desse mercado. Após o cadastramento, o usuário poderá solicitar a compra de livros a partir do fornecimento das seguintes informações: (i) título(s) do(s) livro(s), (ii) nome(s) do

autor(es), (iii) se o(s) livro(s) deve(m) ser novo(s) ou se pode(m) ser usado(s). A partir dessas informações, o agente comprador verifica se o agente vendedor possui o item disponível. Perceba que cada solicitação de compra pode ser composta por informações de um ou mais livros.

Caso o agente vendedor tenha o(s) livro(s) desejado(s), as informações do(s) item(ns) são apresentadas ao usuário para que a compra possa ser efetivada. No entanto, quando o agente vendedor não possui disponibilidade de algum item para venda, ou quando o preço sugerido pelo mercado é maior do que o usuário está disposto a pagar, tal usuário pode solicitar a compra para outro mercado. Para que isso seja possível, o usuário deve definir um critério de compra que será usado por seu agente comprador a fim de encontrar outro agente vendedor (mercado) que o atenda. Um critério de compra é composto pelas seguintes informações:

• **Valor máximo de pagamento**: O usuário deve informar o valor máximo em dinheiro que está disposto a pagar para cada livro que faz parte da sua solicitação de compra.

• **Livros novos e usados**: O usuário deve informar se deseja comprar livros novos ou usados.

• **Reputação do agente vendedor**: Como cada mercado possui um agente vendedor representante, o usuário pode solicitar que seu agente comprador negocie exclusivamente com agentes vendedores que possuam reputação maior ou igual à reputação informada pelo usuário.

Como o sistema armazena e disponibiliza a informação de quais compradores negociaram com quais vendedores, o agente comprador pode solicitar a outros agentes compradores a reputação deles em relação a algum vendedor. Essa reputação é chamada de reputação de testemunho, apresentada pelo modelo FIRE (Dong-Huynha et al., 2004). Nesse modelo, o valor possível das reputações é de [- 1, +1], onde -1 equivale a uma reputação absolutamente negativa, +1 absolutamente positiva, enquanto que 0 (zero) é uma reputação neutra ou incerta. Esses valores foram considerados no sistema.

A ideia geral da reputação de testemunho é ilustrada na Figura 49, onde um agente A (comprador) solicita a reputação a outros agentes compradores (agentes C, D e E) em relação ao agente vendedor B. A partir das informações recebidas, o agente A encontra a reputação final do vendedor a partir da média das reputações fornecidas pelos agentes C, D e E.

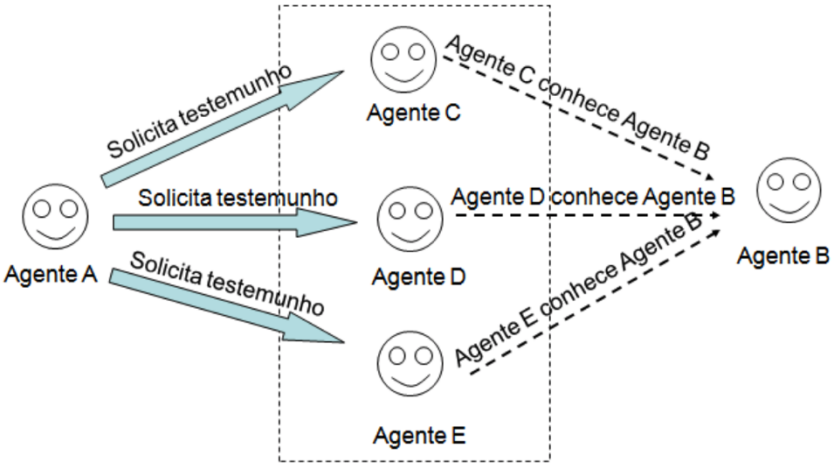

Figura 49 - Visão geral da reputação de testemunho

Quando o agente comprador não encontra ao menos um vendedor que atenda todo o critério de compra definido pelo usuário, o vendedor que chegou mais próximo em atendê-lo é escolhido. Ao encontrá-lo, a proposta de venda é apresentada ao usuário, aguardando a confirmação da compra. Após a confirmação, o agente comprador conclui a negociação com o agente vendedor e em seguida incrementa a reputação do vendedor em +0.1. Todo vendedor inicia com reputação 0 (zero) para cada agente comprador.

Nesse cenário será considerada para fins de estudo apenas a autoadaptação do agente vendedor para lidar com as normas. As seguintes normas foram consideradas neste cenário:

Norma N1: Vendedor é obrigado a entregar mercadoria vendida.

Recompensas: Se recurso é provido:

Recompensa 1: Aumento da reputação.

Recompensa 2: Poderá anunciar outro produto grátis.

Punição 1: Vendedor fica proibido de vender produto até entregar a mercadoria.

Norma N2: Comprador pode pagar a mercadoria comprada à vista.

Recompensa 1: Aumento da reputação.

Norma N3: Vendedor é proibido de vender mercadoria que não esteja no estoque.

Punição 1: Perda da sua reputação.

Norma N4: Vendedor só pode remarcar preço antes da loja abrir.

Punições: Se norma é descumprida:

Punição 1: Perda da sua reputação.

Punição 2: A venda do produto será cancelada.

Norma N5: Vendedor tem que remarcar o preço quando for anunciada promoção.

Recompensa 1: Aumento da sua reputação.

8.3.1 Modelagem a partir da ANA-ML

Nesta subseção são apresentados os diagramas ANA-ML criados seguindo a abordagem apresentada na Seção 7.1. Inicialmente são apresentados os diagramas de classes estáticos, seguido pelo diagrama de atividade dinâmico definido para o sistema.

8.3.1.1 Diagramas Estruturais

O diagrama de organização apresentado na Figura 50 ilustra a organização principal usando a representação simplificada que omite os compartimentos inferior e intermediário dos elementos do diagrama.

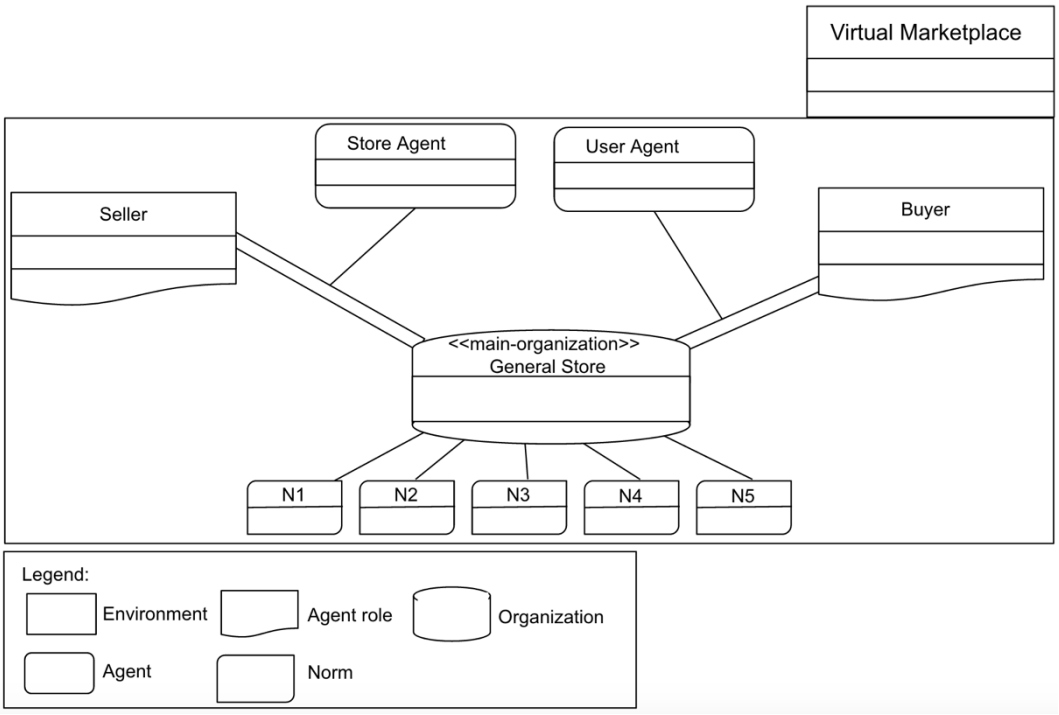

Figura 50 - O diagrama da organização principal

A fim de criar esse diagrama, foram definidas as classes da organização principal e do ambiente, junto com as classes de elemento, suborganização e papel.

O diagrama apresenta a classe da organização principal *General Store* e foram modelados nesse sistema dois tipos de agente: *user agent* e *store agent*. O diagrama também ilustra as classes de papel *seller* e *buyer* definidas pela classe da organização principal e exercidas por *store agent* e *user agent*, respectivamente. Um ponto importante que se deve ressaltar neste cenário é que as normas ***N1*** até ***N5*** foram criadas para a organização principal, sendo válidas também para as suborganizações, caso essas sejam criadas.

8.3.1.2 Descrição Parcial da Organização e Ambiente

Dada a descrição do cenário de uso é possível identificar uma organização principal que reside no ambiente *Virtual Marketplace*. Diferente do cenário de evacuação de pessoas em áreas de risco, o ambiente aqui é um elemento passivo que armazena itens que serão negociados como um de seus atributos. Ele implementa os métodos *get* e *set* a fim de acessar esses itens. Além disso, também armazena informações sobre os agentes e as organizações que residem nele. Portanto, esses métodos são definidos para acessar essas informações.

Para permitir que um agente ou uma organização se mova de um ambiente para outro, o ambiente deverá ter métodos para verificar a permissão relacionada à entrada e à saída dos elementos. O ambiente também define os métodos *get* e *set* para acessar as demais instâncias de ambiente.

A classe *Virtual Marketplace* está ilustrada em Figura 51. Alguns métodos e atributos associados a essa classe foram omitidos. Além disso, foram implementados os *métodos*: (i) *addPercept* para inserir novas características do ambiente, (ii) *executeAction* quais as ações o ambiente irá executar e (iii) *stop* para finalizar o sistema. Como pode ser visto na Figura 40, a qual representa o agente civil.

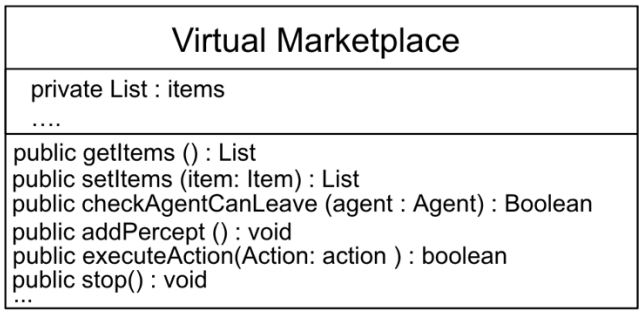

Figura 51 - A classe do ambiente Virtual Marketplace (parcial)

A organização principal *General Store* (parcialmente ilustrada na Figura 52), é a organização principal do sistema, não exercendo nenhum papel e apenas uma instância dessa classe pode ser criada por ambiente. Como os usuários podem comparar itens no mercado principal, a organização principal define os papéis do agente *buyer* e *seller*. Esses papéis serão detalhados na Seção 7.3.1.4.

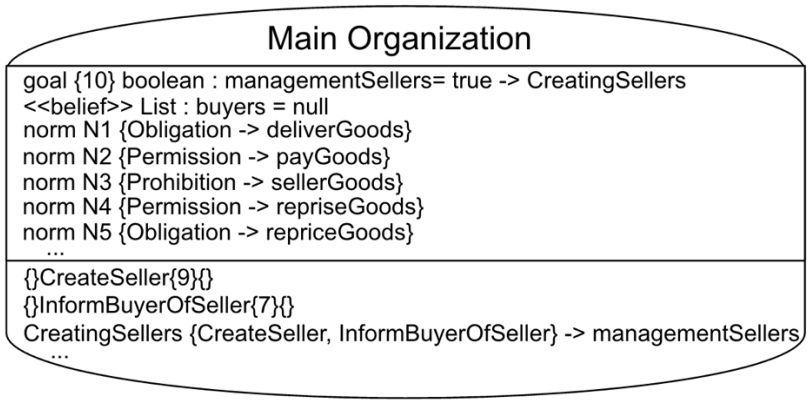

Figura 52 - A classe da organização principal (parcial)

Os objetivos da organização principal são o gerenciamento dos vendedores, pedidos e lucros. Para alcançá-los, a organização principal define planos para (i) criar vendedores para negociar com compradores, (ii) atualizar o ambiente a fim de informar que um item não está mais disponível e (iii) avaliar os lucros resultantes das vendas. Para garantir que a organização principal receba as informações relativas às vendas, foram definidas as normas da Seção 7.3. As crenças da organização principal estão relacionadas às informações relativas aos compradores, vendedores e vendas.

8.3.1.3 Descrição das Normas

A seguir, na Figura 53, as normas ativas no cenário de uso (Ver Seção 7.3). Como pode ser visto, as normas seguem a estrutura definida na Seção 2.3. A norma ***N1***: (i) restringe a ação de entrega de mercadoria vendida; seu (ii) conceito deôntico é obrigatório; (iii) é endereçada ao papel de vendedor; (iv) está ativa quando a loja abre e (v) expira quando o mercado fecha; (vi) já suas recompensas caso seja cumprida é ter a reputação aumentada e poderá anunciar um outro produto grátis no mercado; e como (vii) punição o vendedor fica proibido de vender outros produtos nesse mercado até entregar a mercadoria.

Já a norma ***N2*** (i) permite ao agente comprador pagar o produto à vista; (ii) sua atuação deôntica é permissiva; (iii) endereçada ao papel de comprador; (iv) está ativa quando a loja abre; (v) expira quando o mercado fecha e; (vi) como recompensa terá sua reputação aumentada.

A norma ***N3*** (i) incide sobre a ação de vender mercadorias que não estejam no estoque; (ii) sua ação deôntica é proibitiva; (iii) é endereçada ao papel de vendedor; (iv) está ativa quando a loja abre; (v) expira quando o mercado fecha; e (vi) caso o vendedor faça uma venda sem produto do estoque terá sua reputação diminuída.

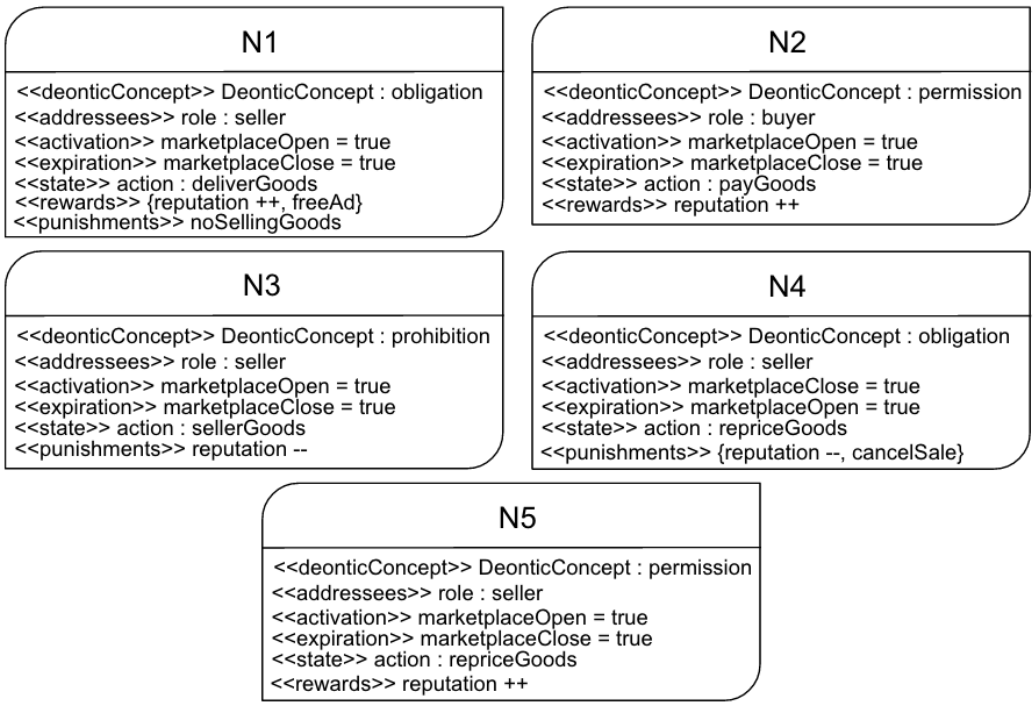

Figura 53 – Normas vigentes no Virtual Marketplace

As normas ***N4*** e ***N5*** são muito interessantes e mais a diante será realizada uma discussão sobre a possibilidade de que normas diferentes tenham atuações distintas sobre a mesma ação ou estado. A norma ***N4*** (i) atua sobre a marcação de preço dos produtos vendidos no mercado; (ii) o seu conceito é de permissão; (iii) é endereçada ao papel de vendedor; (iv) mas ela está ativa apenas quando a loja está fechada; (v) ao contrário das outras normas, ela expira quando o mercado abre; e (vi) caso o agente no papel de vendedor a viole terá sua reputação decrementada. Já a norma ***N5*** (i) atua sobre a mesma ação de remarcar preço que a norma ***N4***, entretanto, é válida somente quando acontece promoções no mercado virtual; (ii) e seu conceito deôntico é de obrigação; (iii) é endereçada ao papel de vendedor; (iv) fica ativa apenas quando o mercado está aberto; (v) expira quando o mercado fecha e, por fim; (vi) o agente vendedor terá como recompensa o aumento da sua reputação.

8.3.1.4 Descrição dos Papéis Exercidos por Agentes na Organização Principal

Conforme já visto, a organização principal define o papel de *comprador* (Figura 54) e o papel de *vendedor* (Figura 55) cujos objetivos são comprar um item e vender um item, respectivamente. Para alcançar esses objetivos, os agentes negociam itens armazenados no ambiente. O *dever* de um comprador é procurar vendedores. Depois de consultar o ambiente, o vendedor envia o preço ao comprador que pode aceitar ou recusar a proposta. As opções de aceitar ou recusar uma determinada proposta são os *direitos* do papel de comprador. Se o comprador aceitar a proposta, o vendedor enviará a fatura a ele. Então, o vendedor enviará as informações relativas à venda para a organização principal, conforme especificado em seu dever.

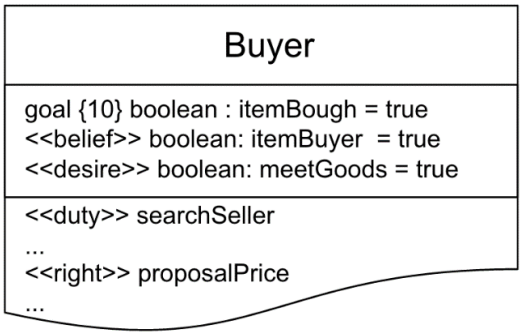

Figura 54 - A classe do papel Buyer (parcial)

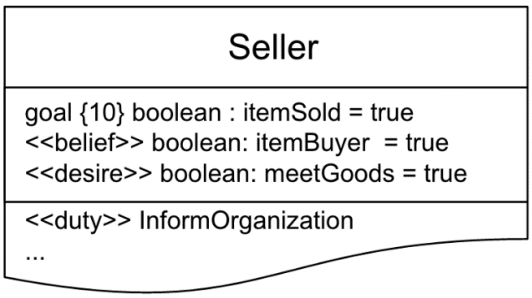

Figura 55 - A classe do papel Seller (parcial)

Os compradores e os vendedores têm diferentes visões dos itens que negociam. Um item é um desejo para os compradores e uma oferta para os vendedores. O desejo e a oferta possuem diferentes características. Sabendo que o item negociado no mercado é um livro.

8.3.1.5 Descrição dos Agentes

Conforme mencionado, há dois tipos de agentes. A classe *User Agent* (Figura 56) representa os usuários no sistema. Uma *User Agent* é criada quando um novo usuário deseja ter um "*itemBought*" ou um "*itemSold*". A instância do agente de usuário depende dos objetivos do usuário. A classe do agente do usuário descreve os objetivos. Entretanto, se o objetivo de uma instância do agente de usuário for ter um item comprado, o objetivo relacionado a ter um item vendido será excluído. O oposto também é verdadeiro.

Para alcançar seus objetivos, o agente de usuário deve precisar exercer os papéis de comprador de livros. O agente do usuário pode ter planos associados a seus papéis e objetivos.

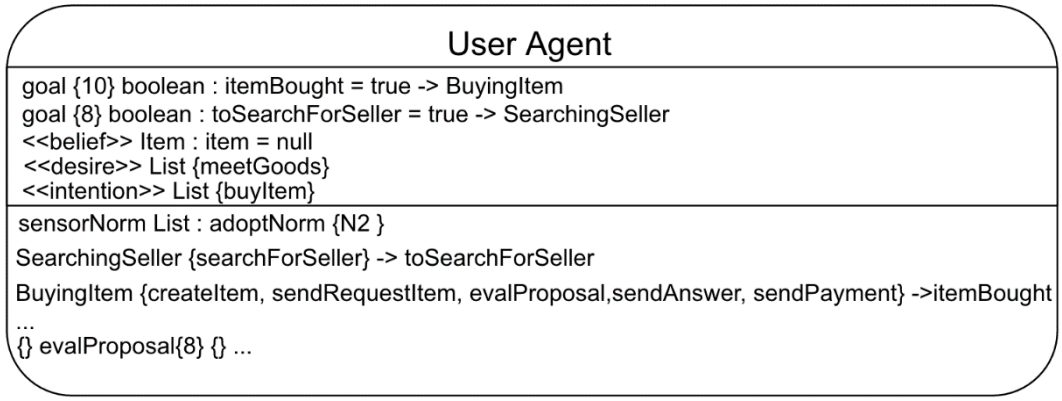

Figura 56 - A classe User agent (parcial).

O objetivo "*itemBought*" é dividido nas seguintes ações: "*createItem*", "*sendRequestItem*", "*evalProposal*", "*sendAnswer*" "*sendPayment*". Ao realizá-las, o objetivo também é alcançado. Antes de tudo, um agente deve tentar alcançar o objetivo "*toSearchForSeller*". Em seguida, se ele encontrar um vendedor, o agente deverá começar a negociar com ele. Se o agente não puder comprar o item, ele poderá entrar em outra loja (organização).

Já o objetivo "*itemSold*" é composto pelas ações de "*createProposal*", "*calculateBill*" e "*informOrganization*". Além de outras crenças, a classe do agente do usuário define a crença "*item*" que é usada para armazenar o item que as instâncias do agente do usuário desejam vender ou o item que desejam comprar. A Figura 56 ilustra a classe *User Agent* descrevendo (i) o objetivo "*itemBought*" (ii) a crença que armazena um item, (iii) o desejo de aumentar sua reputação e (iv) sua lista de intenções a serem realizadas, no seu caso de comprar um determinado item, representado por "*buyItem*".

Já a parte inferior da classe *User Agent* tem as normas que foram identificadas pelos monitores do agente e foram entendidas pelo agente que são para o papel que ele realiza na organização. Além disso, tem os planos que estão ligados aos objetivos dos agentes.

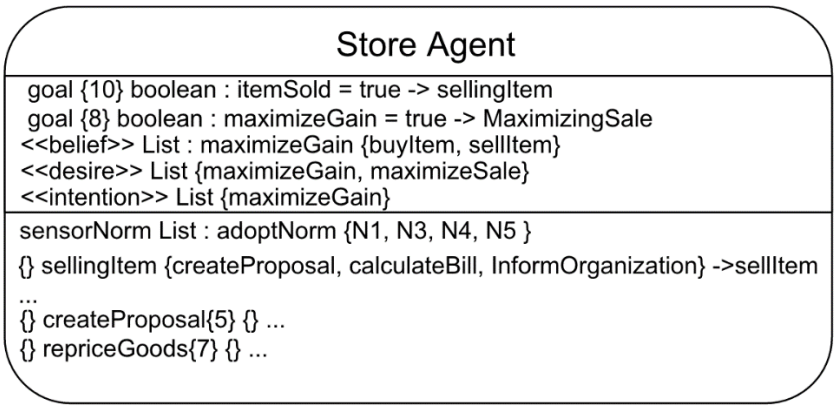

Figura 57 - A classe Store agent (parcial).

Uma *Store Agent* (Figura 57) representa as preferências do sistema. Essa classe tem os objetivos de vender um item e maximizar os ganhos. Ela pode apenas exercer o papel de vendedor. Na parte inferior, tem-se *sensorNorm*, que identifica quais normas foram endereçadas ao papel de vendedor. O plano que essa classe executa é o "*sellingItem*" está relacionado ao objetivo de "*itemSold*".

8.3.2 Diagramas Dinâmicos e a Autoadaptação Realizada pelo Agente Vendedor

Os diagramas de atividades criados na ANA-ML irão ajudar a inserir os elementos necessários para o *store agent* (exercendo o papel *Seller*) e *user agent* (exercendo o papel *Buyer*) entendam as normas vigentes no sistema endereçadas a eles e como eles irão detectar e resolver conflitos entre essas normas. Assim, cada atividade é representada por um retângulo arredondo. As características do agente são representadas com quadrados com a identificação de qual estereótipo seria do agente.

8.3.2.1 Store Agent no Papel de Vendedor

Quando um agente *vendedor* entra no mercado virtual com o objetivo de oferecer itens para os possíveis agentes *compradores* o processo do raciocínio autoadaptativo do agente vendedor é executado. Esse processo procura por outro lado verificar as restrições da loja virtual para ver se é possível fazer a venda da mercadoria ao usuário. Entretanto acontece algo diferente do primeiro cenário de uso (Ver Seção 7.2). O agente ao executar o algoritmo de detecção e resolução de

conflitos (Ver Seção 7.2.1.2) percebe que as normas ***N4*** e ***N5*** são normas conflitantes, ou seja, agem sobre a mesma ação, mas com conceitos deônticos distintos, uma de obrigação e a outra de permissão. Com isso, o agente precisará decidir qual das normas cumprir e qual delas será violada. Essa tomada de decisão é realizada através das atividades de Coleta, Análise, Decisão e Efetuação, as quais compõem o processo de autoadaptação.

A atividade de Coleta é executada através do algoritmo 1 (Ver Seção 7.2.1.1) assim que o *user agent* recebe o papel de *vendedor* e começa a atuar no mercado virtual. Com isso, o *user agent* entende as normas ***N1***, ***N3***, ***N4*** e ***N5*** são endereçadas a ele no sistema.

Em seguida, a atividade de Análise é executada em relação às normas endereçadas ao papel de vendedor. O *user agent* começa o processo verificando quais as normas estão ativas ou expiraram (Algoritmo 2, Ver Seção 7.2.1.2), depois verifica quais normas já foram cumpridas por ele (Algoritmo 3, Ver Seção 7.2.1.2), em seguida avalia a contribuição normativa de cada norma (Algoritmo 4, Ver Seção 7.2.1.2), e finalmente o algoritmo de detecção e solução de conflito entre normas (Algoritmo 5, Ver Seção 7.2.1.2). No algoritmo 5 percebe-se o conflito entre as normas ***N4*** e ***N5***. A norma ***N4*** fala que o vendedor só pode (norma permissiva) remarcar o preço de suas mercadorias antes da loja *abrir*, entretanto, a norma ***N5*** diz que o *vendedor* tem que (norma obrigatória) remarcar os preços dos produtos quando for anunciada uma *promoção* e a loja estiver aberta.

A Figura 58 mostra um pedaço do código desenvolvido no *framework* JSAN 2.0 para detecção e resolução de conflitos entre normas (parte do algoritmo 5, ver Seção 7.2.1.2). São comparadas todas as normas do ambiente endereçadas ao papel de vendedor pelo agente. As normas ***N4*** e ***N5*** entram na condição de detecção de conflito (linha 284 a 293), ***N4*** de permissão e ***N5*** de obrigação com a sua ativação posterior ao prazo de expiração de ***N4***. Assim, calculando a contribuição normativa de ***N4*** e ***N5*** com o algoritmo 4 (Ver Seção 7.2.1.2) tem-se que:

A contribuição normativa de ***N5*** dado o algoritmo 4 (Ver Seção 7.2.1.2), é a soma do conceito deôntico ser de obrigação (CDO) com valor 1, mais a recompensa por cumprir com a norma ser um objetivo do agente (RCN) com valor 1. Assim, a contribuição normativa de N5 é: CDO + RCN = 2. Já a contribuição normativa de N4 é o conceito deôntico de permissão para o objetivo do agente (CDPE) com valor 1, mais o cálculo das punições recebidas (PR) que é de -2, logo a contribuição

normativa de N4 é CDPE + PR = -1. Portanto ***N5*** tem uma contribuição normativa maior do que ***N4***.

```
284 if (    (normContribution1.getDeonticConcept().equalsIgnoreCase("obligation") &&
285          normContribution2.getDeonticConcept().equalsIgnoreCase("permission"))
286         &&
287         ((normContribution1.getActivation() < normContribution2.getActivation() &&
288           normContribution1.getExpiration() >= normContribution2.getExpiration()
289          )
290                 ||
291         (normContribution1.getActivation() > normContribution2.getExpiration() ||
292          normContribution1.getExpiration() < normContribution2.getActivation()))
293    ){
294     if (unifier.unifies(term1, term2)) {
295         if ((normContribution1.getFullfillContribution()+normContribution2.getViolateContribution())
296                 >= (normContribution2.getFullfillContribution()+normContribution1.getViolateContribution())) {
297             deletedNorms.add(normContribution2);
298
299         }else{
300             if ((normContribution2.getFullfillContribution()+normContribution1.getViolateContribution())
301                     > (normContribution1.getFullfillContribution()+normContribution2.getViolateContribution())) {
302                 deletedNorms.add(normContribution1);
303
304             }
305         }
306     }
307 }
```

Figura 58 – Código parcial de detecção e resolução de conflitos

Logo depois, a atividade de Decisão é executada para escolher o melhor plano para atender à solicitação de venda, dada as seguintes variáveis: (i) a satisfação de realizar uma ação; (ii) a motivação de realizar um objetivo individual; e (iii) a contribuição de realizar uma dada norma (Ver Algoritmo 6, Seção 7.2.1.3). Na tomada de decisão, o *framework* JSAN 2.0 calcula para todos os planos do agente a importância de cumprir ou violar uma norma. Por exemplo, a importância da norma ***N5*** ser cumprida é a soma da *satisfação* de realizar a ação de remarcar preço, de valor 7, somada à contribuição normativa de 2. Logo a importância de realizar aquele comportamento é de 9. Depois serão comparados todos os planos e aquele que for o de maior importância será adicionado à base de intenções do agente para ser realizado.

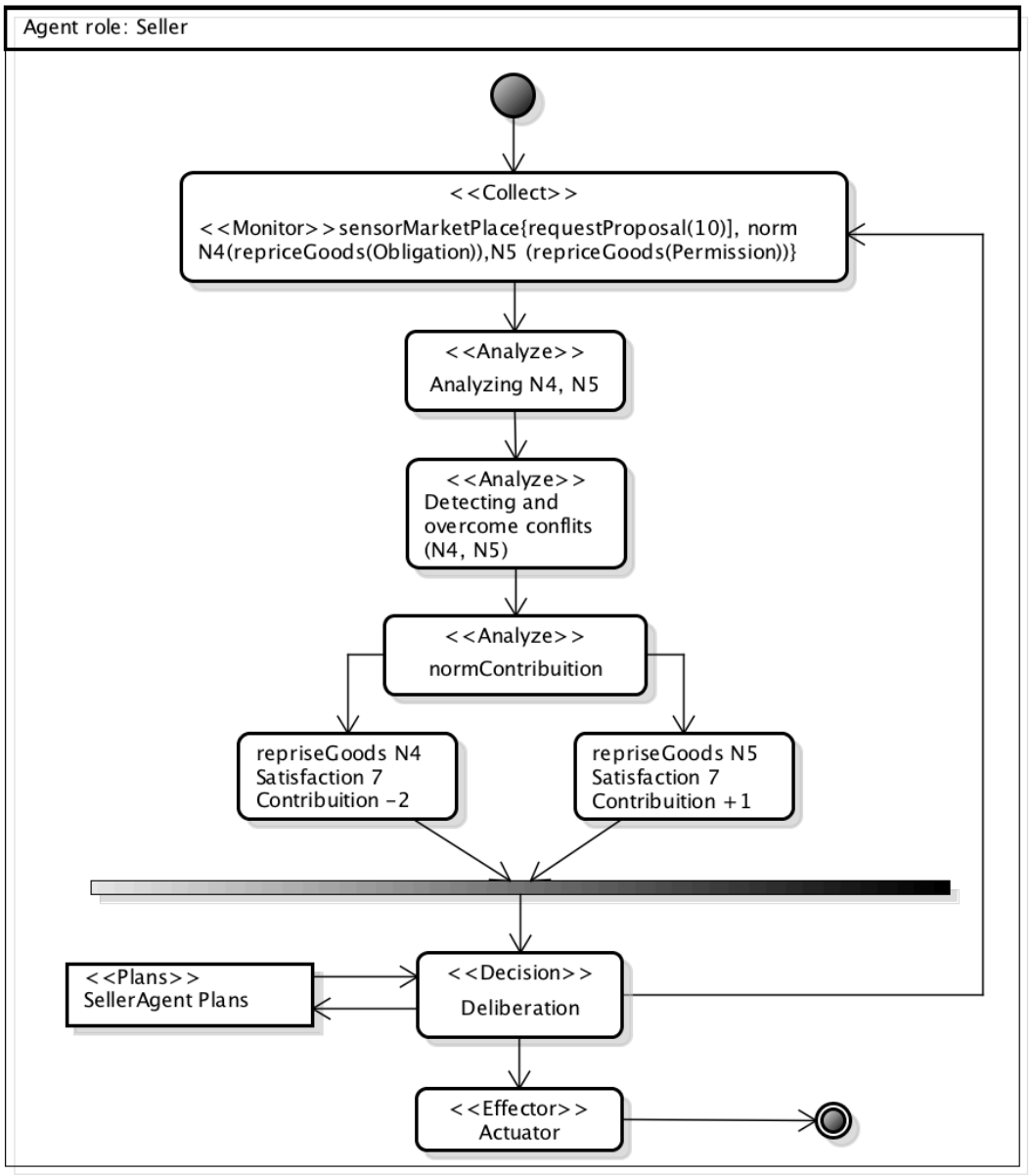

Figura 59 – Diagrama de atividades do *store agent* no papel de *Seller*

Em seguida, a atividade de Efetuação é realizada dado o plano escolhido na atividade de Decisão (Ver Seção 7.2.1.4). Caso contrário, a atividade Decisão é executada novamente para escolher outro plano candidato. Esse ciclo de execução continua até que o plano seja considerado apto para realizar a remarção de preço do produto com o mercado virtual aberto. A Figura 59 mostra o diagrama de atividades do agente no papel de *Seller*.

9
Conclusões e Trabalhos Futuros

Engenharia de software baseada em agentes é um poderoso paradigma para design e a implementação de software (Gonçalves et al., 2015; Lind, 2001; Wooldridge et al., 2001). Sistemas desenvolvidos com a tecnologia baseada em agentes de software requerem metodologias, linguagens de modelagem, plataformas de desenvolvimento e linguagens de programação que explorem seus benefícios e características próprias. Depois de analisar muitas das linguagens de modelagem para SMAs publicadas na literatura (Gowri, 2014; Gómez-Rodríguez, 2014; Bresciani et al., 2004; Van Riemsdijk et al., 2015a; Mefteh, 2015; Cernuzzi et al., 2014; Gonçalves et al., 2015; Da Silva Figueiredo et al., 2011; Beydoun et al., 2009), percebeu-se a falta de linguagens de modelagem que exploram o uso de abstrações relacionadas a agentes e promovam o refinamento dos modelos de design para código.

Além disso, a maioria das linguagens de modelagem não modelam os aspectos estruturais (entidades e relacionamentos) tampouco os aspectos dinâmicos (comportamento independente do domínio) normalmente descritos em SMAs e definidos no modelo conceitual ANA, como normas e adaptação. Algumas propostas descrevem um subconjunto dessas abstrações, e outras não modelam a interação entre as entidades definidas. A diferença entre o metamodelo proposto em ANA e os demais apresentados como trabalhos relacionados, foi a incorporação de abstrações pouco discutidas até o momento para SMAs em um nível menor de granularidade. Neste nível foi possível mostrar como as entidades de um SMA e suas propriedades relacionam com as propriedade de normas e do ciclo de adaptação.

A fim de definir uma linguagem de modelagem para SMAs que contemple todos os conceitos descritos em ANA, foi proposta a linguagem de modelagem ANA-ML (Viana, 2016). ANA-ML é uma linguagem de modelagem que estende MAS-ML com base nos aspectos estruturais e dinâmicos apresentados no modelo conceitual ANA. Com a incorporação das abstrações de normas e adaptação como

metaclasses e seus atributos como estereótipos, foi possível criar modelos com uma maior facilidade e menor esforço. Isto foi demonstrado no capítulo 6, através de um estudo realizado com pessoas para validar o uso da linguagem de modelagem ANA- ML. Os usuários que participaram do experimento perceberam que a inserção destes conceitos facilitou na modelagem de sistemas complexos, apresentados no capítulo 1.

Para auxiliar a construção de agentes de software autoadaptativos normativos, ou seja, agentes capazes de executar um processo de autoadaptação para lidar com questões normativas, a arquitetura ANA-BDI foi proposta e posteriormente, sua implementação foi realizada no *framework* JSAN 2.0. Assim, agentes desenvolvidos a partir da abordagem proposta são capazes de realizar tarefas, tais como: (i) verificar as normas do sistema que são dirigidas a ele; (ii) distinguir normas ativas ou expiradas; (iii) verificar quais as normas foram cumpridas ou violadas; (iv) avaliar os efeitos de cumprir com uma norma ou violar dado a contribuição normativa e a importância de realizar um determinado comportamento; (v) detectar e resolver conflitos entre normas; e (vi) efetivar o cumprimento ou violação dessas normas dado as escolhas dos planos.

Finalmente, os resultados apresentados mostraram um resultado muito interessante que é a de utilizar agentes capazes de se adaptarem para lidar com normas restritivas. Conseguindo assim um equilíbrio entre os desejos do agente e os objetivos organizacionais do ambiente que ele reside.

9.1 Limitações do Trabalho

A seguir são apresentadas as principais desvantagens das abordagens apresentadas na tese:

- Como ANA-ML baseia-se no metamodelo ANA para criar os diagramas de UML é necessário entender todos os conceitos definidos em ANA. Como há muitas novas abstrações no modelo conceitual, entender todas as definições, relacionamentos e diferentes tipos de interações é uma tarefa muito complexa.

- Os diagramas de ANA-ML podem ficar muito grandes quando sistemas grandes são modelados e todos os elementos do diagrama são expressos usando a forma completa (sem omitir qualquer compartimento).
- O diagrama de atividades pode se tornar muito complexo dada a quantidade de ações e planejamentos que são realizados dadas as condições e restrições do ambiente.
- Os elementos regulados pelas normas podem ser objetivos ou ações. No entanto, a arquitetura ANA-BDI ser estendido para regular conjunto de planos, ações ou objetivos.
- Agentes criados em JSAN 2.0 não considera os interesses dos outros agentes do sistema. Isto pode levar o agente a violar todas as normas, levando-o a um comportamento extremamente individual (Santos Neto, 2012b);
- O agente só tem a capacidade de ver as normas endereçadas a ele; não consegue saber as normas endereçadas aos outros papéis do sistema. Isto pode levar o sistema ao declínio;
- Um sistema de governança não é levado em consideração, assim podem surgir situações onde agentes podem criar normas a fim de satisfazer seus interesses pessoais (Lopez e Marquez, 2004);
- As recompensas e punições vêm atreladas às normas, mas o agente não avalia que é o responsável por fornecê-las, ou seja, é assumido que a entidade fornecedora seja sempre confiável. É interessante verificar o quão confiável é esta entidade, como já discutido em (Silva, 2008);

9.2 Trabalhos Futuros

Alguns trabalhos futuros foram identificados a partir das abordagens oferecidas no documento. Veja a seguir esses trabalhos:

- Uma ferramenta de modelagem ANA-ML deve ajudar o desenvolvedor de sistemas multiagentes durante a modelagem e a implementação de suas aplicações. Os objetivos da ferramenta de modelagem são simplificar e acelerar os designs de diagramas ANA-ML. Além disso, a ferramenta deverá possibilitar a tradução automática dos modelos para código;

- A formalização da linguagem ANA-ML traria alguns benefícios de uma semântica precisa que são: clareza, equivalência, consistência, capacidade de estender, refinamento e prova (Evans et al., 1999);
- Inserir o conceito de sistemas de Governança (Silva, 2007b) para ter uma maior transparência na interação entre os agentes, além de regular os recursos do sistema através das normas que poderão ser criadas por entidades do sistema. Garantir também que o agente que violou a norma sofra a punição. A avaliação do comportamento do agente seria baseada nos testemunhos do agente que recebe de outros agentes sobre violações de normas;
- Planos de reparação podem ser criados, com o objetivo de reparar o sistema caso as normas sejam violadas;
- Diferenciar as normas organizacionais das normas individuais. As normas organizacionais, que são definidas pela organização, restringem o comportamento dos agentes que desempenham papéis na organização e são punidos dadas as suas violações. Já as normas individuais são quando os agentes têm expectativas sobre o comportamento de outros agentes. Então, como seriam definidas? Como saber quais são os agentes que melhor atingem estas expectativas? Assim, estas normas serão definidas por um agente e as violações não recebem as punições vindas da organização.
- Como o valor de motivação em realizar um objetivo ou a satisfação em efetuar uma ação é uma função matemática que mapeia estes em um inteiro, e esses valores podem ser os mesmos, a tomada de decisão para priorizar qual objetivo ou ação realizar primeiro poderá levar em contar fatores discutidos em (Barbosa et al., 2014), como os traços de personalidade – atitudes e emoções, que poderiam resolver o problema de qual objetivo priorizar dado que esses têm o mesmo valor de motivação.

10
Referências bibliográficas

ATZORI, LUIGI, ANTONIO IERA, AND GIACOMO MORABITO. "**The internet of things: A survey**." Computer networks 54.15, 2010.

BAIA, DE MEDEIROS, D. **Modelagem de Contextos Dinâmicos em Simulação de Gestão de Projetos de Software Baseada em Multiagentes**. Tese de Doutorado. PUC-rio, 2016.

BARBIER, F., CARIOU, E., GOAER, O. L., & PIERRE, S. **Software Adaptation: Classification and a Case Study with State Chart XML**. IEEE Software, 32(5), 2015.

BARBOSA, S. D., GUILHERME DA SILVA, F. A., FURTADO, A. L., & CASANOVA, M. A. **Plot Generation with Character-Based Decisions**. Computers in Entertainment (CIE), 2014.

BEAMON, BENITA M., AND STEPHEN A. KOTLEBA. "**Inventory management support systems for emergency humanitarian relief operations in South Sudan**." The International Journal of Logistics Management 17.2, 2006.

BEHESHTI, RAHMATOLLAH; ALI, AWRAD MOHAMMED; SUKTHANKAR, GITA REESE. **Cognitive Social Learners: An Architecture for Modeling Normative Behavior**. In: AAAI. 2015. p. 2017-2023.

BERNON, C., GLEIZES, M. P., PEYRUQUEOU, S., & PICARD, G. **ADELFE: a methodology for adaptive multi-agent systems engineering**. In Engineering Societies in the Agents World III (pp. 156-169). Springer Berlin Heidelberg, 2003.

BEYDOUN, G., LOW, G., HENDERSON-SELLERS, B., MOURATIDIS, H., GOMEZ-SANZ, J. J., PAVON, J., & GONZALEZ-PEREZ, C. **FAML: a generic metamodel for MAS development**. Software Engineering, IEEE Transactions on, 35(6), 841-863, 2009.

BOGDANOVYCH, A., RODRÍGUEZ, J. A., SIMOFF, S., COHEN, A., & SIERRA, C. **Developing virtual heritage applications as normative multiagent systems**. In International Workshop on Agent-Oriented Software Engineering (pp. 140-154). Springer Berlin Heidelberg, 2011.

BOISSIER, O., BORDINI, R., HüBNER, J., RICCI, A., and SANTI, A. **Multi-agent oriented programming with JaCaMo**. Science of Computer Programming, (0):–. 1.2, 3.1, 2011.

BOSSE, T.; GERRITSEN, C. **An Agent-Based framework to Support Crime Prevention**, AAMAS, Toronto, 2010. 525-532.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming Multi-Agent Systemns in AgentSpeak using Jason**. [S.l.]: [s.n.], 2007.

BRATMAN, MICHAEL. **Intention, plans, and practical reason**. 1987.

BRESCIANI, P., PERINI, A., GIORGINI, P., GIUNCHIGLIA, F., & MYLOPOULOS, J. **Tropos: An agent-oriented software development methodology**. Autonomous Agents and Multi-Agent Systems, 8(3), 203-236, 2004.

BROERSEN, J., DASTANI, M., HULSTIJN, J., HUANG, Z., & VAN DER TORRE, L. **The BOID architecture: conflicts between beliefs, obligations, intentions and desires**. In Proceedings of the fifth international conference on Autonomous agents, ACM, pp. 9-16, 2001.

CAIRE, G., COULIER, W., GARIJO, F., GOMEZ, J., PAVÓN, J., LEAL, F., & MASSONET, P. **Agent oriented analysis using MESSAGE/UML**. In International Workshop on Agent-Oriented Software Engineering, Springer Berlin Heidelberg, pp. 119-135, 2001.

CASTELFRANCHI, ROSARIA CONTE CRISTIANO. **ARE INCENTIVES GOOD ENOUGH TO ACHIEVE (INFO) SOCIAL ORDER?**. Social Order in Multiagent Systems, v. 2, p. 45, 2001.

CERNUZZI, LUCA; MOLESINI, AMBRA; OMICINI, ANDREA. **The Gaia Methodology Process**. In: Handbook on Agent-Oriented Design Processes. Springer Berlin Heidelberg, 2014. p. 141-172.

CERQUEIRA, S. L. R. et al. **Plataforma GeoRisc Engenharia da Computação Aplicada à Análise de Riscos Geo-ambientais**. PUC-RIO. Rio de Janeiro, 2009.

CERVENKA, R., & TRENCANSKY, I. **The Agent Modeling Language-AML: A Comprehensive Approach to Modeling Multi-Agent Systems**. Springer Science & Business Media, 2007.

COOK, D., J. **How Samart Is Your Home?**, Science Volume 35, p. 1579-1581, March, 2012.

CRIADO, N., ARGENTE, E., NORIEGA, P., & BOTTI, V. **Towards a normative BDI architecture for norm compliance**. COIN@ MALLOW2010, 65-81, 2010.

DA, KELING, MARC DALMAU, AND PHILIPPE ROOSE. "**A Survey of adaptation systems**." International Journal on Internet and Distributed Computing Systems 2.1, 2011.

DA SILVA FIGUEIREDO, KAREN, VIVIANE TORRES DA SILVA, AND CHRISTIANO DE OLIVEIRA BRAGA. **Modeling norms in multi-agent systems**

with NormML. Coordination, organizations, institutions, and norms in agent systems VI. Springer Berlin Heidelberg, 2011. 39-57.

DAM, HOA KHANH; WINIKOFF, MICHAEL. **Towards a next-generation AOSE methodology**. Science of Computer Programming, v. 78, n. 6, p. 684-694, 2013.

DASTANI, MEHDI; VAN DER TORRE, LEENDERT. **Programming BOID-Plan Agents deliberating about conflicts among defeasible mental attitudes and plans**. In: Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on. IEEE, 2004. p. 706-713.

D'INVERNO, M.; LUCK, M. **Understanding Agent Systems**. New York: Springer. 2001.

DONG-HUYNHA, T.; JENNINGS, N.; SHADBOLT, N. FIRE: **An integrated trust and reputation model for open multi-agent systems**. In: 16th European Conference on Artificial Intelligence, Valencia, Spain. IOS Press, 2004. p. 18-22.

EVANS, A. S.; FRANCE, R. B.; LANO, K. C.; RUMPE, B. **Metamodeling semantics of UML**. In: KILOV, H. (Ed.), Behavioral Specifications for Businesses and Systems. Kluwer, 1999.

FELICÍSSIMO, CAROLINA HOWARD; DE LUCENA, CARLOS JOSÉ PEREIRA; BRIOT, JEAN-PIERRE. **An approach to operationalize regulative norms in multiagent systems**. INTECH Open Access Publisher, 2011.

FIGUEIREDO, KAREN, AND VIVIANE TORRES DA SILVA. **Norm-ML: A Modeling Language to Model Norms**. ICAART (2), edited by J. Filipe and ALN Fred (2011): 232-237.

FIPA. **FIPA Agent Management Specification**, 2002. Disponível em: <http://www.fipa.org/ specs/fipa00023/SC00023J.html>. Acesso em: 16 fev. 2016.

GARCIA, A., LUCENA, C., ZAMBONELLI, F., OMICINI, A., & CASTRO, J. (Eds.). **Software Engineering for large-scale multi-agent systems: research issues and practical applications** (Vol. 2603). Springer, 2003.

GÓMEZ-RODRÍGUEZ, A., FUENTES-FERNÁNDEZ, R., GONZÁLEZ-MORENO, J. C., & RODRÍGUEZ-MARTÍNEZ, F. J. **INGENIAS with the unified development process. In Handbook on Agent-Oriented Design Processes.** Springer Berlin Heidelberg. (pp. 371-405), 2014.

GONÇALVES, E. J. T., CORTÉS, M. I., CAMPOS, G. A. L., LOPES, Y. S., FREIRE, E. S., DA SILVA, V. T., ... & DE OLIVEIRA, M. A. **MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures**. Journal of Systems and Software, 108, 77-109, 2015.

GOWRI, R., KANMANI, S., PUNITHA, D. **Tropos based adaptation framework for self-adaptive system**. Journal of Theoretical and Applied Information Technology, 63(3), 790-799, 2014.

GUBBI, J., BUYYA, R., MARUSIC, S., & PALANISWAMI, M. **Internet of Things (IoT): A vision, architectural elements, and future directions**. Future Generation Computer Systems, 29(7), 1645-1660, 2013.

HE, M.; JENNINGS, N.; LEUNG, H. **On agent-mediated electronic commerce**. In: IEEE Transaction on Knowledge and Data Engineering, volume 15, n.4, p. 985-1003. 2003.

HOLLANDER, Christopher D.; WU, ANNIE S. **The current state of normative agent-based systems**. Journal of Artificial Societies and Social Simulation, v. 14, n. 2, p. 6, 2011.

HUBER, N., VAN HOORN, A., KOZIOLEK, A., BROSIG, F., & KOUNEV, S. **Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments**. Service Oriented Computing and Applications, 8(1), 73-89, 2014.

HÜBNER, J. F., SICHMAN, J. S., AND BOISSIER, O. **S-moise+: A middleware for developing organized multi-agent systems**. In COIN I, volume 3913 of LNAI, pages 64–78. Springer. 3.1, 2006.

IBM. **An architectural blueprint for autonomic computing – Technical Report**. 2003.

JANSSEN, M., LEE, J., BHAROSA, N., & CRESSWELL, A. **Advances in multi-agency disaster management: Key elements in disaster research**. Information Systems Frontiers, 12(1), 1-7, 2010.

JACOBSON, I., BOOCH, G., RUMBAUGH, J., RUMBAUGH, J., & BOOCH, G. **The unified software development process** (Vol. 1). Reading: Addison- wesley, 1999.

JENNINGS, NICHOLAS R.; WOOLDRIDGE, MICHAEL J. **Software agents**. IEE review, p. 17-20, 1996.

JENNINGS, N. R.; WOOLDRIDGE, M. **Applications of Intelligent Agents**. Em: Jennings, N. R.; Wooldridge, M (eds), Agent Technology: Fouhdations, Appplications, and Markets, volume 3, n.28. 1999.

JENNINGS, N. R.; WOOLDRIDGE, M. **Agent-Oriented Software Engineering**, Handbook of Agent Technology, AAAI/MIT Press, 2000.

KALAREH, MEHDI AMOUI. **Evolving Software Systems for Self-Adaptation**. 2012. Tese de Doutorado. University of Waterloo.

LAROUM, TOUFIK, AND BORNIA TIGHIOUART. "**A multi-agent system for the modelling of the HIV infection.**" Agent and Multi-Agent Systems: Technologies and Applications. Springer Berlin Heidelberg, p.94-102, 2011.

LEITE, J. C. S. P. **Livro Vivo: Engenharia de Requisitos**. 2008.

LIND, J. **Issues in agent-oriented software engineering**. Em: CIANCARINI, P.; WOOLDRIDGE, M. (Eds.) Agent-Oriented Software Engineering, LNCS 1957, Germany: Springer, p.45-58. 2001.

LOMUSCIO, A. R.; WOOLDRIDGE, M.; JENNINGS, N. **A classification scheme for negotiation in electronic commerce**. Em: International Journal of Group Decision and Negotiation, v.12, n.1, p.31-56. 2003.

LÓPEZ, FABIOLA LÓPEZ; LUCK, MICHAEL; D'INVERNO, MARK. **Constraining Autonomy through Norms**, 2002.

LÓPEZ, FABIOLA LÓPEZ. **Social Power and Norms**. Diss. University of Southampton, 2003.

LOPEZ, F. and MARQUEZ, A. **An architecture for autonomous normative agents**. In: Proc. of the 5th Int. Conf. in Computer Science. 2004.

LUCENA, C. J. P. D. **Inteligência Artificial e Engenharia de Software**. Rio de Janeiro: Zahar, 1987.

MACHADO, RODRIGO; BORDINI, RAFAEL H. **Running AgentSpeak (L) agents on SIM_AGENT**. In: Intelligent Agents VIII. Springer Berlin Heidelberg, 2001. p. 158-174.

MEFTEH, W., MIGEON, F., GLEIZES, M. P., & GARGOURI, F. (2015). **ADELFE 3.0 Design, Building Adaptive Multi Agent Systems Based on Simulation a Case Study. In Computational Collective Intelligence**. Springer International Publishing, pp. 19-28, 2015.

MENEGUZZI, FELIPE, AND MICHAEL LUCK. "**Norm-based behaviour modification in BDI agents**." Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

MERRIAM-WEBSTER. Merriam-Webster **Online Dictionary**. http://www.merriam-webster.com/dictionary/norm, 2015.

NALLUR, VIVEK; BAHSOON, RAMI. **A decentralized self-adaptation mechanism for service-based applications in the cloud**. Software Engineering, IEEE Transactions on, v. 39, n. 5, p. 591-612, 2013.

OHMURA, HIDEFUMI, DAISUKE KATAGAMI, AND KATSUMI NITTA. "**Development of social adaptive agents in simulation game of cross-cultural**

experience." Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on. IEEE, 2009.

OXFORD DICTIONARY OF SCIENCE. **Oxford Dictionaries Language Matters**. http://www.oxforddictionaries.com/definition/english/adaptation, 2015.

PAVÓN, JUAN, AND JORGE GÓMEZ-SANZ. **Agent oriented software engineering with INGENIAS**. Multi-Agent Systems and Applications III. Springer Berlin Heidelberg, 2003. 394-403.

RAO, ANAND S., AND MICHAEL P. GEORGEFF. "**BDI agents: From theory to practice**." ICMAS. Vol. 95. 1995.

RICCI, A., PIUNTI, M., VIROLI, M., & OMICINI, A. **Environment programming in CArtAgO**. In Multi-Agent Programming, Springer US, pp. 259-288, 2009.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence - A Modern Approach**. New Jersey, USA: Prentice Hall, 2009. 1152 p.

SARDINHA, J. A. R., CHOREN, R., DA SILVA, V. T., MILIDIÚ, R., & DE LUCENA, C. J. **A combined specification language and development framework for agent-based application engineering**. Journal of Systems and Software, 79(11), 1565-1577, 2006.

SANTOS NETO, B. F., Viviane Torres Da Silva, and Carlos Jose Pereira de Lucena. "**Using jason to develop normative agents**." Advances in Artificial Intelligence–SBIA 2010. Springer Berlin Heidelberg,p. 143-152, 2010a.

SANTOS NETO, B. F. **Uma abordagem deontica para o desenvolvimento de agentes normativos autônomos**. Diss. Tese de doutorado. Rio de Janeiro: PUC, Departamento de Informática, 2012b.

SEN, S., and AIRIAU, S. **Emergence of norms through social learning**. In Proceedings of the International Joint Conference on Artifical Intelligence, 2007. 1507–1512.

SHAN, LIJUN; DU, CHENGLIE; ZHU, HONG. **Modeling and Simulating Adaptive Multi-agent Systems with CAMLE**. In: Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. IEEE, 2015. p. 147-152.

SILVA, V. T. **Uma Linguagem de Modelagem para Sistemas Multi-agentes Baseada em um Framework Conceitual para Agentes e Objetos**. PUC-Rio, Rio de Janeiro –RJ. Brazil, 2004.

SILVA, V., GARCIA, A., BRANDÃO, A., CHAVEZ, C., LUCENA, C., & ALENCAR, P. **Taming agents and objects in software engineering**. In International Workshop on Software Engineering for Large-Scale Multi-agent Systems, Springer Berlin Heidelberg. pp. 1-26, 2002.

SILVA, V. T., R. CHOREN, and C.J.P. de LUCENA, "**MAS-ML: A Multi-Agent System Modeling Language**," Int'l J. Agent-Oriented Software Eng., vol. 2, no. 4, pp. 381-421, 2008.

SILVA, V. T. D.; LUCENA, C. J. P. D. **Modeling Multi-agent Systems**, Communications of ACM, 2007a. 103-108.

SILVA, V.; DURAN, F.; GUEDES, J., LUCENA, C. "**Governing Multi-Agent Systems**", In Journal of Brazilian Computer Society, special issue on Software Engineering for Multi-Agent Systems, number 2 volume 13, SBC, pp. 19-34, 2007b.

UML: **Unified Modeling Language Specification**, Version 2.0, OMG, Available in:: <http://www.omg.org/uml/>. Accessed: February 21 2015.

VAN RIEMSDIJK, M. B., DENNIS, L., FISHER, M., & HINDRIKS, K. V. **A semantic framework for socially adaptive agents: Towards strong norm compliance**. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, (pp. 423-432), 2015a.

VAN RIEMSDIJK, M. BIRNA; JONKER, CATHOLIJN M.; LESSER, VICTOR. **Creating Socially Adaptive Electronic Partners: Interaction, Reasoning and Ethical Challenges**. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2015b. p. 1201-1206.

VASCONCELOS, WAMBERTO; KOLLINGBAUM, MARTIN J.; NORMAN, TIMOTHY J. **Resolving conflict and inconsistency in norm-regulated virtual organizations**. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM, 2007. p. 91.

VERHAGEN, HENRICUS JE. **Norm autonomous agents**. Diss. Stockholm Universitet, 2000.

VIANA, M. L.; CUNHA, F. J. P. ; BALDOINO F. SANTOS NETO ; PAULO ALENCAR; LUCENA, C. . **A Framework for Supporting Simulation with Normative Agents**. In: WESAAC, 2015, Niteroí. 9 Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações. Niteroí: Niteroí: UFF, v. 9. p. 167-172, 2015a.

VIANA, M. L.; PAULO ALENCAR; EVERTON GUIMARÃES; CUNHA, F. J. P.; DONALD COWAN; LUCENA, C. **JSAN: A Framework to Implement Normative Agents**. In: SEKE, 2015, Pittsburgh. The 27th International Conference on Software Engineering & Knowledge Engineering, p. 660-665, 2015b.

VIANA, M. L.; PAULO ALENCAR; DONALD COWAN; EVERTON GUIMARÃES; CUNHA, F. J. P. ; LUCENA, C. . **The Development of Normative Autonomous Agents**: an Approach. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2015, Cingapura. 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, p. 9-16, 2015c.

VIANA, M. L.; LUCENA, C.. **An Approach to the Design of Adaptive and Normative Software Agents**. In: CBSOFT, 2015, Belo Horizinte. WTDSoft 2015 5th Workshop on Theses and Dissertations of Cbsoft, v. 01. p. 75-82, 2015d.

VIANA, M. L.; PAULO ALENCAR; LUCENA, C. . **A Metamodel Approach to Developing Adaptive Normative Agents**. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2015, Cingapura. 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, p. 89-91, 2015e.

VIANA, M. L. ; PAULO ALENCAR ; LUCENA, C. . **A Modeling Language for Adaptive Normative Agents**. In: EUMAS, 2016, Valencia. European Conference on Multi-Agent Systems, 2016. (to appear)

YU, L.; SCHMID, B. **A Conceptual Framework for Agent-Oriented and Role-Based Work on Modeling**. Em: WAGNER, G.; YU, E. (Eds.). Anais do 1st International Workshop on Agent-Oriented Information Systems, 1999.

WEISS, Gerhard. **Multiagent systems: a modern approach to distributed artificial intelligence**. MIT press, 1999.

WINER, B_J. **Latin squares and related designs**. 1962.

WOOLDRIDGE, MICHAEL; JENNINGS, NICHOLAS R.; KINNY, DAVID. **A methodology for agent-oriented analysis and design**. In: In Proceedings of the Third International Conference on Autonomous Agents Agents 99. 1999.

WOOLDRIDGE, M.; CIANCARINI, P. **Agent-Oriented Software Engineering: the State of the Art**. Em: CIANCARINI, P.; WOOLDRIDGE, M. (Eds.) Agent-Oriented Software Engineering, LNCS 1957, Berlin: Springer, p. 1-28. 2001.

WOOLDRIDGE, MICHAEL. **An introduction to multiagent systems**. John Wiley & Sons, 2011.

ZAMBONELLI, F., JENNINGS, N. R., & WOOLDRIDGE, M. **Developing multiagent systems: The Gaia methodology**. ACM Transactions on Software Engineering and Methodology (TOSEM), 12(3), 317-370, 2003.

A
Questionários Utilizados no Estudo

A.1
Formação Acadêmica, Conhecimentos Específicos e Experiência

Parte 1. Conhecimentos sobre o Participante:

Nome: _____
Tempo que já trabalhou com modelagem de software: _____
Tipos de modelagem com que trabalha ou já trabalhou (ex: diagrama de classe, sequência, atividades, etc.):
Tempo de experiência com desenvolvimento de software: _____
Formação Acadêmica: _____
Nível da pós-graduação: _____
Conhecimento em UML: ____Alto ____Médio ____Baixo ____Nenhum
Conhecimento em SMA: ____Alto ____Médio ____Baixo ____Nenhum
Conhecimento em MAS- ML: ___Alto ___Médio ____Baixo ____Nenhum

Parte 2. Para todas as questões abaixo, por favor, informe o horário que iniciou a tarefa, o horário em que terminou e o grau de dificuldade em respondê-la.

A ideia central desse questionário é solicitar mudanças em diagramas representados na linguagem de modelagem ANA-ML. O cenário a ser testado é a Prevenção de Crimes. Como alternativa ao uso de padrões de deslocamentos antigos, criminologistas têm unido forças com pesquisadores da Ciência da Computação e

Inteligência Artificial para explorar os benefícios de simulações sociais (Bosse e Gerritsen, 2010), com o uso de agentes, para investigar o deslocamento de crimes. Assim, a perspectiva desta abordagem é usar os ambientes de simulação para predizer a dinâmica do deslocamento de crimes no futuro, melhor que analisar as dinâmicas do passado.

A inserção de normas nessa simulação acontece para regular o comportamento dos agentes guardas. Para incentivar o cumprimento dessas normas, algumas sanções foram especificas para cada norma. Note que as sanções são normas que são ativadas quando uma norma é violada ou cumprida. Portanto, procure atender TODAS as solicitações a seguir:

Atividade 1

Horário Inicial _____:_____:_____

Modelar o ambiente da simulação, com sua organização principal, normas e agentes:

a) Criar o ambiente *Crime Prevention*, fechado.
b) Criar a organização principal *City-Organization*;
c) Agente Pessoa com os papéis de:
    i. Guarda
    ii. Civil
    iii. Bandido
d) Norma N1: Se o número de *Guardas* é menor do que o número de bandidos, guardas são proibidos de efetuar prisão.
    i. Punição: Perda de sua reputação.
e) Norma N2: Se *Guardas* estão indo efetuar prisões, comandantes são obrigados a oferecer recursos.
    i. Recompensa 1: Aumento da reputação dos comandantes.
    ii. Recompensa 2: Envio de reforço blindado.
    iii. Punição: Perda de sua reputação.
f) Norma 3: Se civis estão sendo assaltados em uma determinada área da cidade, guardas são obrigados a se deslocarem para este local.
    i. Recompensas: Se deslocamento é provido:
    Recompensa 1: A reputação dos guardas é aumentada.

Recompensa 2: Mais guardas são disponibilizados para ajudar na prisão de criminosos.

ii. Punição 1: Se guardas não se deslocam para a área de assaltos, sua reputação é diminuída.

iii. Punição 2: Retornar para o batalhão de polícia.

Horário Final _____:_____:_____

Dificuldade: 1__ 2__ 3__ 4__ 5

Atividade 2

Horário Inicial _____:_____:_____

Criar agente *guarda* na prevenção de crimes. Para se criar um agente com o papel de guarda deve-se definir a relação agente X papel e outras propriedades. Para realizar essa tarefa, deve-se seguir TODAS as instruções a seguir:

a) Goal 1: "*cidadeSemCrimes*", motivação (10);

b) Goal 2: "*Aumentar reputação*", motivação (7);

   iv. Crença: "*prenderBandido*" e "*civisSalvos*";

   v. Desejo: "*reputaçãoMáxima*";

   vi. Intenção: "*prenderBandido*" e "*aumentarReputação*".

c) Sensores capturaram normas ***N1***, ***N2*** e ***N3*** endereçadas ao *guarda*.

d) Criar o plano: "*manterOrdem*", composto pelas ações: "*deslocarPelaCidade*" com *satisfação* 8, "*avaliarDensidadeCrimes*" com *satisfação* 4 e "*efetuarPrisao*" com satisfação 7. Este plano está ligado ao *goal cidadeSemCrimes*.

Horário Final _____:_____:_____

Dificuldade: 1__ 2__ 3__ 4__ 5

Atividade 3

Horário Inicial _____:_____:_____

Uma norma do sistema foi modificada e uma nova norma foi criada na organização. Portanto procure seguir TODAS as mudanças a seguir no modelo:

a) Modificar a norma "Norma ***N2***":
   i. Conceito deôntico: Permitido;
   ii. Regula: Provimento de recursos;
   iii. Endereçada: papel de guarda;
   iv. Ativada: tempo 30;
   v. Desativada: tempo 50;
   vi. Recompensa: Aumento da reputação.

b) Criar a norma "Norma ***N4***":
   i. Conceito deôntico: Obrigatória;
   ii. Regula: ação de efetuar prisão;
   iii. Endereçada: papel de guarda;
   iv. Ativa: todo período da simulação;
   v. Recompensa: Aumento da reputação;
   vi. Punição 1: Diminuir reputação;
   vii. Punição 2: Retirada de equipamentos.

Horário Final _____:_____:_____

Dificuldade: 1__ 2__ 3__ 4__ 5

Atividade 4

Horário Inicial _____:_____:_____

Quando um agente no papel de guarda não consegue realizar a prisão de bandido, por exemplo, dada a norma ***N1***, a qual diz que se o número de bandidos for maior do que o de guardas, esses são proibidos de efetuar a prisão, pois não podem colocar suas vidas e de civis em risco. Como o número de guardas se espalha pelas diferentes localidades da cidade, o processo de autoadaptação oferecido pelo JSAN 2.0 em relação a norma ***N1*** é executado pelo agente no papel de *guarda*. Esse processo procura por outras formas de fazer a prisão de bandidos, com os dados que ele tem da situação. Esse processo é composto pelas seguintes atividades: *Collect, Analyze, Decision e Effector*.

Horário Final _____:_____:_____

Dificuldade: 1__ 2__ 3__ 4__ 5