UNIVERSIDADE FEDERAL DA GRANDE DOURADOS

Marcelo Yoshio Hasegawa

Aplicação de Agentes Autônomos em Ambiente Virtual

DOURADOS – MS

2016

Marcelo Yoshio Hasegawa

Aplicação de Agentes Autônomos em Ambiente Virtual

Trabalho de Conclusão de Curso de graduação apresentado para obtenção do título de Bacharel em Sistemas de Informação pela Faculdade de Ciências Exatas e Tecnologia da Universidade Federal da Grande Dourados.
Orientador: Prof. Dr. Joinvile Batista Junior

DOURADOS – MS

2016

Marcelo Yoshio Hasegawa

Aplicação de Agentes Autônomos em Ambiente Virtual

Trabalho de Conclusão de Curso aprovado como requisito para obtenção do título de Bacharel em Sistemas de Informação na Universidade Federal da Grande Dourados, pela comissão formada por:

_____
Orientador Prof. Dr. Joinvile Batista Junior
FACET – UFGD

_____
Prof. Me. Anderson Bessa da Costa
FACET – UFGD

_____
Prof. Me. Claudia Regina Tinós Peviani
FACET – UFGD

Dourados, 29 de Abril de 2016.

RESUMO

Este trabalho tem como objetivo a prototipação de uma aplicação com a utilização de agentes autônomos em ambientes virtuais. Os agentes demonstram comportamentos como a reatividade, pró-atividade e sociabilidade. Sistemas multiagentes tem sido utilizados em Ambientes Virtuais Inteligentes que representam um ambiente virtual que se assemelha ao mundo real, habitado por entidades autônomas inteligentes que demonstram uma variedade de comportamentos. O Modelo BDI (Belief-Desire-Intention) tem sido utilizado para a implementação de aplicações baseadas em sistemas multiagentes. Neste modelo agentes definem e ativam objetivos que acionam atividades encapsuladas em forma de planos. A execução de atividades em planos pode resultar na criação de sub objetivos com o consequente acionamento de outros planos, caracterizando uma hierarquia de objetivos e planos. O desenvolvimento de uma aplicação envolvendo sistemas multiagentes em ambientes virtuais, no Modelo BDI, foi a estratégia escolhida para o aprendizado de sistemas multiagentes. O Modelo BDI é suportado pelo *framework* Jadex, com APIs disponíveis, uma interface gráfica para o desenvolvimento de agentes na linguagem Java e o suporte de um mecanismo de comunicação de agentes, locais ou remotos, através de serviços. Inicialmente foram estudadas duas aplicações de sistemas multiagentes no *framework* Jadex, no Modelo BDI. A seguir foi desenvolvido o protótipo de uma aplicação na qual um avião parte de um porta aviões e ataca alvos em terra, lançando mísseis ar-terra. Uma bateria antiaérea é utilizada na tentativa de derrubar o avião, lançando mísseis terra-ar. Nesta aplicação são utilizados todos os conceitos suportados pelo Modelo BDI: agentes, crenças, objetivos, planos, tarefas e serviços.

Palavras-Chave: sistemas multiagentes, ambientes virtuais, modelo BDI.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ACL – *Agent Communication Language*

API – Interface de Programação de Aplicação (do inglês *Application Programming Interface*)

AVI – Ambiente Virtual Inteligente

BDI – Crenças, Desejos e Intenções (do inglês *Beliefs, Desires and Intentions*)

FIPA – *Foundation for Intelligent Physical Agents*

IDE – Ambiente de Desenvolvimento Integrado (do inglês, *Integrated Development Environment*)

JADE – *Java Agent Development Framework*

POJO – *Plain Old Java Objects*

SCA – Arquitetura Componente Serviço (do inglês, *Service Component Architecture*)

XML – *Extensible Markup Language*

1. Introdução

Este trabalho se refere à prototipação de uma aplicação com a utilização de agentes autônomos em ambientes virtuais. Neste capítulo são apresentados os conceitos utilizados como base desta monografia. O capítulo é iniciado pela motivação do estudo, seguido de um fato histórico referente a uma aplicação do tema pesquisado (Seção 1.1. Histórico e Motivação). Em seguida são expostas as oportunidades de aplicações e pontos relevantes que devem ser levados em consideração que precedem o estudo (Seção 1.2. Oportunidades e Relevância). Ao final são apresentados o objetivo (Seção 1.3 Objetivos do Trabalho) e a organização desta monografia (Seções 1.4. Metodologia Adotada e 1.5. Conteúdo do Trabalho).

1.1. Histórico e Motivação

O surgimento dos agentes teve como ponto inicial o desenvolvimento da Inteligência Artificial, na qual uma de suas definições tem como objetivo a criação de agentes que demonstrem aspectos de um comportamento inteligente. Os agentes foram adotados por outras áreas da Ciência da Computação, tornando-se uma área multidisciplinar.

A definição universal do significado da palavra agente é incerta, sendo descrita de diferentes formas, características e objetivos. Em um consenso geral, um agente é um *software* cuja função é observar um ambiente e realizar ações visando um objetivo. Para Wooldrige, um agente é uma entidade situada em um ambiente que é capaz de realizar ações de forma autônoma/independente para conseguir seus objetivos (WOOLDRIGE, 2002).

Os agentes demonstram comportamentos como a reatividade (interação com o ambiente, respondendo as mudanças no ambiente), pró-atividade (alcançar objetivos, não dirigido somente por eventos, mas tomando a iniciativa e reconhecendo oportunidades) e sociabilidade (habilidade de interagir com outros agentes, similares ou humanos, e possivelmente cooperar com outros agentes).

Em sistemas multiagentes, grupos de agentes trabalham de forma cooperativa ou disputando recursos. Sistemas multiagentes fazem o uso de agentes autônomos para a implementação de aplicações com objetivos de alta complexidade ou impossíveis de serem alcançados com apenas um agente. Para tal é necessário a divisão clara das responsabilidades entre cada agente, visto que devem decidir um comportamento a ser seguido, conforme seus contextos e seus objetivos, quando uma ação é requisitada.

Um exemplo da utilização de um sistema multiagentes, no desenvolvimento de aplicações de alta complexidade, é missão espacial *Deep Space* 1, desenvolvida pela *National Aeronautics and Space Administration* (NASA). A sonda utilizada na missão espacial, lançada em 24 de outubro de 1998 na missão, de mesmo nome, *Deep Space 1*, foi construída com um sistema de controle autônomo capaz de realizar decisões importantes sem a necessidade de uma intervenção humana, permitindo que a sonda continue operando normalmente. Um caso comum em missões espaciais é a ocorrência de falhas em algum componente da sonda, onde é necessário sua identificação e decisão de isolá-lo ou desabilitá-lo. Missões anteriores à DS1 requeriam um grande número de pessoas apenas para monitorar os estados das sondas, além de uma equipe especializada em tomar decisões sobre as ações das sondas. Uma implementação de um sistema autônomo, como na DS1, torna a missão espacial mais robusta, especialmente contra falhas, e reduz de forma considerável os gastos da missão, visto que parte das pessoas responsáveis por monitorar e tomar decisões das ações das sondas, podem assumir outras funções.

Wooldrige (2002) exemplifica outros sistemas multiagentes focados na interação com um usuário de computador, como o sistema ADEPT (JENNINGS *et al*, 1996) para gerenciamento de uma organização e as relações com seus clientes, o sistema Jango (DOORENBOS *et al*, 1997) para gerenciamento de compras e o projeto OZ (BATES et al, 1992) para desenvolver agentes para ambientes virtuais onde os usuários consigam experimentar “viver” ao invés de assistir o ambiente.

Sistemas multiagentes tem sido utilizados em Ambientes Virtuais Inteligentes (AVIs). Segundo Anastassakis (2001), um AVI representa um ambiente que se assemelha ao mundo real, habito por entidades autônomas

inteligentes que demonstram uma variedade de comportamentos. Ainda segundo Anastassakis, tais AVIs podem ser aplicados em diversas áreas, principalmente áreas que fazem uso de simulações, podendo ser citados o entretenimento (jogos de computadores), a engenharia (Desenhos Assistidos por Computador) e a educação (Ambientes Virtuais de Aprendizagem).

1.2. Oportunidades e Relevância

Sistemas multiagentes têm sido suportados por várias linguagens e plataformas. Bordini *et al* (2006) cita algumas linguagens, de diferentes paradigmas, e de plataformas utilizadas para a criação de sistemas multiagentes. Dentre as plataformas citadas, pode-se destacar a plataforma JADEX, variante da plataforma JADE com a adição ao suporte ao Modelo BDI, utilizada como estudo neste trabalho.

O Modelo BDI (Belief-Desire-Intention) foi selecionado para o desenvolvimento de uma aplicação de um ambiente virtual baseado em sistemas multiagentes, em função do seguintes aspectos: (a) o modelo se popularizou muito em função de sua simplicidade; (b) o *framework* Jadex suporta o Modelo BDI e sua interação com ambientes virtuais 2D e 3D; (c) Jadex disponibiliza vários exemplos, dentro os quais dois exemplos selecionados estão sendo estudados como base para este trabalho; e (d) o *framework* Jadex suporta uma API implementada na linguagem Java, que tem sido extensivamente abordada no curso de Sistemas de Informação.

O Modelo BDI tem sido utilizado para a implementação de aplicações baseadas em sistemas multiagentes. Neste modelo agentes definem e ativam objetivos que acionam atividades encapsuladas em forma de planos. A execução de atividades em planos pode resultar na criação de subobjetivos com o consequente acionamento de outros planos, caracterizando uma hierarquia de objetivos e planos.

O desenvolvimento de uma aplicação envolvendo sistemas multiagentes em ambientes virtuais foi a estratégia escolhida para o aprendizado de sistemas multiagentes. O Modelo BDI é suportado pelo *framework* Jadex, com APIs disponíveis e uma interface gráfica (Jadex Control Center) para o

desenvolvimento de agentes na linguagem Java e suportando um mecanismo de comunicação de agentes, locais ou remotos, através de serviços.

Em versões anteriores, as implementações utilizando o *framework* Jadex dependiam fortemente de definições XML, de forma que somente os códigos de execução dos planos era definido em Java. Na sua versão mais recente, BDI v3, as implementações são realizadas quase totalmente em Java através de anotações no código Java, dispensam o usuário das definições em XML (*Extensible Markup Language*) de propriedades de agentes, objetivos e planos, simplificando substancialmente o trabalho de implementação.

O *framework* Jadex disponibiliza uma API para a utilização de ambientes virtuais em 2D e 3D. O *framework* Jadex disponibiliza ainda, exemplos de aplicações de sistemas multiagentes utilizando interfaces gráficas baseadas em componentes gráficos da API “java.Swing” e utilizando a API provida pelo Jadex para ambientes virtuais 2D e 3D.

A proposta deste trabalho se baseia no estudo e aplicações, para assimilação dos conceitos e anotações utilizadas no Modelo BDI e da comunicação entre agentes suportada pelo Jadex, resultando no desenvolvimento de uma aplicação para consolidar os conhecimentos adquiridos.

1.3. Objetivos do Trabalho

O objetivo geral deste trabalho é o desenvolvimento de uma aplicação de sistemas multiagentes em um ambiente virtual.

Os objetivos específicos são: (a) a capacitação na utilização do Modelo BDI para construir sistemas multiagentes, a partir de exemplos disponíveis no *framework* Jadex; e (b) o desenvolvimento de uma aplicação de sistema multiagentes, no Modelo BDI, utilizando a API para suporte a ambientes virtuais do Jadex.

1.4. Metodologia Adotada

A metodologia adotada pode ser descrita em quatro etapas. Na primeira etapa foram feitos estudos sobre os conceitos suportados pelo Modelo BDI, aplicado à agentes, objetivos e planos.

Na segunda etapa foram estudadas duas aplicações, disponíveis em conjunto, do *framework* do Jadex com o objetivo de refinar o entendimento a respeito dos conceitos da utilização dos conceitos do Modelo BDI e da utilização de ambientes virtuais pela Jadex. A primeira aplicação estudada faz uso de um único agente, cuja interface gráfica é baseada na API “javax.swing”. Já a segunda aplicação utiliza vários agentes e um ambiente virtual suportado pelo Jadex.

Na terceira etapa foram feitas a especificação e implementação da aplicação, com base nos conhecimentos adquiridos na etapa anterior, assim como os conceitos estudados na primeira etapa. Foram também realizados testes com a aplicação, verificando as suas principais deficiências.

Na quarta etapa foram consolidados os resultados obtidos da etapa anterior, assim como as dificuldades encontradas e futuros trabalhos.

1.5. Conteúdo do Trabalho

No Capítulo 2 é apresentada a fundamentação teórica para este trabalho: conceituação de sistemas multiagentes e do Modelo BDI, descrição do suporte a este modelo pelo *framework* Jadex, e análise dos exemplos escolhidos para entendimento do Modelo BDI e do ambiente de suporte 2D e 3D suportado pelo Jadex.

No Capítulo 3 será descrito o desenvolvimento da aplicação proposta, abordando as etapas de especificação, implementação e teste.

No Capítulo 4 serão analisados os resultados alcançados, e sugeridos aspectos de melhoria e evolução do trabalho realizado.

2. Fundamentação Teórica

As subseções deste capítulo abordam a conceituação: agentes, sistemas multiagentes, do Modelo BDI e dos *frameworks* JADE e Jadex. Adicionalmente, são descritos os estudos realizados em duas aplicações de sistemas multiagentes, disponíveis como exemplos de aplicação do *framework* Jadex, que serviram de base para o desenvolvimento da aplicação que representa o resultado principal deste trabalho.

2.1. Agentes e Sistemas Multiagentes

Um agente é um sistema informático situado em um ambiente que é capaz de realizar ações de forma autônoma para conseguir seus objetivos. (Wooldridge, 2002). De forma geral, um agente é uma entidade que observa um ambiente através de sensores e realiza ações autônomas através de atuadores, visando atingir uma meta. Para Wooldridge (2002) agentes necessitam de duas características importantes, autonomia e capacidade de comunicação. Autonomia representa a capacidade do agente em tomar decisões dado uma situação ou problema. Tal independência possibilita que ele atinja um objetivo sem a intervenção humana. Capacidade de comunicação é a habilidade do agente de se comunicar com outros agentes ou componentes para realizar a troca de informações.

Um sistema multiagentes é formado por uma coleção de agentes que interagem entre si a fim de cumprirem suas metas. Os agentes representam e agem em benefício de seus usuários, com objetivos e motivações diferentes. A interação entre agentes ocorre tipicamente através de uma infraestrutura de rede. Para que essa interação seja bem-sucedida é necessário que os agentes tenham a habilidade de cooperação, coordenação e negociação. Tais habilidades englobam protocolos de comunicação, formato de mensagens, atos da fala, entre outras técnicas.

O campo de sistema de agentes inteligentes se demonstra promissor devido sua capacidade de tratar problemas de forma autônoma através da abstração de dados e utilização de um raciocínio de alto nível. Agentes vem sendo aplicados a Mundos Virtuais, pois os mesmos proporcionam simulações de eventos do mundo real. Tendo como exemplo o sistema VITAL (Anastassakis, 2001), um sistema multiagentes que suporta aplicações de ambientes virtuais inteligentes, o sistema AdapTIVE (Santos, 2004), um ambiente virtual que tem sua estrutura e apresentação adaptada com base nos interesses e preferências de seus usuários, e o projeto SOAR/Games (LENT e LAIRD, 1998), que tem como objetivo desenvolver agentes inteligentes em jogos de computadores.

2.2. O Modelo BDI para Sistemas Multiagentes

O Modelo BDI (Belief-Desire-Intention) suporta a criação de agentes inteligentes, cujos principais conceitos são: a implementação de crenças, desejos e intenções de agentes. Este modelo representa as atitudes mentais associadas a possíveis estados do ambiente.

Crenças simbolizam as informações que o agente tem sobre si mesmo, outros agentes e do ambiente. Desejos representam os objetivos do agente, estados em que o agente deseja alcançar. Intenções representam as escolhas do agente, meios de como os desejos serão realizados. Tais conceitos têm como fundamentação a teoria de Michael Bratman, "*Intention, Plans, and Pratical Reason*" de 1999. Em sua teoria, Bratman propôs que as ações humanas são geradas a partir de suas crenças, desejos e intenções. As crenças representam as suas atitudes informacionais, desejos as suas atitudes motivacionais e intenções as suas atitudes deliberativas.

Tal modelo foi adaptado para ser computacionalmente tratável, sem perder o objetivo original do modelo. Apenas as crenças são representadas explicitamente em códigos de programação. Os Desejos são reduzidos a eventos que são manipulados por planos. Intenções são representadas implicitamente pela pilha de execução dos planos.

2.3. Frameworks JADE e Jadex

JADE (*Java Agent Development*) é um *framework* utilizado como *middleware* para o desenvolvimento de sistemas multiagentes (Gatti, 2007). Suporta um ambiente onde os agentes podem ser iniciados para execução. Contém uma biblioteca de classes que auxilia no desenvolvimento de agentes. Possui uma interface gráfica para execução, onde os agentes podem ser gerenciados, juntamente com componentes de análises estatísticas (Gatti, 2007).

O JADE segue as especificações da entidade FIPA (*Foudantion for Intelligent Physical Agenets*), que promove a interoperabilidade de agentes com outras tecnologias, através da padronização de agentes e sistemas multiagentes. As especificações da FIPA definem a interconexão de diferentes agentes em uma rede. A Figura 1 representa o modelo FIPA.

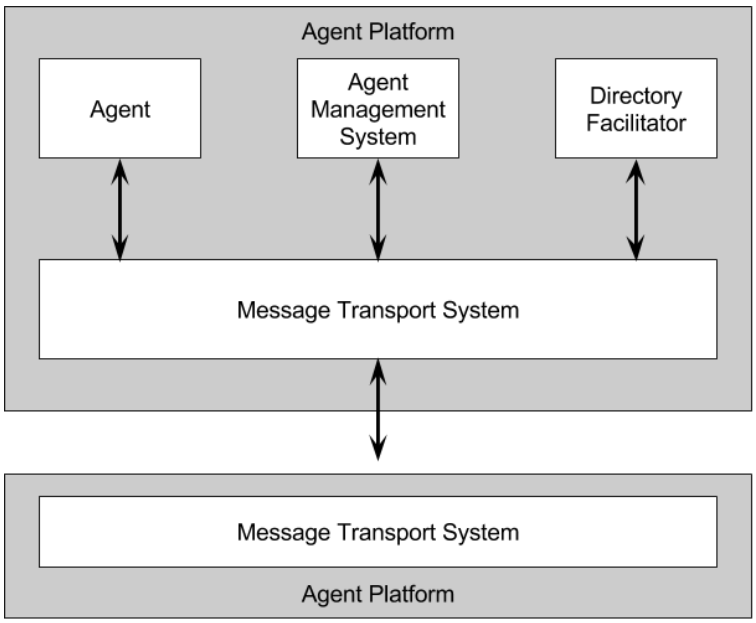

Figura 1 - Estrutura do modelo FIPA. Fonte: GATTI, 2007 (adaptado).

No modelo FIPA, os agentes são executados em contêineres. Um contêiner representa a instância de um ambiente Jade, sendo que vários agentes podem ser executados. O Jade sempre inicia um contêiner denominado *Main-Container*. Uma plataforma representa um conjunto de contêineres ativos. Todos os contêineres são conectados a um barramento, que proporciona a troca de informações entre os componentes.

Existem dois agentes especiais utilizados pela plataforma Jade, que garantem a interoperação dos outros agentes da plataforma. O *Agent Management System* (AMS) representa uma entidade, um agente, que controla o acesso e uso de outros agentes à plataforma. Todo agente deve se registrar no AMS. O *Directory Facilitator* (DF) representa uma entidade que oferece o serviço de páginas amarelas, utilizado para que os serviços possam ser encontrados pelos agentes.

O *framework* Jadex, suporta o Modelo BDI utilizando a infraestrutura da plataforma JADE. Possibilita que agentes sejam construídos com crenças planos e objetivos. O Modelo BDI foi tratado pela plataforma na forma de *tags* XML nas versões iniciais do Jadex. Tal método foi substituído pelo uso de anotações Java na versão três do Modelo BDI, BDIv3. As anotações têm como finalidade definir os componentes do Modelo BDI como sendo objetos simples da linguagem Java, os chamados POJOs (*Plain Old Java Objects*).

2.4. Mundo de Blocos

A aplicação “Mundo de Blocos” é um dos exemplos disponíveis da plataforma Jadex. A aplicação consiste de duas entidades denominadas “mesas”, um conjunto de entidades “blocos”, o usuário e um agente. Em uma das mesas o usuário posiciona os blocos, dispostos em pilhas de blocos. A outra mesa será modificada pelo agente, que tem como objetivo replicar a configuração dos blocos da mesa do usuário. A interface da aplicação, ilustrada na Figura 2, é composta pelas áreas das configurações inicial e desejada, e pelos comandos para definir a configuração desejada e para a execução da aplicação. Esta interface gráfica é implementada com componentes do pacote “javax.Swing”.

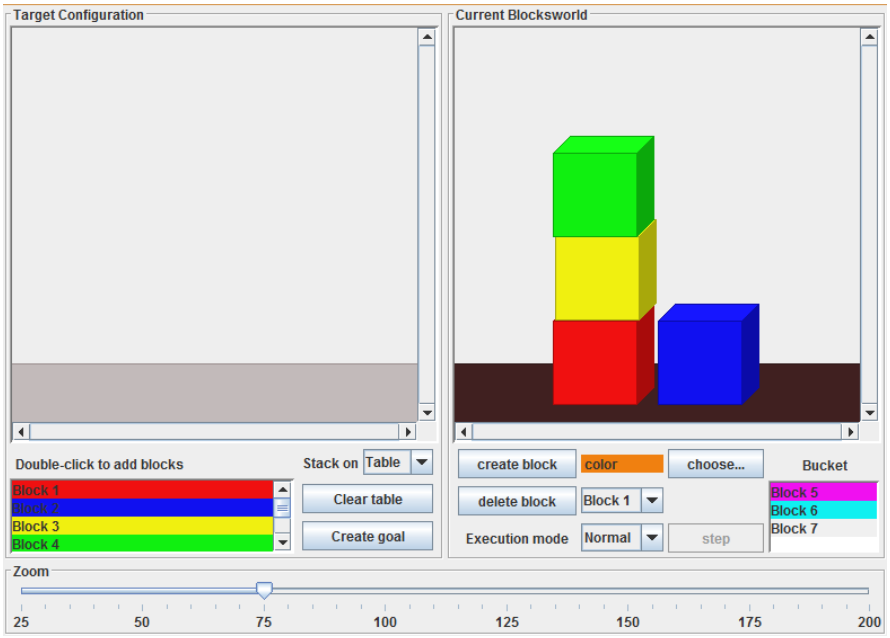

Figura 2 - Interface do Mundo de Blocos. Fonte: elaborado pelo autor.

A mesa da Configuração Alvo (*Target Configuration*) permite que o usuário posicione os blocos em uma configuração de seu interesse. O conjunto de comandos disponíveis e suas funções, da mesa da Configuração Alvo, são representados no Quadro 1.

Quadro 1 - Comandos da Mesa da Configuração Alvo

Comando	Funcionalidade
Stack on	Representa a lista de possíveis locais onde os blocos da mesa da Configuração Alvo possam ser posicionados. Inicialmente apenas a mesa se encontra disponíveis. Posteriormente, conforme os blocos são posicionados, os blocos que estão no topo de cada pilha se tornam locais disponíveis para que outro bloco seja posicionado.
Double-click to add blocks	Representa a lista de possíveis blocos para serem posicionados na respectiva mesa. Um duplo clique do mouse (botão esquerdo) posiciona o bloco selecionado na mesa, conforme a opção selecionada no *Stack on*.
Clear Table	Remove todos os blocos da respectiva mesa.
Create Goal	Realiza a criação do plano que o agente executará.
Zoom	Determina o grau de ampliação da visibilidade de ambas as mesas.

A mesa da Configuração Corrente (*Current Blocksworld*) representa o estado inicial do sistema que deverá evoluir para atingir a configuração alvo,

estabelecida pelo usuário. Com a criação de um objetivo, o agente irá alterar a configuração dos blocos da mesa até atingir a Configuração Alvo. O conjunto de comandos disponíveis e suas funções, da mesa de Configuração Corrente, são representados no Quadro 2.

Quadro 2 - Comandos da Mesa da Configuração Corrente

Comando	Funcionalidade
Create Block	Cria um novo bloco, com base na cor representada em *Color*.
Color - choose	Abre uma janela com uma palheta de cores para escolher a cor de um novo bloco.
Delete Block	Remove um bloco da lista de blocos disponíveis.
Execution Mode	Representa a lista de possíveis modos de execução do objetivo e dos planos: ▪ 'NORMAL' a execução dos planos é feita de forma automática e indisponível para o usuário, sendo mostrada apenas o resultado final do objetivo. ▪ 'STEP' a execução dos planos é feita de forma manual, controlada pelo botão "*Step*". ▪ 'SLOW' a execução dos planos é feita de forma automática, com a existência um tempo de espera de 1s entre a execução dos planos, sendo visível para o usuário.

Após determinar a Configuração Alvo, o usuário ativa a execução do agente para converter a mesa da Configuração Corrente, disparando o objetivo Configurar (*ConfigureGoal*), associado ao agente, que representa o objetivo de mais alto nível desta aplicação. A Figura 3 representa a determinação da Configuração Alvo e o escalonamento do objetivo de mais alto nível da aplicação.

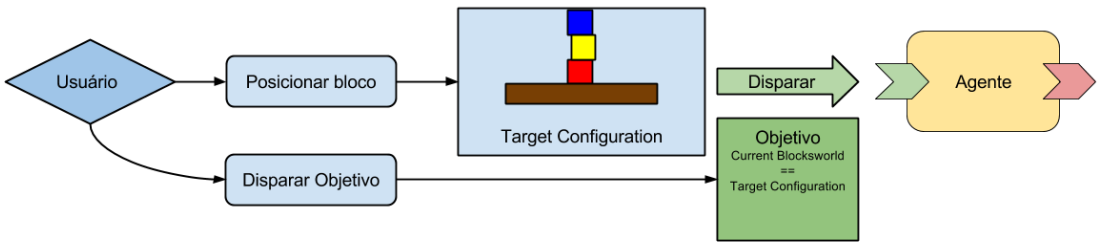

Figura 3 - Disparo de um objetivo. Fonte: elaborado pelo autor.

Ao ser ativado o objetivo de atingir a Configuração Alvo, o agente executa o plano para configurar os blocos para atingir a Configuração Alvo (*ConfigureBlocksPlan*). O plano inicialmente realiza a descoberta da configuração dos blocos da mesa da Configuração Alvo, através de iterações na entidade que armazena as pilhas de blocos.

Com base nas configurações corrente e alvo, o plano Configurar (*ConfigureBlocksPlan*) escala subobjetivos, para modificar a mesa corrente, que ativam a execução do plano Empilhar Blocos (*StackBlocksPlan*). Este plano tem como objetivo posicionar um bloco em um local alvo. Inicialmente o plano Empilhar Blocos escala o objetivo Liberar que ativa o plano Liberar Blocos (*ClearBlocksPlan*) para descobrir se existem blocos posicionados acima do bloco a ser movido. A aplicação pode ser executada passo a passo ou em um único passo, conforme o modo de execução selecionado pelo usuário. A Figura 4 representa a criação de subobjetivos escalonados pelo plano Configurar.

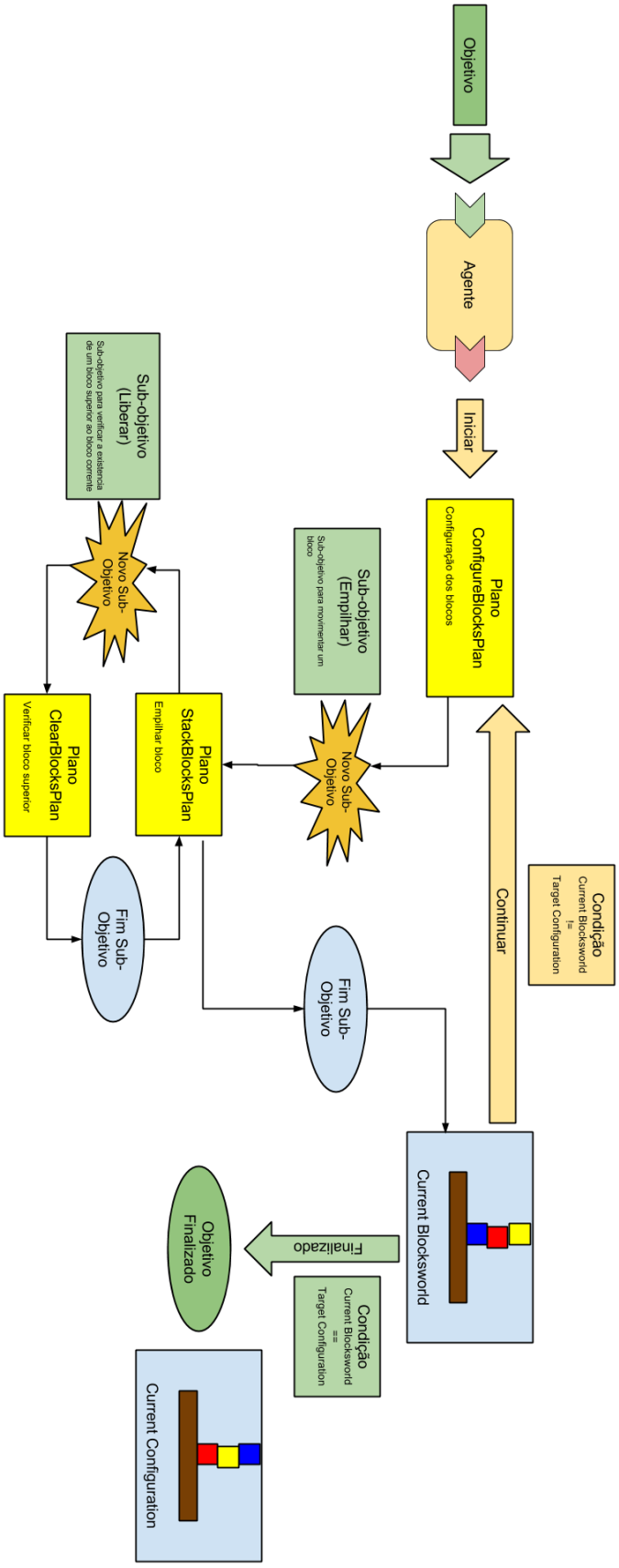

Figura 4 - Execução do plano Configurar e disparo de subobjetivos, até a conclusão do objetivo. Fonte: elaborado pelo autor.

Cada subobjetivo escalonado ativa um plano para realizá-lo. Com base nos objetivos e subobjetivos, o agente realiza iterativamente os planos "*StackBlocksPlan*" e "*ClearBlocksPlan*" até que a configuração da mesa corrente seja equivalente à mesa alvo.

A aplicação faz uso apenas de um agente para a execução dos planos, não necessitando, portanto, de utilizar um mecanismo de comunicação entre agentes.

2.5. Mundo de Marte

A aplicação "Mundo de Marte" é um outro exemplo disponível para ilustrar a utilização da plataforma Jadex. Faz uso de uma API de um ambiente virtual, suportando sua criação em 2D ou 3D. Aplicação que faz uso de agentes distintos, para a execução dos planos. Cada agente corresponde a uma entidade trabalhadora que realiza uma operação específica no processo de coleta de minérios. Os agentes e suas descrições de operações são mostradas no Quadro 3. O objetivo da aplicação consiste na exploração do ambiente e a coleta dos minérios, através da cooperação entre os agentes, no período de tempo estabelecido para a execução da missão.

Quadro 3 - Operações dos Agentes do Mundo de Marte

Agente	Descrição do agente
Sentinela	Responsável por examinar locais no ambiente onde podem existir uma fonte de minério.
Produtor	Responsável por extrair minério nos locais onde o agente Sentinela examinou e constatou a presença do mesmo.
Transportador	Responsável por carregar o minério produzido pelo agente Produtor até a base dos agentes.

Diferente do "Mundo de Blocos", a aplicação "Mundo de Marte" é iniciada através um arquivo XML que armazena informações referentes ao modo de execução da aplicação. São implementados nativamente três modos de execução. O primeiro denominado "Clássico", consistindo em um ambiente do Mundo de Marte em 2D. O segundo com a representação do Mundo de Marte

em 3D. E o terceiro, denominado Mundo de Espaço, representando as mesmas funções do Mundo de Marte aplicados no contexto do espaço, modificando apenas a representação dos agentes para naves espaciais e ao alvos (minérios) se tornam asteroides. Cada arquivo define um conjunto de configurações diferentes para representar o ambiente.

Inicialmente, em cada arquivo são definidos os espaços de nomes e as importações de bibliotecas que serão utilizados pela aplicação, como por exemplo a API para desenvolvimento de ambientes 2D ou 3D. Posteriormente são definidos os objetos específicos da aplicação, como as entidades Sentinela, Produtor, Transportador, Posição alvo e Base da missão, com suas devidas propriedades e parâmetros. Criados as entidades, é realizado o mapeamento dos componentes gráficos de cada entidade, assimilando um arquivo de textura no modo de execução 2D, ou um modelo 3D no modo de execução 3D. Também são definidas demais configurações, como os componentes Java (utilizados para criação dos agentes), as tarefas realizadas em alguns planos, a quantidade de agentes de cada tipo, a quantidade e localização dos minérios presentes no ambiente, indicadores numéricos (posição dos agentes, quantidade de minérios extraídos, tempo da missão).

Outro diferencial da aplicação do Mundo de Marte é a utilização dos conceitos de capacidades e tarefas, em conjunto com os planos.

Uma Capacidade é definida como um encapsulamento de parte de crenças, desejos e intenções de um agente em um módulo, para ser reutilizado quando necessário. Os elementos encapsulados são semelhantes aos que um agente utiliza, porém específicos para um conjunto de funcionalidades. Um agente pode implementar mais de uma capacidade, dividida de forma hierárquica, tendo uma capacidade raiz e outras sub capacidades. Quando o agente é iniciado, os planos, definidos explicitamente ou nas capacidades, são acionados para alcançarem seus objetivos e sub objetivos.

Um Plano, assim como no Mundo de Bloco, representa a configuração e execução de um conjunto de ações a serem realizadas para que um objetivo ou sub objetivo seja alcançado. Contudo, a aplicação do Mundo de Marte tem como diferencial o uso de tarefas como parte das ações planejadas.

Uma tarefa representa uma funcionalidade encapsulada para a manipulação da lógica de um agente. Pode ser construída para ser especifica a

um agente ou para ser reutilizada entre vários agentes. Um exemplo de tarefa consiste em uma tarefa de Movimentação que realiza continuamente o deslocamento de um agente em um ambiente, dado parâmetros como um espaço de tempo, velocidade e direção de movimento.

No contexto do Mundo de Marte é implementada apenas a capacidade de Movimentação (*MovementCapability*), a qual é utilizada em todos os agentes para o deslocamento pelo ambiente. Estão presentes na capacidade:

- Crenças:
    - Tempo da missão: *flag* para determinar se a missão dos agentes ainda está em andamento ou se o tempo já foi expirado. Valor inicial contido no arquivo de configuração.
    - Lista de Alvos: lista de espaços do ambiente para onde o agente irá se locomover.
- Objetivos:
    - Movimentar (*Move*): deslocar o agente para um local destino.
    - Movimentar ao redor (*WalkAround*): deslocar o agente de forma aleatório pelo ambiente.
    - Fim da missão (*Missionend*): deslocar o agente de volta a base da missão.
- Planos:
    - Deslocamento Planejado: acionamento dos objetivos Movimentar e Fim da Missão.
    - Deslocamento Aleatório: acionamento do objetivo Movimentar ao Redor.

O Mundo de Marte implementa as tarefas de movimentação:

- No modo de execução 2D
    - Movimentar (*MoveTask*): realiza o deslocamento de um agente para um local do ambiente, tanto graficamente (mostrada na interface) quanto os valores das variáveis do agente.
    - Girar (*Rotation*): realiza a rotação de um agente, mudando o sentido de deslocamento do agente.
- No modo de execução 3D

- Movimentar 3D (*Move3DTask*): funcionamento semelhante ao Movimentar, porém tratando locais de um ambiente 3D.
- Girar 3D (*Rotation3DTask*): funcionamento semelhante, porém, tratando de rotações em um ambiente 3D.

Ambas as tarefas, em cada modo de execução, são compartilhadas entre todos os agentes.

2.6. Comunicação por serviços

Adicionalmente, deve existir uma comunicação entre os agentes para que eles possam trabalhar de forma cooperativa. Tal comunicação é realizada através da comunicação por serviços utilizando os chamados Componentes Ativos. A intenção dos Componentes Ativos é proporcionar a combinação de componentes da Arquitetura Componente Serviço (SCA) com agentes e serviços. Componentes SCA funcionam de forma análoga a um Circuito Integrado, onde subcomponentes (módulos) são construídos em uma placa de circuito, que detêm de uma interface para acesso e integração a outros componentes SCA. A Figura 5 representa a estrutura de um Componente Ativo.

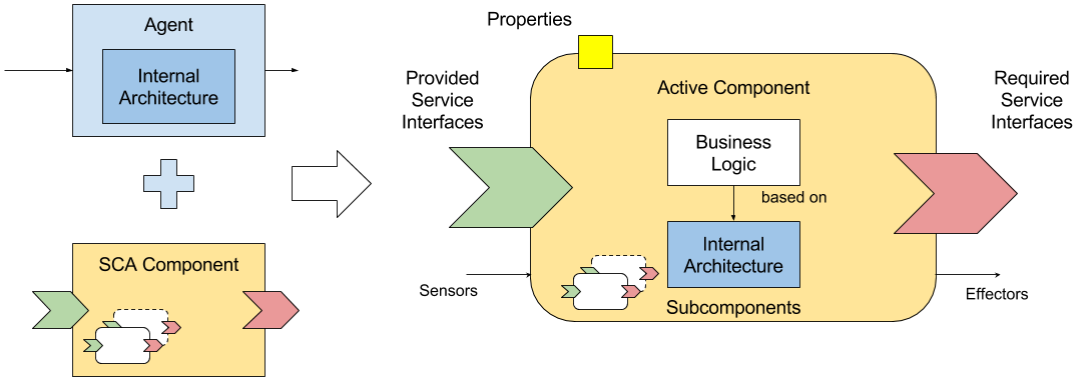

Figura 5 - Estrutura de um Componente Ativo. Fonte: Documentação do Jadex (adaptado).

Componentes Ativos possibilitam que um agente da arquitetura BDI, orientado a objetivos, seja encapsulado em um componente SCA. Possibilita todas as funcionalidades de um componente SCA, como configuração externa, hierarquia de componentes, e adicionalmente a utilização e o provimento de serviços, pelo fato que serviços fazem parte dos componentes, sem modificar as regras de negócios, lógica do componente, de um agente.

Serviços consistem de um arquivo de Especificação, uma interface para funcionar como a fronteira entre o agente e o serviço, e um arquivo que contém a implementação do serviço. Para que os serviços sejam encontrados pelos clientes (agentes), os serviços são registrados no serviço de páginas amarelas (*YellowPages*) do agente *Directory Facilitator*.

A comunicação entre agentes é realizada de forma assíncrona pelo barramento do contêiner onde os agentes estão presentes. A ordem das mensagens é controlada através do formato em que as mensagens são escritas. Um desses formatados é a ACL (*Agent Communication Language*) da especificação da FIPA. A Figura 6 representa a estrutura de uma mensagem ACL.

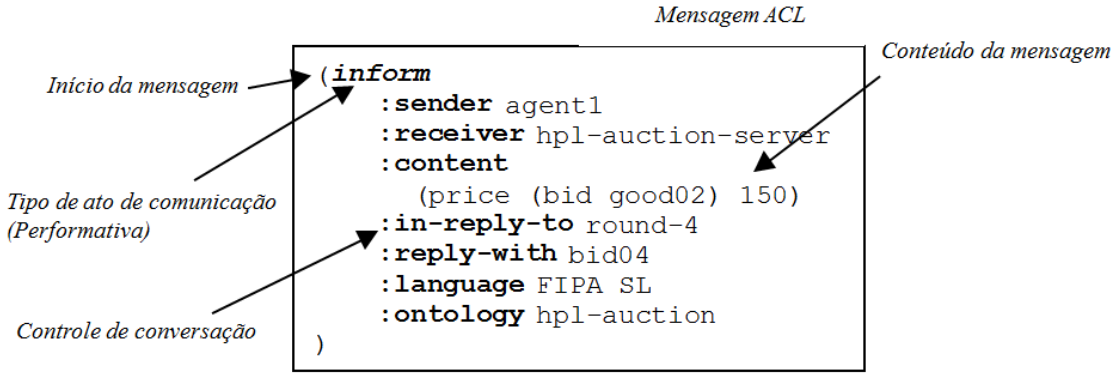

Figura 6 - Exemplo de estrutura de uma mensagem ACL. Fonte: GATTI, 2007.

A FIPA ACL define um modo padrão para empacotar mensagens, de tal forma que satisfaz para outros agentes a finalidade da comunicação (Posland *et al.* 2000). A linguagem define 20 verbos comunicativos (atos de comunicação) que são utilizados para estabelecer uma comunicação. A mensagem ainda contém informações do remetente (:sender) e do destinatário (:receiver), a mensagem em si (:content), a linguagem utilizada (:language), a semântica dos itens do conteúdo (:ontology) e campos de controle de

conversação (:in-reply-to e :reply-with). Formulada a mensagem, é necessário montar um objeto do tipo ACLMessage e realizar a chamada do método *send()* do agente. O barramento realiza a entrega da mensagem na caixa de entrada do agente destinatário. O agente destinatário recupera a mensagem pelo método *receive()*, processando-a em seguida.

2.7. Execução do Mundo de Marte

A execução da aplicação ativa uma interface gráfica, correspondente ao modo de execução desejado, juntamente com seus componentes. Os componentes carregados representam as propriedades do mundo (tempo da missão, quantidade de agentes e minérios), o ambiente (tabuleiro com o "terreno" do ambiente, onde cada posição contém propriedades especificas), os agentes (sentinela, produtor e transportador), dentre outros componentes definidos nos arquivos de configuração.

2.7.1. Agente Sentinela

- O agente Sentinela tem como função explorar o ambiente do Mundo de Marte e analisar locais com possam conter minérios. Por padrão é implementado apenas um agente Sentinela.
- O Sentinela contém a capacidade de Movimentação. Nela está contida a base de crenças do agente, que consiste de uma lista de locais alvos para serem analisados e uma variável responsável por armazenar o tempo restante da missão.
- Tem como objetivo Analisar Alvo (*AnalyzeTarget*) para analisar um local alvo.
- Tem como plano Analisar Alvo (*AnalyzeTargetPlan*) que planeja a verificação do local onde o agente se encontra (local alvo) para verificar a existência de minérios.

- Tem como tarefa Analisar Alvo (*AnalyzeTargetTask*) para analisar a propriedade Minério para descobrir a quantidade de minérios presentes naquele local. Também executa um tempo de espera para simular um tempo de análise. Tanto a quantidade de minérios presentes no local quanto o tempo de espera são definidos no arquivo de configuração XML do ambiente.
- Serve o serviço Alvo que informa um local alvo para serem extraídos minérios.

Durante sua execução:

- O Sentinela inicia com o objetivo Analisar Alvo, mas que é inibido pela capacidade de Movimentação por não existirem inicialmente alvos para serem analisados na base crenças.
- A capacidade de Movimentação tem como objetivo inicial Movimentar ao redor, para deslocar o Sentinela pelo ambiente a fim de se encontrar possíveis locais com minérios.
- Aciona o plano Deslocamento Aleatório para calcular um local aleatório e realizar o deslocamento do agente até o local calculado.
    - As tarefas de Mover e Girar são executadas pelo plano para deslocar o agente para o local destino gerado aleatoriamente.
    - Se durante a tarefa Mover, for descoberto um local que pode conter minérios, o local é armazenado em uma variável e o objetivo é substituído para Movimentar. Um local pode ser descoberto se o mesmo entrar no raio de visão (propriedade definida no arquivo de configuração) do agente Sentinela.
- O objetivo Movimentar desloca o Sentinela até um local destino.
    - É acionado o plano Deslocamento Planejado. Este planeja o deslocamento, através das tarefas Mover e Girar, do agente até um local alvo.
    - Ao final da tarefa Mover, o local onde o agente se deslocou é adicionado à lista de locais alvos da base de crenças da capacidade.
- Quando um local é adicionado a base de crenças da capacidade, a mesma é inibida e o objetivo Analisar Alvo volta a ser executado.

    - É acionado o plano Analisar Alvo.
        - A executada a tarefa Analisar Alvo.
    - Se ao final do plano, a quantidade de minérios no local analisado for superior a zero, é realizada a chamada do agente Produtor através da requisição do serviço de Produção e o provimento do local alvo através do serviço de Alvo.
- Caso existam outros locais alvos para serem explorados, o Sentinela ira se deslocar para esses locais e os analisará.
- Caso contrário o objetivo Analisar Alvo volta a ser inibido pela capacidade Movimentação.
    - Caso o tempo da missão tenha expirado, o objetivo Fim da missão se torna ativo, acionando o plano Deslocamento Planejado para deslocar o agente até a base da missão, encerrando as atividades do agente.

2.7.2. Agente Produtor

O agente Produto tem como função extrair os minérios de um local provido pelo Sentinela. Por padrão são implementados vários agentes Produtores para que o tempo de extração seja menor, caso existam mais de um agente extraindo um mesmo local.

- O Produtor contém a capacidade de Movimentação. Nela está contida a base de crenças do agente, que consiste de uma lista de locais alvos com minérios para serem extraídos e uma variável responsável por armazenar o tempo restante da missão.
- Tem como objetivo Produzir Minério (*ProduceOre*) para extrair minérios de um local alvo.
- Tem como planos Informar Novo Alvo (*InformNewTargetPlan*) para responder o chamado do serviço Alvo do Sentinela e Produzir Minerio (*ProduceOrePlan*) que planeja a extração dos minérios dos locais informados com existência de minérios.

- Tem como tarefa Produzir Minério (*ProduceOreTask*) para extrair a propriedade Minério do local onde o agente se encontra. Executa a extração de uma unidade de minério em uma unidade de tempo para simular um tempo de extração. Quanto mais agentes se encontram extraindo um mesmo local, mais rápido será a extração da quantidade total de minérios. Tanto a quantidade de minérios presentes no local, quanto a unidade de tempo utilizada, são definidas no arquivo de configuração XML do ambiente.
- Serve o serviço Produzir que responde chamados para extração de minérios de um local alvo.
- Requer o serviço Alvo do agente Sentinela.

Durante sua execução:

- O Produtor inicia com o objetivo Produzir Minério, mas que é inibido pela capacidade de Movimentação por não existirem inicialmente alvos com minérios na base de crenças.
- A capacidade de Movimentação tem a execução semelhante ao agente Sentinela. O objetivo Movimentar ao redor é executado até que um local seja adicionado a base de crenças. Um local é adicionado a base de crenças quando um agente Produtor responde a requisição de Produção do agente Sentinela.
- Quando um local é adicionado a base de crenças da capacidade, a capacidade é inibida e o objetivo Produzir Minério volta a ser executado.
- É acionado o plano Produzir Minério.
    - Dispara o objetivo Movimentar em forma de subobjetivo para deslocar o agente até o primeiro local da base de crenças.
    - É executada a tarefa Produzir Minério.
    - Ao final do plano é realizada a chamada do agente Transportador através da requisição do serviço de Transportar, sendo provido um local alvo que contém os minérios extraídos.
- Caso existam outros locais alvos para serem extraídos, o Produtor irá se deslocar para esses locais e irá extrair os minérios presentes.
- Caso contrário o objetivo Produzir Minério volta a ser inibido pela capacidade Movimentação.

- Assim como no agente Sentinela, caso o tempo da missão tenha expirado, o objetivo Fim da missão é acionado e o plano Deslocamento Planejado é executado para deslocar o agente até a base da missão, encerrando as atividades do agente.

2.7.3. Agente Transportador

O agente Transportador tem como função carregar os minérios extraídos de um local provido pelo Produtor. Por padrão são implementados vários agentes Transportadores para que o tempo de transporte/número de viagens seja menor, caso existam diferentes Transportadores transportando minérios de um mesmo local.

- O Transportador contém a capacidade de Movimentação. Nela está contida a base de crenças do agente, que consiste de uma lista de locais alvos com minérios extraídos e uma variável responsável por armazenar o tempo restante da missão.
- Tem como objetivo Transportar Minério (*CarryOre*) para carregar minérios extraídos de um local alvo até a base da missão.
- Tem como planos Informar Novo Alvo (*InformNewTargetPlan*) para responder o chamado do serviço Carregar do Produtor e Transportar Minério (*CarryOrePlan*) que planeja a carga, transporte e descarga de todo o minério de um local alvo para a base da missão.
- Tem como tarefa Carregar Minério (*LoadOreTask*) que realiza o carregamento do minério para o agente e o descarregamento do minério na base da missão. Cada agente possui a propriedade Capacidade que indica a quantidade de minério pode ser carregador por vez. Quando cheio o agente retorna a base da missão para descarregar. A capacidade que cada agente Transportador pode carregar é definida no arquivo de configuração XML do ambiente.
- Serve o serviço Transportar que responde chamados para transporte de minérios de um local alvo.
- Requer o serviço Produzir.

Durante sua execução:

- O Transportador inicia com o objetivo Transportar Minério, mas que é inibido pela capacidade de Movimentação por não existirem inicialmente alvos com minérios na base de crenças.
- A capacidade de Movimentação tem a execução semelhante ao agente Sentinela e do agente Produtor. O objetivo Movimentar ao redor é executado até que um local seja adicionado a base de crenças. Um local é adicionado a base de crenças quando um agente Transportador responde a requisição de Transportar do agente Produtor.
- Quando um local é adicionado a base de crenças da capacidade, a capacidade é inibida e o objetivo Transportar Minério volta a ser executado.
    - É acionado o plano Transportar Minério.
        - Dispara o objetivo Movimentar em forma de subobjetivo para deslocar o agente até o primeiro local da base de crenças.
        - É executada a tarefa Carregar Minério.
        - Dispara novamente o objetivo Movimentar em forma de subobjetivo para deslocar o agente até a base da missão.
        - É executada a tarefa Carrega Minério.
        - O plano é repetido enquanto no local alvo ainda existirem minérios extraídos.
- Caso existam outros locais alvos para serem transportados, o Transportador ira se deslocar para esses locais e irá transportar os minérios presentes.
- Caso contrário o objetivo Transportar Minério volta a ser inibido pela capacidade Movimentação.
    - Assim como no agente Produtor, caso o tempo da missão tenha expirado, o objetivo Fim da missão é acionado e o plano Deslocamento Planejado é executado para deslocar o agente até a base da missão, encerrando as atividades do agente.

2.8. Uso de anotações

Com a substituições das *tags* por anotações Java, a forma em que agentes são construídos também foi modificada. O uso de arquivos XML não é mais necessário para configurar os agentes, sendo realizado em arquivos Java. Algumas anotações utilizadas são:

- @Agent public class Classe { ... } : utilizada para definir que uma classe Java será utilizada para descrever um agente.
- @AgentBody : utilizada para especificar o corpo do agente. Define os métodos utilizados pelo agente.
- @Plans(...) : associa um conjunto de planos ao agente.
- @Plan(trigger = @Trigger(...), body = @Body(...)): associa um plano ao agente. O parâmetro *trigger* associa uma condição que aciona o plano. O parâmetro *body* especifica uma classe Java sendo o corpo do plano.
- @Plan public class ClassePlano { ... } : utilizada para definir que uma classe Java representa um plano. A classe será utilizada pelo anotação @Plan presente no corpo da classe do agente.
- @Belief : utilizada para especificar que uma variável faz parte da base crenças do agente.
- @Capability : utilizada para definir um variável representa uma Capacidade do agente.

Outras anotações também podem ser utilizadas, dependendo do tipo de aplicação que será construída.

3. Desenvolvimento do Trabalho Proposto

Nesta seção será descrita a especificação, implementação e teste de uma aplicação para sistemas multiagentes em um ambiente virtual.

3.1. Especificação de uma Aplicação

Após os estudos das aplicações de exemplo, foi possível conhecer o funcionamento do *framework* Jadex. Para consolidar os conhecimentos adquiridos será desenvolvida uma aplicação para o Jadex.

A especificação da aplicação, a ser desenvolvida, será composta das seguintes atividades:

- especificação dos requisitos de uma aplicação de sistemas multiagentes em um ambiente virtual.
- mapeamento dos requisitos em agentes, objetivos, planos e serviços.
- e caracterização da interface do ambiente virtual da aplicação.

A aplicação utilizará o mesmo ambiente estudado no exemplo do “Mundo de Marte”; contudo, serão implementados outros agentes, crenças, objetivos e planos.

A aplicação deverá conter três tipos de agentes principais (Avião, Alvo e Bateria Antiaérea) e 2 tipos de agentes secundários (Míssil Ar-Terra e Míssil Terra-Ar) que são gerados durante a execução da aplicação.

3.1.1. Especificação dos Agentes

Agentes do tipo Avião:

- Agentes do tipo Avião têm como objetivo encontrar e “destruir” os agentes do tipo Alvo.
- Não sabem de início o local exato dos alvos, contudo sabem que devem “procurar” em uma determinada região do ambiente.

- Possuem um campo de visão, limitado, que permite que o agente enxergue o ambiente e descubra informações de objetos que entram no campo de visão.
- Apenas se deslocam em linha reta, sendo necessário realizar uma movimento de “curva” para trocar a direção do movimento.
- Possuem como posição inicial e final a localização do objeto “Porta Aviões”.
    - O objeto “Porta Aviões” representa um posição no ambiente e não possui nenhum Agente associado, consequentemente não realiza qualquer ação.
- Possuem como principais bases de crenças:
    - Fim da Missão: indica se o tempo da missão foi esgotado;
    - Lista de Alvos Encontrados: lista de todos os agentes do tipo Alvo que foram encontrados pelo Avião.
- Implementam os serviços
    - Destruir: quando solicitado, realizam a remoção da instância do Avião do ambiente.
    - Alvo destruído: recebe a comunicação de que Alvo foi destruído por um agente Míssil Ar-Terra.

Agentes do tipo Bateria AA

- Agentes do tipo Bateria AA (Antiaérea) tem como objetivo proteger os agentes do tipo Alvo, encontrando os agentes do tipo Avião e os “destruindo”.
- Não sabem de início a localização do agentes do tipo Avião.
- Possuem um campo de visão, limitado, que permite que o agente enxergue o ambiente e descubra informações de objetos que entram no campo de visão.
- Apenas se deslocam em linha reta, sendo necessário rotacionar no próprio eixo para trocar a direção do movimento.
- Possuem como posição inicial um local aleatório no ambiente e posição final a localização de um agente do tipo Alvo.
- Possuem como principais bases de crenças:
    - Fim da Missão: indica se o tempo da missão foi esgotado;

    - Lista de Aviões Encontrados: lista que contêm todos os agentes do tipo Avião que foram encontrados pela Bateria Antiaérea.
- Implementam os serviços:
    - Alvo destruído: recebe a comunicação de que Alvo foi destruído por um agente Míssil Ar-Terra.
    - Avião destruído: recebe a comunicação de que Avião foi destruído por um agente Míssil Terra-Ar.

Agentes do tipo Alvo

- São estáticos e não realizam nenhuma ação.
- Servem como posição final para os agentes do tipo Bateria AA.
- Possuem como posição inicial um local aleatório no ambiente.
- Implementam o serviço:
    - Destruir: quando solicitado, realizam a remoção da instância do Alvo do ambiente

Agentes do tipo Míssil Ar-Terra

- Gerados durante a execução da aplicação por agentes do tipo Avião.
- Representam o ataque do Avião contra um Alvo.
- Após serem criados por um Avião se deslocam, em linha reta, até atingir um agente Alvo.
- Possuem como principais bases de crenças:
    - Sem Alvo: indica se o Alvo associado foi destruído.

Agentes do tipo Míssil Terra-Ar

- Gerados durante a execução da aplicação por agentes do tipo Bateria AA.
- Representam o ataque da Bateria AA contra um Avião.
- Possuem um tempo de vida limitado, que ao ser excedido o agente se autodestrói.
- Após serem criados por uma Bateria AA perseguem um agente Avião até atingi-lo.
- Possuem como principais bases de crenças:
    - Sem Alvo: indica se o Avião associado foi destruído.

3.1.2. Especificação do Ambiente

Especificações gerais do ambiente:

- Consiste de uma área com formato quadrilátero, com duas regiões principais.
- A primeira região irá representar o mar, onde estarão presentes o Portão Aviões e os Aviões, e ocupará 25% da área do ambiente.
- A segunda região irá representar a terra, onde estarão presentes os Alvos e Baterias AA, e ocupará 75% da área do ambiente.
- Irá utilizar apenas elementos com representações em duas dimensões (2D).

Especificações dos avatares:

- Cada agente terá uma representação gráfica no ambiente, denominada avatar.
- Cada avatar definirá um conjunto de propriedades que serão utilizadas pelo agente para interagir com o ambiente, seja para manter a sua representação ou encontrar e interagir com outros agentes. O Quadro 4 demonstra as principais propriedades dos avatares de cada tipo de agente.

Quadro 4 - Principais propriedades para cada tipo de agente

Tipo de agente	Propriedades
Avião e Bateria Antiaérea	Posição, rotação, velocidade, velocidade mínima, velocidade máxima, distância máxima do campo de visão.
Alvo	Posição, estado (não descoberto ou descoberto).
Míssil Ar-Terra e Míssil Terra-Ar	Posição, rotação, velocidade, velocidade mínima, velocidade

	máxima, alvo.

- Cada avatar terá uma imagem (textura) associada que melhor represente a natureza do agente.

3.2. Cenários de Execução

Nesta seção serão descritas os cenários de execução para cada agente da aplicação. O Quadro 5 descreve o significado de cada símbolo utilizando durante os cenários de execução.

Quadro 5 - Símbolos utilizados nos Cenários de Execução

Símbolo	Descrição
Numeração (Ex.: 1, 2, 3, etc.)	Ordem do acionamento dos objetivos
● ○ ■	Ação executada em um objetivo, plano ou tarefa.
--	Acionamento de um plano.
>>	Acionamento de uma tarefa ou Término de uma tarefa.
[?]	Desvio condicional.
()	Parâmetros recebidos ou argumentos enviados.
//	Comentário.

3.2.1. Agente Avião

1. Objetivo: Mover Aleatoriamente
    - aciona o Plano: Mover Aleatoriamente

-- Plano: Mover Aleatoriamente

- gera uma ponto aleatório // uma posição/coordenada no ambiente
- aciona o Subobjetivo: Mover para Local (ponto)

2. Objetivo: Mover para Local (ponto)
    - aciona o Plano: Mover para Local (ponto)

-- Plano: Mover para Local

- aciona a Tarefa: Mover (ponto, Agente)

>> Tarefa: Mover (ponto, Agente) // idêntica para os agentes: Avião, Bateria Antiaérea, Míssil Ar-Terra e Míssil Terra-Ar

- verifica se o Agente Avião já está na posição de destino (através das propriedades do ambiente: posição, velocidade, rotação)

[?] posição = destino

- encerra a Tarefa

[?] posição != destino

- calcula o deslocamento em relação ao ponto destino // acelera se estiver na direção do destino e desacelera se precisar de rotação
- recupera do ambiente os agentes próximos e verifica o tipo de cada agente

[?] tipo do Agente != Alvo

- reexecuta Tarefa: Mover

[?] tipo do Agente == Alvo

- insere o Agente Alvo na Crença: "Lista de Alvos Encontrados", que aciona o Objetivo: Atacar Alvo (Alvo)

- finaliza a Tarefa

>> Fim da Tarefa: Mover

3. Objetivo: Atacar Alvo (Alvo)

- aciona o Plano: Atacar Alvo (Alvo)

-- Plano: Atacar Alvo (Alvo)

- calcula a distância e inclinação (em relação à direção do avião) da posição do Avião com posição do Alvo

[?] Condição (distância inferior ou igual à distância mínima de ataque)

- gera um ponto aleatório com base na inclinação
- aciona o Subobjetivo: Mover para Local (ponto)

- aciona a Tarefa: Lançar Míssil Ar-Terra (Alvo, Avião)

>> Tarefa: Lançar Míssil Ar-Terra (Alvo, Avião)

- cria um agente: Míssil Ar-Terra (Alvo, propriedades do Avião {posição, rotação, velocidade})
- encerra a Tarefa

X. Objetivo: Retornar ao Porta Aviões // acionado pela Crença: Fim da Missão
   - aciona o Plano: Mover para Local (posição do porta aviões)

   // com nenhum objetivo restante ativo: o a gente fica em espera (*stand-by*)

3.2.2. Agente Bateria Antiaérea

1. Objetivo: Proteger
   - aciona o Plano: Proteger

-- Plano: Proteger
   - aciona a Tarefa: Proteger (Agente)

>> Tarefa: Proteger
   - inicializa o tempo de início da proteção
   - recupera do ambiente os agentes próximos e verifica o tipo de cada agente

     [?] tipo do Agente != Avião
       - continua a execução

     [?] tipo do Agente == Alvo
       - insere o Agente Alvo na Crença: "Lista de Aviões Encontrados", que aciona o Objetivo: Atacar Avião (Avião)
   - recupera o tempo de proteção // tempo decorrido desde o início da tarefa

     [?] tempo proteção > tempo mínimo proteção
       - encerra a Tarefa

     [?] tempo proteção < tempo mínimo proteção
       - reexecuta a Tarefa

>> Fim da Tarefa: Proteger
   - gera um ponto aleatório
   - aciona o Subobjetivo: Mover para Local (ponto)

2. Objetivo: Mover para Local (ponto)
   - aciona o Plano: Mover para Local (ponto)

-- Plano: Mover para Local
   - aciona a Tarefa: Mover (ponto, Agente)

>> Tarefa: Mover (ponto, Agente)

- verifica se o Agente Bateria Antiaérea já está na posição de destino (através das propriedades do ambiente: posição, velocidade, rotação)

[?] posição = destino

- encerra a Tarefa

[?] posição != destino

- calcula o deslocamento em relação ao ponto destino

// primeiro o agente roda em seu próprio eixo para se alinhar com o destino

// posteriormente, acelera na direção do destino e desacelera se precisar de rotação ou estar próximo ao destino

- recupera do ambiente os agentes próximo e verifica o tipo de cada agente

[?] tipo do Agente != Avião

- reexecuta a Tarefa: Mover

[?] tipo do Agente == Avião

- insere o Agente Avião nas Crenças "Lista de Aviões Encontrados" que aciona o Objetivo: Atacar Avião (Avião)
- finaliza a Tarefa

>> Fim da Tarefa: Mover

3. Objetivo: Atacar Avião (Avião)

- aciona o Plano: Atacar Avião (Avião)

-- Plano: Atacar Avião (Avião)

- aciona a Tarefa: Lançar Míssil Terra-Ar (avião, bateria antiaérea)

>> Tarefa: Lançar Míssil Terra-Ar (avião, bateria antiaérea)

- cria um agente: Míssil Terra-Ar (avião, propriedades da Bateria Antiaérea {posição, rotação})
- encerra a Tarefa

>> Fim da Tarefa: Lançar Míssil Terra-Ar

X. Objetivo: Retornar para Base // ativado pela Crença: Fim da Missão

- verifica se existem agentes do tipo Alvo que não foram destruídos

[?] agentes vivos > 0

- recupera a posição de um Agente Alvo (escolha de forma arbitraria)

- aciona o Plano: Mover para Local (posição do Agente Alvo)

[?] agentes vivos = 0

- permanece na posição em que se encontra

// com nenhum objetivo ativo: o a gente fica em espera (*stand-by*)

3.2.3. Agente Míssil Ar-Terra

1. Objetivo: Destruir Alvo (Alvo)
   - aciona o Plano: Destruir Alvo (Alvo)

-- Plano: Destruir Alvo (Alvo)

- aciona a Tarefa: Destruir Alvo (Alvo)

>> Tarefa: Destruir Alvo (Alvo)

- verifica se o Agente Míssil Ar-Terra já está na mesma posição do Alvo (através da propriedade do ambiente: posição)

[?] posição = posição do Alvo

- encerra a Tarefa

[?] posição != posição do Alvo

- calcula o deslocamento em relação ao ponto destino/posição do Alvo

>> Fim da Tarefa: Destruir Alvo

- requisita o Serviço: Destruir, do Agente Alvo recebido
- altera a sua Crença: “Sem Alvo”, que aciona o Objetivo: Autodestruir

2. Objetivo: Autodestruir (Míssil Ar-Terra)
   - aciona o Plano: Autodestruir (míssil Ar-Terra)

-- Plano: Autodestruir (míssil Ar-Terra)

- remove a instancia do Agente Míssil Ar-Terra recebida

// o agente encerra as ações que estava realizando e remove a sua instancia no ambiente

3.2.4. Agente Míssil Terra-Ar

1. Objetivo: Destruir Avião (Avião)
    - aciona o Plano: Destruir Avião (Avião)

-- Plano: Destruir Alvo (Avião)

- aciona a Tarefa: Destruir Avião (Avião)

>> Tarefa: Destruir Avião (Avião)

- verifica se o Agente Míssil Terra-Ar já está na mesma posição do Avião (através da propriedade do ambiente: posição)

[?] posição = posição do Avião

- encerra a Tarefa

[?] posição != posição do Avião

- calcula o deslocamento em relação ao ponto destino/posição do Avião
- atualiza o ponto de destino/posição do Avião // devido o Avião ser um agente móvel

>> Fim da Tarefa: Destruir Avião

- requisita o Serviço: Destruir, do Agente Avião recebido
- altera a sua Crença: “Sem Alvo”, que aciona o Objetivo: Autodestruir

3. Objetivo: Autodestruir (míssil Terra-Ar)
    - aciona o Plano: Autodestruir (míssil Terra-Ar)

-- Plano: Autodestruir (míssil Terra-Ar)

- remove a instancia do Agente Míssil Terra-Ar recebida

// o agente encerra as ações que estava realizando e remove a sua instancia no ambiente

3.3. Desenvolvimento da aplicação

Nesta seção, serão descritos a implementação e os testes que serão realizados na aplicação, especifica na seção anterior, bem como, a análise dos resultados obtidos no desenvolvimento da aplicação.

3.3.1. Pré-Requisitos

O primeiro passo para desenvolver uma aplicação com o *framework* Jadex consiste em obter o projeto disponibilizado pelos desenvolvedores do Jadex. Por mais que não existam restrições para se criar o próprio projeto “do zero”, utilizar o projeto disponibilizado tem a vantagem de já estar configurado para obter dependências (bibliotecas) e utilizar Jadex.

O projeto foi originalmente feito para ser utilizado pelo Ambiente de Desenvolvimento Integrado (IDE) Eclipse, mas para a execução do trabalho foi migrado para o IDE NetBeans. A migração foi possível pois o projeto original foi baseado no gerenciador de projetos Apache Maven, que possibilita maior flexibilidade na construção de aplicações Java.

Com o projeto pronto é necessário definir a classe principal (*Main Class*) da aplicação. A classe principal terá como função iniciar uma instância do Jadex para então poder ser executada. A biblioteca do Jadex disponibiliza uma classe que pode ser utilizada como classe principal, bastando associar a classe “jadex.base.Starter” à classe principal do projeto, por meio da IDE. Contudo, criar uma implementação classe principal possibilita habilitar funcionalidades adicionais do Jadex, uteis para realizar testes e depurar a aplicação.

3.3.2. Definição do Ambiente

Como foi dito na Seção 3.1 (Especificação de uma aplicação), a aplicação utilizará o mesmo ambiente gráfico utilizado no exemplo “Mundo de Marte”. Para habilitar o suporte ao ambiente gráfico, denominado EnvSupport, é necessário que seja escrito um arquivo XML para definir as propriedades do ambiente. Este arquivo XML também tem como função:

- mapear os componentes ativos (agentes) com as suas respectivas representações gráficas (avatares).
- mapear as tarefas dos agentes.
- determinar processos (semelhante a uma tarefa, mas executada pelo ambiente sem estar associada à algum agente).
- determinar observadores (meios de o usuário visualizar o ambiente).

- determinar diferentes configurações para executar a aplicação.
- Exemplo: controlar o número de agentes.
- iniciar aplicação pelo Jadex Control Center.

Com o ambiente configurado, a próxima etapa consiste na descrição da lógica de negócios de cada tipo de agente. Como dito anteriormente, na versão BDIv3 a definição dos componentes (agentes, crenças, objetivos, planos, tarefas e serviços) é feita inteiramente por códigos Java, sendo necessário apenas mapear as classes Java que interagem com o ambiente por meio do arquivo XML que instancia o EnvSupport.

3.3.3. Definição dos Agentes

Seguindo o Modelo BDIv3, cada agente possuirá uma classe Java para representá-lo. O nome da classe do agente sempre deve terminar com “BDI”, caso contrário a mesma não será reconhecida pelo Jadex. Adicionalmente, todas as classes de agentes devem conter (representado pela Figura 7):

- A anotação @Agent, referenciando a declaração da classe.
- A anotação @Agent, dentro da classe, referenciando uma variável do tipo BDIAgent;
- A anotação @AgentBody, dentro da classe, referenciando um método para representar a lógica de negócio do agente.

```
01 @Agent
02 public class ClasseAgenteBDI {
03     @Agent
04     private BDIAgent agente;
05 
06     @AgentBody
07     public void corpo() {
08         // Lógica do agente
09     }
10 }
```

Figura 7 - Estrutura mínima da classe de um Agente BDI. Fonte: elaborada pelo autor.

3.3.4. Definição das Crenças

Em seguida, para cada agente, são definidas as suas crenças, objetivos e planos. Para otimizar essa definição é possível utilizar uma “capacidade”, representado pela Figura 8. Uma capacidade possibilita encapsular crenças, objetivos e planos para que possam ser reutilizados por diferentes agentes que utilizam esta capacidade.

```
01 @Agent
02 public class ClasseAgenteBDI {
03     @Agent
04     private BDIAgent agente;
05
06    // Agente que utiliza uma
capacidade
07    @Capability
08    private Capacidade
capacidade;
09
10     @AgentBody
11     public void corpo() {
12         // Lógica do agente
13     }
14 }
```

```
01 @Capability
02 public class Capacidade {
03
04    @Belief
05    protected Object crenca;
06
07    @Goal
08    public class Objetivo {
09          // definição das
condições (criação, desistência,
recorrência, etc.), parâmetros e
resultados
10    }
11
12    @Plan(trigger =
@Trigger(goals = Objetivo.class))
13    public void corpoPlano() {
14          // Lógica do plano
15    }
16 }
```

Figura 8 - Estrutura de um agente que utiliza uma Capacidade. Fonte: elaborado pelo autor.

O mapeamento entre as crenças, objetivos e planos representa uma etapa importante para definir o comportamento autônomo do agente após a sua criação. Seguindo o Modelo BDI, as crenças representarão as informações que o agente têm sobre o ambiente. Consequentemente, o estado de uma crença pode acionar um objetivo especifico, que consequentemente executa a lógica de seu plano. Crenças são definidas pela anotação @Belief referenciando uma variável de qualquer tipo. Toda vez que ocorrer a atualização do estado desta variável, o agente terá conhecimento da mudança.

3.3.5. Definição dos Objetivos

Já os objetivos, os desejos do agente, são definidos pela anotação @Goal, referenciando a declaração de uma classe. Para que o objetivo seja acionado pelo estado de uma crença é necessário adicionar a anotação @GoalCreationCondition(belief = “crença”) ao construtor da classe. Esta anotação aciona a criação do objetivo toda vez que o estado da crença (definida pelo parâmetro *belief*) mudar. Contudo, nem sempre a mudança do estado de uma crença representa a condição ideal para acionar um objetivo. Como solução é possível utilizar classes estáticas (*static*) para definir os objetivos, que permite que a anotação @GoalCreationCondition referencie um método estático para controlar a criação do objetivo. O método referenciado então controla as condições para acionar o objetivo. Quando as condições são atingidas, o método retorna uma nova instancia da classe do objetivo, acionando o objetivo e seu Plano. Caso as condições não sejam estabelecidas, o método retorna uma instancia nula, que impede o acionamento do objetivo.

3.3.6. Definição dos Planos

Em resposta ao acionamento de um objetivo, o seu respectivo plano será executado. Planos, ações que o agente realiza para alcançar os seus desejos, são definidos pela anotação @Plan de duas formas:

- Referenciando um método.
- Referenciando uma classe.
    - Requer a anotação @PlanBody, dentro da classe, referenciando um método para representar a lógica do plano.
    - Pode ser utilizada a anotação @PlanAPI, para controlar a lógica do plano após o seu acionamento, desde refletir a atualização de uma crença, até o acessar informações de capacidades e agentes para controlar a execução.

Independente da abordagem utilizada, todos os planos necessitam que o parâmetros “*trigger*” seja definido, referenciando a classe de seu objetivo. A

Figura 9 mostra atualização da capacidade, refletindo a utilização de um plano referenciando uma classe.

```
01 @Capability
02 @Plans(
03    @Plan(trigger =
@Trigger(goals = Objetivo.class),
body = @Body(Plano.class))
04 )
05 public class Capacidade {
06
07    @Belief
08    protected Object crenca;
09
10    @Goal
11    public class Objetivo {
12      // definição das condições
(criação, desistência,
recorrência, etc.), parâmetros e
resultados
13    }
14 }
```

```
01 @Plan
02 public Plano {
03
04    @PlanAPI
05    protected IPlan plan; //
anotação e variável necessária
para a modificação do contexto
do plano durante a execução
06
07    @PlanBody
08    public void corpoPlano() {
09          // Lógica do plano
10    }
11 }
```

Figura 9 - Atualização da Capacidade com a implementação do plano em forma de classe. Fonte: elaborado pelo autor.

3.3.7. Definição das Tarefas

A partir do acionamento de um plano, diferentes ações podem ser executadas em sua lógica. Entre as ações mais comuns estão a chamada de tarefas. Tarefas desempenham os papeis de atuadores e receptores do agente. Permitem que o agente realize ações no ambiente e que reações sejam percebidas. Por meio das tarefas que as crenças de um agente podem ser atualizadas. Diferentemente de crenças, objetivos e planos, as tarefas não são definidas por meio de anotações. Cada tarefa deve ser definida por uma classe, que estenda a classe "*AbstrackTask*", e mapeada por meio de um arquivo XML que configura as propriedades do ambiente. Para definir a lógica da tarefa é necessário sobrescrever (*override*) ao menos o método "*execute*". A Figura 10 mostra a definição da classe de uma tarefa.

```
01 public class Tarefa extends AbstractTask {
02
03     @Override
04     public void execute(IEnvironmentSpace space, ISpaceObject obj,
long progress, IClockService clock) {
05           // Lógica da tarefa
06     }
07 }
```

Figura 10 - Estrutura mínima de uma tarefa. Fonte: elaborado pelo autor.

3.3.8. Definição dos Serviços

Além de seguir as suas crenças, objetivos e planos, um agente pode requisitar e prover serviços. O uso de serviços permite que agentes comuniquem-se entre si, possibilitando a coordenação para trabalharem de forma cooperativa. Serviços são definidos por meio de interfaces que são implementadas pelos agentes. Adicionalmente cada agente mapeia, por meio de anotações, os serviços que ele deseja requisitar (anotação @RequiredServices) e quais ele pode prover (anotação @ProvidedServices). E assim como as crenças, os serviços também podem ser encapsulado em capacidades.

3.4. Análise dos Resultados

Nesta seção serão apresentadas ilustrações que refletem as implementações das especificações da seção 3.1. (Especificação de uma Aplicação) e dos cenários da seção 3.2 (Cenários de Execução).

A Figura 11 representa a execução de agente Avião desde a sua criação, até atingir o seu objetivo principal, destruir um agente Alvo.

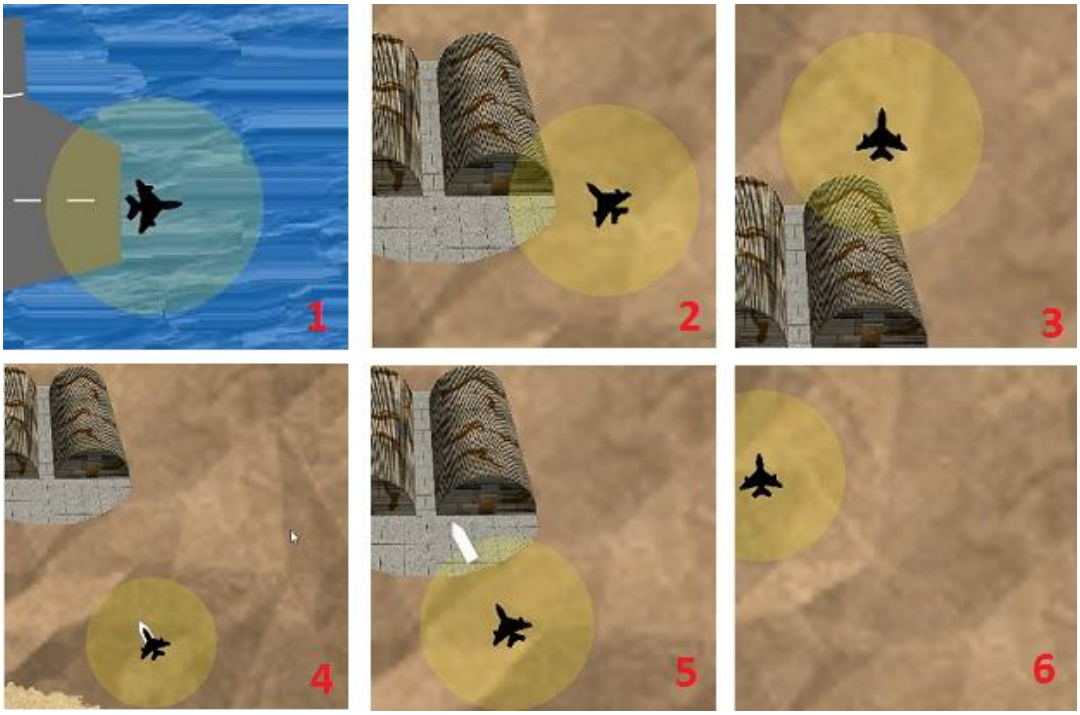

Figura 11 - Execução de um agente Avião. Fonte: elaborado pelo autor.

O primeiro quadro representa a criação do agente Avião no Porta-Aviões. A primeira ação do Avião será de levantar vôo para a partir de então, executar o objetivo Mover Aleatoriamente até que seja encontrado um agente Alvo, como representado no segundo quadro. O Avião então sobrevoa o Alvo sem realizar qualquer ação, representado pelo terceiro quadro, enquanto se dirige a ponto intermediário para então poder atacar o alvo. Ao chegar no ponto intermediário o Avião cria um agente Míssil Ar-Terra que irá se dirigir até a posição do Alvo, representado pelos quadros quatro e cinco. Ao alcançar o Alvo, o Míssil Ar-Terra então solicita o serviço Destruir do Alvo, ocorrendo a destruição de ambos os agentes, restando apenas o Avião, como representado no sexto quadro. A partir deste ponto, o Avião verifica se existem outros Alvos para serem destruídos. Em caso positivo volta a executar o objetivo Mover Aleatoriamente. Caso contrário retorna para o Porta-Aviões.

A Figura 12 representa a Execução de agente Bateria Antiaérea desde a sua criação, até atingir o seu objetivo principal, destruir um agente Avião.

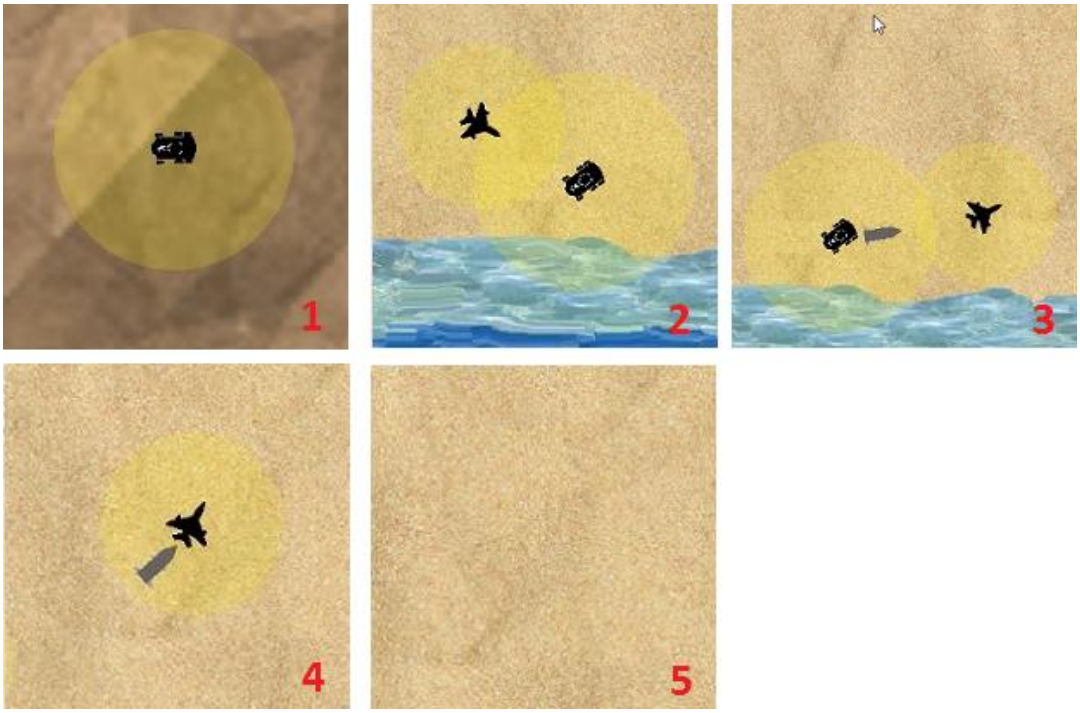

Figura 12 - Execução de um Agente Bateria Antiaérea. Fonte: elaborado pelo autor.

O primeiro quadro representa a criação do agente Bateria Antiaérea em um ponto aleatório no ambiente. A partir de então a Bateria já inicia o objetivo Proteger, onde permanece por alguns segundos na posição atual verificando se agentes do tipo Avião entram em seu campo de visão. Caso nenhum Avião entre em seu campo de visão no tempo determinado, a Bateria Antiaérea se desloca para outra posição (aleatória) no ambiente e então volta a executar a proteção do local. Caso um Avião entre em seu campo de visão, durante a proteção ou mesmo enquanto o agente se movimenta, será criado um agente Míssil Terra-Ar e disparado em direção do Avião, representado pelos quadros dois e três. O Míssil Terra-Ar irá perseguir Avião até alcança-lo ou enquanto o seu tempo de vida não foi esgotado, representado pelo quadro quatro. Caso o Míssil Terra-Ar consiga alcançar o Avião, será feita a solicitação do serviço Destruir do Alvo, ocorrendo a destruição de ambos os agentes, representado pelo quinto quadro. Em caso contrário, apenas o Míssil Terra-Ar será destruído. A partir deste ponto, a Bateria Antiaérea verifica se existem outros Aviões para serem destruídos. Em caso positivo volta a executar o objetivo Proteger. Caso contrário retorna para posição de um agente Alvo ou permanece na posição atual caso todos os agentes do tipo Alvo forem destruídos.

4. Considerações Finais

Nesta seção são apresentadas: as conclusões, as dificuldades encontradas a respeito do trabalho desenvolvido.

4.1. Conclusões

Até o presente momento foram realizadas as seguintes atividades de estudo:

- Estudo sobre sistemas multiagentes
- Estudo sobre o Modelo BDI
- Estudo sobre o *framework* Jadex
- Estudos sobre as aplicações Mundo de Blocos e Mundo de Marte

Com base nos estudos realizados, foi possível consolidar o conhecimento da utilização de sistemas multiagentes aplicados a mundos virtuais, as oportunidades que o Modelo BDI proporciona no desenvolvimento de agentes e a arquitetura de aplicações baseadas no *framework* Jadex.

Após o desenvolvimento da aplicação, foi possível observar que Jadex se possibilita que uma vasta gama de aplicações possam ser desenvolvidas. O *framework* se demonstrou completo quanto ao controle e gerenciamento de sistemas multiagentes. De forma geral, o desenvolvimento dos agentes fica restrito à sua definição. O Jadex proporciona toda a arquitetura que possibilita a criação, interação e comunicação dos agentes, aliado ao suporte de um ambiente virtual pronto para uso. A definição de agentes seguindo o Modelo BDI e a arquitetura de Componentes Ativos, se demonstrou ser bastante simples, principalmente pela familiaridade com linguagem Java. Contudo algumas técnicas utilizadas requereram um conhecimento mais detalhado e específico do Jadex. Adicionalmente, algumas dificuldades foram encontradas, que impossibilitaram que o trabalho avançasse em todos os pontos desejados.

4.2. Dificuldades Encontradas

O objetivo deste trabalho consiste no desenvolvimento de uma aplicação de sistemas multiagentes em um ambiente virtual. A ausência de uma documentação bem estruturada foi uma das maiores dificuldades para o desenvolvimento do trabalho. Muitas técnicas para programar os agentes no Jadex não estão descritas de forma clara em sua documentação. Mesmo com o estudo de aplicações de exemplo, a estrutura do códigos de alguns exemplos contêm pouca ou nenhuma explicação. Exemplo: na documentação do Jadex é descrita que a anotação @GoalCreationCondition deve referenciar o construtor da classe do objetivo. Contudo no exemplo do Mundo de Marte a mesma anotação é utilizada em um método estático. A compreensão de seu uso se deu apenas pela análise da execução da aplicação, por meio da tentativa e erro.

A segunda dificuldade encontrada foi a ausência de captura de Exceções (*Exceptions*) pelo Jadex. Dado uma programação inválida, erros não são mostrados ou são mostradas mensagens genéricas. Exemplo: dado uma declaração inválida na classe de um objetivo, exceções de qualquer natureza (conversão de tipos, objetos nulos, estouro de vetor, etc.) não são mostradas. Diferente da primeira dificuldade, onde a tentativa e erro se demonstrou uma opção para compreensão do problema, a natureza desta dificuldade impossibilita a mesma solução. A dificuldade de encontrar o erro também dificulta a formulação de uma solução.

Outra dificuldade encontrada, que têm como consequência as duas dificuldades anteriores, é a impossibilidade de criar diferentes agentes Misseis em uma mesma execução. Dado que um Míssil Terra-Ar já foi disparado mas não conseguiu atingir o agente Avião, ao ser disparado um Míssil Ar-Terra a aplicação é encerrada sem que qualquer mensagem de erro seja mostrada. Ao ser analisado o depurador da aplicação são mostradas mensagens genéricas que impossibilitam encontrar o problema. Por consequência não foi possível implementar a coordenação de agentes do mesmo tipo, visto que o disparo de Misseis de ambos os tipos se tornaria constante.

Outra dificuldade encontrada foi a divergências entre os padrões do ambiente (EnvSupport) e do Jadex em relação da representação de valores

matemáticos. A tarefa Mover, recupera e a atualiza informações do ambiente, servindo como um sensor dos agentes. Uma das informações recuperadas é o grau de rotação do avatar do agente (inclinação), representado por um vetor de 3 posições (valores do tipo ponto flutuante). Contudo as informações recuperadas do ambiente seguem uma representação diferente da representação utilizada por classes do Jadex. Exemplo: Ao ser recuperado o valor -90º do ambiente, o agente deve levar em consideração que a rotação de seu avatar se encontra em 180º. Para que o agente mude a rotação do seu avatar para 90º, ele deve informar ao ambiente para atualizar o valor para 0º. A Figura 13 demonstra as diferenças no modo de representar ângulos de um plano (geométrico) entre a representação convencional americana, a representação utilizada no ambiente e a representação utilizada em uma das classes do Jadex.

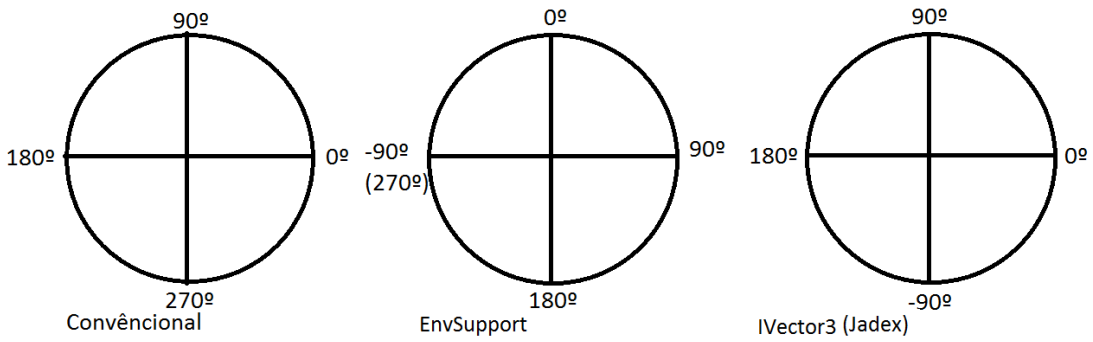

Figura 13 - Diferença da representação de ângulos do EnvSupport com o Jadex. Fonte: elaborado pelo autor.

Em função da existência de divergência entre os padrões matemáticos do ambiente e as classes do Jadex, foi necessário desenvolver métodos para converter as informações antes de serem utilizadas, tanto do ambiente para o agente quanto para o inverso.

4.3. Trabalhos Futuros

Como trabalhos futuros podem ser propostas:

- Finalizar a implementação da coordenação de agentes do mesmo tipo, após a resolução do problema da criação de agentes misseis.
- Explorar a capacidade da ambiente em manipular representações em 3D e implementar uma versão utilizando esta representação.
- Aplicar os conceitos de agentes e do *framework* Jadex em outras áreas e temas.

Um ponto a ser levado em consideração para os trabalhos futuros é que desde o início deste trabalho, novas versões do Jadex foram lançadas, com correções e aprimoramentos do *framework*. É proposto então que sejam realizados testes para verificar os impactos das mudanças do Jadex em sua versão mais atual, que precedam o desenvolvimento de outros trabalhos.

REFERÊNCIAS

ACTIVE COMPONENTS. **Jadex Documentation**. Disponível em: https://download.actoron.com/docs/releases/latest/jadex-mkdocs/. Acesso em 01 de abr 2016.

ANASTASSAKIS, G.; RITCHING, T.; PANAYIOTOPOULOS, T. (2001). **Multi-agent Systems as Intelligent Virtual Environments**. 2001. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.4776&rep=rep1&type=pdf. Acesso em: 01 de abr 2016.

BATES, J.; LOYALL, A. B.; REILLY, W. S. **Integrating Reactivity, Goals, and Emotion in Broad Agent**. 1992. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.9013&rep=rep1&type=pdf. Acesso em 01 de abr 2016.

BORDINI, H. F.; Braubach, L.; Dastani, M.; Seghrouchni, A. E. F.; Gomez-Sanz, J. J.; Leite, J.; O'Hare, G.; Pokahr, A.; Ricci, **A. A Survey of Programming Languages and Platforms for Multi-Agent Systems**. 2006. Disponível em: http://wen.ijs.si/ojs-2.4.3/index.php/informatica/article/viewFile/71/63. Acesso em 25 de jun 2014.

BRATMAN, M. E. **Intentions, Plans, and Practical Reason**. 1999. Cambridge University Press.

DOORENBOS, R.; ETZIONI, O.; WELD, D. S. **A Scalable Comparison-Shopping Agent for World-Wide Web**. 1997. Disponível em: http://eprints.soton.ac.uk/252104/1/IJCIS96.pdf. Acesso em 01 de abr 2016.

GATTI, M. **JADE/JADEX**. 2007. Disponível em: http://www.les.inf.puc-rio.br/wiki/images/e/ef/07-02-JADE-JADEX.ppt. Acesso em 26 de jun 2014.

JENNINGS, N. R.; FARATIN P.; JOHNSON, M. J.; NORMAN, T. J.; O'BRIEN, P.; WIEGAND, M. E. **Agent-Based Business Process Management**. 1996.

Disponível em: http://eprints.soton.ac.uk/252104/1/IJCIS96.pdf. Acesso em 01 de abr 2016.

LENT, M. V.; LAIRD, J. **Developing an Artificial Intelligence Engine**. 1998. Disponível em: http://ai.eecs.umich.edu/people/laird/papers/GDC99.pdf.pdf. Acesso em: 01 de abr 2016.

POSLAND, S.; BUCKLE, P.; HADINGHAM R. **The FIPA-OS agent platform: Open Source for Open Standards**. 2000. Disponível em: http://www2.ic.uff.br/~julius/compmov/FIPAOS.pdf. Acesso em 01 de nov 2014.

SANTOS, C. T. **Um Ambiente Virtual Inteligente e Adaptativo Baseado em Modelos de Usuário e Conteúdo**. 2004. Disponível em: http://osorio.wait4.org/publications/Mestrados/Dissertacao-Cassia-Santos.pdf. Acesso em: 26 de out 2014.

WOOLDRIGE, M. **An Introduction to MultiAgent Systems**. 2002. Disponível em: http://coltech.vnu.edu.vn/httt/media/courses/AI++/Tai%20lieu/TLTK.pdf. Acesso em 28 de jun 2014.