UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO TOMIO HAMA

Uma Plataforma Orientada a Agentes para o Desenvolvimento de Software em Veículos Aéreos Não-Tripulados

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Rafael Heitor Bordini
Orientador

Porto Alegre, abril de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hama, Marcelo Tomio

Uma Plataforma Orientada a Agentes
para o Desenvolvimento de Software em
Veículos Aéreos Não-Tripulados / Marcelo Tomio Hama. – Porto Alegre: PPGC da UFRGS, 2012.

89 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Rafael Heitor Bordini.

1. Inteligência Artificial. 2. Agentes. 3. Veículos Aéreos Não-Tripulados. 4. Plataforma. I. Bordini, Rafael Heitor. II. Título.

“If I have seen farther than others,
it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

“O destino é quem nos embaralha as cartas,
mas nós somos os que jogamos.”
— William Shakespeare

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais Antônio e Aglair por sempre me apoiarem em minhas decisões, e ao meu irmão Ricardo pelo seu companheirismo. Agradeço aos professores do INF pelas aulas ministradas, e aos grandes colegas de laboratório, Luiz, Gabriel e Ricardo. Sem dúvidas, este trabalho teria sido praticamente inconcebível sem algumas sugestões, sem as aulas ministradas ou sem apoio familiar.

Agradecimentos extras vão ao Departamento de Engenharia Elétrica da UFRGS, aos amigos do Seinenkai de São Miguel Arcanjo e aos antigos professores e amigos de Presidente Prudente.

Agradecimentos especiais ao Prof. Dr. Rafael H. Bordini pela paciência e esforço em me orientar neste trabalho, e aos amigos com quem dividi apartamento, Guilherme, Marco, Ivan e Fernando.

Muito obrigado a todos.

Marcelo Tomio Hama

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

3APL	Artificial Autonomous Agents Programming Language
A&A	Agents & Artifacts
API	Application programming interface
BDI	Belief/Desire/Intention
DARPA	Defense Advanced Research Projects Agency
DPS	Distributed
DSL	Damn Small Linux
FIPA	Foundation for Intelligent Physical Agents
FIFO	First In First Out
FMS	Flight Model Simulator
GPS	Global Position System
IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
IDE	Integrated Development Environment
IO	Input/Output
JADE	Java Agent DEvelopment Framework
JDK	Java Development Toolkit
JNI	Java Native Interface
KQML	Knowledge Query and Manipulation Language
MDD	Model Driven Development
OMG	Object Management Group
POO	Programação Orientada à Objetos
POA	Programação Orientada à Agentes
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
SACI	Simple Agent Communication Infrastructure
SDP	Solucionamento Distribuído de Problemas
SMA	Sistema Multi-Agente
UAV	Unmanned Aerial Vehicle
UAVAS	Unmanned Aerial Vehicles AgentSpeak
VANT	Veículo Aéreo Não-Tripulado

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Veículos aéreos não-tripulados (VANTs) são relativamente recentes no meio acadêmico, onde muitas tecnologias e algoritmos vêm sendo pesquisados e desenvolvidos. A engenharia de software apliacada a este âmbito possui poucas abordagens em relação a sistemas autônomos e inteligentes, enquanto que sistemas multi-agentes e a programação orientada a agentes vêm sendo cada vez mais utilizadas. Este trabalho foca na aplicação do paradigma da programação orientada a agentes para o controle de VANTs, de modo a conceber um framework e utilizar arquitetura, teoria e ferramentas orientados a agentes como forma de prover uma abstração mais sofisticada para a programação de comportamentos inteligentes em VANTs. Na pesquisa, propõem-se o modelo ***UAVAS*** – *Unmanned Aerial Vehicles AgentSpeak* que é um *framework* de programação de comportamentos para VANTs que possui um modelo de abstração de veículos aéreos tripulados para veículos aéreos não-tripulados. Ao final, a pesquisa foi avaliada e validada por meio de resultados obtidos em simulações com a infraestrutura implementada. Dois estudos de caso foram realizados, um com ênfase nas comunicações inter-VANTs e cooperação de time, e outro com ênfase nas verificações dos mapeamentos de sinais com o envio de dados da infraestrutura. Para cada um dos casos, simuladores específicos foram criados no intuito de observar as características pertinentes de cada estudo de caso.

Palavras-chave: Inteligência Artificial, Agentes, Veículos Aéreos Não-Tripulados, Plataforma.

An Agent-Oriented Platform For Development and Programming Unmanned Aerial Vehicles

ABSTRACT

Unmanned aerial vehicles (UAVs) are relatively new in civilian context, where many technologies and algorithms have been the focus at much research and development. Software engineering applied to this field has few approaches in relation to autonomous systems and intelligent behavior development, while multi-agent system and agent-oriented programming are being increasingly used. This work focuses on applying the paradigm of agent-oriented programming for the control of UAVs, in order to design a framework and use architecture, theory and agent oriented tools as a way to provide a more sophisticated abstraction for programming intelligent behaviors in UAVs . The main contribution of this work is an architecture that allows the use of the *Jason* platform to program multi-agent system which can control teams of autonomous unmanned aerial vehicles. In this research, we propose the ***UAVAS*** - *Unmanned Aerial Vehicles AgentSpeak* model, which is a *framework* to program intelligent behaviors to UAVs and owns an abstraction model of manned aircraft to unmanned aerial vehicles. At the end, the survey was evaluated and validated by means of results from simulations in the implemented infrastructure. Two case studies were performed, with emphasis on inter-UAV communication and cooperation of team, and the another one focusing on mapping verifications of data signals sent to the infrastructure. For each case, specific simulators have been created in order to observe the relevant characteristics of each case study. end abstract

Keywords: Artificial Intelligence, Agents, Unmanned Aerial Vehicles, Platform.

1 INTRODUÇÃO

Ao longo da história de suas existências, VANTs (veículos aéreos não-tripulados) vêm desempenhando importante e influente papel na tecnologia aérea dos países. Neste escopo, pode-se afirmar que algoritmos para gerenciamento da eletrônica, técnicas computacionais para criação de autonomia, teorias e conceitos voltados para o controle, a navegabilidade, as arquiteturas de sistema, a eficiência do consumo de energia, os sistemas de segurança, os cuidados de cargas, os sensores inteligentes, entre outros, são os principais desafios das últimas pesquisas realizadas.

Verifica-se que as pesquisas voltadas a VANTs não são triviais de serem realizadas, onde uma das causas é a curta idade (aprox. os últimos 20 anos) das principais tecnologias que os suportam. Outrora, os principais engenhos de VANTs foram realizados pelos governos e com foco à guerra e à defesa de estado, fazendo com que somente nestas últimas décadas os meios acadêmicos e civis viessem a usufruir de suas diversas funcionalidades, que além de bélicas, podem ser também de monitoramentos, de coberturas de áreas de risco, de transporte ou de resgates. Apesar da maior abertura da tecnologia ao público não-governamental, VANTs são parte de uma tecnologia de grande impacto e ainda possuem certa restritividade decorrente de leis de confidencialidade (como é o caso de muitos projetos militares), o que oculta algumas de suas pesquisas aos olhos públicos e diminui consideravelmente o número de artigos e plataformas abertas/livres na literatura. Alguns outros grandes obstáculos além destes conceitos burocráticos são a especificidade do hardware e do software, que atualmente são caros por demandarem de conhecimentos especializados. Assim, através de efeitos cumulativos de custo, protótipos de VANTs assumem altos preços e longos tempos de concepção e implementação de projeto. A questão dos locais de realização de testes é também outra grande preocupação visto que estes devem ser cuidadosamente selecionadas por motivos de segurança civil. Pesquisas para VANTs são geralmente complexas e caras devido a uma série de outros fatores advindos da contemplação de múltiplas áreas de conhecimento (engenharias, informática e computação, dispositivos embarcados, aeronáutica, questões de segurança, entre outros).

Adentrando e especializando o contexto de VANTs, podemos abordar a engenharia de comportamentos, onde o principal objetivo é lidar com a modelagem, concepção, cria-

ção e análise de algoritmos que permitam aos dispositivos robóticos em geral assumirem comportamentos autônomos e complexos (DORIGO; COLOMBETTI, 1997; CHEVALIER, 2003). Em geral, a engenharia de comportamentos é realizada principalmente com linguagens da Programação Orientadas a Objetos (POO), como **C++**, **C#** ou **Java**, ou com linguagens de baixo nível como **C** ou **Assembly**, onde a implementação dá-se em *firmware*. Em muitos casos, trabalhar com estes tipos de linguagens dentro do contexto dos VANTs é uma tarefa árdua devido à incompatibilidade de níveis abstrativos: VANTs possuem complexas rotinas e operações (decolar, seguir alvos, enviar mensagens de comunicação, estabelecer rotas ou desviar de objetos) e estas linguagens, por mais bem estruturadas que venham a ser escritas, podem assumir blocos de códigos de difícil compreensão. Uma das recentes abordagens para o auxílio da escrita de comportamentos são os *frameworks*, modelos e arquiteturas da Programação Orientada à Agentes (POA), que oferecem uma metodologia onde as operações e os conceitos dos sistemas são mapeados para outros mais intuitivos e próximos da maneira do raciocínio humano. Apesar destas metodologias serem empregadas, muitos destes *frameworks* utilizam linguagens da POO, fazendo com que ainda existam as incompatibilidades técnicas (e.g. inexpressividade de código). Assim, a implementação de comportamentos de VANTs exige altos conhecimentos teóricos e técnicos por parte dos desenvolvedores, o que implica em um considerável decréscimo do número de profissionais aptos a desenvolver comportamentos e sistemas de controles.

As principais motivações para o estudo desta abordagem baseiam-se na relevância deste assunto nos últimos anos além da relativa escassez de ferramentas de desenvolvimento para VANTs. Segundo pesquisas realizadas por especialistas, prevê-se que, dentro de 3 anos, as aquisições de VANTs venham a dobrar na América Latina. Algumas das possíveis causas para este fenômeno são o avanço da eletrônica, da informática, da globalização dos meios de mídia e de incentivos governamentais. Assim sendo, o desenvolvimento para VANTs vem a se tornar uma promissora área para pesquisas, com aplicações em diversos setores como a agricultura ou e a segurança civil. As poucas abordagens encontradas na bibliografia dão ênfase em pesquisas não totalmente abertas, com ferramentas comerciais. Assim como foi mencionado, a aplicação de agentes inteligentes e sistemas multi-agentes para o controle de VANTs vem se tornando bastante discutida, principalmente a partir desta última década, como uma forma de prover auxílios na abstração, na concepção e no desenvolvimento dos sistemas para possibilitar implementações de comportamentos sofisticados. A abordagem da aplicação de agentes para VANTs vem a ser conveniente e intuitiva devido às características de sistema distribuído que VANTs podem vir a possuir em frotas, grupos ou times. A inteligência e autonomia que podem vir a ser necessárias e os tratamentos de formação de coalizões encaixam-se de forma natural nos modelos da POA. Uma das linhas de pesquisa do universo dos SMA é a engenharia de software orientada à agentes, que ocupa-se da concepção, modelagem e implementação

de linguagens de mais alta abstratividade através de arquiteturas da POA. A arquitetura *Belief/Desire/Intention* (BDI) possui notável histórico na literatura acadêmica sobre os agentes racionais, onde um considerável número de linguagens tiveram suas implementações com base em seu modelo. Tais linguagens, por herdarem as características da POA, são descritivas e expressivas em muitos aspectos, permitindo a construção de expressões de código mais inteligíveis, curtas e robustas.

A principal proposta desta pesquisa é um modelo que possibilita a escrita de comportamentos inteligentes para VANTs, que utiliza linguagens da POA como forma de descrevê-los, e que oferece rotinas e métodos especificos à construção de comportamentos complexos, possibilitando a codificação de missões interativas entre VANTs de maneira mais inteligível aos seres humanos. O modelo realiza a diminuição do "*gap* abstrativo", estabelecendo uma ponte tecnológica entre o programador de comportamentos inteligentes e as arquiteturas em baixo nível. Esta ponte é feita através do mapeamento das instruções pré-definidas em *firmware* para rotinas inerentes ao controle de VANTs (deslocar-se, patrulhar, enviar mensagens, entre outros) por meio de uma infraestrutura orientada a SMA para o desenvolvimento de comportamentos para VANTs. Pressupõem-se que, com a adoção do modelo proposto, desenvolver algoritmos voltados para VANTs venha a ser uma tarefa factível de maneira menos complexa e dispendiosa.

Nesta pesquisa, *AgentSpeak*, cuja principal implementação encontra-se no interpretador Jason[1], foi a linguagem escolhida para a engenharia de comportamentos, por possuir vasta documentação acadêmica e ser uma plataforma aberta. Cada um dos VANTs foi modelado como um SMA, que pode ser habitado por um ou mais agentes, em analogia a uma aeronave real, que pode ser controlada por um ou mais tripulantes (aspectos detalhados serão dados no capítulo 4). Na pesquisa, propõem-se o modelo ***UAVAS*** – *Unmanned Aerial Vehicles AgentSpeak* que é um *framework* de programação de comportamentos para VANTs que possui um modelo de abstração de veículos aéreos tripulados para veículos aéreos não-tripulados. Ao final, a pesquisa foi avaliada e validada por meio de resultados obtidos em simulações com a infraestrutura implementada. Dois estudos de caso foram realizados, um com ênfase nas comunicações inter-VANTs e cooperação de time, e outro com ênfase nas verificações dos mapeamentos de sinais com o envio de dados da infraestrutura. Para cada um dos casos, simuladores específicos foram criados no intuito de observar as características pertinentes de cada estudo de caso.

Esta dissertação se divide da seguinte maneira: na primeira parte da dissertação, nos capítulos 2 e 3, são oferecidas as bases teóricas gerais em SMA e veículos aéreos não-tripulados, com abordagens dos principais e atuais problemas em relação ao desenvolvimento para POA, VANTs e explanações do estado da arte. A segunda parte, nos capítulo 4 e 5, define como os SMA inserem-se no contexto dos VANTs. No contexto deste trabalho, são mostradas também as técnicas e abordagens adotadas, bem como a definição

[1]Página eletrônica em: `http://jason.sourceforge.net/`

e os detalhes das implementações do modelo proposto. A terceira parte, nos capítulos 6 e 7, detalha os resultados pertinentes às validações gerais através de estudos de caso e simulações específicas, apresentando também as conclusões gerais do estudo.

2 AGENTES E SISTEMAS MULTI-AGENTES

Na ciência da computação, a inteligência artificial (IA) é a área de pesquisa que dedica-se a realizar estudos de conceitos, concepções de modelos, e desenvolvimento de métodos e técnicas no intuito de fornecer habilidades de autonomia e adaptabilidade às máquinas. A inteligência artificial distribuída (IAD) é uma das subáreas da IA e aborda-a com a proposta da divisão de problemas em sub-problemas, pequenos e mais simples destacando a interação como fundamental solução. Desta forma, sistemas que venham a utilizar as técnicas da IAD podem resolver tanto os problemas virtualmente distribuídos quanto os fisicamente distribuídos, tornando as modelagens mais intuitivas e compreensíveis em diversos casos. Por sua vez, a IAD é uma união de dois grandes tópicos, onde um deles são os sistemas multi-agentes (SMA) e o outro são os solucionamentos distribuídos de problemas (SDP). Os SMA diferenciam-se dos SDP na forma como são solucionados os problemas. Em um SDP os componentes autônomos operam sobre um mesmo problema (tendo em analogia, pessoas tentando resolver um mesmo quebra-cabeças), enquanto que em um SMA os sistemas são implementados como entidades autônomas, os quais operam em diversos tipos de problemas (tendo em analogia, pessoas trabalhando em uma bolsa de valores, com diferentes objetivos). Um agente é, neste contexto, uma entidade particular de um SMA, onde tal SMA pode ser homogêneo (composto por um único tipo de agente) ou heterogêneo (composto por diferentes tipos de agentes, com diferentes capacidades e habilidades).

2.1 Definições e Contexto Histórico

Preliminarmente, é interessante mencionar que as idéias e conceitos que fornecem base aos modelos de agentes e SMA provêm de muitas e diferentes áreas, não prendendo-se somente à computação, mas vindos também dos modelos de sistemas organizacionais, filosofia lógica, psicologia, entre outras. No transcorrer histórico da área de agentes e SMA os sistemas lógicos possuem forte influência (WRIGHT, 1951; HINTIKKA, 1962; HOARE, 1969; SMULLYAN, 1986; WIERINGA; MEYER, 1993; HUTH; RYAN, 2004; DAWSON, 2006; MELO et al., 2009), com a utilização da lógica modal para a concepção

de linguagens abstratas, modelagens de raciocínios e estruturação de regras, imbuídas de muitas de suas características. Um dos maiores produtos dos modelos lógicos foi a *Knowledge Query and Manipulation Language* (KQML) que é uma linguagem que protocola a comunicação entre agentes e sistemas baseados em conhecimento (FININ et al., 1994) desenvolvida no início dos anos 90 como parte do projeto DARPA, e que tinha como um dos objetivos iniciais o desenvolvimento de técnicas para a construção de bases reusáveis de conhecimento em larga escala. A *KQML* foi originalmente concebida como uma interface para sistemas de bases de conhecimento e, posteriormente, direcionada ao âmbito da comunicação de agentes.

2.1.1 Agente

O conceito de agentes inteligentes possui diversas definições. Uma das vertentes literárias define agentes como sendo um sistema computacional capaz de executar, de forma autônoma, ações em um ambiente no qual este se situa, com o propósito de realizar objetivos a ele delegados (WOOLDRIDGE, 2009). Outra definição diz que agentes são entidades autônomas as quais realizam observações através de sensores e agem sob um determinado ambiente, guiados por seus objetivos (RUSSELL; NORVIG, 1995). Nestas últimas décadas, agentes inteligentes e sistemas multi-agentes vêm ganhando bastante espaço na literatura como abordagem para modelar resoluções de problemas em diversos setores tanto na área acadêmica quanto no comercial e/ou industrial, e são grandes os números de definições.

De modo geral, um agente difere de um objeto (noção abstrativa da POO) principalmente nos aspectos da arquitetura interna e na maneira com que realizam envio e recebimento de mensagens (SHOHAM, 1993). Na POO, os objetos são instanciações de classes que declaram variáveis de tipos quaisquer sob estruturas quaisquer, e enviam mensagens uns aos outros através de métodos. Na programação orientada a agentes, um agente possui em sua arquitetura interna uma estrutura utilizada para armazenar regras e premissas lógicas chamada de *base de crenças*[1]. Com esta estrutura, um agente tem a capacidade de guardar informações a respeito de seus ambientes e/ou de outros agentes. Agentes podem ainda realizar performativas de comunicação definidas (*tell*, *ack*, *ask*, etc) ao invés de métodos tipados, fazendo com que a teoria da comunicação e pesquisas sobre modelos de atos de fala (i.e. (COHEN; LEVESQUE, 1988, 1990)) venham a ter forte influência nas modelagens dos agentes. Assim, a POA acrescenta níveis abstrativos à POO, especializando-a de maneira mais intuitiva e próxima dos modelos de raciocínio humano. Nestas definições, o termo **agente** faz sentido apenas quando as noções de **ambiente** também estão presentes, dado que os agentes agem sob e percebem as informações externas destes ambientes. Assim, um agente tem em sua autonomia o recebimento de informações, o raciocínio sobre o ambiente, o discernimento sobre quais ferramentas fazer uso,

[1] Em relação aos agentes que implementam a arquitetura *Belief/Desire/Intention.*

e a reflexão de conceitos sobre os demais possíveis agentes que habitam o sistema, decidindo então as ações mais cabíveis a serem tomadas, dado o corrente contexto, e quais objetivos deverão ser seguidos. Paralelamente aos agentes, ambientes possuem entidades passivas desprovidas do poder de controle sobre suas ações, os quais podem possuir diversas denominações como ferramentas, artefatos, ambientes de rede e objetos. A Figura 2.1 retrata um modelo de um agente simples.

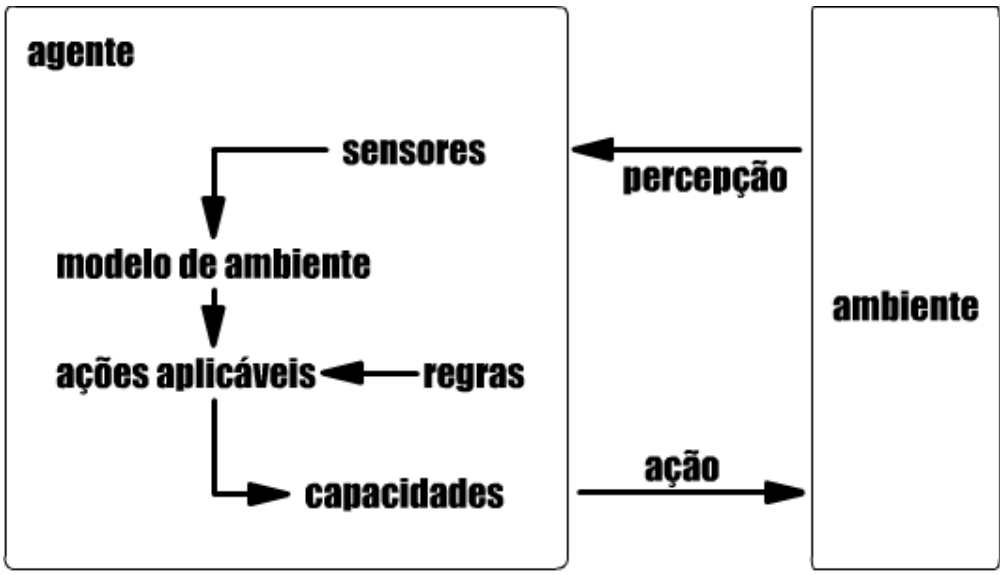

Figura 2.1: Modelo de um agente simples.

Agentes desempenham papel importante em sistemas inteligentes onde, após implementados, podem vir a dispor da autonomia que lhes é característica. O termo autônomo designa a característica de existência própria do agente, independente da existência de outros. Usualmente, cada um dos agentes possui capacidades comportamentais, objetivos e poder de raciocínio necessários para sua atuação no ambiente. De maneira conceitual, um agente difere de um objeto (referente à POO) principalmente através dos parâmetros que os definem (módulos, conjunto de regras, bases de conhecimentos) e das noções de comunicação com o exterior. Objetos realizam operações por terem sido programados para tal, e agentes realizam operações porque decidiram realizá-la (WOOLDRIDGE, 2002).

2.1.2 Sistema Multi-Agente

Já em sentido mais amplo, um SMA é um sistema complexo composto por iguais ou diferentes tipos de agentes, cada qual podendo realizar interações com o ambiente do sistema. Em um SMA, os agentes são cada uma das entidades ativas onde um conjunto destes pode vir a formar uma sociedade complexa e interativa. No contexto de troca de informações, denomina-se **interação entre agentes** a troca de informações de agente a agente e **interação entre agente e ambiente** as interações realizadas entre os agentes

e seus ambientes podendo ou não alterá-los através de suas ações. As interações entre agentes podem ser realizadas de forma direta (comunicação explícita) ou de modo indireto, com a emissão de sinais através do ambiente. Um SMA é composto por agentes que podem ser reativos ou cognitivos e tem por principais características de implementação a distributividade e concorrência de seus agentes onde cada um destes possui sua própria *thread* de execução ou processo de controle (ODELL, 2000).

2.1.3 Classes de Ambientes

Tão importantes quanto os agentes de um SMA, ambientes possuem papel único nos modelos. É a partir da noção de ambiente que o sentido da existência do agente toma forma, pois um agente diferencia-se de um módulo computacional ou um objeto da POO por ter a capacidade de interagir de forma autônoma neste ambiente. Em (WOOLDRIDGE, 2002) há uma discussão das noções gerais e classificações de ambientes em quatro tipos de parâmetros: **acessível/inacessível**, **determinístico/indeterminístico**, **estático/volátil** e **discreto/contínuo**. A tabela 2.1 explana resumidamente cada um destes parâmetros.

Parâmetro	**Característica**
Acessível/Inacessível	Diz respeito à acurácia, confiabilidade e completeza das informações que os agentes podem obter do ambiente.
Determinístico/Indeterminístico	Diz respeito às noções de causa e efeito do ambiente. Um ambiente determinístico é aquele em que, do ponto de vista do agente, para cada ação, há um único e garantido efeito.
Estático/Volátil	Diz respeito às alterações que ocorrem no ambiente. O ambiente estático é aquele que tem seu estado alterado somente através das ações de um agente.
Contínuo/Discreto	Diz respeito às ações e percepções que o ambiente disponibiliza. O ambiente discreto é aquele em que há um número fixo e finito de ações e percepções.

Tabela 2.1: Classificações de ambientes.

Tomando como exemplo um tabuleiro de xadrez [2] temos um ambiente acessível pois todas as informações relevantes de um jogo podem ser observadas pelos agentes que o jogam. O tabuleiro é determinístico pois enquanto um jogo não acaba, cada estado é atingido somente com uma ação de um agente (mover uma peça), segundo uma transição do tipo $tabuleiro_{(p)}$ + $Ag_{(jogada)} \rightarrow tabuleiro_{(q)}$, onde $p \neq q$, $p \in configuracoes$, $q \in$

[2]Regras do xadrez: `http://www.fide.com/component/handbook/`, acessada em 25/11/2011.

$configuracoes$, e tal que $configuracoes$ é o conjunto de todas as possíveis e válidas configurações de arranjos das peças no tabuleiro. O ambiente é estático, visto que as peças e casas do jogo não podem alterar suas posições de lugar enquanto os agentes estiverem deliberando. O tabuleiro é um ambiente discreto, visto que o jogo possui um número finito de percepções e ações possíveis.

2.1.4 Arquitetura BDI

A arquitetura *Belief/Desire/Intention* (BDI) (BRATMAN, 1981) é um modelo de software desenvolvido para a programação de agentes inteligentes caracterizado pela aplicação das noções de crenças (o que se acredita), desejos (um objetivo real, instanciado) e intenções (sequência de atos ou planos, escolhidos deliberadamente de modo a satisfazer os desejos). Em essência, este modelo fornece um mecanismo para separar a atividade de seleção de um plano (que pode vir de uma biblioteca ou um planejador externo) da execução de planos atualmente ativo. Como consequência, os agentes que herdam as características do modelo BDI são capazes de equilibrar o tempo gasto em deliberar sobre os planos (escolher o que fazer) e executar os planos propriamente ditos (fazê-lo). A atividade de criar planos (planejar), não está dentro do escopo do modelo, e é deixada para os projetistas do sistema e programadores.

2.2 Tipos de Sistemas Multi-Agentes

Segundo (RUSSELL; NORVIG, 1995; ALMEIDA, 2007) um SMA pode ser classificado em duas principais classes a se saber: os Sistemas Multi-Agentes Reativos, que trabalham com o desenvolvimento de sistemas que geralmente utilizam um grande número de agentes simples; e os Sistemas Multi-Agentes Cognitivos, que em geral utilizam poucos agentes, mas que realizam tarefas mais complexas.

2.2.1 Sistemas de Agentes Reativos

Na abordagem de SMA Reativo é muito comum os agentes serem baseados em modelos de organização biológica ou etológica (formigas, cupins, abelhas, entre outros), onde o modelo de funcionamento de um agente é formado pelo par Estímulo-Resposta (Ação-Reação). As principais características dos agentes e dos SMA Reativos são destacadas a seguir:

- O conhecimento (regras de comportamento) dos agentes é implícito e a manifestação externa de inteligência do sistema se dá através do comportamento dos agentes;

- Não há uma explícita representação interna do ambiente nos agentes: o comportamento (resposta) de cada agente é baseado no que ele percebe (estímulo) a cada instante;

- Não há registro das ações: os agentes reativos não mantém nenhum tipo de histórico de suas ações, de modo que o resultado de uma determinada ação passada não influencia diretamente nas decisões de uma ação futura;

- Organização etológica: a forma de organização dos SMA reativos é similar à observada por animais que vivem em grandes comunidades;

- Grande número de membros: em geral, os SMA reativos possuem um grande número de agentes, com populações que podem chegar à ordem dos milhares.

2.2.2 Sistemas de Agentes Cognitivos

Os agentes cognitivos são baseados em organizações sociais humanas como grupos, hierarquias e mercados. As principais características dos SMA cognitivos são:

- Representação explícita do ambiente e dos outros agentes da sociedade;

- Registro histórico de ações e estímulos, sendo capazes de planejar ações futuras através de suas experiências;

- Sistema de percepção que permite examinar o ambiente, bem como sistema de comunicação que permite a troca de mensagens entre agentes, sendo os dois distintos entre si. A comunicação entre agentes é realizada de modo direto através do envio e recebimento de mensagens;

- Mecanismo de controle deliberativo. Os agentes cognitivos podem raciocinar e decidir em conjunto sobre quais ações devem ser tomadas, que planos devem seguir e que objetivos devem alcançar;

- Usualmente poucos agentes, na ordem de algumas dezenas.

Os SMA são modelos computacionais que podem usufruir da inteligência artificial para prover níveis de proatividade para os agentes e permitir a resolução de tarefas gerais em que planos, decisões, objetivos e crenças são as variáveis dos entes do sistema. Dentre os modelos de arquiteturas para SMA a BDI é uma das mais abordadas pela literatura. Cada uma das abordagens apresenta inúmeras importâncias diante dos diferentes contextos a que podem estar relacionadas. Cabe a ressalva de que diferentes problemas necessitam de diferentes tipos de abordagens, arquiteturas ou número de agentes para a obtenção de melhores resultados. Deve-se ter em mente os problemas em que se está trabalhando de modo a saber-se o porquê da utilização do modelo e paradigma da POA, bem como suas melhores funcionalidades e seus notáveis defeitos.

2.2.3 Inteligência Distribuída

A principal idéia de um SMA vem da noção de que um comportamento global pode ser alcançado a partir do comportamento individual de cada um dos agentes. Neste sentido, em um SMA não é estritamente necessário que os agentes que o compõem sejam individualmente inteligentes para que seja alcançado um comportamento global inteligente. Em organizações, podem ainda existir definições de restrições e leis aplicadas aos agentes que a ela pertencem, de modo que se possa garantir que cada agente desejará a resolução dos problemas propostos seguindo determinadas regras do sistema.

2.3 Metodologias em SMA

Além dos modelos teóricos no percurso das pesquisas de agentes e SMA, há também muitas frentes em metodologias de projeto para sistemas e técnicas para automatizações nas implementações de SMA. Em muitos casos, para se conceber e implementar um SMA são utilizadas metodologias específicas com o objetivo de facilitar o entendimento da abstração e a automatização na fase de design. A POA diversifica-se assim como diversificam-se as linguagens, metodologias e ferramentas criadas. Para o auxílio das metodologias e incremento da eficiência das codificações, existem *frameworks* e IDEs em desenvolvimento. Existem algumas (e.g. (ODELL; PARUNAK; BAUER, 2000; COLLINOT; DROGOUL, 1998; BRAZIER et al., 1997)) abordagens de metodologias de engenharia de SMA onde em particular, duas delas são bastante referenciadas por serem abordagens maduras e englobarem tanto o nível macro (sistemas e organizações de agentes) quanto o nível micro (aspectos do agente no sistema). A seguir, uma breve descrição das duas metodologias.

2.3.1 A Metodologia Prometheus

Prometheus (PADGHAM; WINIKOFF, 2002, 2004) é uma metodologia de desenvolvimento de agentes e SMA BDI através de uma linguagem descritivamente genérica independente de design, arquitetura ou implementação. A metodologia enfoca a noção de diagramas compostos por entidades, cada qual definindo abstrativamente as principais noções de um SMA (percepção, ação, agente, protocolo, capacidades, mensagens, etc). A Figura 2.2 mostra o diagrama do *Prometheus*, onde ilustram-se as relações entre o design, a arquitetura e a especificação do SMA a ser desenvolvido.

2.3.2 A Metodologia Gaia

O modelo *Gaia* (WOOLDRIDGE; JENNINGS; KINNY, 2000) permite aos desenvolvedores de SMA realizar a implementação direta do design a partir da captura dos requisitos pré-definidos (o qual pode ser realizado independente da metodologia). *Gaia* provê mecanismos para realizar um passo-a-passo das fases mais abstratas para as fa-

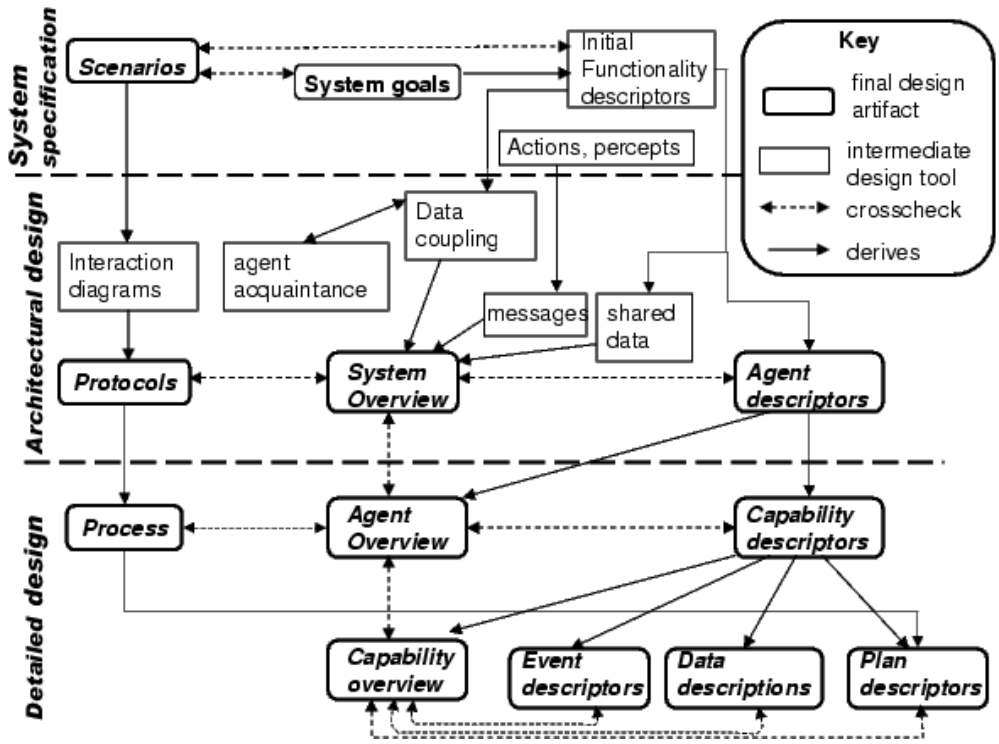

Figura 2.2: Diagrama do *framework Prometheus*. Fonte: Padgham L. and Winikoff M.

ses mais concretas do sistema de maneira iterativa. A Figura 2.3 mostra as relações dos modelos da metodologia *Gaia*.

2.4 *Jason/AgentSpeak*

Jason/AgentSpeak é uma implementação de linguagem orientada à agentes baseada em lógica de predicados. A *AgentSpeak*(L) (RAO, 1996) é uma linguagem orientada à agentes que possui sua mais recente versão implementada no interpretador Jason (BORDINI; HüBNER, 2005; BORDINI; HüBNER; WOOLDRIDGE, 2007), tendo relevante histórico documentado na literatura e tendo raízes em duas grandes frentes de pesquisas: a arquitetura e a lógica modal BDI. A arquitetura BDI adota um modelo de software similar ao raciocínio humano a respeito de intenções, crenças e desejos, onde tais noções são interpretadas por um raciocinador que realiza transições internas para descrever um modelo de pensamento através de ciclos. A lógica modal é utilizada em *AgentSpeak* através de literais que são empregados para definir os objetivos, e as crenças de seus agentes.

Os modelos de representação do conhecimento possuem relativa idade, onde os primeiros estudos pertencem aos anos 50 com a representação através da lógica modal (HINTIKKA, 1962). Em 1972, surge a Prolog que é uma linguagem de programação que se enquadra no paradigma de programação em lógica matemática, de uso geral e especialmente associada com a inteligência artificial e lingüística computacional. Pode-se dizer

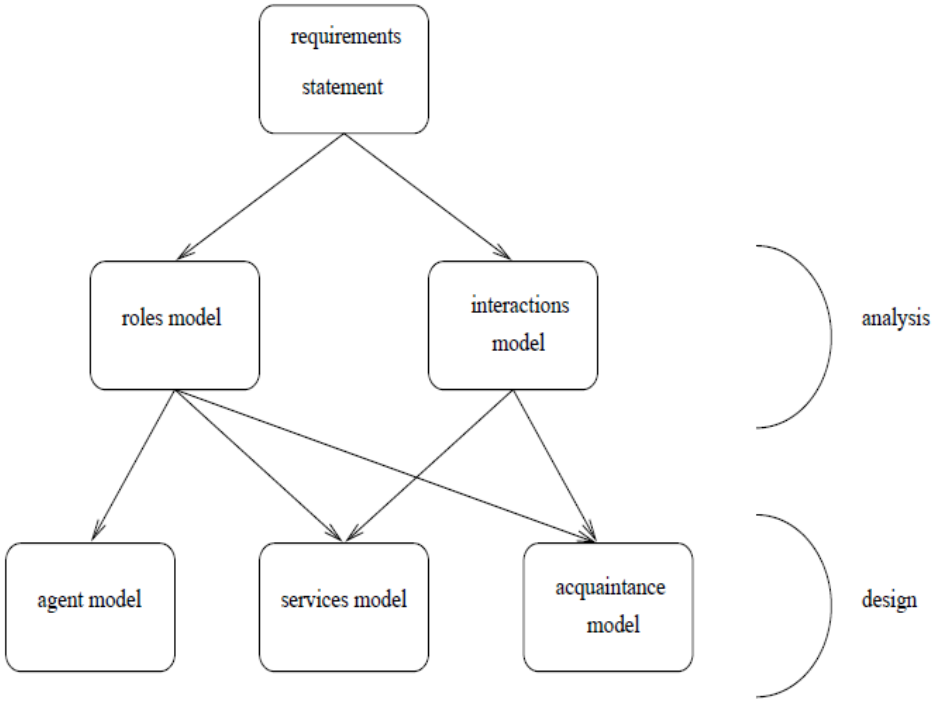

Figura 2.3: Relação dos modelos em *Gaia*. Fonte: Wooldridge M., Jennings N. R. and Kinny D.

que as pesquisas em lógica aplicada à represetação de crenças e os fundamentos das intenções e atos de fala (COHEN; LEVESQUE, 1990, 1988) foram os pontos de partida para as primeiras modelagens em linguagens orientadas à agentes. *AgentSpeak(L)* foi uma das diversas linguagens concebidas, e parte das idéias propostas pela arquitetura e lógica BDI (RAO; GEORGEFF, 1993, 1991). Através de todos estes conceitos e com a utilização da linguagem JAVA, *JASON* implementa *AgentSpeak* com a semântica operacional. A Figura 2.4 mostra a relação entre *AgentSpeak* e o tempo transcorrido, desde pesquisas iniciais com a lógica modal até os dias atuais da linguagem, com algumas das mais importantes pesquisas em destaque.

2.4.1 Funcionalidades e Extensões

Em relação à *AgentSpeak*, Jason/*AgentSpeak* possui diversas características extras como a *Strong Negation*[3] que permite a utilização das duas noções de mundo (*close world assumption*[4] e *open world assumption*[5]), gerenciamento de falha de planos, gerenciamento de atos comunicativos entre agentes, anotações para meta-informações, modo debugging, entre outros.

[3]Negação forte, que denota o conhecimento de que uma dada assertiva é falsa.
[4]Modelo de semântica de crenças onde o não-conhecimento de uma assertiva define-a como sendo falsa
[5]Modelo de semântica de crenças onde o não-conhecimento de uma assertiva define-a como sendo desconhecida.

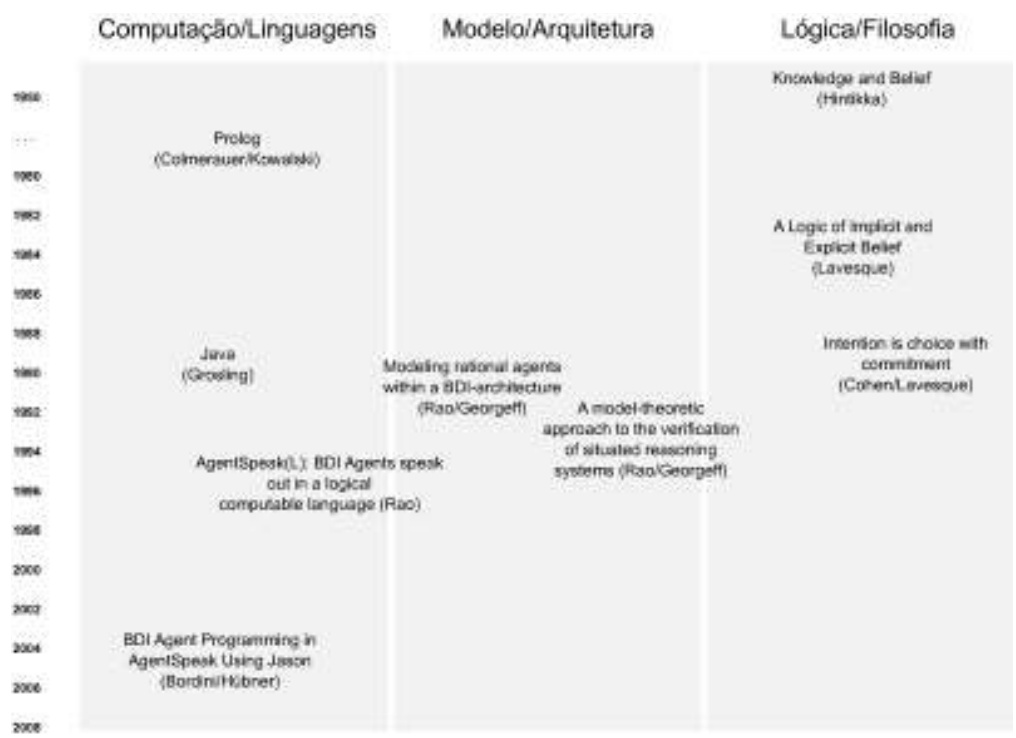

Figura 2.4: Contexto das pesquisas relacionadas à AgentSpeak ao longo do tempo.

2.4.2 Sintaxe e Estruturação

Um agente *AgentSpeak* é definido por um conjunto de crenças que provêm o estado inicial de sua base de crenças (sua memória inicial). Este conjunto de crenças é definido através de fórmulas lógicas atômicas (de primeira ordem). O agente pode ainda ter *planos* registrados em sua biblioteca de planos que podem ser ativados dependendo do contexto a que ele se encontrar. Um plano possui um cabeçalho que consiste de um evento de ativação (especificando o tipo de evento para o qual o plano é relevante), e uma conjunção de crenças literárias que representam seu contexto. A conjunção de literais no contexto precisa ser uma consequência lógica das atuais crenças que o agente possui no momento em que o plano venha a ser considerado apropriado, por exemplo, o momento em que o evento de ativação ocorreu (assim, apenas planos aplicáveis podem ser escolhidos para serem executados). Outro componente da estrutura dos planos é o corpo, o qual é composto por uma sequência de ações básicas ou sub-objetivos, os quais o agente deverá executar para completar seus objetivos. Ações básicas são representadas por operações atômicas as quais o agente realiza para alterar as condições do ambiente de acordo com seus desejos. Estas ações podem ainda serem escritas como fórmulas atômicas, utilizando-se um conjunto de símbolos especiais ao invés de símbolos de predicados. Um agente *AgentSpeak* distingue dois tipos principais de objetivos: *Achievement Goals* e *Test Goals*. *Achievement Goals* são executados através de uma fórmula atômica com o operador-prefixo $\{!\}$,

enquanto que os *Test Goals* são executados através do operador-prefixo {?}. Planos são ativados através dos operadores {+} (planos principais) ou {−} (planos de contingência). No excerto nas listagens 2.1 é mostrado um pequeno trecho em linguagem *AgentSpeak*, definindo um plano de reação a uma mudança de crença, um plano para atingir um objetivo e um plano de contingência. A BNF[6] da linguagem pode ser vista no apêndice A.1.

Listing 2.1: Exemplos da linguagem *AgentSpeak*

```
+rule_one(Arg):
   context_of_rule <-
      !calculate_something(Arg).

+!main_plan:
   context_one & context_two(X) & not context_three <-
      !subplan_one;

-!contingence_plan:
   context_one <-
      !subplan_two;
```

2.4.3 O Interpretador *Jason*

Como mencionado anteriormente, *Jason* é um interpretador para a linguagem *AgentSpeak* (versão extendida da *AgentSpeak*(L)) que implementa a semântica operacional (PLOTKIN, 1981) e provê uma plataforma de desenvolvimento para SMA com diversas funcionalidades customizáveis. Atualmente, JASON fornece plugins de integração com IDEs, plataformas como SACI[7] (HUBNER; SICHMAN, 2000) e JADE[8] (BELLIFEMINE; POGGI; RIMASSA, 1999), e recursos adicionais como biblioteca de ações. Muitos projetos de pesquisa no âmbito de SMA incluem *Jason* como principal interpretador da camada de agentes. Recentemente, alguns dos projetos de pesquisa que utilizam *Jason* são:

- UAVAS[9]: Um *framework* orientado à agentes para programação e desenvolvimento de veículos aéreos não-tripulados (HAMA et al., 2011), descrito nesta dissertação;
- JASDL[10] Uma combinação entre agentes e ontologias, através da extensão da plataforma Jason (KLAPISCAK; BORDINI, 2008);
- MADeM[11] Agentes capazes de realizar decisões em niveis sociais (GRIMALDO;

[6] Fonte em `http://jason.sourceforge.net/Jason.pdf`

[7] *Framework* cujo principal objetivo é disponibilizar protocolos de comunicação para a programação de agentes. É construído sob a *API Java Standard Edition* e composto de ferramentas que oferecem facilidades no desenvolvimento de sociedade de agentes e IA distribuída.

[8] Um *framework* implementado em Java que objetiva simplificar as codificações de SMA através de um *middleware* em conformidade com as especificações da *FIPA*, provendo ferramentas gráficas que suportam *debugging* e implantação por fases.

[9] Unmanned Aerial Vehicles AgentSpeak

[10] *Jason* AgentSpeak-DescriptionLogic.

[11] Multi-modal Agent Decision Making. Webpage em `http://www.uv.es/grimo/jmadem/index.html`, acessada em 25/11/2011.

LOZANO; BARBER, 2008);

- JaCaMo[12]: Uma combinação das tecnologias Jason (BORDINI; HüBNER; WOOLDRIDGE, 2007), CArtAgO (Common ARTifact infrastructure for AGents Open environments) (RICCI et al., 2009) e Moise (HUBNER et al., 2007) para o desenvolvimento de sistemas multi-agentes. Nesta combinação, a camada de agentes é implementada em Jason, a camada de ambientes é implementada em CArtAgO e os niveis organizacionais são implementados em Moise.

Uma listagem mais detalhada pode ser obtida na página web do interpretador *Jason*, referenciada anteriormente. Tais ferramentas e modelos apresentam-se como objetos de estudos atuais, onde cada uma disponibiliza extensões e customizações extras visando facilitar o desenvolvimento de modelos específicos, a criação de novas metodologias, o aperfeiçoamento da engenharia de software e o aumento da eficiência no contexto da programação de agentes. Paralelamente à *AgentSpeak*, existem outros modelos de linguagens e ferramentas para a POA sendo desenvolvidas e implementadas. Algumas delas são:

- *3APL*: Uma linguagem com propósitos experimentais para desenvolvimento, implementação e testes de múltiplos agentes cognitivos utilizando a arquitetura BDI (HINDRIKS et al., 1999). Desenvolvida e mantida por pesquisadores de computação da University of Utrecht. A *3APL* é relativamente simples, com uma sintaxe constituída de operadores booleanos básicos *AND*, *OR* e *NOT*, sentenças condicionais *IF-THEN-ELSE* e controladores de fluxo *WHILE-DO*;

- *IndiGolog*: É uma outra linguagem que aparece bastante na literatura, e definida como uma linguagem de alto nível para a programação de robôs e agentes inteligentes (GIACOMO et al., 2009) que suporta formulações e execuções de planos *on-the-fly* em ambientes. A *IndiGolog* pode ser utilizada para escrever controladores de programas para robôs que venham a combinar planejamento, percepção e reatividade;

- *GOAL*: Uma linguagem utilizada para a programação de agentes racionais, que derivam suas escolhas em função de suas crenças e desejos (HINDRIKS; HOEK, 2008). Provê construções básicas que facilitam a manipulação BDI e a estruturação das implementações das decisões, oferecendo um *framework* baseado em senso comum ou raciocínio prático.

- *Moise*: Um modelo organizacional para SMA baseado nas noções de papéis, obrigações, grupos e missões, permitindo que um SMA possua especificações explícitas de sua organização, que podem ser utilizadas tanto pelo raciocínio dos próprios agentes quanto pela plataforma que executa tal organização.

[12] Webpage em `http://jacamo.sourceforge.net/`, acessada em 25/11/2011

- *CArtAgO*: CArtAgO (RICCI; VIROLI; OMICINI, 2008) é uma infraestrutura genérica que disponibiliza ferramentas para a execução de ambientes virtuais de SMA baseada no meta-modelo *A&A* (OMICINI; RICCI; VIROLI, 2008) para modelar e projetar um SMA. O meta-modelo *A&A* introduz abstrações de alto nível inerentes ao trabalho cooperativo entre seres humanos em um ambiente onde agentes são entidades computacionais que executam algum tipo de tarefa (em analogia a trabalhadores humanos), e artefatos são recursos e ferramentas dinamicamente construídos usados e manipulados pelos agentes;

- *JACK Intelligent Agents* Um *framework* escrito em JAVA e baseado em POA para a programação de agentes inteligentes com a utilização de noções da arquitetura BDI (BUSETTA et al., 1999), é um sistema comercial de código fechado.

2.5 Possíveis Alvos das Aplicações

Agentes e SMA possuem um grande leque de aplicações, as quais necessitam de *frameworks* e plataformas. Desde suas iniciais concepções até a atualidade, houveram muitas e diferentes subáreas de pesquisas. Agentes e SMA são modelados, implementados e utilizados em inúmeros possíveis contextos (JENNINGS; SYCARA; WOOLDRIDGE, 1998) e contemplam uma grande variedade de possíveis aplicações. Algumas destas possíveis aplicações estão em simulação de entidades biológicas (KHAN et al., 2003), em comércio eletrônico (GUTTMAN; MOUKAS; MAES, 1998), em controle de veículos (SISLAK et al., 2008; BAUZA; GOZALVEZ; SANCHEZ-SORIANO, 2010), em sistemas operacionais (SVAHNBERG; DAVIDSSON; GRAHN, 1998), em simulações de sociedades (BERRY; KIEL; ELLIOTT, 2002), em jogos (DIXON, 2011), em monitoramentos (MANGINA, 2005), entre muitos outros. Este trabalho enfatiza a aplicação de agentes para o controle de VANTs, de modo que teorias, técnicas de programação e abstratividades inerentes aos SMA venham a ser possíveis de serem aplicadas no contexto de VANTs e frotas de VANTs. O próximo capítulo abordará, com maiores detalhes, este recente contexto de aplicação de agentes.

3 VEÍCULOS AÉREOS NÃO-TRIPULADOS

VANTs (veículos aéreos não-tripulados) vêm desempenhando importante em um contexto geral. VANT ou UAV (Unmanned Aerial Vehicle)[1] é um veículo aéreo (aeronave) que dispensa a presença de piloto em cabine, podendo ser controlado através de controladores remotos (i.e. um piloto com rádio-transmissor em uma estação) ou voar de forma autônoma por meio de algoritmos pré-programados. Esta característica lhe dá a importante capacidade de poupar a presença de vidas humanas em cabine durante situações de riscos. Apesar de existirem tecnologias recentes, o conceito de VANT não o é e as primeiras utilizações para fins militares datam de períodos próximos a 1880, como mostra a Figura 3.1, uma concepção artística do ataque aéreo austríaco na cidade de Veneza.

Figura 3.1: Ataque aéreo por balões em 1880. Fonte: Prof. Jurij Drushnin.

[1] Como é conhecido internacionalmente.

3.1 Definições e Conceitos de VANTs

Após algumas décadas, as definições para VANTs formalizaram-se de modo que um conceito mais técnico veio a ser estabelecido. Na atualidade, um VANT é definido (NATO, 2001) como sendo um veículo que contempla quatro características:

1. Utiliza-se de forças aerodinâmicas para locomover-se no espaço aéreo;

2. Não possui piloto, podendo ser controlado remotamente através de transmissor em radio-frequência e/ou assumindo comportamento autônomo;

3. Pode ser customizado e reutilizado;

4. Tem capacidade de carga, que pode ser tanto uma carga letal quanto não-letal.

3.1.1 Classificações de Veículos Aéreos Não-Tripulados

Pelas dadas definições, podemos comparar algumas diferentes categorias de veículos aéreos. A tabela 3.1 mostra as características de um VANT em comparação com outros tipos de categorias.

Veículo Aéreo	Locomoção	Cap. de Carga	Reuso	Piloto em Cabine
Balão	Convecção	Sim	Sim	Dispensável
Míssel Teleguiado	Combustão/Aerodinâmica	Sim	Não	Ausente
Caça F-16	Combustão/Aerodinâmica	Sim	Sim	Presente
Bimotor	Aerodinâmica	Sim	Sim	Presente
Dirigível	Convecção	Sim	Sim	Presente
Foguete V-2	Combustão/Aerodinâmica	Sim	Não	Ausente
VANT	Combustão/Aerodinâmica	Sim	Sim	Ausente

Tabela 3.1: Tipos de Veículos Aéreos e suas Características

Um Balão locomove-se através dos fenômenos térmicos da convecção de ar quente e frio ou através das diferenças de densidade de gases. Têm a capacidade de transportar cargas e podem ser usados em muitas viagens e passeios. Balões dispensam a presença de pilotos, pois podem transportar pessoas, no caso dos balões de passeio, ou simplesmente servir de enfeites para eventos. Mísseis teleguiados são utilizados por aeronaves bélicas em campanhas militares e, em sua grande maioria, utilizam a combustão térmica para gerar impulso. A aerodinâmica é utilizada para a sustentação e manipulação de direção. Não possuem cabine e não transportam pilotos. Mísseis transportam cargas explosivas ou incendiárias e são destruídos após a colisão com seus alvos. Um caça F-16[2], assim como os mísseis teleguiados, utiliza a combustão para gerar força de impulsão e aerodinâmica para manter sua sustentação e realizar manobras aéreas. Possui capacidade de carga de dezenas de mísseis/acessórios, e necessita de piloto em cabine, podendo ser utilizado em mais de uma missão. Um bimotor possui dois motores que alimentam suas duas hélices

[2]Monomotor militar desenvolvido pela General Dynamics para a Força Aérea dos Estados Unidos.

no intuito de gerar impulso. A combustão existente é apenas a do combustível para alimentação dos motores e não está ligada diretamente à sustentabilidade ou fornecimento de força de impulsão ao veículo aéreo, logo as únicas forças que mantém um bimotor em ar são as forças aerodinâmicas. O bimotor possui capacidade de carga, reutilidade e necessidade de piloto em cabine. Um dirigível possui as mesmas características em relação ao bimotor, com exceção ao modo como se sustenta no ar, que é através do fenômeno da convecção, onde gases menos densos tendem a subir. A diferença de densidade de gases é obtida através do uso hélio (He) ou hidrogênio (H), ou então por meio de aquecimento. Foguetes V-2[3] são muito similares aos mísseis teleguiados, em todas as características. Veículos aéreos não-tripulados obtém sua sustentabilidade através das forças aerodinâmicas (similar ao bimotor), podem carregar equipamentos (câmeras, explositos, radares, etc), podem ser reutilizados e não possuem piloto em cabine. Com mais formalidade, pode-se observar que existem certas características que separam VANTs de demais tipos de veículos aéreos. Balões são considerados pela história como VANTs, mas é interessante notar que o principal recurso utilizado por eles nas manobras aéreas é a convecção e a diferença de densidades de ar, o que não os classifica necessariamente como VANTs segundo as definições técnicas modernas. Na Figura 3.2 os diferentes tipos de aeroveículos do quadro 3.1.

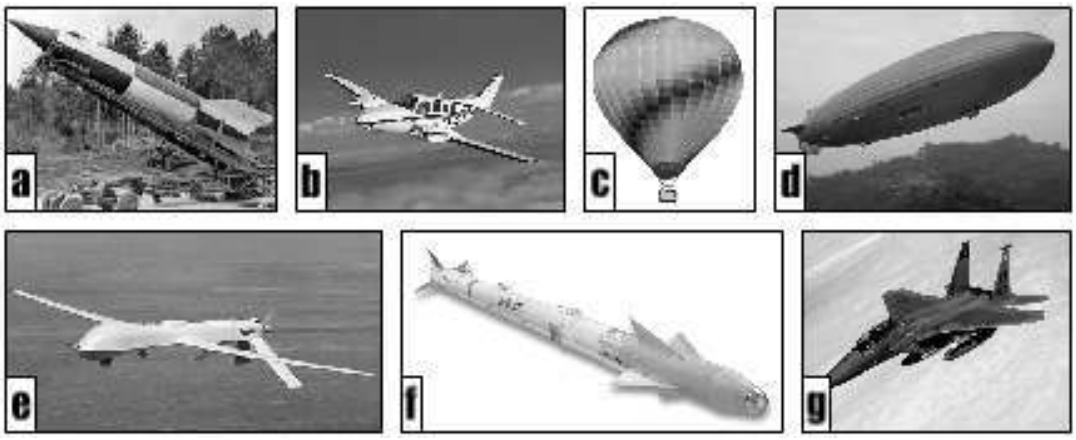

Figura 3.2: Veículos Aéreos. a) Foguete V2; b) Bimotor; c) Balão; d) Dirigível; e) VANT Predator; f) Míssil Sidewinder; g) Caça F-16.

3.1.2 Tipos de VANTs

Além da separação das diferentes classes de veículos aéreos, é possível a realização de uma observação mais próxima de cada um dos grupos e a classificação em diferentes tipos de classes. No contexto dos VANTs, a classificação é geralmente realizada de acordo com

[3] Arma balística alemã utilizada durante a segunda guerra mundial.

suas funcionalidades, nichos militares ou alcances/altitudes máximas. De acordo com as funcionalidades, VANTs podem ser classificados nos seguintes grupos:

Falso Alvo : Fornecem cobertura aérea e funcionam como alvos-iscas com o objetivo de enganar unidades hostis;

Reconhecimento : Fornecem dados geográficos, informações de campo ou da infraestrutura de instalações;

Monitoramento : Englobam o contexto de monitoramentos de áreas de difícil acesso ou ambientes hostis;

Combate : Fornecem apoio bélico às missões de alto-risco às vidas dos pilotos, seja como força de frente ou cobertura e apoio;

Transporte : Possuem a tarefa de transportar cargas em geral;

Resgate : Utilizados em missões de resgate tático, em lugares de difícil acesso;

Pesquisa e Desenvolvimento : Utilizados para o desenvovimento da tecnologia em VANTs através de testes de software e integrações com outras plataformas e sistemas;

Civil e Comercial : Especificados e concebidos com o objetivo de atender aos propósitos civis, comerciais ou de entretenimento, como é o caso de hobbie em aeromodelismo e apresentações para o marketing.

Além desta classificação, de acordo com a padronização militar norte-americana VANTs podem também ser enquadrados em três distintos grupos em função das forças militares que os utilizam (*US Air Force tiers*, *US Marine Corps tiers* ou *US Army tiers*) (DOD, 2005). Assim, outro possível tema de classificação é através dos alcances/altitudes máximos (i.e. Handheld[4] ou HALE[5]).

3.1.3 Software para VANTs

Diversas instituições realizam pesquisas nos mais variados temas. Estes temas podem ser separadas de acordo com suas finalidades e objetivos. Listam-se a seguir alguns resumos dos temas de pesquisas voltadas ao desenvolvimento com VANTs.

Planejamento de Caminhos : O planejamento de caminhos tem por principal objetivo a concepção de algoritmos que forneçam aos VANTs a capacidade de definir, calcular, e executar deslocamentos de forma inteligente e menos custosa em termos de tempo e processamento. Em uma situação real, um VANT poderá estar diante da necessidade de navegar por ambientes desconhecidos, com obstáculos multi-formes e

[4] 2.000 pés (600 m) de altitude, e aproximadamente 2 km de alcance
[5] Altas altitudes, por longa duração de vôo, acima de 30.000 pés (9.100 m) e alcance indefinido

desprovido de informações preliminares. A capacidade de examinar e construir seu próprio caminho e de percorrê-lo dentro de possíveis labirintos é uma necessidade crucial na autonomia de um VANT;

Interações em Larga Escala : Muitas tarefas são complexas ao ponto de necessitarem de mais de um VANT. Estas tarefas podem vir a ser mais complexas ainda, de modo que dezenas ou centenas de protótipos venham a ser necessários. Em larga escala, muitos problemas podem vir a ocorrer como o incompatível tempo de resposta de comunicação inter-VANT, paralelismos de comunicação e perdas de sinais. Soluções algorítmicas e estruturais vêm sendo propostas frequentemente na tentativa de solucionar os mais diversos tipos de situações deste contexto;

Navegabilidade & Telemetria : A navegabilidade é um tema de estudo que ocupa-se com as funcionalidades básicas de geo-localização e referenciamento, importantes para que o veículo aéreo possa situar-se geograficamente. Triangulação de pontos, referenciamento por GPS, infraestruturas de redes sem-frio são alguns dos tópicos publicados;

Engenharia e Construção : A construção de um protótipo de VANT não é trivial e aborda conceitos de engenharia mecânica, eletrônica e aeronáutica. O tema da engenharia e construção ocupa-se das questões de compatibilidade de hardware, pesos de equipamentos, velocidades, acelerações, e outros aspectos de baixo nível abstrativo;

Detecção e Prevenção de Colisões : Neste tema são estudados e desenvolvidos os algoritmos que possibilitam aos VANTs a realização de movimentos táticos, no intuito de desviarem-se de objetos fixos e móveis. Existem quatro tipos possíveis de obstáculos: fixos conhecidos geograficamente, fixos desconhecidos geograficamente, móveis conhecidos geograficamente e móveis desconhecidos geograficamente. Para cada tipo de obstáculo são necessárias diferentes abordagens;

Concepção e Pesquisa de Modelos : Para a construção de protótipos é necessária a pré-modelagem e concepção do VANT a ser criado, levando em consideração o ambiente em que o protótipo será utilizado e as funcionalidades requeridas. Este tema estuda aspectos das ciências aeronáuticas e modela o número de hélices que deve ser utilizado, altitudes/alcances máximos, pesos e capacidades de cargas, entre outros.

3.2 O Estado da Arte das Pesquisas em VANTs

Nestes últimos anos pôde-se ver um grande crescimento da utilização de VANTs, principalmente no campo acadêmico e civil. No Brasil são poucas as instituições que

possuem infraestrutura para realizar testes e estudos avançados, sendo o estudo da viabilidade de utilização bastante emergente nestes momentos atuais. Internacionalmente, as aplicações e testes são mais amplos, com abordagens mais sofisticadas e estudos de algoritmos avançados.

3.2.1 VANTs na Atualidade Nacional

No Brasil, algumas poucas instituições acadêmicas e laboratórios de tecnologias desenvolvem e estudam VANTs, tanto na construção de protótipos e modelos como na elaboração e concepção de algoritmos e software. Pode-se dizer que tais instituições cunham o estado-da-arte na área de VANTs no âmbito nacional, com publicações recentes em conferências internacionais ou com projetos dentro do território nacional.

- Verlab/UFMG – No estado de Minas Gerais, Brasil, o laboratório de Visão e Robótica da Universidade Federal de Minas Gerais (UFMG)[6] trabalha principalmente com a elaboração de algoritmos para a autonomia de VANTs e com ferramentas de vôo. No campo dos algoritmos e técnicas, os principais trabalhos estão relacionados com a geração e planejamento de caminhos (NETO; CAMPOS, 2009; NETO; MACHARET; CAMPOS, 2009) através de conceitos matemáticos (hodógrafos pitagóricos) e computacionais. No campo do desenvolvimento de ferramentas de vôo, alguns dos trabalhos são um sistema para testes de estratégias de controle (NETO; CAMPOS, 2008) e um horizonte artificial para *feedback* de vôo (ISCOLD et al., 2007). Na Figura 3.3 a ferramenta de horizonte virtual desenvolvida pelo laboratório Verlab.

Figura 3.3: Horizonte virtual: ferramenta desenvolvida pela UFMG/Verlab. Fonte: UFMG/Verlab.

- IME – No estado do Rio de Janeiro, o Instituto Militar de Engenharia realizou

[6]Webpage em `http://www.ufmg.br/`, acessada em 28/12/2011

recentemente a utilização[7] de um VANT para uma missão de mapeamento e apoio às operações de busca e salvamento nas regiões afetadas pelas chuvas na região serrana do Rio de Janeiro.

- ITA – O Instituto Tecnológico da Aeronáutica possui grande histórico na área de desenvolvimento de protótipos de aeronaves, tanto tripuladas quanto não-tripuladas, além de fornecer cursos nas áreas de engenharia e ciências aeronáuticas. Uma das recentes pesquisas está relacionada às simulações de dispositivos de piloto automático (RIBEIRO; OLIVEIRA, 2010).

- USP/Escola Politécnica – Recentemente, a USP vem direcionando algumas de suas pesquisas na área de VANTs, com levantamentos acerca das possibilidades de utilização das aeronaves e análises das questões e dos impactos na economia, segurança, treinamentos e custos (LONGHITANO, 2010). Ainda há o desenvolvimento de protótipos de VANTs sendo realizado por uma empresa incubada nas dependências da universidade, com aplicação nacional nos monitoramentos de desmatamentos[8] na região da hidroelétrica de Jirau, estado de Rondônia.

- GCAR/UFRGS – O Grupo de Controle Automação e Robótica[9] da Universidade Federal do Rio Grande do Sul possui diversas frentes de pesquisa em robótica e automação. O desenvolvimento de infraestrututra para veículos aéreos não-tripulados é um dos temas correntes.

3.2.2 VANTs na Atualidade Internacional

No âmbito internacional, a diversificação das linhas de pesquisa é ainda maior, com a aplicação de abordagens de sistemas multi-agentes, análises e simulações de algoritmos para *swarm-scale* (interação de grande número de VANTs), *frameworks* para pilotos automáticos, direcionamento à indústria bélica, entre outros.

- ETH/IDSC – O Instituto de Sistemas Dinâmicos e Controle da Universidade Federal de Zurique, possui *Testbed* para desenvolvimentos ou testes e realiza diversas pesquisas relacionadas ao âmbito dos VANTs e robótica. As principais linhas de pesquisa estão inseridas no tratamento de coalizões e gerência de frotas. Algumas das recentes publicações relatam a construção de um sistema que permite aos VANTs realizarem danças aéreas sincronizadas com músicas (SCHOELLIG; AUGUGLIARO; D'ANDREA, 2010). Outras pesquisas estão relacionadas com algoritmos para aprendizado em manobras aéreas (LUPASHIN et al., 2010), com

[7] Maiores detalhes em: `http://www.faperj.br/boletim_interna.phtml?obj_id=6952`, de 19/01/2011.

[8] Maiores informações em: `http://www.aereo.jor.br/2011/07/15/vant-inovador-feito-na-usp-monitora-desmatamento-em-jirau/`, de 15/07/2011.

[9] Página WWW em: `http://www.ece.ufrgs.br/`.

planejamento de caminhos (JUN; D'ANDREA, 2002) em ambientes desconhecidos e com a concepção de vôos de cadeias aéreas distribuídas[10]. A Figura 3.4 mostra o *Testbed* do laboratório da *Institute for Dynamic Systems and Control*.

Figura 3.4: *Testbed* do Instituto de Sistemas Dinâmicos e Controle da Universidade Federal de Zurique. Fonte: ETH/IDSC.

- ATG/Gerstner Lab – Localizado na República Tcheca, o laboratório Gerstner possui diversos grupos de pesquisa, dentre os quais o *Agent Technology Group* efetua trabalhos especialmente na área de aplicações de SMA. Alguns dos projetos vigentes tratam as questões do tráfego aéreo civil de VANTs em larga escala, sob uma perspectiva orientada à SMA de maneira descentralizada e com utilização de algoritmos para evitar colisões entre os veículos aéreos. Recentes publicações relatam estudos nos quais foram simulados planejamento de caminhos e monitoramentos por VANTs em um cenário urbano com diversos obstáculos multiformes (JAKOB et al., 2010). Outro trabalho relata a utilização de SMA para o cálculo e predição de rotas e análises de riscos em viagens marítimas (VANEK et al., 2011). Outros temas são o projeto *AgentScout*[11], financiado pelo exército norte americano, na área de cobertura aérea e coordenação tática em ambientes militarizados e *AgentDrive*, que aborda o tema de agentes para gerenciamento de tráfego (VOKRíNEK; KOMENDA; PECHOUCEK, 2010) e algoritmos para gerenciamento de frotas (PECHOUCEK et al., 2006; PECHOUCEK; SIALAK, 2009). A Figura 3.5 mostra uma das screenshot do projeto *AgentFly* (SISLAK et al., 2008), uma plataforma de desenvolvimento e simulações de VANTs.

- Grasp Lab/Penn Engineering – Fundando em 1979, o laboratório GRASP (General Robotics, Automation, Sensing and Perception) na University of Pennsylvania é um

[10]Uma cadeia aérea distribuída consiste de VANTs que se conectam e realizam manobras aéreas em conjunto. Página do projeto em `http://www.idsc.ethz.ch/Research_DAndrea/DFA`

[11]Página web em `http://agents.felk.cvut.cz/projects/agentscout/`, acessada em 29/11/2011.

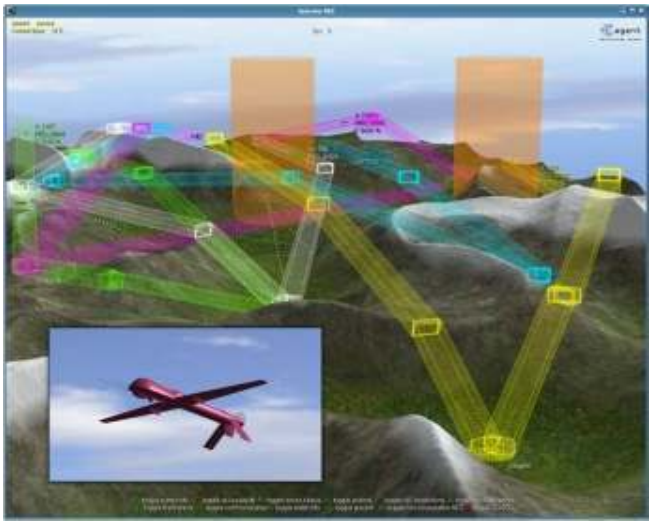

Figura 3.5: Simulação de VANTs com a plataforma AgentFly. Fonte: Gerstner Lab.

dos principais centros de pesquisa com ênfase multidisciplinar em robótica. Especificamente em VANTs, os principais trabalhos abordam manobras aéreas de alta precisão através de câmeras de alta frequência de captura instaladas em um *Testbed* (MICHAEL et al., 2010; AHMADZADEH et al., 2006; CHENG; KELLER; KUMAR, 2008). Os movimentos dos VANTs são realizados através de *feedbacks* gerados das coordenadas geo-espaciais obtidas nas câmeras e transmitidas ao sistema. Neste sistema, vôos com múltiplos VANTs podem ser realizados e manobras em alta velocidade são testadas. A Figura 3.6 mostra os testes realizados com algoritmos de manobras aéreas através de uma janela artificial.

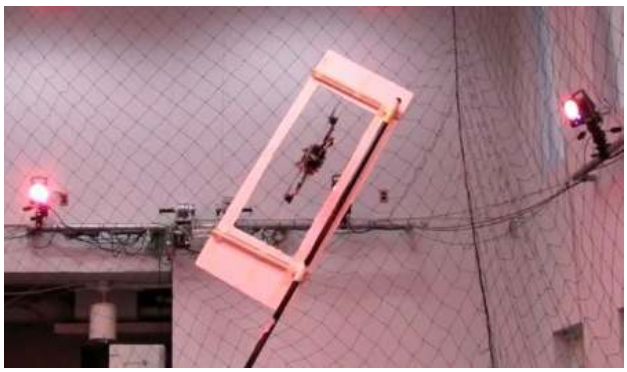

Figura 3.6: Testes com algoritmos de manobras aéreas através de uma janela. Fonte: GRASP Lab.

- Outros Laboratórios – Existem muitos outros laboratórios e instituições acadêmicas que realizam pesquisas com VANTs no mundo, confirmando a assertiva de que VANTs vêm se tornando cada vez mais presentes na tecnologia dos países e no con-

texto internacional. Os Estados Unidos investem fortemente nesta área de pesquisa e é atualmente o país que detém o maior orçamento para pesquisas relacionadas.

Algumas instituições acadêmicas de vários países que podem ser citadas são a *North Dakota State University* (DARGAR et al., 2002; HUFF; KAMEL; NYGARD, 2003), a *University of Central Florida* (DEJONG, 2005), a *Arizona State University* (MARSH, 2007), a *South Dakota School of Mines and Technology* (OURSLAND, 2010), e a *Carnegie Mellon University* (SCERRI et al., 2008; GLINTON; SCERRI; SYCARA, 2011). A *University of Freiburg* (GRZONKA; GRISETTI; BURGARD, 2009) e a *Humboldt-Universitat* (HAFNER et al., 2010), na Alemanha, trabalham especialmente com o contexto da engenharia de protótipos e algoritmos para navegabilidade de VANTs. A *University of Melbourne* (KARIM; HEINZE, 2005) na Austrália, a *St.Petersburg Institute for Informatics and Automation of RAS* (GORODETSKY et al., 2007) na Rússia e a *University of Glasgow* (KIM; CRASSIDIS, 2010) na Escócia são outras instituições acadêmicas com dezenas de publicações em jornais e revistas da IEEE e/ou ACM. A Figura 3.7 mostra um mapa global das instituições acadêmicas que pesquisam VANTs e que foram observadas neste trabalho.

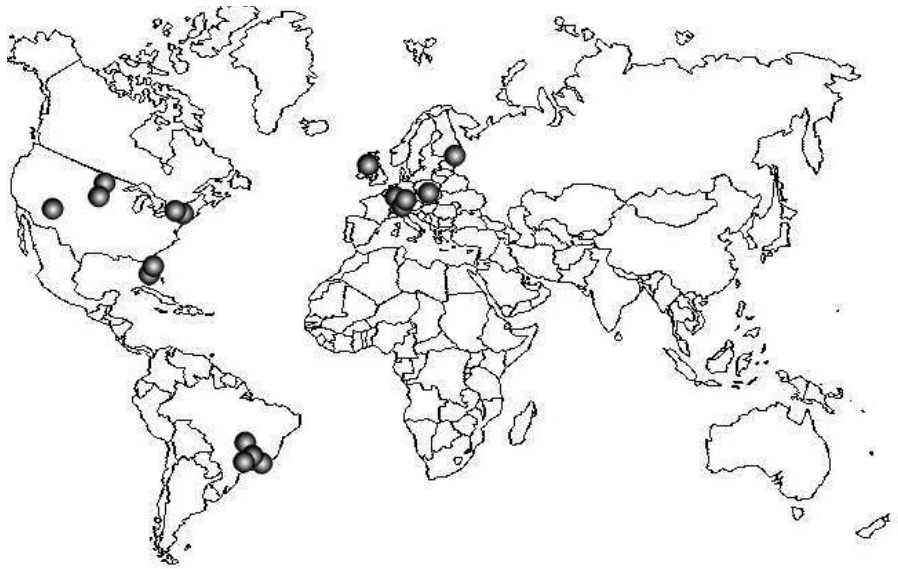

Figura 3.7: Visão geral de algumas das instituições acadêmicas que trabalham com VANTs.

3.3 Desafios em Agentes Aplicados a VANTs

Os modelos de agentes e SMA aplicam-se com naturalidade à modelagem de VANTs devido às principais características de sistema inteligente distribuído que uma frota de

VANTs possa vir a ter. Pode-se ainda realizar uma modelagem na qual um VANT é controlado por vários agentes inteligentes de modo que cada um destes agentes esteja realizando uma tarefa específica. Assim como mostram os trabalhos descritos anteriormente, as idéias não são novas e abordam usos, simulações e testes com aplicações de agentes em VANTs. Em relação à aplicação de agentes em VANTs, alguns dos desafios que foram observados nestes primeiros momentos herdam muitas de suas causas da pouca idade das tecnologias e da complexidade das infraestruturas de hardwares e de softwares.

Situações Adversas e Imprevistas – Em determinadas situações, agentes que pilotam VANTs poderão vir a estar diante de problemas dos quais não poderão comportar-se de maneira inteligente ou não poderão agir de forma consistente. Nestes tipos de situação espera-se que, ao menos, o VANT não venha a ser derrubado e que o sistema mantenha-se operante. Tratar todos os possíveis casos é virtualmente impossível e o desenvolvimento de uma autonomia com um mínimo de consistência é sempre uma tarefa de extrema complexidade;

Interatividade e Cooperatividade – O desenvolvimento de agentes que pilotam VANTs deve considerar ainda a abordagem de missões onde a utilização de mais de uma entidade é necessária como, por exemplo, em missões de aerofotografia, em que diversas fotografias devem ser tiradas de um objeto ou ambiente em um mesmo instante, sendo a consideração estratégica muito mais importante do que a tática. Este desafio herda os desafios dos estudos de interação em larga escala e não é um tópico trivial;

Suporte e Escalabilidade – Outro grande desafio é prover suporte e infraestruturas extensíveis para futuras implementações dado que a tecnologia dos VANTs sofre avanços a cada dia. Estar munido com a capacidade de gerenciar múltiplos VANTs com diferentes habilidades e características de uma maneira flexível e escalável é um dos desafios da aplicação de agentes e SMA em VANTs;

Confiança & Comunicação – Um outro grande desafio vem a ser o tempo de resposta dos agentes, dada a importância de VANTs em exercerem, precisamente, suas ações. Um VANT não pode dar-se à liberdade de cometer falhas grosseiras em missões, pois pode colocar em risco um hardware de alto custo, a soberania de um país e até mesmo vidas. Assim, uma infraestrutura confiável é necessária para diversos tipos e números de VANTs, tratando possíveis perdas de sinais, duplicatas ou reenvios de mensagens, gerenciamento e confirmações de mensagens recebidas, paralelismos, entre outros;

Abstratividade de Implementação – A tecnologia dos VANTs é bastante nova ainda e muitas plataformas e infraestruturas utilizam linguagens não convenientes para

a programação e desenvolvimento de autonomia, característica dos agentes e dos SMA. Um VANT possui rotinas de alta abstratividade (deslocar-se, patrulhar, enviar mensagens, seguir alvos, entre outros) e as linguagens de programação geralmente utilizadas nos *firmwares* dos protótipos de VANTs são de baixo nível (i.e. **C**, **C++** ou **Assembly**). A diminuição deste *gap* abstrativo e a construção de modelos é outro desafio da engenharia de software para VANTs.

3.4 Proposta de Modelo

Podem existir muitos outros desafios tecnológicos de VANTs dos quais alguns podem até mesmo ser desconhecidos nestes primeiros momentos, vindo a ser descobertos posteriormente. Em contrapartida, o fato de que VANTs vêm se tornando importantes é inegável. Pesquisas realizadas pela *Market Intel Group*[12], empresa de consultoria e levantamento de dados de mercado e indústria apontam que dentro de três anos a aquisição de VANTs venha a aumentar drasticamente no mundo todo, como mostra a Figura 3.8, o que torna o assunto da tecnologia dos VANTs de grande importância não só nacionalmente mas também globalmente. Ainda, o conceito de agentes autônomos e sistemas multi-agentes vêm provando ser uma abordagem interessante para lidar com muitos tipos de situações no universo dos VANTs.

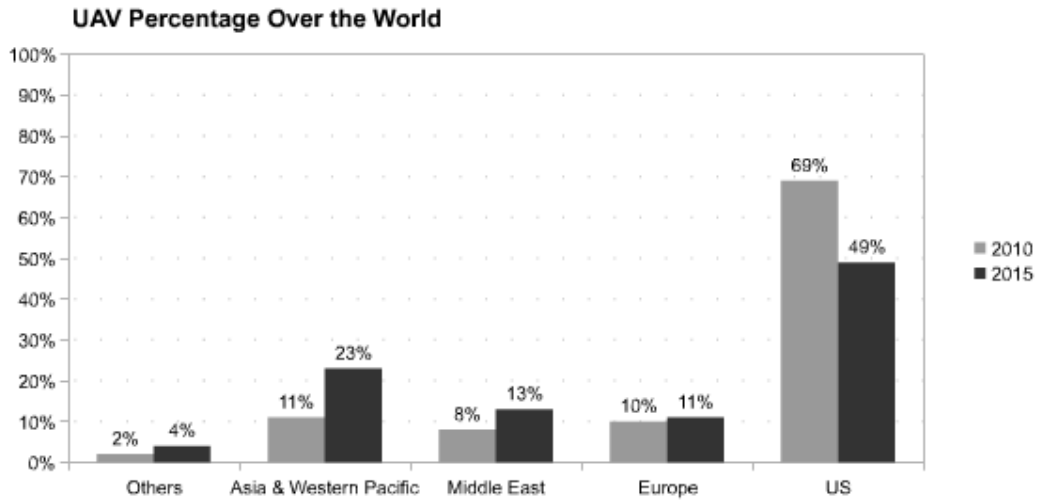

Figura 3.8: Percentual de aquisições de VANTs no contexto mundial. Fonte: Adaptação de *Market Intel Group*.

Tomando este cenário por base, propõe-se um modelo que contemple uma infraestrutura orientada a agentes para o estudo, simulação e aplicação de agentes a VANTs, de

[12]Página web em `http://marketintelgroup.com/`, acessada em 29/12/2011.

modo que venha a ser possível o emprego de linguagens de programação orientadas a agentes e SMA para o desenvolvimento de missões, atacando um dos desafios da engenharia de software para VANTs: a abstratividade de desenvolvimento. O próximo capítulo abordará com detalhes o modelo UAVAS (HAMA et al., 2011), que tem como principal objetivo a proposta da diminuição do *gap* abstrativo na programação e desenvolvimento de autonomia para VANTs.

4 O MODELO UAVAS

A Engenharia de Software é uma área do conhecimento da computação que objetiva o estudo e a aplicação de abordagens sistemáticas, disciplinadas, quantificadas e bem definidas, nas práticas do desenvolvimento, operação e manutenção dos softwares com o intuito de obter maiores qualidades na organização e na produtividade (ABRAN; MOORE, 2004). Bons modelos fornecem uma visualização mais expressiva do sistema como um todo e fornecem facilidades extras em extensões posteriores, tornando o sistema manutenível e escalável. A área de agentes e SMA possui uma engenharia de software particular, que pesquisa modelos e metodologias para a concepção de agentes autônomos e que pode ser interessantemente aplicada ao desenvolvimento em VANTs.

O desenvolvimento de VANTs com a utilização da abordagem de agentes e SMA é relativamente recente, onde existem alguns poucos trabalhos que datam de menos de uma década em relação a esta pesquisa. Neste escopo, alguns trabalhos relacionados ou similares que merecem menção são: (DARGAR et al., 2002; HUFF; KAMEL; NYGARD, 2003), que propõem abordagens para concepções de modelos e coalizões de VANTs através de plataformas orientadas a agentes com serviços de mensagens inter-agentes; (KARIM; HEINZE, 2005), que realiza relatos técnicos da aplicação de agentes para a construção de um controlador de vôo e inserção de autonomia aos VANTs, com dois tipos diferentes de complexidade para as modelagens; (MARSH, 2007), que é uma abordagem orientada à Model Driven Development (MDD) para a definição de comportamentos de VANTs; (SISLAK et al., 2008), que é uma plataforma implementada sob o paradigma de SMA para simulações de VANTs em larga escala; (WALLIS et al., 2002), que implementa um modelo orientado à agentes inteligentes para programação de VANTs através da utilização de JACK (BUSETTA et al., 1999); e (PINHEIRO, 2006), que implementa um simulador de frotas de VANTs, com serviços de comunicação inter-agentes.

Além de serem poucas as abordagens existentes, nenhum destes trabalhos mencionados trata a união de VANTs com SMA com enfoque na POA. Verificou-se também que na atual literatura não há[1] uma definição para a Engenharia de Software aplicada a VANTs que tenha como principal ênfase a implantação da inteligência e autonomia através de

[1] Segundo pesquisas no estado da arte realizadas no estudo descrito por esta dissertação.

abordagens em SMA ou a aplicação de linguagens orientadas a agentes. Assim sendo, para fins de referência neste documento, propõem-se a Definição 1, para a engenharia de software que mescla as técnicas da programação orientada a agentes e o desenvolvimento de VANTs.

Definição 1 *Engenharia de Sistema Multi-Agentes Aero-Veiculares: engenharia de software que tem por principal objetivo a aplicação dos conceitos de agentes e sistemas-multi-agentes no desenvolvimento de autonomia nos veículos aéreos não-tripulados.*

A principal ênfase da engenharia de sistema multi-agentes aero-veicular é a aplicação dos conceitos da engenharia de software aplicados a veículos aéreos não-tripulados, sob uma perspectiva orientada a agentes. O elemento em foco é o **sistema multi-agentes aero-veicular**, formalizado segundo a definição 2.

Definição 2 *Sistema Multi-Agente Aero-Veicular: sistema multi-agente que tem por objetivo principal prover autonomia a um veículo aéreo não-tripulado, seja de forma parcial ou completa.*

Ressalva-se que existem pesquisas que realizam implementações e experimentos com o conceito de agentes e SMA aplicados a VANTs. As definições realizadas não estabelecem novas idéias acerca do universo dos VANTs, mas sim tentam estabelecer uma formalidade para o ato de modelar agentes e SMA cujas funcionalidades sejam intrínsecas à criação de autonomia nos VANTs. Através das duas definições anteriores, pode-se propor uma terceira, que formaliza um conceito de Engenharia de Comportamentos para VANTs, descrita em definição 3.

Definição 3 *Engenharia de Comportamentos para Veículos Aéreos Não-Tripulados: Processo de desenvolver a autonomia dos veículos aéreos não-tripulados através de programação orientada a agentes e sistemas multi-agentes.*

4.1 Expressividade na Engenharia de Comportamentos

A expressividade de desenvolvimento é um ponto interessante na criação de sistemas, ainda mais quando estes possuem alta complexidade de infraestrutura. Geralmente, quando o VANT possui autonomia, esta é pré-programada dentro de seu *firmware* e o comportamento "inteligente" em si é complexo de ser implementado. Grande parte desta complexidade vem como consequência da baixa abstratividade das linguagens de programação utilizadas (e.g. **C** ou **Assembly**) para implementar operações de alto nível, como a gestão de manobras de vôo, sistema de inter-comunicação ou operações de geolocalização. Em diversas pesquisas de VANTs, ao se tentar desenvolver a autonomia em protótipos, habitualmente podemos nos encontrar com arquiteturas complexas e de difícil

entendimento devido ao fato de que o desenvolvimento da autonomia não deve ser uma tarefa ocupada com questões de *Input*/*Output* de *hardware*, conversões de bytes, ponteiros, interrupções, entre outros. As vezes, este *gap* tecnológico é grande o suficiente para implicar em um aumento considerável de complexidade do sistema como um todo.

4.1.1 O *Gap* Abstrativo

Como mencionado anteriormente, uma das características do desenvolvimento de autonomia para VANTs é o peculiar uso de linguagens de programação de baixo nível (próximos da linguagem de máquina), o que torna bastante difícil a construção de comportamentos complexos a VANTs. Ilustra-se aqui um caso real do *gap* abstrativo. O excerto em Listing 4.1 é parte da implementação, em linguagem **C**, do *firmware* dos protótipos de VANTs da *MikroKopter*, um projeto de Veícuo Aéreo Não-Tripulado conduzido por engenheiros e pesquisadores alemães.

Listing 4.1: Método para adicionar um ponto geo-espacial.

```
u8 WPList_Append(Waypoint_t* pwp) {
   if (WPNumber < WPLISTLEN) {
      memcpy(&WPList[WPNumber], pwp, sizeof(Waypoint_t));
      WPNumber++;
      NaviData.WaypointNumber = WPNumber;
      return TRUE;
   }
   else return FALSE;
}
```

Sob o ponto de vista técnico, o conjunto de instruções realiza a inserção de um novo ponto geo-espacial como alvo a ser alcançado pelo VANT. O VANT possui, definido em seu *firmware*, uma lista de pontos geo-espaciais, os quais deverão ser ordenadamente alcançados para que a missão ou tarefa seja concluída. Pode-se ainda adotar uma abordagem de patrulhamento, onde os pontos geo-espaciais da lista deverão ser visitados ciclicamente (ao chegar no final da lista, reinicia-se novamente pelo primeiro ponto geo-espacial).

Sob o ponto de vista semântico observa-se que tanto a linguagem adotada quanto as instruções escritas não possuem uma abstratividade compatível sendo difícil a simultânea especialização das técnicas, dos conhececimentos específicos da aplicação, da arquitetura, da linguagem de programação e do desenvolvimento de autonomia. Reutilização da implementação, robustez de arquitetura e expressividade tornam-se extremamente comprometidas nestes tipos de situações.

4.2 Bases Teóricas

A solução explorada nesta pesquisa para o mencionado *gap* tecnológico é a abordagem de agentes e SMA. Assim como foi mencionado anteriormente, o desenvolvimento de VANTs possui peculiaridades como a complexidade da arquitetura, as questões do teste

de algoritmos, experimentos e computações distribuídas. Em vários casos, é muito perceptível a importância de *frameworks* especializados e ambientes de simulação. Agentes e SMA são abordagens consolidadas para modelar sistemas computacionais inteligentes e distribuídos, e podem ser vistos como uma interessante e conveniente solução para modelar sistemas de VANTs dadas as características em comum. Duas destas características mostram-se mais perceptíveis:

- **Modelo de Comunicação**: VANTs devem implementar capacidades de comunicação para enviar ou receber informações para outros sistemas ou VANTs. Analogamente, os agentes devem se comunicar uns com os outros, ou com servidores e serviços. Em termos de comunicação inter-sistema e intra-sistema, tanto VANTs quanto agentes devem usar um protocolo para se comunicar e trocar dados, sendo muito semelhantes entre si neste aspecto;

- **Sistemas Distribuídos e Concorrentes**: agentes e VANTs possuem em comum as propriedades de um sistema concorrente e distribuído. Podemos trabalhar com uma equipe de VANTs ou uma sociedade de agentes onde cada sistema possui as inteligências particulares de cada uma das entidades e a inteligência global fruto do conjunto das inteligências particulares. Além disso, internamente, um agente pode ter vários objetivos simultâneos (também chamado de *Desejos* no contexto BDI), e um VANT pode ter várias tarefas a realizar e monitoramentos de estados a processar, podendo vir a serem computados e processados simultaneamente. Tendo em vista estas peculiaridades, a modelagem de um VANT com abordagens de agente mostra-se muito plausível.

A principal base teórica utilizada neste trabalho é a abordagem BDI (BRATMAN, 1981), um conhecido modelo no desenvolvimento de agentes Cognitivos com noções abstratas de sistemas inteligentes. Na literatura acadêmica, o agente cognitivo é uma entidade autônoma e deliberativa de software capaz de assumir planos racionais a fim de realizar seus objetivos por meio de regras lógicas sendo assim capaz de resolver tarefas complexas. O tipo de agente cognitivo melhor se adapta aos designs do modelo proposto devido ao nível de autonomia do software, do bom conjunto de ferramentas e linguagens de programação, da ampla documentação e do alto número de trabalhos que utilizam esta abordagem.

4.3 O Modelo UAVAS

Os conceitos de agentes e SMA tornam-se interessantes assim como as linguagens de programação orientadas a agentes. Nesta linha, propõem-se o modelo *UAVAS*, onde a visualização de um SMA aero-veicular é vista de forma análoga a uma aeronave tripulada por pilotos, co-pilotos e comissários, dotados de certa inteligência e capacidade de agir em

torno de determinadas circunstâncias. O modelo utiliza agentes como entidades capazes de realizar funções similares às de um piloto automático oferecendo um conjunto de ações pré-definidas em linguagens de maior expressividade e de maior nível abstrativo. Dessa forma um usuário final que tenha apenas o objetivo de descrever missões e criar autonomia para VANTs poderia descrever planos de vôo, instruções de missões ou comportamentos complexos inerentes ao VANT através de uma linguagem de alto nível abstrativo. Nesta analogia as seguintes modelagens são criadas:

Modelo *UAVAS*		**Veículo Aéreo Tripulado**
Agentes *UAVAS*	abstrai	Tripulação do Veículo Aéreo Tripulado
Ambiente Virtual	abstrai	Cabine de Comando
Arquivos AgentSpeak .ASL	abstrai	Inteligência da Tripulação e Planos de Vôo
Protocolo de Ações	abstrai	Comandos do Veículo Aéreo Tripulado
VANT	abstrai	Veículo Aéreo Tripulado

Tabela 4.1: Analogias do modelo *UAVAS*.

Os Agentes *UAVAS* herdam a arquitetura dos agentes *Jason* (agentes definidos pela arquitetura da plataforma *Jason*) e são providos de uma API para poderem receber as percepções, executar as ações, enviar mensagens e deliberar/raciocinar sobre planos no contexto do VANT. Estes agentes estão dentro de um SMA de *Jason* (uma instância de sistema multi-agente em *Jason*, que possui runtime interna e ciclos deliberativos de agentes) e podem interagir uns com os outros, a fim de transmitirem dados relevantes, como podemos ver em um veículo aéreo tripulado onde pilotos, co-pilotos e comissários interagem entre si. Em *Jason* os comportamentos inteligentes são desenvolvidos através da linguagem *AgentSpeak* em arquivos *.ASL*. Esses comportamentos são a modelagem dos planos de vôo e da inteligência da tripulação do veículo aéreo tripulado. No veículo aéreo tripulado um conjunto de controladores e comandos (ativadores de ailerons, profundores, lemes, etc) está disponível para manipular o veículo aéreo, sendo estes modelados por um protocolo de ações que estabelece uma comunicação direta com a API do *firmware* enquanto que os ambientes virtuais dos agentes podem ser vistos como a cabine de comando. Em outras palavras, o painel de controle da aeronave pilotada é modelado no *UAVAS* através do protocolo de ações. Estas ações são mapeamentos de blocos semânticos pré-programados em linguagens de baixo nível para métodos e rotinas intrinsecas às operações executadas por VANTs. Desta forma os métodos escritos em *firmwares* e as operações em VANTs escritas em linguagens de alto nível vêm a ser conectadas através de uma espécie de ponte tecnológica. O conceito final é abstrair o VANT com Agentes como um veículo aéreo com tripulação humana, onde as ações dos Agentes *UAVAS* são ações escritas em *Jason/AgentSpeak*, sendo o modelo concebido apto a assumir um papel genérico em termos de aplicação, que poderia ser tanto uma estação em terra quanto um piloto automático dentro do VANT.

4.3.1 Analogia

O modelo utiliza uma linguagem da POA para descrever tanto a inteligência dos agentes quanto o que cada um deles deverá executar durante as operações de vôo do VANT. O arquivo que contém o código desta linguagem pode ser comparado a uma espécie de plano de vôo de uma aeronave pilotada por tripulação humana. Na aeronave pilotada, a tripulação realiza manobras de vôo e operações gerais através do painel de controle, enquanto que no modelo proposto, este painel é modelado como um protocolo de ações que possui rotinas de relevância operacional (seguir alvos, ativar trem de pouso, patrulhar, entre outros) e de rotinas simples de vôo (controle de leme, elevador ou ailerons). Em resumo, o objetivo principal é construir uma ponte tecnológica entre *firmwares* e operações de alto nível de VANTs, usando agentes inteligentes e SMA para que se possa oferecer uma abordagem que venha a facilitar o desenvolvimento de aplicações aos usuários em geral através do encapsulamento de rotinas que não fazem parte do contexto de VANTs. A figura 4.1 mostra um paralelo entre o esquema de modelagem *UAVAS* e missões pilotadas.

4.3.2 A Arquitetura do Modelo

A arquitetura do modelo baseia-se em um esquema de fluxo e conversão de dados. No modelo, um agente poderá ter dois tipos de interação com o ambiente: agir sobre ele (ações) e percebê-lo (percepções). No âmbito das ações, um Agente executará um método do protocolo de ações através do contexto em que se encontrará suas crenças, desejos e planos (de acordo com a arquitetura BDI). Esta ação é convertida em sinais de IO e então são passados para a camada do sistema operacional vigente no *hardware* do VANT que através de métodos de *firmware* poderá realizar as ações executadas pelo Agente. No âmbito das percepções, dados e condições obtidas por rotinas definidas no *firmware* são passados através de portas IO da camada de sistema operacional para a camada do *framework*, onde então serão convertidos e traduzidos em percepções para a base de crenças dos agentes. A figura 4.2 mostra o modelo conceitual do fluxo de dados desta arquitetura.

O *framework* é executado dentro de um sistema operacional em um *hardware* embarcado no VANT. O VANT envia e recebe bytes de dados à plataforma *UAVAS* através de um receptor/transmissor assíncrono universal (***Universal Asynchronous Receiver/Transmitter*** – UART), que faz chamadas diretamente à API do *firmware* do VANT com chamadas de funções de baixo nível. Os bytes de dados são convertidos em fluxos de dados legíveis por um gerenciador de Input/Output (**IO Manager**), e processados por um conversor de Ações/Percepções (**Act/Percept Converter**) que usa um interfaceamento de um modelo de *firmware* (**Firmware Model**) para a construção de valores válidos a partir dos parâmetros e argumentos. Estes valores são enviados ou recebidos pelas ações geradas no **Protocolo de Ações**, onde tais ações são chamadas pela implementação da arquitetura dos Agentes (***UAVAS* Agent Architecture**). Os agentes são interpretados por um sis-

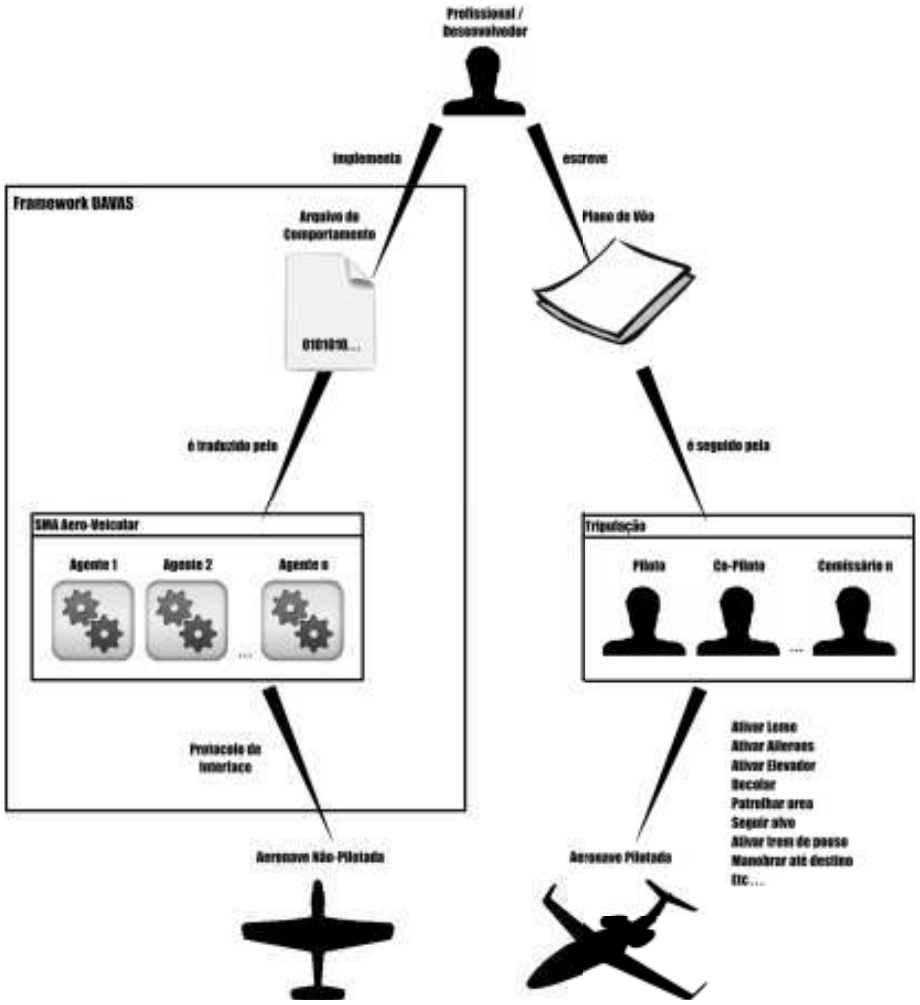

Figura 4.1: Paralelo entre o esquema de modelagem *UAVAS* e missões pilotadas.

tema multi-agente de *Jason* (***SMA Jason***) onde ciclos de raciocínio deliberativo ocorrem. Cada um destes Agentes têm seus comportamentos inteligentes descritos em arquivos .asl (***.ASL Files***).

4.4 Elementos do *Framework*

A arquitetura do modelo proposto engloba três elementos básicos em sua concepção: um protocolo de ações executadas pelos Agentes Aero-Veiculares que estabelece uma ponte entre o *firmware* e a plataforma, e que expõem as ações e operações de vôo; um *engine* de conversão de métodos para ações e sinais para percepções; e um modelo de interface que serve de mapeamento para a API do *firmware* do VANT. Cada um destes elementos é descrito com detalhes nas subseções seguintes.

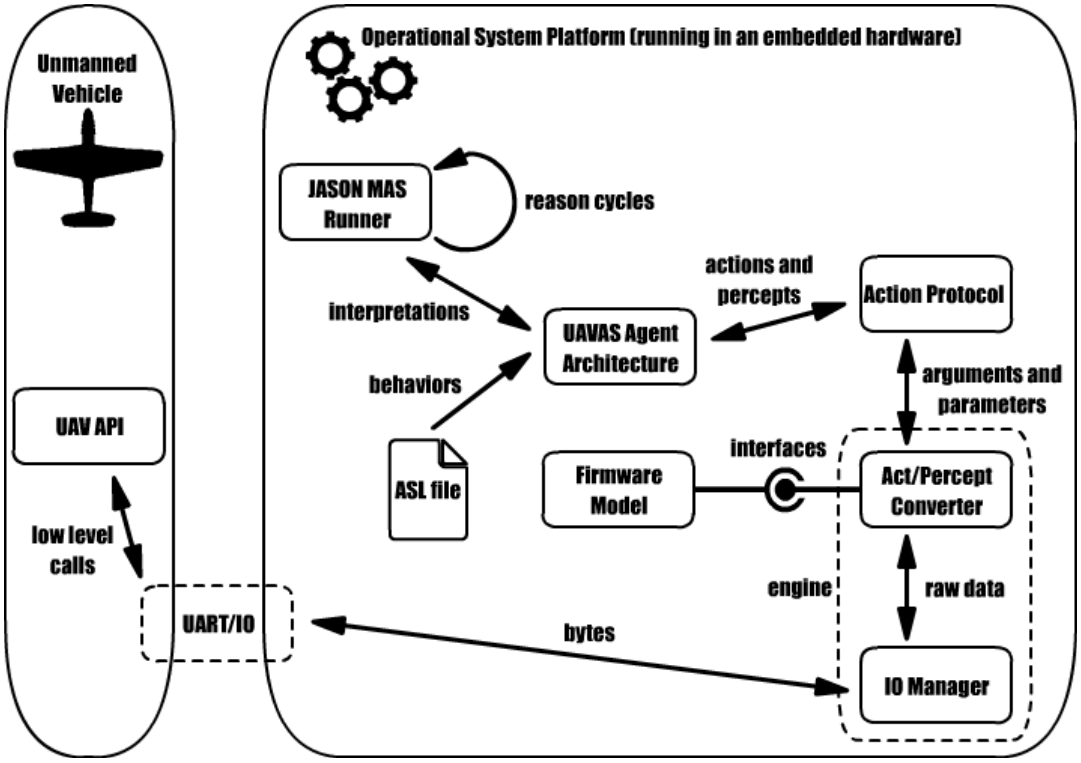

Figura 4.2: Modelo conceitual do framework UAVAS.

4.4.1 Protocolo de Ações

Em sua atual versão, o protocolo de ações define 16 operações que englobam ações de deslocamento, checagem de ambiente e sistema, comunicação inter-agentes e configurações. Estas ações não são apenas procedimentos chamados pela infra-estrutura *UAVAS* mas métodos dos quais os agentes VANTs podem fazer uso. Existem três grupos principais de ações: **locomoção**, **comunicação** e **percepção**. Os principais objetivos das ações de locomoção é o deslocamento físico do VANT através de dados geográficos obtidos via telemetria ou similares. As ações de comunicação possuem como objetivo a troca de mensagens entre agentes de modo que seja possível a realização de cooperação e trabalho em equipe entre os VANTs. As ações de percepção estão relacionadas ao sensoriamento, à consulta de ambiente e à checagem das condições do agente ou do VANT. O protocolo de ações define:

- `pitch`⟨`Value`⟩ – Quando um Agente executa esta ação, o VANT é solicitado a configurar o *nick*/*elevator* em função do argumento especificado **Value**;

- `roll`⟨`Value`⟩ – Quando um Agente executa esta ação, o VANT é solicitado a configurar o *inclination*/*aileron* em função do argumento especificado **Value**;

- **`yaw`**⟨**`Value`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a configurar o *tail/rudder* em função do argumento especificado **Value**;

- **`accelerator`**⟨**`Value`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a configurar o *collective/accelerator* em função do argumento especificado **Value**;

- **`moveTo`**⟨**`P`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a se deslocar para o dado ponto geográfico **P**, descartando quaisquer pontos geográficos passados anteriormente e cancelando modos de patrulha;

- **`doPatrol`**⟨**`PList`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a patrulhar ciclicamente os pontos geográficos na dada lista **Plist**;

- **`goHome`** – Quando um agente executa esta ação, o VANT é solicitado a se deslocar para sua posição inicial (definida no momento da inicialização do sistema);

- **`setHome`**⟨**`P`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a sobreescrever sua posição inicial, assumindo que o dado ponto geográfico **P** é uma atualização;

- **`addWayPoint`**⟨**`P`**⟩ – Internamente, um VANT possui uma lista *FIFO* de pontos geográficos a serem alcançados de forma ordenada. Quando um agente executa esta ação, o VANT é solicitado a adicionar o ponto geográfico **P** nesta lista interna;

- **`request`**⟨**`destinatary,logicExpression`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a realizar uma requisição a outro VANT, onde o objetivo é tornar verdadeira a expressão lógica **logicExpression** passada como parâmetro;

- **`inform`**⟨**`destinatary,logicExpression`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a enviar a expressão lógica **logicExpression** para outro VANT. O outro VANT assume esta expressão como verdadeira;

- **`ask`**⟨**`destinatary,logicExpression`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a enviar uma pergunta a outro VANT. O outro VANT é solicitado a enviar um *booleano* (true ou false) para o VANT que fez a pergunta;

- **`ack`**⟨**`destinatary`**⟩ – Quando um agente executa esta ação, o VANT é solicitado a enviar um sinal de confirmação para outro VANT. O outro VANT assume que sua última mensagem enviada ao primeiro VANT foi entregue sem quaisquer problemas;

- **`checkEnvironment`** – Quando um agente executa esta ação, o VANT é solicitado a verificar os estados do ambiente atual, como velocidade e direção do vento, pressão atmosférica ou umidade do ar;
- **`checkBattery`** – Quando um agente executa esta ação, o VANT é solicitado a verificar os seus níveis de combustível ou cargas de bateria;
- **`checkLocation`** – Quando um agente executa esta ação, o VANT é solicitado a verificar os seus valores atuais de latitude, longitude e altitude.

Utilizando *AgentSpeak* (introduzido na seção 2.4) como linguagem para executar estas ações, o protocolo de ações permite que cada uma destas ações possa ser executada por agentes *Jason/AgentSpeak*, de modo que a implementação dos comportamentos e instruções de missões venham a possuir o nível de abstração de programação de Agentes em arquiteturas BDI.

4.4.2 O Motor de Conversão

A principal tarefa do *engine* de conversão é realizar as traduções dos sinais recebidos/enviados entre a camada de agentes e a camada do sistema operacional e *firmware* através da implementação de métodos definidos na interface do protocolo de ações. O *engine* trabalha tanto em nível abstrativo alto quanto baixo pois realiza as tarefas de abertura de portas IO, traduz sinais enviados/recebidos, captura métodos chamados pelos agentes e envia dados aos agentes em forma de percepções. A figura 4.3 mostra os dois fluxos nos quais os dados são convertidos no *engine*.

Figura 4.3: Esquema do fluxo de dados no *engine*.

4.4.3 Interface de Mapeamento

No modelo proposto, a interface do *firmware* é independente da plataforma. A separação da interface de mapeamento permite que o modelo possa ser utilizado em diferentes configurações e modelos de *firmwares* de VANTs. A Figura 4.4 mostra uma abstração onde o modelo usa diferentes *firmwares* de interfaces. Através deste chaveamento é possível implementar o modelo UAVAS para outros tipos de VANTs com diferentes tipos de *firmwares*, desde que este disponibilize uma API aberta.

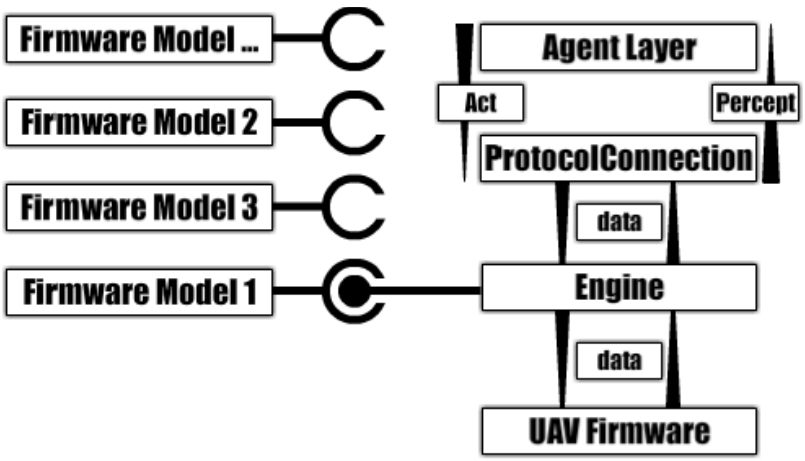

Figura 4.4: Abstração onde o modelo usa diferentes *firmwares*.

4.5 Modelo de Comunicação Inter-Agente

No modelo UAVAS, pela Figura 4.1, um SMA aero-veicular pode ser visto como um conjunto de agentes, que se comunicam com outros conjuntos de agentes, formando um complexo sistema que se comunica através de mensagens que podem ser tanto internas aos SMA aero-veiculares, quanto externas num contexto inter-VANT.

O modelo proposto é uma abstração de uma aeronave real, com múltiplos agentes modelando múltiplos tripulantes. A comunicação pode dar-se por dois níveis diferentes, interna ao SMA aero-veicular e inter SMA aero-veicular, da mesma forma com que tripulantes podem trocar informações e aeronaves podem enviar mensagens à outras aeronaves. A Figura 4.5 mostra como o modelo da comunicação é concebido no *UAVAS*, que vem a ser um tipo de organização de agentes. A seguir, no próximo capítulo, será abordada a implementação do modelo.

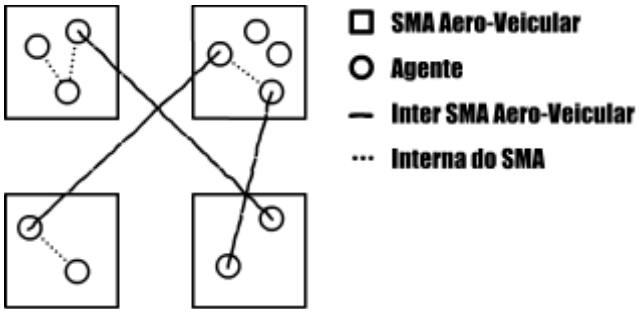

Figura 4.5: Abstração do modelo de comunicação do *UAVAS*.

5 PROTÓTIPO DO MODELO UAVAS

Neste capítulo é descrita a implementação de um protótipo inicial do modelo *UAVAS*, que foi feito através de duas camadas distintas, de agentes e de *Framework/Engine*, com uso de um modelo de interface de *firmware* dos VANTs da *MikroKopter* (introduzido na seção 4.1.1). Tanto o projeto *MikroKopter* quanto seu modelo de *firmware* serão explicados com maiores detalhes neste capítulo.

Na implementação do modelo *UAVAS*, a camada dos Agentes é formada pela arquitetura de agentes *Jason* (seção 2.4) sobrescritos pelas ações pré-definidas provenientes da IDE *Jason* e pelos arquivos de definição dos comportamentos escritos em *AgentSpeak*. A IDE *Jason* é conveniente pois além de ser uma plataforma *open-source* para projeto e implementação de agentes cognitivos com características BDI, também define uma interface que disponibiliza a implementação de métodos inerentes ao contexto de agentes, como agir, perceber, enviar mensagens, checar mensagens recebidas ou realizar *broadcast* de informações. Com esta interface e sobreescrevendo-se métodos da arquitetura de *Jason*, é possível a captura das ações dos agentes. Tais ações são provenientes dos ciclos deliberativos de raciocínios que foram interpretados das traduções dos arquivos de definição de comportamento. Os ambientes virtuais dos SMA Aero-Veiculares são definidos através de herança da interface *UAVAS* e expõem o protocolo introduzido na seção 4.4.1, que tem por objetivo a atualização das percepções e a execução de ações dos agentes. Os arquivos de definição de comportamento são escritos em linguagem *AgentSpeak* e possuem o paradigma orientado a SMA, sendo dessa forma mais inteligíveis e mais próximos do modelo de expressão do raciocínio humano. Na seção seguinte, algumas noções iniciais específicas à implementação sobre os modelos de orientação em ambiente geo-espacial, e detalhes do projeto *MikroKopter* serão apresentados.

5.1 Manobras de Veículos Aéreos

Ao contrário de veículos como carros que podem se locomover em um espaço 2D abrangendo quatro tipos diferentes de direções (oeste-leste e norte-sul), um veículo aéreo pode se locomover em um espaço 3D com seis diferentes tipos de direções (3 angulares

e 3 lineares). Na maioria do casos, no veículo aéreo o vetor velocidade em cada uma destas direções é construído a partir de uma rotação em torno de si mesmo e uma força aerodinâmica linear, que combinada a esta direção fornece o vetor desejado. O sistema adotado no *UAVAS* é o Pitch/Roll/Yaw (PRY) como podemos ver na Figura 5.1.

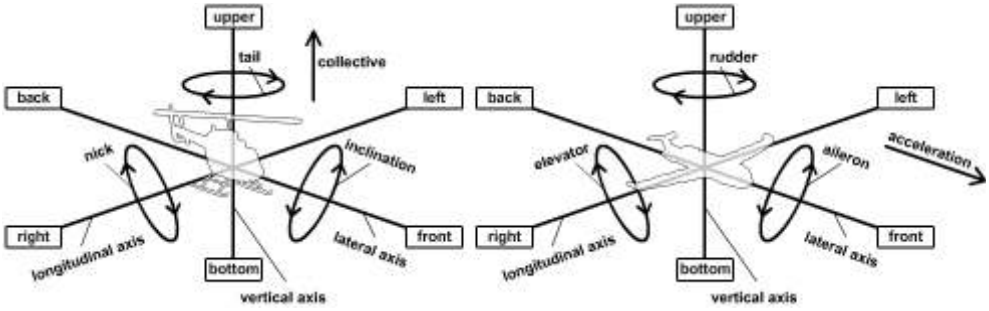

Figura 5.1: Sistema PRY, usado na representação de manobras geo-espaciais.

Os dois tipos mais comuns de veículos aéreos utilizam a aceleração vertical ou a aceleração-frontal para criar forças aerodinâmicas para seus deslocamentos. Na vertical (a maioria dos tipos de helicópteros), as manobras são feitas através do ajuste do **nick** para rotações em relação ao eixo longitudinal, o **tail** para rotações em relação ao eixo vertical e a **inclination** para rotações em relação ao eixo lateral. No caso da aceleração frontal (aviões em geral), as manobras são feitas através do ajuste do **aileron** para rotações na direção do eixo longitudinal, o **rudder** para rotações em relação ao eixo vertical e o **elevator** para rotações em relação ao eixo lateral.

5.2 O Projeto *MikroKopter*

A atual implementação do modelo toma como base o *firmware* do projeto *MikroKopter*, um projeto de engenharia elétrica/aeronáutica que desenvolve protótipos de VANTs com diversas configurações e diferentes números de hélices. Em geral, nestas configurações existem três principais componentes de *hardware* embarcado, sendo placas de circuitos integrados, onde cada uma delas realiza uma tarefa específica no sistema do VANT. Estes *hardwares* trabalham com protocolos de sinais para comunicações entre si ou com sistemas externos, gerenciando intruções programadas ou recebendo dados e informações via UART. Alguns trabalhos relacionados com o projeto *MikroKopter* são (HAFNER et al., 2010; OURSLAND, 2010; GRZONKA; GRISETTI; BURGARD, 2009), onde são trabalhadas principalmente as questões de algoritmos envolvendo autonomia e planejamento de caminhos. Em termos mais técnicos, os componentes de *hardware* embarcado são:

- `Flight-Ctrl`: A placa de *Flight-Ctrl* realiza a manipulação de dados referentes ao sistema e ao gerenciamento de medidas de ambiente como velocidades angula-

res dos eixos, aceleração, pressão atmosférica, carga da bateria e processamentos de posição. Esta placa é distribuída nas versões 1.0 (placa verde) e 1.3 (placa vermelha), e trabalha com um processador de 8 bits *Atmel ATMEGA644* de 20MHz, que é um circuito controlado por um microcontrolador de alta performance e baixo consumo de energia escrito sob arquitetura RISC[1];

- `Navi-Ctrl`: Esta placa extende as funcionalidades da *Flight-Ctrl* com um módulo de GPS e gerenciamento de geo-localização. Ações como *goHome* tornam-se possíveis de serem implementadas graças a este componente;

- `Brushless-Ctrl`: Esta placa atualiza o sistema com as diminuições de erros de interfaces e riscos de interferência além de aumentar a eficiência e a performance geral através de gerencia e controle dos motores por ativamento via *Pulse Width Modulation*[2] (PWM).

5.2.1 Configurações de Hélice e Manobra

No *UAVAS* atual, a modelagem das funções de vôo foram feitas com base na configuração de 4 hélices[3], como é mostrado na Figura 5.2, do modelo *MikroKopter* tomado como base.

Figura 5.2: Protótipo *MikroKopter* configurado com 4 hélices. Fonte: DELET (Departamento de Engenharia Elétrica, UFRGS).

Para executar as rotações de *Pitch*, *Roll* e *Yaw*, o protótipo de 4 hélices utiliza dife-

[1]Linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas.

[2]Técnica usada comumente para controlar a energia de componentes elétricos.

[3]A configuração de 4 hélices foi escolhida como base tendo-se em vista que a instituição na qual esta pesquisa foi realizada possui tais protótipos, sendo maiores as possibilidades de testes futuros no VANTs.

rentes tipos de configurações de força nos motores, de modo que as forças aerodinâmicas resultantes venham a ter seus torques anulados ou sobrepostos. A figura 5.3 mostra cada uma das configurações para a execução das manobras. Três tipos de setas indicam a intensidade da força atuante (velocidade de rotação da hélice), sendo forte para as mais grossas, média para as de tamanho médio e fracas para as pequenas.

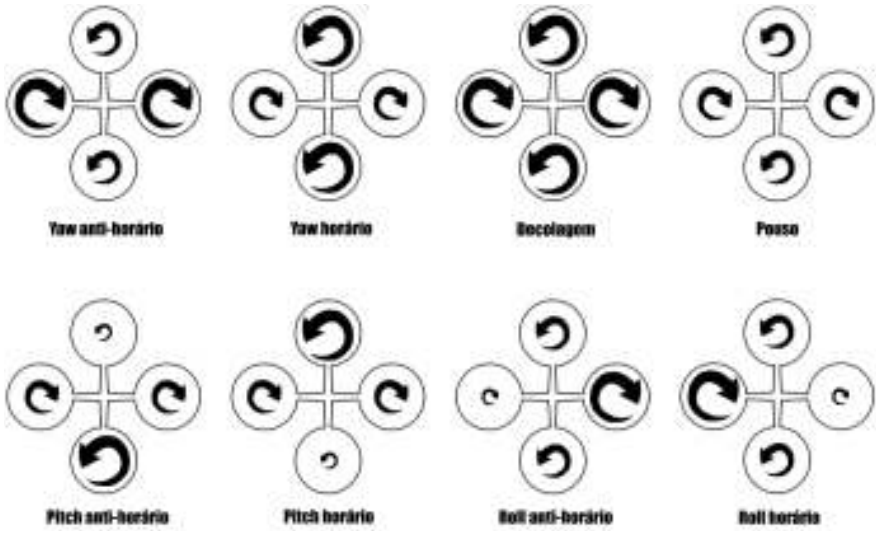

Figura 5.3: Esquema das forças que atuam nas hélices para manobras do VANT.

Como mostrado na Figura 5.3, tem-se as configurações de:

Yaw Anti-Horário : Basicamente a manobra de leme, onde as hélices das laterais giram mais rápido que as da frente e da traseira, resultando uma força que faz com que o VANT gire em sentido anti-horário;

Yaw Horário : Utiliza o mesmo princípio do leme anti-horário, mas as hélices que giram mais rápido são as da frente e da traseira. Estas hélices giram num sentido oposto ao horário, resultando uma força que faz com que o VANT gire em sentido horário;

Decolagem : Se todas as hélices girarem em mesma intensidade, todas hão de anular suas resultantes do plano sagital. A única força que fará diferença será o vetor vertical, fazendo com que o VANT suba;

Pouso : Todas as hélices giram com pouca intensidade, fazendo com que as resultantes do plano sagital sejam anuladas entre si e, consequentemente, o VANT vem a descer;

Pitch Anti-Horário : A hélice traseira gira com maior intensidade e a dianteira com menor intensidade, em relação às hélices laterais. Neste caso a manobra de elevador para baixo é gerada por meio da resultante maior na traseira do VANT;

Pitch Horário : O mesmo princípio do elevador para baixo é utilizado no elevador para cima, com a diferença de que a hélice que gira com maior intensidade é a frontal, sendo a resultante maior na frente do VANT;

Roll Anti-Horário : Corresponde à manobra gerada pelos ailerons de um avião. A resultante maior fica na lateral direita do VANT e faz com que o mesmo incline para a esquerda;

Roll Horário : Mesmo princípio da manobra de aileron anti-horário, mas a inversão de intensidade entre as hélices esquerda e direita faz com que o VANT incline para o lado direito.

5.2.2 Protocolo do *Firmware*

No projeto *MikroKopter* há um protocolo serial a nível de *firmware* com o qual é possível a realização da troca de mensagens e envio de requisições, passando pontos geo-espaciais como parâmetros ou mesmo podendo-se ler dados de condições gerais de vôo. Este protocolo[4] é baseado em blocos de dados seriais individuais e organizados como mostra a tabela 5.1. A Figura B.1 e a Figura B.2 (no apêndice) mostram cada uma das chamadas de métodos de *firmware* da *Navi-Ctrl* e *Flight-Ctrl*. O *UAVAS* faz uso de chamadas a estas rotinas de *firmware* e implementa métodos de maior nível abstrativo. Por exemplo, o método de alto nível **moveTo** disponível no protocolo de ações pode ser escrito através de uma chamada à rotina **SendTargetPosition** da API do *firmware*. Na implementação do modelo *UAVAS*, esta chamada está pré-prototipada em uma interface com os parâmetros tipados corretamente e prontos para serem sobreescritos.

Start-Byte	Address Byte	ID-Byte	n Data-Bytes coded	CRC-Byte1	CRC-Byte2	Stop-Byte
'#'	'a'+ Addr	'V','D' etc	"modified-base64"	variable	variable	'\r'

Tabela 5.1: Protocolo de comunicação do firmware MikroKopter. Fonte: MikroKopter Serial Protocol.

5.3 Implementação do *UAVAS*

A respeito das camadas do *framework* e do *engine* da implementação, a camada dos agentes é composta por pares de arquivos *.java* e *.asl*, que serão as próprias definições dos agentes. Uma é a arquitetura e a outra a definição dos comportamentos inteligentes. Observando-se ainda que o paradigma SMA prevê o par {*Agente*,*Ambiente*}, a implementação de uma classe que extende os ambientes virtuais provenientes da API de *Jason* é realizada de modo que cada um dos modelos possa se conectar convenientemente ao

[4]Definições e maiores informações na página WWW, em `http://www.mikrokopter.de/ucwiki/en/SerialProtocol`.

protocolo de ações, realizando uma comunicação entre o *firmware* e o *framework*/*engine*. Na camada do *engine*, o principal objetivo é a realização do processamento de dados recebidos e enviados de forma que as conversões sejam compatíveis.

Em relação às ferramentas utilizadas, listam-se e descrevem-se aqui os aplicativos e ambientes usados na implementação do modelo, tanto a plataforma principal quanto para a criação de simuladores. Todos os itens listados são *open-source* e podem ser obtidos da web de forma gratuita.

Java SE : A versão 6 do kit de desenvolvimento de software (Software Development Kit – SDK) da linguagem Java, juntamente com a máquina virtual (jre). Usada como principal linguagem na implementação do sistema;

RXTXComm : Biblioteca Java que utiliza *Java Native Interface* (JNI) para prover comunicação serial/paralela com a JDK. Usada na construção da comunicação entre o *UAVAS* e a API/UART;

Jason/AgentSpeak : Interpretador da linguagem *AgentSpeak*, que provê a camada de *runtime* dos Agentes e a arquitetura da SMA. Utilizado como *engine* de deliberação e interpretação das ações e percepções dos Agentes *UAVAS*;

JBullet : Biblioteca de simulações de física que possui um ambiente gráfico ambutido que utiliza *OpenGL* para renderização de gráficos. É na verdade um *porting* de C para Java, da biblioteca de simulações físicas **Bullet**. Utilizada principalmente no simulador, para criar cenários gráficos e ambientes com conceitos de física e resultantes vetoriais;

Flight Model Simulator : O simulador FMS é uma ferramenta leve de simulação de aeromodelismo, distribuído livremente e com diversas funcionalidades e configurações, permitindo simulações com uma grande diversidade de modelos. O FMS também permite a utilização de Radio-Transmissores conectados via USB ou portas seriais de modo que seja possível o controle dos VANTs virtuais através de controles reais. O FMS possui API fechada, o que não permite requisições internas à plataforma do FMS. Foi utilizado nas simulações de *firmware* e chamadas internas do *framework*.

Virtual Serial Port Emulator : O VSPE é uma pequena ferramenta de simulação de portas seriais e paralelas. Utilizado para a criação de portas virtuais na simulação com o FMS;

Eclipse : Ambiente de desenvolvimento geral que possui diversos *plugins* e *add-ons*, funcionando em muitas linguagens e perspectivas de arquiteturas diferentes. A implementação atual do modelo é um *source-code* cujo projeto foi criado no Eclipse e pode ser importado sem maiores dificuldades.

As classes implementadas no núcleo da plataforma são mostradas na tabela 5.2, onde são descritas as funções de cada uma delas, no contexto do *engine* do modelo *UAVAS*.

Classe	Função
GeoPoint	Modelo do ponto geo-espacial interno do *engine*
IActPerceptConverter	Interface para chaveamento
IIOManager	Interface de entrada e saída para conexão serial
IOManager	Implementação do gerenciador de entrada e saída serial
ProtocolConnection	Ambiente virtual que disponibiliza o protocolo de ações aos VANTs

Tabela 5.2: Classes do *engine* UAVAS.

5.4 Características Internas do Protótipo

Algumas características internas, inerentes ao funcionamento do protótipo podem ser descritas. Tais características dizem respeito ao funcionamento das entradas e saídas de sinais, do tratamento de concorrências intrínsecas às arquiteturas de SMA e Agentes, e às classes escritas.

5.4.1 Gerenciamento de Sinais de *Input/Output*

Os sinais gerados para as ações e percepções são enviados através de um canal de portas seriais/paralelas. Na implementação do modelo há a criação de uma classe que realiza a tarefa de abrir estas portas e converter sinais em blocos de dados legíveis pelo *engine*. Os dados de percepção podem ser criados nos *hardwares* do *firmware* (*Flight-Ctrl*, *Navi-Ctrl* ou *Brushless-Ctrl*) através de medidores de velocidade, pressão atmosférica, aceleração, entre outros. Os dados de ações são criados nas arquiteturas dos agentes, através dos ciclos de raciocínios deliberativos internos ao interpretador *Jason* utilizando os arquivos de definição de comportamento (*.ASL*). Nos dois sentidos de fluxo (entrada ou saída), os dados são codificados em bytes específicos pela **RXTXComm**.

5.4.2 Concorrência & Paralelismo

O *framework UAVAS* serve de ponte tecnológica entre os SMAs e *firmwares*, de modo que têm-se diversos SMAs comunicando-se entre si, onde a IDE *Jason* implementa os aspectos de concorrência de Agentes através de threads de execução onde cada um destes Agentes possui sua própria linha de execução. O fato de que um VANT é um conjunto de Agentes provenientes da arquitetura implementada em *Jason* faz com que um VANT seja internamente concorrente (sob os aspectos de múltiplas threads), tanto em termos de Agentes quanto em termos de intra-comunicações. No entanto, a comunicação inter-plataforma (entre SMA Aero-Veiculares) é feita através do ambiente extendido, e não há um tratamento formal quanto à checagem de aspectos de concorrência. No entanto, em simulador criado, a comunicação inter-VANT é feita por meio de mensagens enviadas

pelos Agentes, adotando um modelo FIFO, onde a primeira mensagem enviada é sempre a primeira recebida.

5.5 Estudo de Casos

Com o objetivo de validar a concepção e implementação do modelo, foram realizadas duas simulações para observação de algumas de suas características esperadas. Seguem, no próximo capítulo alguns resultados pertinentes, obtidos com estudos de caso específicos, um deles com ênfase nas comunicações inter-VANTs e cooperação de time, e outro com ênfase nas verificações dos mapeamentos de sinais com o envio de dados da infraestrutura. Para cada um dos casos, simuladores específicos foram criados no intuito de observar as características pertinentes de cada estudo de caso.

6 ESTUDOS DE CASO

Assim como foi mencionado no capítulo anterior, duas simulações foram criadas para a realização de verificações do protocolo e simulações com a utilização do *framework* concebido. Estas duas simulações foram criadas de uma maneira em que uma delas focou o objetivo de observar a comunicação entre os SMA aero-veiculares através do estabelecimento de grupos de trabalho, e outra o objetivo de observar as chamadas internas do *framework* e as conexões do protocolo com rotinas de baixo nível abstrativo com o uso de portas seriais virtuais onde o motor comunica-se com um *firmware* simulado.

6.1 Simulação TSim

O simulador **TSim** (*Team-Simulator*) foi criado com o intuito de realizar simulações com as ações de comunicação, trabalhando principalmente as noções de gerência e coordenação de times, com instanciações de mais de um SMA aero-veicular. O simulador utiliza uma interface visual pertencente a uma biblioteca de elementos de simulações físicas. A criação desta plataforma de simulação foi idealizada devido ao fato de que não foram encontradas, na literatura e na web, ferramentas que simulassem as características desejadas na implementação do modelo *UAVAS*, que são a existência de uma API aberta para leitura e escrita de rotinas de vôo, com possibilidade de utilização de múltiplas instanciações de SMA aero-veiculares (uso de mais de um VANT) na simulação.

6.1.1 Direções de Aceleração e Velocidade

No **TSim** foi implementado um ambiente virtual 3D onde os vetores de aceleração e velocidade dos modelos virtuais dos VANTs são criados através de um ambiente de física simulada, proveniente da biblioteca *JBullet* introduzida na seção 5.3, onde o conjunto de movimentos do VANT baseia-se no controle das quatro manobras básicas de uma aeronave (*Pitch, Roll, Yaw* e *Collective/Acceleration*). *Pitch* é a manobra que realiza a rotação em torno do eixo lateral de uma aeronave, fazendo com seu nariz suba ou desça. *Roll* é a manobra que realiza rotação em torno do eixo longitudinal da aeronave, e faz com que ela se incline para os lados. *Yaw* é a manobra que realiza rotação em torno do eixo

vertical da aeronave, fazendo com que ela gire para os lados, num eixo perpendicular ao seu plano sagital. Estes conceitos podem ser melhor visualizados no sistema de coordenadas que define rotações de objetos em espaço 3D, e permitem a realização de manobras gerais em um VANT com suas conversões. Na Figura 6.1 é mostrado como *Pitch*, *Roll* e *Yaw* compõem as manobras aéreas de quadrotores. A combinação das forças geradas pela *Collective/Acceleration*) com estas rotações é o que permite as manobras aéreas.

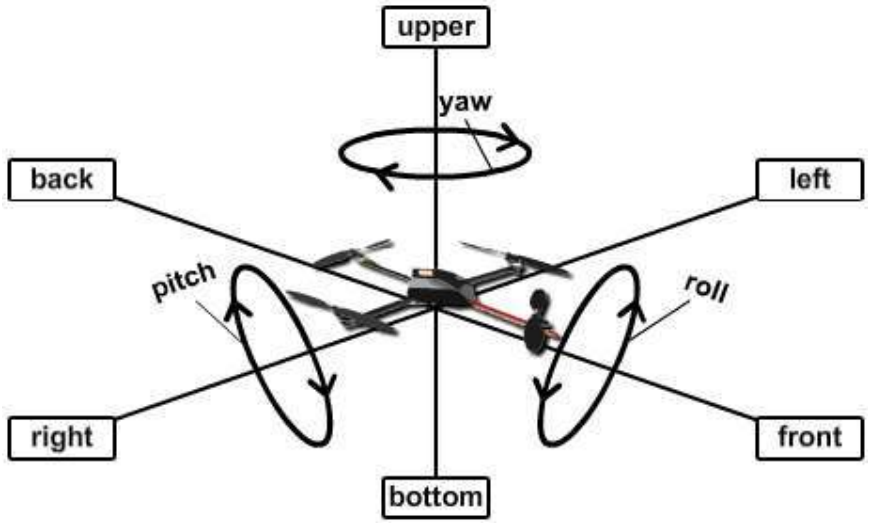

Figura 6.1: Sistema de coordenadas cartesianas para rotação em quadrotores: Pitch, Yaw e Roll.

6.1.2 Estudo De Caso: Coordenação De Times

O caso de estudo para o **TSim** ilustra um cenário onde 4 VANTs possuem a missão de obter um conjunto de fotografias panorâmicas de um objeto desconhecido, em um mesmo instante e de maneira sincronizada, para que todas as fotos possam ser sintetizadas em uma só posteriormente. A simulação da missão possui um líder do time sendo este líder o responsável pela gerência das ações do grupo de VANTs. Ainda, é necessário que os VANTs sempre andem em grupo dadas as características de comunicação (supondo um cenário onde o alcance do WiFi de comunicação seja restrito a alguns poucos kilômetros). Assim sendo, é necessário um trabalho em conjunto de forma coordenada para a conclusão da missão. A seguir, descreve-se resumidamente cada uma das instruções dos comportamentos escritos em *AgentSpeak* para os VANTs.

6.1.3 Descrevendo o Comportamento

Em um arquivo *AgentSpeak* existem, geralmente, quatro partes estruturais principais. As regras de domínio, as crenças iniciais, os desejos iniciais, e a definição de planos de objetivos e planos de contingência, como será visto a seguir.

Na primeira parte do código estão definidas as regras gerais de raciocínio. O seguinte trecho define a regra de cálculo de distância entre dois pontos geo-espaciais, dadas as suas latitudes, longitudes e altitudes. O resultado é retornado através de um processo de unificação similar à regras de Prolog, em Listing 6.1.

Listing 6.1: Implementação das regras lógicas.

```
// == Rules ==
distance(LtP, LnP, HtP, LtC, LnC, HtC, D) :-
   D = math.sqrt(
       math.sqrt((LtP-LtC)*(LtP-LtC)) +
       math.sqrt((LnP-LnC)*(LnP-LnC)) +
       math.sqrt((HtP-HtC)*(HtP-HtC))
).
```

Na segunda parte do código, encontrado em Listing 6.2, de descrição de comportamento estão listados as sentenças literárias as quais o agente VANT possuirá em sua base de conhecimentos assim que os ciclos do interpretador se iniciarem. Pode-se dizer que são os conhecimentos iniciais do agente. Na simulação, o agente VANT possui as crenças de que a missão não está completa (*missionIncomplete*) e que a missão ainda não se iniciou (*notInitiated*).

Listing 6.2: Implementação das crenças iniciais.

```
// == Pre-beliefs ==
missionIncomplete.
notInitiated.
```

Na terceira parte, em Listing 6.3, inicialmente existe unicamente o objetivo de inicialização.

Listing 6.3: Implementação dos objetivos iniciais.

```
// == Actions ==
!init.
```

Na quarta parte, os planos gerais são definidos. O plano **init** possui como subobjetivos o registro de um ID de chamada em relação ao *framework* UAVAS e *nicknames* de chamada (nesta simulação foram adotados as letras gregas: *alpha*, *beta*, *gamma*, etc). Além disso, inicializam-se dados do protocolo de ações do agente e define-se a posição *home*. Em seguida, toma-se como próximo objetivo a organização do time. O plano de contingência adota como objetivo a reinicialização do plano **init** caso o contexto seja de que o plano ainda não tenha se inicializado. Observe em Listing 6.4.

Listing 6.4: Estrutura do plano de inicialização.

```
// == Plans ==
+!init :
   myId(ID) & notInitiated <-
```

```
        registerId(ID); // register the ID in framework platform
        ?home(X,Y,Z); // uploads my current home location
        -notInitiated; // drops this belief
        !assignTeam(X,Y,Z,ID); // assign the team formation
        !initMission(ID). // initiates the mission
-!init :
    notInitiated <-
        !!init.
```

O plano **assignTeam** em Listing 6.5 toma como contexto o caso em que o agente é o líder, definindo requisições para os demais agentes, setando o destino e concluindo a fase de arranjo do time. Em Listing 6.6 são mostrados os planos que definem as tarefas de cada um dos agentes, no caso de não serem o líder. Na simulação, os agentes VANTs não-líderes são chamados de **alpha**, **beta** e **gamma**, onde os planos definem as tarefas de acordo com o referido *nickname*. Observa-se ainda, que o contexto dos planos assegura que os mesmos serão executados apenas no caso de terem recebido as requisições do líder, garantindo assim um acordo na missão.

Listing 6.5: Estrutura do plano de formação do grupo para o líder.

```
+!assignTeam(X,Y,Z,ID) :
    status(ok) & ID == leader <- // I'm the leader
        // send the request to alpha
        request(alpha, letsMoveOn([X,15,7],[-20,15,10]));
            // send the request to beta
        request(beta, letsMoveOn([X,Y,10],[-15,10,10]));
            // send the request to gamma
        request(gamma, letsMoveOn([X,5,7],[-20,5,10]));
        +checkPoint(X,Y,15); // my checkpoint
        +destination(-17,10,15); // the final destination
        +team(ok).
```

Listing 6.6: Estrutura dos planos de formação do grupo.

```
+!assignTeam(X,Y,Z,ID) :
    <- // We are the assistents
    status(ok) & not (ID == leader) & letsMoveOn([CkPt],[Dst]))
            +checkPoint(CkPt);
            +destination(Dst);
        +team(ok).
```

Chamada em **init**, o plano **initMission** em Listing 6.7 antes não possuia o contexto para ser executado. Porém, após a organização do time, a missão pode ser executada tendo-se em consideração os pontos de destino e de *checkpoint*. O *checkpoint* é um ponto entre o ponto-objetivo final e o ponto inicial da missão e tem por objetivo sincronizar as posições. Neste plano, são adicionados ainda os objetivos de monitoramento da posição atual e da realização de fotografia. Observemos também o uso da chamada do protocolo

através da ação *addWayPoint* a qual adiciona o ponto-objetivo final na lista de pontos do agente VANT, via seu *firmware*.

Listing 6.7: Estrutura do plano de inicialização da missão.

```
+!initMission(ID) :
    status(ok) & team(ok) & destination(Px,Py,Pz) &
    checkPoint(Pox,Poy,Poz) <-
        moveTo(Pox,Poy,Poz);
        addWayPoint(Px,Py,Pz);
        checkLocation;
        !listenToDestination(Px,Py,Pz);
        !photograph.
```

Os planos **listenToDestination** em Listing 6.8 são os que executam o monitoramento da posição global do agente VANT. Assim que o agente VANT estiver em local convenientemente próximo, é setada a crença **atLocation**. A contingência realiza uma nova checagem da localização e realiza uma nova chamada do plano.

Listing 6.8: Estrutura do plano de monitoramento de posição.

```
+!listenToDestination(Px,Py,Pz) :
    location(X,Y,Z) & missionIncomplete &
    distance(X,Y,Z,Px,Py,Pz,D) & D < 0.75 <-
        +atLocation.
-!listenToDestination(Px,Py,Pz) :
    true <-
        checkLocation;
        !!listenToDestination(Px,Py,Pz).
```

O plano **photograph** em Listing 6.9 define os subplanos a serem realizados quando no local do destino. O *delay* de 5 segundos é apenas para simular a realização da fotografia. Assim que a foto for tirada, caso o agente VANT seja o líder será enviado aos demais agentes a mensagem de que a missão foi concluída, além de realizar nova comunicação com o objetivo de retornar à posição de origem.

Listing 6.9: Estrutura do plano da operação de fotografia.

```
+!photograph :
    atLocation & myId(ID) & not leader <-
        .wait(5000); // taking photograph...
        -missionIncomplete;
        !getBackToHome.
+!photograph :
    atLocation & myId(ID) & leader <-
        .wait(5000); // taking photograph...
        -missionIncomplete;
        inform(2,letsGoHome);
        inform(3,letsGoHome);
```

```
        inform(4,letsGoHome);
        goHome.
-!photograph :
    missionIncomplete <-
        !!photograph.
```

O Plano **getBackToHome** em Listing 6.10 toma como contexto as ordens recebidas do agente líder e realiza a chamada do protocolo de ações para o retorno à posição inicial.

Listing 6.10: Estrutura do plano de retorno à base.

```
+!getBackToHome :
    lastInform(Who, letsGoHome) <-
        goHome.
-!getBackToHome :
    true <-
        !!getBackToHome.
```

A sequência de ações descritas pode ser visualizada pela sequência de *screenshots* mostradas na Figura 6.2. O momento **t1** mostra o objeto a ser fotografado ainda sem a presença do time de agentes VANTs. Em **t2** os agentes VANTs se sincronizam através do *checkpoint*. No momento **t3** é realizada a simulação de fotografia. Em **t4** os agentes VANTs estão retornando à posição de origem. A Figura 6.3 mostra, sob outro ângulo, o momento da simulação de fotografia.

6.1.4 Implementação do TSim

As classes implementadas no modelo da simulação **TSim** são mostradas na tabela 6.1 onde são descritas as funções de cada uma.

Classe	Função
TSimEnvironment	Extensão da arquitetura do ambiente em *Jason*
TSimUAVAS	Construtor do ambiente físico 3D
UFOModel	Modelo do objeto a ser fotografado

Tabela 6.1: Classes do simulador **TSim**.

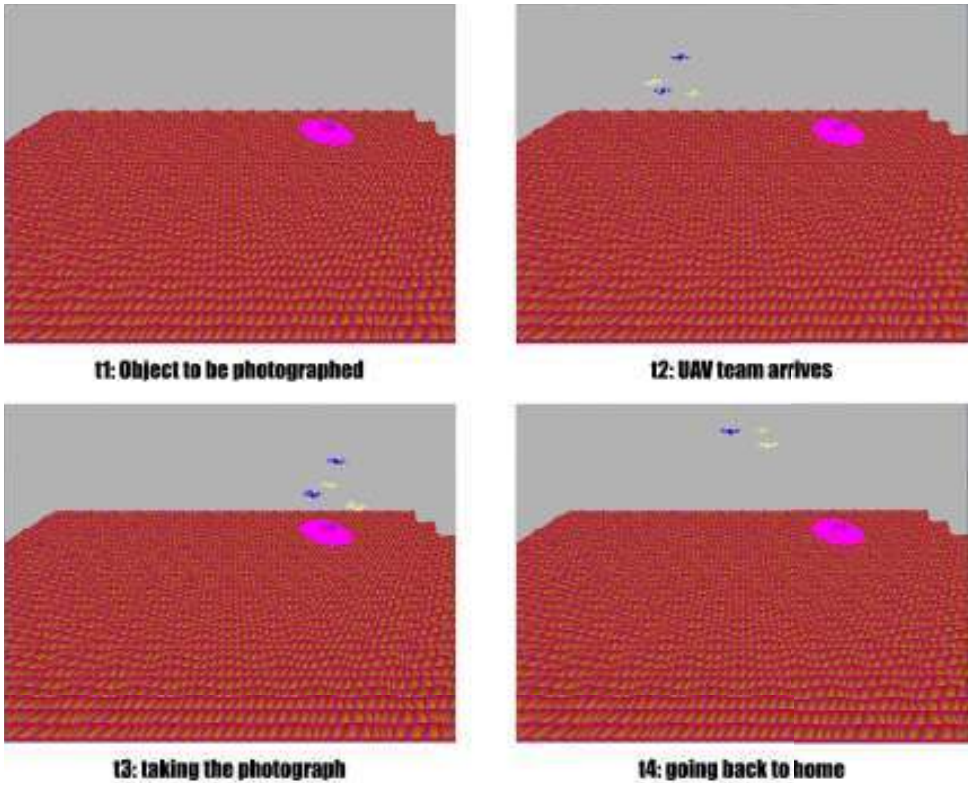

Figura 6.2: Sequência de screenshots do grupo de VANTs realizando a missão.

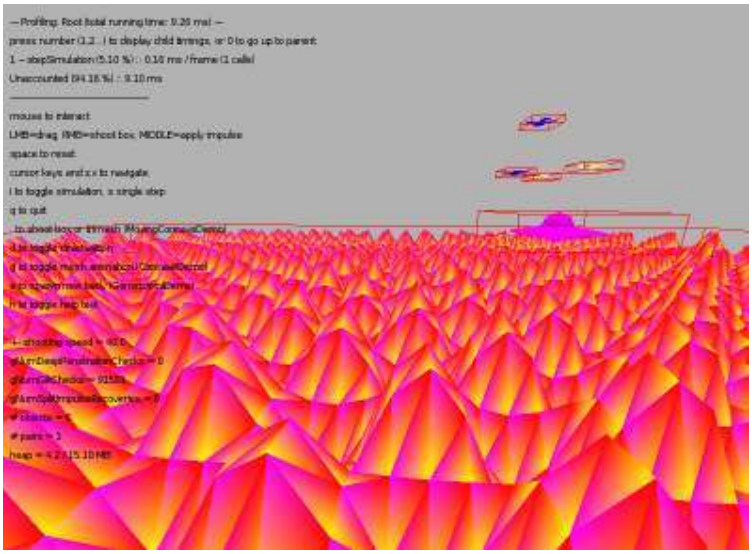

Figura 6.3: Visão panorâmica da equipe de VANTs realizando a missão.

6.2 Simulação FMSSim

A simulação **FMSSim** tem por objetivo principal a realização de observações referentes aos métodos de ações básicas (*Pitch*, *Roll*, *Yaw*) e com conexões entre o *framework* e o *firmware*. Inicialmente objetivava-se a realização de simulações com percepção de ambiente, mas estas não foram possíveis de serem realizadas devido ao fato de que a API do simulador FSM é fechada, fazendo com que as simulações com percepções de ambientes fossem deixadas exclusivamente para o simulador **TSim**. O simulador **FMSSim** realiza abertura de portas seriais, conecta-se ao SMA Aero-Veicular virtual e faz o envio de pacotes de bytes com a mesma codificação de radio-transmissores (controles-remoto) usados para controlar um VANT *MikroKopter*. Como as cadeias são enviadas conforme os resultados dos ciclos de raciocínios dos agentes, é correto afirmar que o controle dos VANTs do FMS é realizado pelo SMA Aero-Veicular escrito em *AgentSpeak*.

6.2.1 Flight Model Simulator

A ponte tecnológica entre o *framework UAVAS* e *firmware* é verificada com simulações no RC-Sim FMS[1] (mencionado no capítulo anterior), que é um simulador de rádio-comunicação. Normalmente, quando não totalmente autônomo, um VANT é manipulado através de um controlador de rádio por pilotos humanos. Um RC-Sim é uma ferramenta projetada para ajudar pilotos de VANTs a ganharem experiência no controle de VANTs, simulando VANTs em um universo virtual. Estes VANTs virtuais recebem comandos através dos sinais de saídas do rádio-controlador que são passados por portas USB ou serial. As configurações de *range* dos sinais são feitas por meio de processo de calibragem, como mostra a Figura 6.4.

As simulações no FMS são feitas por codificação das ações e percepções geradas no *UAVAS* para a mesma combinação de bytes gerados em um rádio-transmissor FMS e, em seguida, enviados como sinais de entrada para o FMS por meio de porta serial virtual.

6.2.2 Estudo de Caso: IO/Serial e Manobras Básicas

Ilustra-se aqui uma forma muito simples de operação de decolagem com um VANT *MikroKopter* virtual, usando o FMS. A simulação com o FMS faz uso de comandos muito mais simples em relação aos que foram desenvolvidos no **TSim**, devido principalmente às restrições de API. No entanto, pelo fato de haver emulação com portas seriais simulando dados de transmissores, esta simulação é a que mais se aproxima de uma abordagem realística do modelo *UAVAS* em um VANT real. As principais funcionalidades simuladas foram os comandos de *Pitch*, *Roll* e *Yaw*. Na Figura 6.5 encontra-se uma screenshot da simulação.

O seguinte excerto, em Listing 6.11, mostra a codificação *AgentSpeak* que descreve o

[1] A página web do FMS pode ser acessada em: `http://n.ethz.ch/~mmoeller/fms//`.

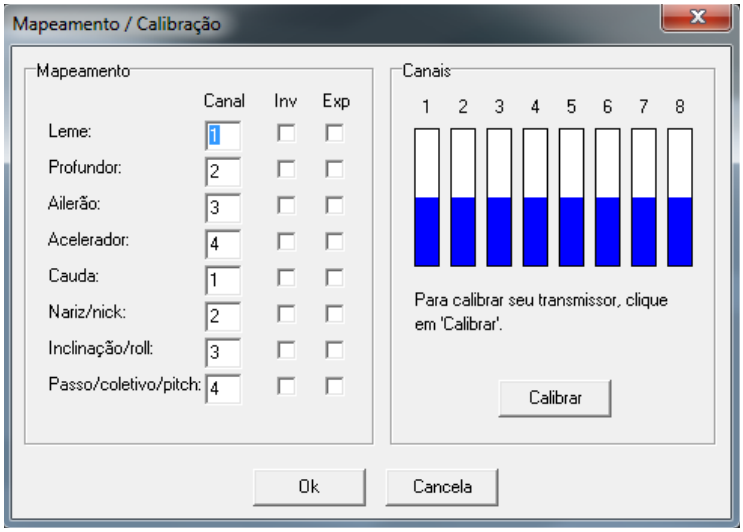

Figura 6.4: Calibragem do FMS para receber sinais do *framework* UAVAS.

comportamento do agente VANT no FMS. Existem dois planos principais: a calibragem e o de teste das operações.

Listing 6.11: Definição do comportamento do agente VANT no FMS.

```
+!init : true <-
    !calibrate;
    !test.

+!test : calibrated <-
    .wait(5000);
    collective(8);
    .wait(10000);
    collective(1);
    pitch(4);
    .wait(2000);
    collective(5);
    pitch(5);
    !!test.

+!calibrate : true <-
    .wait(3000); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait( 500); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait( 500); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait(5000); calibrateMidValue;
   +calibrated.
```

Figura 6.5: Simulação de ascenção de posição através do FMS.

6.2.3 Implementação do FMSSim

As classes implementadas no modelo da simulação **FMSSim** são mostradas na tabela 6.2 onde são descritas as funções de cada uma.

Classe	Função
FMSSimEnvironment	Extensão da arquitetura do ambiente em *Jason*
FMSActPerceptConverter	Construtor do ambiente físico 3D
IFMSUAVASBridge	Interface dos métodos das manobras

Tabela 6.2: Classes do simulador **FMSSim**.

6.3 Aplicação em VANTs

Dentro das implementações do modelo, um dos tópicos em corrente discussão e análise é a aplicação do modelo UAVAS para VANTs reais através de uma plataforma de hardware que sirva de infraestrutura para o *framework*, como a *Beagle Board*[2]. Atualmente, foram realizados testes com uma versão de Linux minimalista DSL (SHINGLEDECKER; ANDREWS; NEGUS, 2008), que tem como principais características o baixo consumo de processamento e o pequeno tamanho de instalação. A aplicação para VANTs reais toma como foco as questões de compatibilidades de hardware para um possível *deploy* nos protótipos da *MikroKopter*, em particular nas aeronaves do Departamento de Engenharia Elétrica da UFRGS. A Figura 6.6 mostra o atual *deploy* do *framework* UAVAS para o Linux DSL em uma máquina virtual de configuração reduzida (2GB de disco e 128 de memória RAM).

[2]Webpage em `http://beagleboard.org/`. Acessada em 30/11/2011.

Figura 6.6: Simulação do AddWayPoint com o *framework* UAVAS rodando no DSL.

6.4 Análise dos Resultados

O modelo UAVAS aplica o paradigma SMA para implementar, em alto nível, métodos inerentes ao contexto dos VANTs utilizando neste primeiro momento o protocolo de comunicação disponível no *firmware* do projeto *MikroKopter*. Ao longo de sua história, o paradigma SMA possui boas referências, projetos relacionados, ferramentas e bases teóricas. O modelo toma *AgentSpeak* como linguagem de implementações para descrever missões e comportamentos dos VANTs provendo assim, uma solução para a engenharia de software aplicada ao desenvolvimento de autonomia na área das ciências aeronáuticas.

Nos estudos de casos implementados, caso fossem abordadas a utilização de linguagens como **C** ou **Assembly** (linguagens padrão do *firmware*) para a escrita do comportamento autônomo, estima-se que o número de linhas de código fosse muito maior dado que as estruturas de **C** e **Assembly** são pouco expressivas. Ainda o grau técnico necessário para desenvolver a mesma autonomia seria muito maior, pois haveriam preocupações extras com ponteiros, interrupções e gerenciamentos de entrada/saída. Estes mecanismos existem nas implementações do modelo, mas são implementados e encapsulados internamente ao *framework*, poupando o desenvolvedor do conhecimento técnico das linguagens de baixo nível. *AgentSpeak* não utiliza laços de controle como **while** ou **if-else** e sim sequências de instruções ativadas através de contextos lógicos, fazendo com que o fluxo de execução seja substituído por ciclos deliberativos.

Estas peculiaridades ficam mais perceptíveis quando confrontamos uma codificação do *firmware* com os comandos do protocolo de ações. O paradigma da programação orientada a agentes é utilizado na camada do *framework*, em particular no *engine* da plataforma. Na implementação do modelo proposto, a mesma instrução de código **WPList_Append** (adicionar ponto geo-espacial na lista) mostrada na listagen 4.1 é escrita inteiramente com

a instrução **addWayPoint(Wp)**, de maneira mais clara e simples sob o ponto de vista relativo à expressividade semântica. Seus aspectos internos são construídos e encapsulados, e sua chamada é exposta para uso dentro do cenário do agente, fazendo com que o sistema usufrua do paradigma da POA e da arquitetura BDI.

Durante as implementações dos casos, o maior resultado obervado foi a diminuição do *gap* semântico, bem como a simplificação das expressões através de encapsulamentos das rotinas que realmente são importantes e/ou utilizadas pelas aeronaves. O sistema torna-se mais claro e fácil de ser desenvolvido, bem como o comportamento inteligente individual e coletivo dos VANTs modelados.

Em geral, as linguagens interpretadas possuem maiores cargas de processamento e exigências de memória e assim, certamente *AngetSpeak* é mais pesada que as linguagens de baixo nível. No entanto, a rapidez do avanço do hardware permite que se pense em um *framework* de software relativamente pesado mesmo estando dentro do contexto de VANTs. Algoritmos eficientes e arquiteturas robustas não acompanham com a mesma velocidade com que se desenvolvem o hardware para computadores. Assim, espera-se que a infraestrutura proposta nesta pesquisa venha a ser útil como plataforma de desenvolvimento, visto que a concepção de bons algoritmos é mais impactante, tanto no âmbito de complexidade de desenvolvimento e codificação quanto no de escalabilidade e reusabilidade.

7 CONCLUSÕES

Até onde foram realizadas as pesquisas no estado da arte, pouco foi encontrado em relação a trabalhos similares. Comparações com outros modelos não puderam ser realizadas devido ao fato de que tais tipos de infraestruturas ou são comerciais ou não são orientadas a agentes e sistemas multi-agentes. Como visto nos primeiros capítulos, a literatura em engenharia de software aplicada a VANTs é, apesar de crescente, ainda bastante restrita e carente de trabalhos, em relação a sistemas inteligentes e autônomos.

No intuito de fornecer novas abordagens para a área da engenharia de software para VANTs, este trabalho realizou a concepção de um modelo orientado a agentes e SMA com uso do POA através de linguagens de alto nível abstrativo e arquitetura BDI. Uma modelagem comparativa entre uma aeronave real e um VANT foi criada, assim como algumas definições. Neste primeiro momento, muitos outros tópicos de importante caráter foram deixadas para pesquisas em trabalhos futuros, alguns destes englobando os assuntos da semântica das ações do modelo, análises de *swarm scale*, concorrência entre planos dos agentes, análise algorítmica da concorrência e do paralelismo de comunicação, entre outros. Para validar o modelo, foram realizadas implementações de dois simuladores, trabalhando aspectos das operações de comunicação inter-agentes por meio do *framework* e as conversão de sinais através de portas seriais por meio de emulação. Os momentos finais do estudo analisaram as compatibilidades e especifiedades da implamntação do modelo em protótipos reais através de componentes de hardware embarcado, onde vislumbra-se que a aplicação para VANTs reais é factível.

No que diz respeito às principais propostas deste estudo, crê-se que o corrente trabalho tenha atingido os seus objetivos, o qual era a concepção e a definição de um modelo orientado a agentes e SMA para a engenharia de software aplicado a VANTs e aeromodelos.

APÊNDICE A BNF AGENTSPEAK

```
agent           ::= beliefs plans
beliefs         ::= ( literal "." )*
                    N.B.: um erro semântico é gerado se o
                    literal tiver variáveis não instanciadas.
plans           ::= ( plan )+
plan            ::= [ atomic_formula "->" ]
                    triggering_event ":" context "<-" body "."
triggering_event ::= "+"     literal
                   | "-"     literal
                   | "+" "!" literal
                   | "-" "!" literal
                   | "+" "?" literal
                   | "-" "?" literal
literal         ::= "~" atomic_formula
                  | "~" "(" atomic_formula ")"
                  | atomic_formula
default_literal ::= "not" literal
                  | "not" "(" literal ")"
                  | literal
context         ::= "true"
                  | default_literal ( "&" default_literal )*
body            ::= "true"
                  | body_formula ( ";" body_formula )*
body_formula    ::= literal
                  | "!" literal
                  | "?" literal
                  | "+" literal
                  | "-" literal
atomic_formula  ::= <ATOM> [ "(" list_of_terms ")" ]
                    [ "[" list_of_annotations "]" ]
structure       ::= <ATOM> "(" list_of_terms ")"
list_of_terms   ::= term ( "," term )*
list_of_annotations ::= as list_of_terms, but generating a
                    semantic error if not ground.
list            ::= "["
                    [ term ( ( "," term )*
                           | "|" ( list | <VAR> )
                           )
                    ] "]"
term            ::= structure
                  | list
                  | <ATOM>
                  | <VAR>
                  | <NUMBER>
                  | <STRING>
```

Figura A.1: BNF da *AgentSpeak* aceita em JASON.

APÊNDICE B MÉTODOS DE *FIRMWARE* DA *NAVI-CTRL* E *FLIGHT-CTRL*

	Received by FC			Sent by FC			
Description	ID	Address	Data	ID	Address	Data	Above FC Firmware
Compass Heading	'V'	FC-Addr	s16 Compass Value	'V'	MK(MAG)-Addr	Nick Roll Attitude ...	0.73f
Engine Test	'T'	FC-Addr	u8[16] values for the engines	'T'	FC-Addr	-	0.73f
Settings Request	'e'	FC-Addr	u8 Settings Index (1..3 READ or Diff for actual setting) u8 Settings Index (11..13 RESET setting to default (channel mapping) will not be changed!) u8 Settings Index (21..23 RESET setting to default (complete reset including channel settings!)	'E'	FC-Addr	u8 Settings Index, u8 Settings Version, Settings Struct	0.73f
Write Settings	'e'	FC-Addr	u8 Settings Index, u8 Settings Version, Settings Struct	'E'	FC-Addr	u8 Settings Index (1 ..3, 0=Error)	0.73f
Read PPM Channels	'g'	FC-Addr	none	'P'	FC-Addr	s16 PPM-data[11]	0.73f
Set 3D-Data Interval	'y'	AnyAddr	u8 Interval	'C'	FC-Addr	struct Data3D	0.72g
Mixer Request	'o'	FC-Addr	none	'M'	FC-Addr	u8 MixerVersion, u8 Name[12], u8 MixerTable[16][4]	0.73
Mixer Write	'm'	FC-Addr	u8 MixerVersion, u8 Name[12], u8 MixerTable[16][4]	'm'	FC-Addr	u8 ack (1 = okay, 0 = error)	0.73
Change setting	'P'	FC-Addr	u8 Number of new Setting	'P'	FC-Addr	u8 Number	0.77
Serial Poti	'p'	FC-Addr	u8 Poti[12]	-	-	-	0.77
BL parameter request	'a'	FC-Addr	u8 BL_Addr	'A'	FC-Addr	u8 Status1, u8 Status2, u8 BL_Addr, BLConfig Struct	0.80
BL parameter write	'a'	FC-Addr	u8 BL_Addr, BLConfig Struct	'W'	FC-Addr	u8Status1, u8 Status2	0.80

Figura B.1: Protocolo da Flight-Ctrl.

	Received by NC			Sent by NC			
Description	AT	Address	Data	AT	Address	Data	since NC Firmware
Serial Link Test	'b'	NC-Addr	c(10 EchoPattern	'B'	NC-Addr	c(10 EchoPattern	0.14f
Error Text Request	'e'	NC-Addr	none	'E'	NC-Addr	char[] Error Message String	0.12h
Send target Position	'c'	NC-Addr	WayPointStruct	-	-	-	0.12h
Send Waypoint	'w'	NC-Addr	WayPointStruct (sending an invalid position will clear the WPList)	'W'	NC-Addr	u8 Number of WPs	0.12h
Request Waypoint	'r'	NC-Addr	u8 WP-Index	'R'	NC-Addr	u8 Number of WPs, u8 WP-Index, WayPointStruct	0.14f
Request DSI-Data	'd'	NC-Addr	1 byte sending interval (in 20ms steps)	'D'	NC-Addr	NaviDataStruct	0.12h
Redirect UART	'u'	NC-Addr	1 byte param for uart selector (0=FC, 1=MK3MAG, 2=MK3FC), can be switched back to NC debug by sending the magic packet 'Yx18,0x38,0x30,0x44,0x00'	-	-	-	0.12h
NC 3D-Data Interval	'c'	AnyAddr	u8 Interval	'C'	NC-Addr	struct Data3D	0.14a
Set/get NC-Parameter	'P'	NC-Addr	u8 get/(0:set/1):u8 parameterId, s16 value (only when set)	'P'	NC_Addr	u8 parameterId, s16 value	0.20

Figura B.2: Protocolo da Navi-Ctrl.

APÊNDICE C ARQUIVOS DE COMPORTAMENTOS DAS SIMULAÇÕES

Listing C.1: uavas_fms.asl

```
+!init : true <-
    !calibrate;
    !test.

+!test : calibrated <-
    .wait(5000);
    collective(8);
    .wait(10000);
    collective(1);
    pitch(4);
    .wait(2000);
    collective(5);
    pitch(5);
    !!test.

+!calibrate : true <-
    .wait(3000); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait( 500); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait( 500); calibrateMinValue;
    .wait( 500); calibrateMaxValue;
    .wait(5000); calibrateMidValue;
    +calibrated.
```

Listing C.2: uavas_tsim.asl

```
// == Rules ==
distance(LtP, LnP, HtP, LtC, LnC, HtC, D) :-
   D = math.sqrt(
       math.sqrt((LtP-LtC)*(LtP-LtC)) +
       math.sqrt((LnP-LnC)*(LnP-LnC)) +
       math.sqrt((HtP-HtC)*(HtP-HtC))
).

// == Pre-beliefs ==
missionIncomplete.
notInitiated.

// == Actions ==
!init.

// == Plans ==
+!init :
   myId(ID) & notInitiated <-
       registerId(ID); // register the ID in framework platform
       ?home(X,Y,Z); // uploads my current home location
       -notInitiated; // drops this belief
       !assignTeam(X,Y,Z,ID); // assign the team formation
       !initMission(ID). // initiates the mission
-!init :
   notInitiated <-
       !!init.

+!assignTeam(X,Y,Z,ID) :
   status(ok) & ID == leader <- // I'm the leader
       // send the request to alpha
       request(alpha, letsMoveOn([X,15,7],[-20,15,10]));
           // send the request to beta
       request(beta, letsMoveOn([X,Y,10],[-15,10,10]));
           // send the request to gamma
       request(gamma, letsMoveOn([X,5,7],[-20,5,10]));
       +checkPoint(X,Y,15); // my checkpoint
       +destination(-17,10,15); // the final destination
       +team(ok).
+!assignTeam(X,Y,Z,ID) :
   <- // We are the assistents
   status(ok) & not (ID == leader) & letsMoveOn([CkPt],[Dst]))
           +checkPoint(CkPt);
           +destination(Dst);
       +team(ok).

+!initMission(ID) :
   status(ok) & team(ok) & destination(Px,Py,Pz) & checkPoint(Pox,Poy,Poz) <-
       moveTo(Pox,Poy,Poz); // adds the checkpoint as a waypoint
       addWayPoint(Px,Py,Pz); // adds the final destination
       checkLocation;
       !listenToDestination(Px,Py,Pz); // initiates the position monitoring
       !photograph.

+!listenToDestination(Px,Py,Pz) : // plan for listening to the position
   location(X,Y,Z) & missionIncomplete & distance(X,Y,Z,Px,Py,Pz,D) & D < MinDist <-
       +atLocation.
-!listenToDestination(Px,Py,Pz) :
   true <-
```

```
        checkLocation;
        !!listenToDestination(Px,Py,Pz).

+!photograph :
    atLocation & myId(ID) & not (ID == leader) <-
        .wait(5000); // taking photograph...
        -missionIncomplete;
        !getBackToHome.
+!photograph :
    atLocation & myId(ID) & ID == 1 <-
        .wait(5000); // taking photograph...
        inform(alpha,letsGoHome);
        inform(beta,letsGoHome);
        inform(gamma,letsGoHome);
        -missionIncomplete;
        goHome.

+!getBackToHome :
    letsGoHome <-
        goHome.
```

APÊNDICE D INTERFACE DO PROTOCOLO

Listing D.1: IActPerceptConverter.java

```
package mind.engine;

import jason.asSyntax.Literal;

public interface IActPerceptConverter {
        // logistic actions
        public boolean move_to(GeoPoint p);

        public boolean add_waypoint(GeoPoint p);

        public void set_home(GeoPoint p);

        public GeoPoint get_home();

        // communication actions
        public boolean request(String agName, Literal l);

        public boolean inform(String agName, Literal l);

        public boolean ask(String agName, Literal l);

        public boolean ack(String agName);

        // perceive actions
        public String check_system();

        public float check_battery();

        public GeoPoint check_location();
}
```

REFERÊNCIAS

ABRAN, A.; MOORE, J. W. **Software Engineering Body of Knowledge**. SWEBOK: IEEE Computer Society, 2004.

AHMADZADEH, A. et al. Multi-UAV Cooperative Surveillance with Spatio-Temporal Specifications. In: IEEE CONFERENCE ON DECISION AND CONTROL, San Diego, CA. **Anais...** IEEE, 2006. p.5293 – 5298.

ALMEIDA, G. O. de. **Utilização da Abordagem de Sistemas Multiagentes no Problema do Controle do Espaço Aéreo**. 2007. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pelotas, Depto de Informática.

BAUZA, R.; GOZALVEZ, J.; SANCHEZ-SORIANO, J. Road Traffic Congestion Detection through Cooperative Vehicle-to-Vehicle Communications. In: WORKSHOP ON USER MOBILITY AND VEHICULAR NETWORKS, Denver, CO. **Anais...** IEEE, 2010. p.606 – 612.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. **JADE - A FIPA-compliant agent framework**. Centro Studi E Laboratori Telecomunicazioni: Universita Degli Studi di Parma, 1999.

BERRY, B.; KIEL, D.; ELLIOTT, E. Adaptive agents, intelligence, and emergent human organization: capturing complexity through agent-based modeling. In: NATIONAL ACADEMY OF SCIENCES OF THE UNITED STATES OF AMERICA, University of Texas at Dallas, Richardson. **Proceedings...** The National Academy of Sciences, 2002. p.7187 – 7188.

BORDINI, R. H.; HüBNER, J. F. BDI Agent Programming in AgentSpeak Using Jason. In: INTERNATIONAL WORKSHOP ON COMPUTATIONAL LOGIC IN MULTI-AGENT SYSTEMS, City University London, UK. **Anais...** Springer, 2005. p.143 – 164.

BORDINI, R. H.; HüBNER, J. F.; WOOLDRIDGE, M. **Programming Multi-Agent Systems in AgentSpeak Using Jason**. Nova Jersey, USA: John Wiley and Sons Ltd, 2007. 292p.

BRATMAN, M. E. Intention and means-end reasoning. **Philosophical Review**, Stanford University, v.90, n.2, p.252 – 265, 1981.

BRAZIER, F. M. T. et al. DESIRE: modelling multi-agent systems in a compositional formal framework. **International Journal of Cooperative Information Systems**, University of Southampton, p.67 – 94, 1997.

BUSETTA, P. et al. JACK Intelligent Agents - Components for Intelligent Agents in Java. **AgentLink Newsletter**, Melbourne, Australia, v.Volume 2, 1999. 2 – 5.

CHENG, P.; KELLER, J.; KUMAR, V. Time-optimal UAV trajectory planning for 3D urban structure coverage. In: INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, Univ. of Pennsylvania, Philadelphia. **Anais...** IEEE/RSJ, 2008. p.2750 – 2757.

CHEVALIER, R. Updating the Behavior Engineering Model. **ISPI Performance Improvement Global Network Chapter**, [S.l.], v.Volume 42, p.8 – 14, 2003.

COHEN, P. R.; LEVESQUE, H. J. **Rational Interaction as the Basis for Communication**. 333 Ravenswood Ave., Menlo Park, CA 94025: AI Center, SRI International, 1988.

COHEN, P. R.; LEVESQUE, H. J. Intention Is Choice With Commitment. **Artificial Intelligence**, Essex, UK, v.Volume 42 Issue 2-3, p.213 – 261, 1990.

COLLINOT, A.; DROGOUL, A. Using the Cassiopeia Method to Design a Robot Soccer Team. **Applied Artificial Intelligence**, Paris, France, v.Volume 12, Issue 2-3, p.127 – 147, 1998.

DARGAR, A. et al. An Agent-based Framework for UAV Collaboration. In: CONFERENCE ON INTELLIGENT SYSTEMS, North Dakota State University. **Anais...** ISCA, 2002. p.54 – 59.

DAWSON, J. W. Gödel And The Limits Of Logic. **Scientific American**, Academic Search Premier, v.Volume 280, Issue 6, p.1 – 7, 2006.

DEJONG, P. **Coalition Formation in Multi-Agent UAV Systems**. 2005. Dissertação (Mestrado em Ciência da Computação) — University of Central Florida.

DIXON, D. S. **An Agent-Based Adaptation of Friendship Games**: observations on network topologies. Albuquerque NM 87131, USA: University of New Mexico, 2011.

DOD, D. O. D. **Unmanned Aircraft Systems Roadmap**. United State of America, 2005.

WILSON, S. W. (Ed.). **Robot Shaping**: an experiment in behavior engineering (intelligent robotics and autonomous agents). [S.l.]: A Bradford Book, 1997.

FININ, T. et al. KQML as an agent communication language. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, New York, NY, USA. **Anais...** ACM Press, 1994. p.456 – 463.

GIACOMO, G. D. et al. IndiGolog: a high-level programming language for embedded reasoning agents. **Multi-Agent Programming**, Springer, v.Part 1, p.31 – 72, 2009.

GLINTON, R.; SCERRI, P.; SYCARA, K. An Investigation of the Vulnerabilities of Scale Invariant Dynamics in Large Teams. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, Carnegie Mellon University. **Anais...** ACM Press, 2011. p.677 – 684.

GORODETSKY, V. et al. Agent-Based Air Traffic Control in Airport Airspace. In: IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT AGENT TECHNOLOGY, SPIIRAS, St. Petersburg. **Anais...** IEEE/WIC/ACM, 2007. p.81 – 84.

GRIMALDO, F.; LOZANO, M.; BARBER, F. MADeM: a multi-modal decision making for social mas. **International joint conference on Autonomous agents and multiagent systems**, Richland, SC, v.Volume 1, p.183 – 190, 2008.

GRZONKA, S.; GRISETTI, G.; BURGARD, W. Towards a navigation system for autonomous indoor flying. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2009. p.1679 – 1684.

GUTTMAN, R. H.; MOUKAS, A. G.; MAES, P. Agent-mediated electronic commerce: a survey. **The Knowledge Engineering Review**, New York, NY, USA, v.Volume 13, Issue 2, p.147 – 159, 1998.

HAFNER, V. V. et al. An autonomous flying robot for testing bio-inspired navigation strategies. In: INTERNATIONAL SYMPOSIUM ON AND 2010 6TH GERMAN CONFERENCE ON ROBOTICS, Munich, Germany. **Anais...** VDE Verlag, 2010. p.1 – 7.

HAMA, M. T. et al. UAVAS: agentspeak agents for unmanned aerial vehicles. In: II WORKSHOP ON AUTONOMOUS SOFTWARE SYSTEMS, São Paulo, SP, Brazil. **Anais...** Autosoft, 2011. p.7.

HINDRIKS, K.; HOEK, W. GOAL Agents Instantiate Intention Logic. In: EUROPEAN CONFERENCE ON LOGICS IN ARTIFICIAL INTELLIGENCE, 11., Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2008. p.232 – 244.

HINDRIKS, K. V. et al. Agent Programming in 3APL. **Autonomous Agents and Multi-Agent Systems**, Kluwer Academic Publishers, v.Volume 2, p.357 – 401, 1999.

HINTIKKA, J. **Knowledge and Belief**. University of Cambridge: King's College Publications, 1962.

HOARE, C. A. R. An Axiomatic Basis For Computer Programming. **Communications of the ACM**, ACM, v.Volume 12 Issue 10, p.576 – 580, 1969.

HUBNER, J. F. et al. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. **International Journal of Agent-Oriented Software Engineering**, Inderscience Publishers, Geneva, SWITZERLAND, v.Volume 1 Issue 3/4, p.370 – 395, 2007.

HUBNER, J. F.; SICHMAN, J. S. SACI: uma ferramenta para implementação e monitoração da comunicação entre agentes. In: IBERO-AMERICAN CONFERENCE ON AI, 15TH BRAZILIAN SYMPOSIUM, 7., São Carlos, São Paulo. **Proceedings...** ICMC/USP, 2000. p.47 – 56.

HUFF, N.; KAMEL, A.; NYGARD, K. E. An Agent Based Framework for Modeling UAV's. In: COMPUTER APPLICATIONS IN INDUSTRY AND ENGINEERING, Las Vegas, Nevada, USA. **Anais...** ISCA, 2003. p.139 – 144.

HUTH, M.; RYAN, M. **Logics in Computer Science**. Cambridge University: Cambridge University Press, 2004.

ISCOLD, P. et al. Desenvolvimento de Horizonte Artificial para Aviação Geral baseado em Sensores MEMS. In: V SIMPóSIO BRASILEIRO DE ENGENHARIA INERCIAL, VerLab. **Anais...** UFMG, 2007. p.6.

JAKOB, M. et al. Occlusion-Aware Multi-UAV Surveillance of Multiple Uraban Areas. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS, Toronto, Canada. **Anais...** ACM Press, 2010. p.1407 – 1408.

JENNINGS, N. R.; SYCARA, K.; WOOLDRIDGE, M. A Roadmap of Agent Research and Development. **Autonomous Agents and Multi-Agent Systems**, Springer, v.Volume: 1, Issue: 1, p.7 – 38, 1998.

JUN, M.; D'ANDREA, R. **Cooperative Control**: models, applications and algorithms. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2002. v.18-23 July, p.95 – 111.

KARIM, S.; HEINZE, C. Experiences with the design and implementation of an agent-based autonomous UAV controller. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, University of Melbourne. **Proceedings...** ACM Press, 2005. p.19 – 26.

KHAN, S. et al. A multi-agent system for the quantitative simulation of biological networks. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, Melbourne, Australia. **Proceedings...** ACM New York: NY: USA, 2003. p.385 – 392.

KIM, J.; CRASSIDIS, J. L. UAV Path Planning for Maximum Visibility of Ground Targets in an Urban Area. In: CONFERENCE ON INFORMATION FUSION, Dept. of Aerosp. Eng., Univ. of Glasgow, Glasgow, UK. **Anais...** IEEE, 2010. p.1 – 7.

KLAPISCAK, T.; BORDINI, R. H. **Declarative Agent Languages and Technologies VI**. Berlin, Heidelberg: Springer-Verlag, 2008. v.Volume 5397, p.91 – 110.

LONGHITANO, G. A. **VANTS para sensoriamento remoto**: aplicabilidade na avaliação e monitoramento de impactos ambientais causados por acidentes com cargas perigosas. 2010. Dissertação (Mestrado em Ciência da Computação) — Universidade de São Paulo, Escola Politécnica.

LUPASHIN, S. et al. A Simple Learning Strategy for High-Speed Quadrocopter Multi-Flips. In: INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, Anchorage, AK. **Anais...** IEEE, 2010. p.1642 – 1648.

MANGINA, E. Intelligent Agent-Based Monitoring Platform forApplications in Engineering. **International Journal of Computer Science & Applications**, Technomathematics Research Foundation, v.Volume 2, No. 1, p.38 – 48, 2005.

MARSH, W. E. **An Initial Methodology For The Definition And Implementation Of Unmanned Aerial Vehicle Agent Behaviors**. 2007. Dissertação (Mestrado em Ciência da Computação) — Arizona State University.

MELO, B. P. N. et al. **Lógica Modal**. Departamento de Informática e Matemática Aplicada: Universidade Federal do Rio Grande do Norte, 2009.

MICHAEL, N. et al. The GRASP Multiple Micro UAV Testbed. **IEEE Robotics and Automation Magazine**, Philadelphia, Pennsylvania, v.Volume 17 Issue:3, p.56 – 65, 2010.

NATO. **Department of Defense Dictionary of Military and Associated Terms**. Joint Publication 1-02, 2001.

NETO, A. A.; CAMPOS, M. F. M. Implementação de um Sistema Hardware-in-the-Loop para Veículos Aéreos Autônomos Não-tripulados. In: XVII CONGRESSO BRASILEIRO DE AUTOMáTICA, Juiz de Fora, MG. **Anais...** CBA, 2008. p.1.

NETO, A. A.; CAMPOS, M. F. M. On the Generation of Feasible Paths for Aerial Robots with Limited Climb Angle. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, Kobe, Japan. **Proceedings...** ICRA, 2009. p.6.

NETO, A. A.; MACHARET, D. G.; CAMPOS, M. F. M. On the Generation of Trajectories for Multiple UAVs in Environments With Obstacles. In: IEEE INTERNATIONAL SYMPOSIUM ON UNMANNED AERIAL VEHICLES, Nevada, USA. **Anais...** IEEE, 2009. p.6.

ODELL, J. Objects and Agents: how do they differ? **Journal of ObjectOriented Programming**, Laboratoire d'Informatique Fondamentale de Lille, v.Volume 10, Issue: 1, p.1 – 9, 2000.

ODELL, J.; PARUNAK, H. V. D.; BAUER, B. Extending UML for Agents. **Computer and Information Science**, Citeseer, v.Volume 1001, Issue 6, p.1 – 15, 2000.

OMICINI, A.; RICCI, A.; VIROLI, M. Artifacts in the A&A meta-model for multi-agent systems. **Autonomous Agents and MultiAgent Systems**, Springer Netherlands, v.Volume 17, Issue 3, p.432 – 456, 2008.

OURSLAND, J. The Design and Implementation of a Quadrotor Flight Controller Using the QUEST Algorithm. In: MIDWEST INSTRUCTION AND COMPUTING SYMPOSIUM, Department of Mathematics and Computer Science. **Anais...** South Dakota School of Mines and Technology, 2010. p.11.

PADGHAM, L.; WINIKOFF, M. Prometheus: a pragmatic methodology for engineering intelligent agents. In: OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES & APPLICATIONS, Seattle, Washington. **Proceedings...** ACM Press, 2002. p.97 – 108.

PADGHAM, L.; WINIKOFF, M. **Developing Intelligent Agent Systems**: a practical guide. RMIT University, Melbourne: John Wiley and Sons, 2004. 230p.

PECHOUCEK, M. et al. Autonomous agents for air-traffic deconfliction. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, New York, NY, USA. **Anais...** ACM Press, 2006. p.1498 – 1505.

PECHOUCEK, M.; SIALAK, D. Agent Based Approach to Free Flight Planning, Control, and Simulation. **IEEE Intelligent Systems**, University of Southhampton, v.Volume 24, Issue 1, p.14 – 17, 2009.

PINHEIRO, C. A. P. **Veículos Aéreos Autonomos Não Tripulados para Monitoramento de Ambientes Desestruturados e Comunicação de Dados**. 2006. Dissertação (Mestrado em Ciência da Computação) — IME - Instituto Militar de Engenharia.

PLOTKIN, G. **A Structural Approach to Operational Semantics**. University of Aarhus: University of Aarhus Press, 1981.

RAO, A. S. AgentSpeak(L): bdi agents speak out in a logical computable language. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 7., Secaucus, NJ, USA. **Proceedings...** Springer-Verlag, 1996. p.42 – 55.

RAO, A. S.; GEORGEFF, M. P. Modeling rational agents within a BDI-architecture. In: KNOWLEDGE REPRESENTATION AND REASONING. **Anais...** Morgan Kaufmann Publishers, 1991. p.473 – 484.

RAO, A. S.; GEORGEFF, M. P. A model-theoretic approach to the verification of situated reasoning systems. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, Chambéry, France. **Anais...** [S.l.: s.n.], 1993. p.318 – 324.

RIBEIRO, L. R.; OLIVEIRA, N. M. F. UAV Autopilot Controllers Test Platform Using Matlab / Simulink and X-Plane. **Frontiers in Education Conference**, IEEE, FIE, p.6, 2010.

RICCI, A. et al. Environment Programming in CArtAgO. **Multi-Agent Programming**, Springer-Verlag, v.Part 2, p.259 – 288, 2009.

RICCI, A.; VIROLI, M.; OMICINI, A. The A&A Programming Model and Technology for Developing Agent Environments in MAS. In: PROGRAMMING MULTI-AGENT SYSTEMS, 5., Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2008. p.89 – 106.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence**: a modern approach. Englewood Cliffs, New Jersey: Alan Apt, 1995.

SCERRI, P. et al. Transitioning multiagent technology to UAV applications. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 7., Carnegie Mellon University. **Proceedings...** International Foundation for Autonomous Agents and Multiagent Systems, 2008. p.89 – 96.

SCHOELLIG, A.; AUGUGLIARO, F.; D'ANDREA, R. A Platform for Dance Performances with Multiple Quadrocopters. **International Conference on Intelligent Robots and Systems**, Workshop on Robots and Musical Expressions, v.July Issue, p.1 – 8, 2010.

SHINGLEDECKER, R.; ANDREWS, J.; NEGUS, C. **The Official Damn Small Linux Book**. Prentice Hall: Prentice Hall, 2008. 448p.

SHOHAM, Y. Agent-oriented programming. **Artificial Intelligence**, Elsevier Science Publishers, v.Volume 60, p.51 – 92, 1993.

SISLAK, D. et al. AGENTFLY: towards multi-agent technology in free flight air traffic control. **Defense Industry Applications of Autonomous Agents and Multi-Agent Systems**, Birkhauser Verlag, p.73 – 97, 2008.

SMULLYAN, R. Logicians Who Reason About Themselves. In: CONFERENCE ON THEORETICAL ASPECTS OF REASONING ABOUT KNOWLEDGE, 1986., San Francisco, CA. **Proceedings...** Morgan Kaufmann Publishers Inc, 1986. p.341 – 352.

SVAHNBERG, M.; DAVIDSSON, P.; GRAHN, H. **ABOS - an Agent-Based Operating System**. Department of Software Engineering and Computer Science: University of Karlskrona, 1998.

VANEK, O. et al. Using Multi-Agent Simulation to Improve the Security of Maritime Transit. In: INTERNATIONAL WORKSHOP ON MULTI-AGENT-BASED SIMULATION, 12., Gerstner Lab. **Proceedings...** Springer Verlag, 2011.

VOKRíNEK, J.; KOMENDA, A.; PECHOUCEK, M. Agents Towards Vehicle Routing Problems. In: NINTH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS, Gerstner Lab. **Proceedings...** AAMAS, 2010. p.8.

WALLIS, P. et al. The Automated Wingman - Using JACK Intelligent Agents for Unmanned Autonomous. In: AEROSPACE CONFERENCE PROCEEDINGS, Carlton, Vic., Australia. **Proceedings...** IEEE, 2002. p.2615 – 2622.

WIERINGA, R. J.; MEYER, J. J. C. Deontic logic: a concise overview. **Deontic logic in computer science**, John Wiley and Sons Ltd, p.3 – 16, 1993.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. United Kingdom: John Wiley and Sons Ltd, 2002.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. United Kingdom: John Wiley and Sons Ltd, 2009. 461p.

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The Gaia Methodology For Agent-Oriented Analysis And Design. **Journal of Autonomous Agents and Multi-Agent Systems**, London, UK, v.3, p.285 – 312, 2000.

WRIGHT, G. H. von. Deontic Logic. **Oxford Journals**, Mind Association, v.Volume 60, Issue 237, p.1 – 15, 1951.