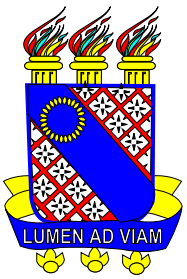

UNIVERSIDADE ESTADUAL DO CEARÁ

MARÇAL JOSÉ DE OLIVEIRA MORAIS II

MAS-COMMONKADS+: UMA EXTENSÃO À METODOLOGIA MAS-COMMONKADS PARA SUPORTE AO PROJETO DETALHADO DE SISTEMAS MULTIAGENTES RACIONAIS

FORTALEZA, CEARÁ

2010

MARÇAL JOSÉ DE OLIVEIRA MORAIS II

MAS-COMMONKADS+: UMA EXTENSÃO À METODOLOGIA MAS-COMMONKADS PARA SUPORTE AO PROJETO DETALHADO DE SISTEMAS MULTIAGENTES RACIONAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Gustavo Augusto Campos de Lima

Co-Orientadores: Francisco Milton Mendes Neto

FORTALEZA, CEARÁ

2010

A000z Morais II, Marçal José de Oliveira.

MAS-CommonKADS+: Uma Extensão à Metodologia MAS-CommonKADS para Suporte ao Projeto Detalhado de Sistemas Multiagentes Racionais / Marçal José de Oliveira Morais II. – Fortaleza, 2010.

120p.;il.

Orientador: Prof. Dr. Gustavo Augusto Campos de Lima

Dissertação (Mestrado Acadêmico em Ciência da Computação) - Universidade Estadual do Ceará, Programa de Pós Graduação em Ciência da Computação.

1. Agentes racionais 2. Engenharia de software 3. Sistemas multiagentes I. Universidade Estadual do Ceará, Programa de Pós Graduação em Ciência da Computação.

CDD:000.0

MARÇAL JOSÉ DE OLIVEIRA MORAIS II

MAS-COMMONKADS+: UMA EXTENSÃO À METODOLOGIA MAS-COMMONKADS PARA SUPORTE AO PROJETO DETALHADO DE SISTEMAS MULTIAGENTES RACIONAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Gustavo Augusto Campos de Lima
Universidade Estadual do Ceará - UECE
Orientador

Prof. Dr. Francisco Milton Mendes Neto
Univ. Fed. Rural do Semi-Árido - UFERSA
Co-orientador

Prof. Dr. Mariela Inés Cortés
Universidade Estadual do Ceará - UECE

Prof. Dr. Pedro Fernandes Ribeiro Neto
Univ. do Estado do Rio Grande do Norte - UERN

Agradecimentos

A minha mãe que heroicamente ultrapassou todas as adversidades que a vida lhe impôs, dentre as quais ser pai e mãe ao mesmo tempo, tarefa que soube fazer de maneira tão sublime que me fez chegar até aqui.

Aos meus avós que sempre me apoiaram e me deram todo carinho e amor que podiam dar a uma pessoa.

A Patrícia pela amizade sincera e por está sempre presente me apoiando e torcendo por mim.

Ao meu orientador, Prof. Dr. Gustavo Augusto Campos de Lima e ao meu co-orientador Prof. Dr. Francisco Milton Mendes Neto, pelos conhecimentos que comigo compartilharam, pela paciência e apoio durante todo o tempo em que estive envolvido no trabalho de pesquisa e redação dessa dissertação.

A Patrícia Ligia, secretária do departamento de ciência da computação da UERN, pela amizade e os momentos de descontração durante o periodo que fui professor substituto.

Ao IFRN pelas vezes que tive de me ausentar durante essa reta final e principalmente a UFERSA que me liberou durante o periodo em que cursei as disciplinas do mestrado.

Aos meus mestres de Taekwondo Paulo e José Afonso (Gaúcho) pelos ensinamentos e pela amizade.

Aos meus amigos José Ismaelton (Belém), João Diniz (Gordo), Julihermes, Lenilton e Igor.

E finalmente mas não menos importante a Deus em todas as suas formas de existência pois sem sua ajuda não conseguiria completar mais essa etapa de minha vida.

"Mesmo desacreditado e ignorado por todos, não posso desistir, pois para mim, vencer é nunca desistir."
Albert Einstein

Resumo

Desenvolver aplicações de softwares industriais é uma tarefa bastante complexa, pois, muitas vezes envolve grandes desafios como sistemas embutidos, sistemas distribuídos, tempo real e multitarefas. Com isso os Sistemas Multiagentes (SMAs) vêm ganhando atenção na área de desenvolvimento de software, pois os engenheiros de software vêm reconhecendo a necessidade da utilização de inteligência em seus softwares para auxiliar no controle das atividades. Devido ao crescente interesse na tecnologia de agentes no contexto da engenharia de software, diversas metodologias foram criadas para suportar o desenvolvimento de sistemas orientados a agentes. A MAS-CommonKADS é uma metodologia orientada a agentes vinda da engenharia do conhecimento que guia os projetistas durante a modelagem de um SMA, porém apesar de possibilitar a modelagem de uma grande quantidade de características dos SMAs ela, assim como a maioria das metodologias, não possibilita a modelagem interna de agentes racionais. Este tipo de modelagem é importante devido à necessidade da representação da estrutura interna de um agente para a sua posterior implementação. Neste trabalho, foi desenvolvido uma extensão para o desenvolvimento de SMA inteligentes (racionais) que engloba todas as características atuais de agentes racionais, baseados em uma integração proposta nesta dissertação que abrange a visão de agentes racionais de dois renomados pesquisadores da área de inteligência artificial, empregando diagramas que oferecem a melhor formalização para os modelos, incluindo a noção de objetos e uma ferramenta que possibilite a modelagem do sistema de uma forma clara e simples.

Palavras-Chave: Agentes racionais, Engenharia de software, Sistemas multiagentes

Abstract

Developing applications software industry is a very complex task, because often involves major challenges such as embedded systems, distributed systems, real time and multitasking. With that Multiagent Systems (MAS) have been gaining attention in the field of software development because software engineers are recognizing the need to use intelligence in their software to assist in controlling the activities. Due to the growing interest in agent technology in the context of software engineering, many methodologies were created to support the development of agent-oriented systems. The MAS-CommonKADS methodology is focused to agents coming from engineering knowledge that guides the designers during the design of an SMA, but despite allowing the modeling of a large number of characteristics of SMAs It, like most methods, doesn't make possible internal modeling of rational agents. This type of modeling is important because of the need for representation of the internal structure of an agent for its subsequent implementation. In this work, was developed an extension to the development of SMA intelligent (rational) that encompasses all the current features of rational agents, based on an integration proposed in this dissertation that covers the vision of rational agents in two renowned researchers in the field of artificial intelligence using diagrams that provide the best formalization for models, including the notion of objects and a tool that enables the modeling system in a clear and simple way.

Keywords: Rational Agents, Software Engineering, Multiagent Systems

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Na última década, a Tecnologia da Informação (TI) tem desempenhado um papel cada vez mais importante no cotidiano das pessoas, influenciando o desenvolvimento de software em diversos campos de aplicação, desde pequenos softwares pessoais a grandes sistemas industriais. Desenvolver aplicações de softwares industriais é uma tarefa bastante complexa, pois muitas vezes envolve grandes desafios, como sistemas embutidos, sistemas distribuídos, sistemas de tempo real e sistemas multitarefas. Estas situações têm motivado os engenheiros de software a obterem um melhor entendimento das características de um software complexo, reconhecendo a necessidade da utilização de inteligência em seus softwares para auxiliar no controle das atividades (LIN et al., 2007). Dentro desta proposta, a tecnologia de agentes inteligentes oferece vários aspectos que auxiliam o desenvolvimento de software complexo.

Agentes são entidades especiais de software, que podem apresentar características como mobilidade, comunicação, reatividade, autonomia e pró-atividade. Estas entidades podem ainda apresentar características inteligentes como, por exemplo, as capacidades de raciocínio e de aprendizagem, recebendo, neste caso, a denominação de agentes inteligentes. Atualmente, os agentes inteligentes são concebidos e avaliados em seus ambientes de execução por meio de uma medida de avaliação objetiva, estabelecida a priori pelo projetista do agente, sendo assim denominados agentes racionais (RUSSEL; NORVIG, 1995).

Um sistema que possua diversos agentes executando tarefas e procurando realizar objetivos comuns é denominado sistema multiagente (SMA). O desenvolvimento de sistemas de software no formato de SMA tem crescido como um novo paradigma de computação, abrangendo uma grande diversidade de aplicações, variando desde pequenos sistemas até sistemas mais complexos e de missão crítica como, por exemplo, controle de tráfego aéreo (GORODETSKY et al., 2005).

A Engenharia de Software Orientada a Agentes (ESOA) tem como principal objetivo criar metodologias e ferramentas que possibilitem o desenvolvimento e a manutenção de software baseado em agentes.

O estudo de metodologias para análise e projeto de SMA ainda está em um estado embrionário (AGUILAR; VIZCARRONDO; PEROZO, 2008). Atualmente existem diversas metodologias para modelagem de SMA, que são, geralmente, extensões de metodologias orientadas a objetos (ex. GAIA (STERLING; TAVETER, 2009), MASE (DELOACH; KUMAR, 2005), Prometheus (ALHASHEL; BALACHANDRAN; SHARMA, 2008)) ou da engenharia de conhecimento (ex. MAS-CommonKADS (SCHREIBER et al., 2000), CoMoMAS (TRAN; LOW, 2008)).

As metodologias que são extensões de metodologias orientadas a objetos apresentam um processo semelhante à orientação a objetos o que facilita na adaptação dos projetistas a um novo paradigma. Contudo estas metodologias representam o SMA dando uma maior atenção à interação entre os agentes, não conseguindo representar de forma adequada o conhecimento dos agentes assim como a aquisição deste conhecimento.

A engenharia do conhecimento auxilia na concepção de sistemas especialistas, que podem ser considerados sistemas de único agente. Este tipo de metodologia tem um foco no conhecimento dos agentes e como ele realizará seus objetivos. Sendo assim, seria interessante que uma metodologia unificasse os poderosos modelos de interação da orientação a objetos e a capacidade de representação de conhecimento dos agentes da engenharia do conhecimento.

As metodologias que proveem da engenharia de conhecimento, em sua maioria, são extensões da CommonKADS (SCHREIBER et al., 2000), que é a principal metodologia da engenharia de conhecimento. Esta abrange aspectos de um projeto de desenvolvimento de um Sistema Baseado em Conhecimento, incluindo gerenciamento de projetos, análise organizacional, aquisição de conhecimento, modelagem conceitual, interação com o usuário, integração de sistemas e implementação computacional. Essas metodologias buscam aproveitar todo o potencial que esta área proporciona para representação do conhecimento e incluem novas características como interação, coordenação e organização para permitir a modelagem de vários agentes.

A principal extensão para a modelagem de SMA baseada na engenharia do conhecimento é a MAS-CommonKADS. Ela estende a metodologia CommonKADS seguindo o mesmo modelo de ciclo de vida em espiral para o desenvolvimento de SMAs e sua abordagem de gestão de projetos dirigido por riscos. Ela engloba as seguintes fases: Conceituação, Análise, Codificação, Integração e Operação e Manutenção. A MAS-CommonKADS é uma metodologia muito interessante pois permite a integração entre engenharia de conhecimento e técnicas da orientação a objetos para a modelagem de SMA. Em seus modelos, ela utiliza *templates*

textuais que detalham todas as caracteristicas dos agentes e, através de diagramas de sequencia de mensagens e outras técnicas, é possivel demonstrar as interações entre agentes. Outro ponto importante desta metodologia é sua capacidade de utilização de ontologias para a representação do conhecimento e a troca deste com outros agentes.

1.1 Problemática

Um grande numero de metodologias tem sido proposto para auxiliar no suporte à analise e ao projeto de SMA, porém a grande maioria das metodologias falha em uma ou mais das seguintes áreas (TRAN; LOW, 2008): (i) projeto interno do agente; (ii) projeto de interação; e (iii) modelagem da organização do SMA.

A MAS-CommonKADS, por se tratar de uma metodologia orientada a agentes vinda da engenharia do conhecimento, possui um grande potencial para a modelagem de agentes. No entanto, ela utiliza uma grande variedade de tecnologias para auxiliar na sua modelagem como, por exemplo: técnicas estruturadas (ex. Diagramas de Fluxo de Dados), técnicas de colaboração (ex. cartões de Classes-Responsabilidade-Colaboração - CRC), diagramas da UML, *templates* herdados do CommonKADS e um grande número de planilhas. Tudo isso dificulta a referência cruzada dos mesmos e a rastreabilidade dos requisitos. Esta grande quantidade de tecnologias também dificulta o processo de aprendizado de novos projetistas.

Essas metodologias, mesmo tendo um dos seus focos principais no conhecimento do agente, não são capazes de representar todos os comportamentos do projeto interno do agente, dificultando a representação de forma clara de alguns aspectos importantes dos SMA como, por exemplo, a noção de agentes racionais e, principalmente, as estruturas de agentes reativos, reativos com estado interno, orientado por objetivos, orientado por utilidade e com aprendizado. Um outro problema existente é que, mesmo sendo demonstrado o poder do paradigma orientado a agentes para a concepção de sistemas, nem todas as entidades de um sistema precisam possuir autonomia. Dentro deste contexto, é importante que as metodologias também sejam capazes de modelar objetos. Além dos problemas citados, existe uma dificuldade para modelagem de metodologias que vem da engenharia do conhecimento, pois a grande quantidade de tecnologias utilizadas na sua modelagem faz com que a construção de uma ferramenta seja uma tarefa bastante complexa.

1.2 Objetivos

Tendo em vista a problemática apresentada, este trabalho tem como objetivo principal propor uma extensão para o desenvolvimento de SMA inteligentes (racionais) que englobe todas as características atuais de agentes racionais. A extensão proposta nesta dissertação engloba a visão de agentes racionais propostas por Russel e Norvig (1995) e a extensão proposta por Wooldridge (2002), empregando diagramas que oferecem uma melhor formalização para os modelos incluindo a noção de objetos e uma ferramenta que possibilite a modelagem do sistema de uma forma clara e simples. Por SMA racional entenda-se um sistema multiagente capaz de selecionar ações para serem executadas em seu ambiente por seus atuadores, que maximize a medida de avaliação de desempenho preestabelecida pelo projetista do sistema. Esta medida de avaliação deve levar em consideração as informações perceptivas que chegam do ambiente pelos seus sensores e todo e qualquer conhecimento embutido previamente no sistema e/ou aprendido com o decorrer de sua experiência.

Mais especificamente, este trabalho tem como objetivos:

- Padronizar os modelos da metodologia através de uma linguagem de modelagem escolhida nesta dissertação;
- Adicionar novas características aos modelos de forma que possamos representar as técnicas de Inteligência Artificial (IA), como lógica fuzzy, sistemas baseados em regras, redes neurais, etc;
- Possibilitar a modelagem de agentes e objetos em uma única metodologia;
- Possibilitar a modelagem da metodologia através de uma ferramenta case (SUN; THANGARAJAH; PADGHAM, 2010).

1.3 Estrutura da Dissertação

Esta dissertação foi organizada em quatro capítulos. O Capítulo 2 descreve o embasamento teórico da área de sistemas multiagentes demonstrando características da engenharia de software orientada a agentes e as principais metodologias existentes, assim como as principais linguagens de modelagem que usam a abstração de agentes inteligentes. Neste capítulo também é apresentado um estudo comparativo sobre as diferentes metodologias e linguagens de modelagem de SMAs, demonstrando suas limitações para definição do comportamento interno

do agente. O Capítulo 3 descreve a extensão proposta nesse trabalho, à linguagem de modelagem AML e a metodologia MAS-CommonKADS para a realização dos objetivos propostos. Também é apresentada uma ferramenta de suporte às extensões propostas. O Capítulo 4 tem como objetivo demonstrar a validação dos modelos propostos para as arquiteturas de agentes racionais, demonstrando o seu uso para problemas que utilizam regras, redes neurais, buscas e lógica fuzzy. Neste capítulo também é demonstrado um pequeno exemplo de uso da metodologia após as modificações realizadas; por fim o Capítulo 5 traz nossas conclusões, demonstrando os resultados alcançados, as limitações e os trabalhos futuros a serem realizados.

2 Referencial Teórico

Nesse capítulo, inicialmente será apresentado o embasamento teórico da área de sistemas multiagente. Em seguida serão descritas características da engenharia de software orientada a agentes, demonstrando algumas das principais metodologias existentes e as principais linguagens de modelagem que usam a abstração de agentes inteligentes. Além disso, será apresentado um estudo comparativo sobre as diferentes metodologias e linguagens de modelagem de SMA, demonstrando suas limitações para definição do comportamento interno do agente.

2.1 Agentes e Sistemas Multiagentes

Não existe um consenso na literatura sobre uma definição do que são agentes, porém Wooldridge (1999) define um agente como sendo um sistema computacional situado em um ambiente e que é capaz de ações autônomas neste ambiente a fim de alcançar seus objetivos. Russel e Norvig (1995) definem a noção de agente racional como uma entidade de software autônoma que percebe seu ambiente de atuação por meio de sensores, processa essas informações e conhecimentos, e atua no ambiente por meio de atuadores visando realizar algum objetivo, conforme estabelecido em uma medida de avaliação de desempenho pré-especificada pelo projetista do agente. A Figura 2.1 ilustra esta definição.

Algumas propriedades devem ser observadas em um agente computacional. São elas (FRANKLIN; GRAESSE, 1996):

- Autonomia – os agentes devem executar a maior parte de suas ações sem interferência direta de agentes humanos ou de outros agentes computacionais, possuindo controle total sobre suas ações e estado interno;
- Habilidade social – os agentes devem poder interagir com outros agentes (humanos ou computacionais), para completarem a resolução de seus problemas, ou ainda para auxiliarem outros agentes;

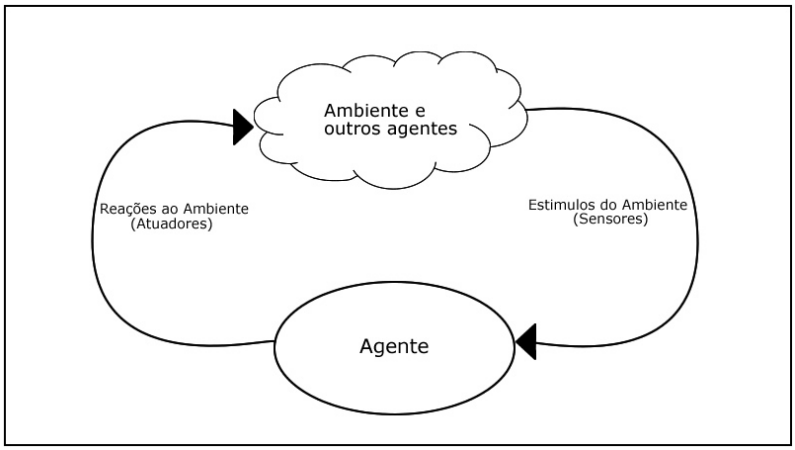

Figura 2.1: Visão de um agente padrão

- Reatividade – os agentes devem perceber e reagir a alterações nos ambientes em que estiverem inseridos;
- Pró-atividade – os agentes, além de atuarem em resposta às alterações ocorridas em seu ambiente, devem apresentar um comportamento orientado a objetivos, tomando iniciativas quando considerarem apropriado;
- Adaptação – os agentes devem poder mudar o seu comportamento devido a uma experiência anterior;
- Mobilidade – os agentes devem poder se movimentar de uma máquina para outra.
- Persistência ou continuidade temporal: capacidade do agente de manter um estado interno conciso através do tempo, isto é, o agente estar continuamente executando um processo.

Segundo Cafarate (2008), para ser um agente, nem todas as propriedades citadas acima precisam estar presentes, mas quanto mais propriedades o agente possuir, em geral, maior será seu grau de inteligência, que também definirá a qual classificação o agente pertence, que pode ser dividida entre a noção forte e fraca de agência. Na noção fraca de agência, os agentes têm vontade própria (autonomia), sendo capazes de interagir uns com os outros (habilidade social), responder a estímulos (reatividade) e tomar iniciativa (pró-atividade). Na noção forte de agência, as noções fraca de agência são preservadas com a adição de que os agentes: (i) podem se mover de um ambiente para outro (mobilidade); (ii) são confiáveis (veracidade); (iii) fazem o que é dito para ser feito (benevolência); (iv) e operam de maneira ótima para atingir suas metas (racionalidade) (TVEIT, 2001).

SMA podem ser utilizados para a representação de uma sociedade de agentes. Segundo Jennings (1996), SMA refere-se à sub-área da IA que investiga o comportamento de um conjunto de agentes autônomos objetivando a solução de um problema que está além da capacidade de um único agente.

Existem diversos motivos para utilizar SMA, dentre eles estão (BITTENCOURT, 1998):

- Melhorar a adaptabilidade, a confiabilidade e a autonomia do sistema;
- Reduzir os custos de desenvolvimento e manutenção;
- Aumentar a eficiência e a velocidade;
- Permitir a integração de sistemas inteligentes existentes de maneira a aumentar a capacidade de processamento e, principalmente, a eficiência na solução de problemas.

Jung e Liu (2003) citam algumas características importantes que os SMA devem possuir, como coordenação e organização. A coordenação nos SMA é necessária, pois os agentes agem de forma autônoma e, devido a essa propriedade, existe a possibilidade deles entrarem em conflito. Esses conflitos podem ser de dois tipos: internos e externos. O primeiro acontece quando um agente individual toma uma decisão que pode não ser precisa e o segundo quando um ou vários agentes divergem entre si. A organização de um SMA, segundo Rezende (2003), pode ser vista simplificadamente como um conjunto de restrições adotadas por um grupo de agentes para que possam atingir seus objetivos globais mais facilmente.

SMA também devem prover protocolos para a interação e comunicação dos agentes. Segundo (WEISS, 2000), um protocolo de comunicação pode especificar os seguintes tipos de mensagens para troca entre agentes: (i) propor um curso de ação; (ii) aceitar um curso de ação; (iii) rejeitar um curso de ação; (iv) cancelar um curso de ação; (v) discordar de um curso de ação proposto; e (iv) contrapropor um curso de ação.

O agente tem, geralmente, um repertório de ações disponíveis capazes de modificar o seu ambiente. Estas ações não são executadas em todas as situações. Além disso, por ter em si pré-condições associadas, apenas as situações possíveis destas associações ocorrem. O problema surge da decisão de quais ações precisam ser executadas para satisfazer, da melhor forma, os objetivos buscados pelo agente. Disso são introduzidas as arquiteturas de agentes, confirmando o seu uso como sistemas de tomada de decisão embutidos em um ambiente.

Os dois principais autores que trataram das arquiteturas de agentes foram Wooldridge (2002) e Russel e Norvig (1995).

Wooldridge (2002) define duas arquiteturas abstratas de agente que são uma extensão do agente padrão demonstrado na Figura 2.1. Na sua primeira arquitetura, ele decompõe o agente em duas funções: a função ver e a função ação. A primeira função é responsável por tratar as informações perceptivas capturadas pelo sensor enquanto a segunda função é responsável por definir a ação que será realizada no ambiente. A Figura 2.2 demonstra a primeira decomposição do agente.

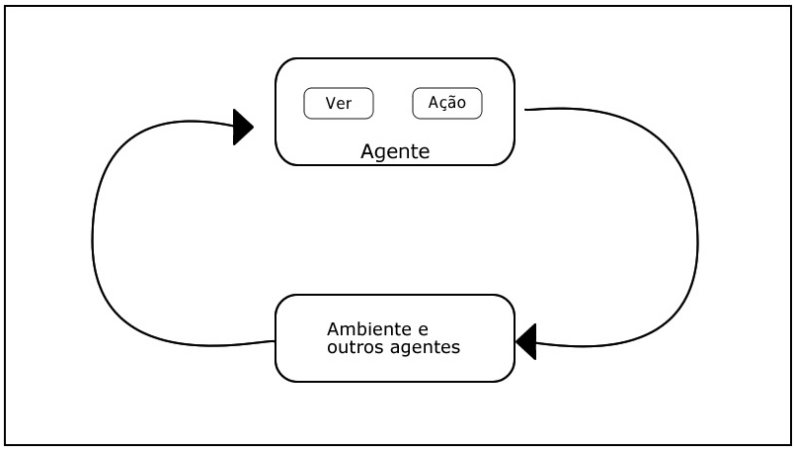

Figura 2.2: Primeira decomposição do agente padrão

Em uma segunda arquitetura abstrata, Wooldridge (2002) faz um refinamento da função ação e engloba uma função próximo, que é responsável por atualizar o estado interno de um agente. Essa arquitetura é responsável por definir agentes capazes de guardar informações do mundo em que está inserido. A Figura 2.3 demonstra o segundo refinamento proposto.

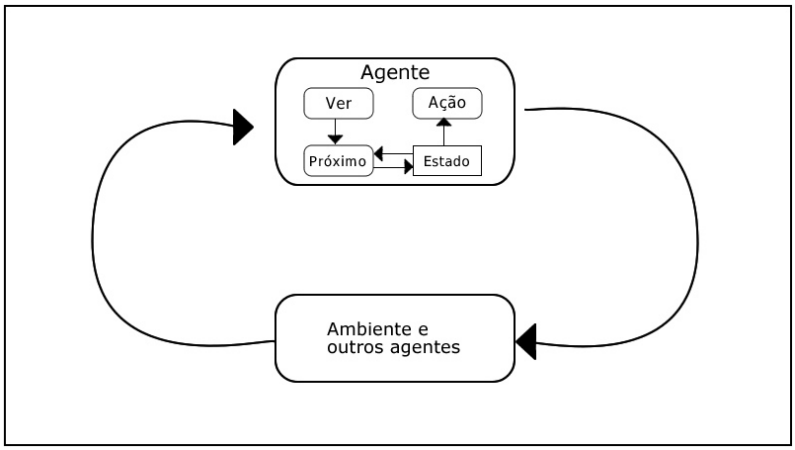

Figura 2.3: Segunda decomposição do agente padrão

Enquanto Wooldridge (2002) define a arquitetura de dois agentes através de refinamentos de sua função ação, Russel e Norvig (1995) descreve quatro arquiteturas de agentes que levam em consideração não apenas como realizar suas ações mas também as informações de

como o agente irá evoluir e como aquelas ações irão modificar o ambiente, definindo assim um conceito mais racional do agente. As arquiteturas propostas por Russel e Norvig (1995) são:

- Agentes reativos: são os agentes mais simples que percebem o ambiente e agem baseados na suas percepções atuais sem considerar o histórico de suas percepções. Geralmente são providos com uma base de conhecimento formada por regras Se-Então, sendo que seu comportamento está totalmente codificado nessas regras;
- Agentes baseados em modelos: são agentes que guardam estados do ambiente e que sabem como o ambiente evolui em função do tempo e em função de suas ações. Portanto as regras de produção podem se basear tanto na sequência de percepção quanto no estado do ambiente para decidir o que agente deve fazer;
- Agentes baseados em objetivos: este tipo de agente toma suas decisões sempre levando em consideração a tentativa de alcançar seus objetivos;
- Agentes baseados em utilidade: pode acontecer de existirem várias sequências distintas de ações que levem o agente a atingir seu objetivo. Então esse tipo de agente toma suas decisões baseado na função de utilidade que melhor se adeque para a resolução do seu objetivo;
- Agentes com aprendizado: são agentes que possuem um componente de aprendizado em conjunto com uma das arquiteturas citadas anteriormente.

2.2 Engenharia de Software Orientada a Agentes

Segundo Castro, Alencar e Silva (2006), para gerenciar com sucesso a complexidade associada ao desenvolvimento, manutenção e distribuição de SMA, técnicas e ferramentas de engenharia de software são necessárias ao longo do ciclo de vida do software. A engenharia de software orientada a agentes tem como objetivo prover técnicas e ferramentas para o desenvolvimento de SMA.

A fase de modelagem conceitual é um passo cada vez mais importante no desenvolvimento de software. Um software bem modelado se torna fácil de implementar, de realizar manutenções e de ser compreendido (NUNES et al., 2009). A modelagem e a especificação de software são atividades fundamentais para fazer do desenvolvimento de software uma atividade de engenharia. O uso de modelos nas diversas áreas da engenharia sempre foi fundamental

para o desenvolvimento de bons produtos. Os modelos permitem uma visão antecipada dos produtos a serem desenvolvidos e permitem análise e avaliação prévias, antes mesmo de serem construídos (CARVALHO, 2005).

A exemplo do paradigma Orientado a Objetos (OO), é necessário que o paradigma Orientado a Agentes (OA) seja também suportado por técnicas e métodos que garantam a qualidade do processo de produção do software. Porém, segundo os estudos de Bastos e Oliveira (2000a) *apud* Bassani et al. (2006), apesar da existência de diversos trabalhos aplicando a abordagem multiagentes, ainda não existe um consenso sobre metodologias para modelar este tipo de sistema.

O conceito de agente apresenta muitas semelhanças em relação ao conceito de objeto, mas, apesar dessas características comuns, eles possuem distinções significativas, pois um agente possui uma postura mais ativa, ou seja, é uma entidade do mundo que possui um ciclo de vida e que durante esse ciclo estará adquirindo continuamente informações do mundo. Por outro lado, um objeto possui uma postura mais passiva diante do mundo, ou seja, apenas recebe mensagens do mundo, efetuando um comportamento em resposta a elas.

Segundo Castro, Alencar e Silva (2006), a construção de SMA não é fácil, pois se tem todos os problemas dos sistemas distribuídos e concorrentes tradicionais, mais as dificuldades adicionais que surgem dos requisitos de flexibilidade e interações sofisticadas. Qualquer metodologia para a Engenharia de Software Orientada a Agentes (ESOA) deve prover abstrações adequadas e ferramentas para modelar não só as tarefas individuais, mas também as tarefas sociais dos agentes além do comportamento inteligente de cada agente. Nesse sentido, várias metodologias para a modelagem de SMA têm sido propostas nos últimos anos como, por exemplo: Multiagent Systems Engineering (MaSE) (DELOACH; KUMAR, 2005), Gaia (ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2005), Tropos (GIORGINI et al., 2005), MAS – CommonKADS (IGLESIAS; GARIJO, 2005), ADELFE (ROUGEMAILLE et al., 2009), MESSAGE (GARIJO; GóMEZ-SANZ; MASSONET, 2005), INGENIAS (GOMEZ-SANZ et al., 2008), PASSI (NUNES et al., 2009) e Prometheus (PADGHAM; WINIKOFF, 2005).

2.2.1 ADELFE

A metodologia ADELFE[1] é especializada no desenvolvimento de sistemas multiagentes adaptativos (ROUGEMAILLE et al., 2009). A teoria dos SMA adaptativos postula que o

[1] ADELFE é um acrônimo francês para " Atelier de Développement de Logiciels à Fonctionnalité Emergente"

comportamento global dos SMA emerge do comportamento coletivo de diferentes agentes que os compõem. A ADELFE é uma metodologia baseada no RUP (Rational Unified Process), contendo modificações necessárias para o projeto de agentes adaptativos desde os requisitos até o projeto. Ela usa as notações da Unified Modeling Language (UML) e uma das extensões para agentes chamada Agent UML (AUML).

O processo de desenvolvimento é feito de forma iterativa e consiste de 4 (quatro) fases (TRAN; LOW, 2008): requisitos, análise, projeto e implementação. A fase de requisitos é dividida em requisitos iniciais e finais. Os requisitos iniciais são capturados conforme as atividades apresentadas na tabela 2.2.1 não ocorrendo nenhuma modificação ao processo padrão do RUP. Nos requisitos finais são incluídas atividades para reconhecer características do ambiente e as suas interações com as entidades passivas e ativas (uma entidade ativa é aquela que possui comportamento autônomo). Na atividade de caracterizar o ambiente, são definidas as entidades que interagem com o ambiente de forma ativa ou passiva e as características do ambiente (como dinâmico, acessível, não determinístico e continuo).

Requisitos Preliminares	Requisitos Finais
Definir requisitos de usuário	Características do ambiente
Validar requisitos de usuário	Determinar entidades
Definir requisitos consensuais	Definir contexto
Estabelecer palavras chaves	Caracterizar ambiente
Extrair Limites e Restrições	Determinar casos de uso
	Desenvolver casos de uso
	Identificar falhas de cooperação
	Elaborar diagramas de seqüência
	Elaborar protótipos da interface com o usuário
	Validar Protótipos

Tabela 2.1: Fase de requisitos

A fase de análise é composta pelos seguintes passos: (i) análise do domínio, que analisa os casos de uso gerados e desenvolve os diagramas de classes preliminares; (ii) verificação da necessidade de uso de sistemas adaptativos, que auxilia o projetista a verificar se realmente devem ser utilizados agentes adaptativos; (iii) identificação dos agentes, que verifica quais entidades são adequadas para serem implementadas como agentes, gerando um diagrama de classes refinado; (iv) e estudo das interações, que demonstra a interação entre entidades através de diagrama de seqüência de mensagens e diagramas de comunicação.

O primeiro passo da fase de projeto é projetar a arquitetura detalhada do sistema

e do modelo multiagente. Deste passo são gerados pacotes, subsistemas, agentes, objetos e os relacionamentos entre eles. O segundo passo consiste em escolher uma linguagem de comunicação para os agentes. No passo seguinte é feito o projeto dos agentes, onde é descrita a estrutura interna do agente que é composta por técnicas (determinadas pelo esteriótipo <<*skill*>>), representações (determinadas pelo esteriótipo <<representations>>), aptidões (determinadas pelo esteriótipo <<aptitudes>>), e situações não cooperativas (determinadas pelo esteriótipo <<non-cooperative situations>>). Por fim, é realizada uma prototipação rápida para simulação do sistema.

A fase de implementação é onde o sistema é codificado, contudo a metodologia apenas cita esta fase sem dar maiores detalhes de como realizar a codificação.

A ADELFE possui algumas ferramentas que ajudam na modelagem de um sistema, como, por exemplo, a AMAS Adequancy Tool, a OpenTool e a Interactive Tool. A AMAS adequancy tool é uma ferramenta que auxilia os projetistas a decidir se é necessário o uso de adaptabilidade no sistema a ser desenvolvido. O estudo de adequabilidade é feito através de um questionário dividido em dois níveis: em nível global (sistema) e um nível local (componentes). Após o projetista responder esse questionário, a ferramenta de apoio à decisão irá responder se o sistema necessita ou não de adaptabilidade.

A OpenTool é uma ferramenta gráfica para a modelagem da UML. Ela foi modificada para suportar os novos esteriótipos e os protocolos de comunicação da AUML. A Interctive Tool é uma ferramenta que auxilia o desenvolvedor a aplicar o processo da metodologia.

Esta metodologia foi desenvolvida utilizando os diagramas UML 1.4. Com o surgimento da UML 2.0, muitas características da AUML foram incorporadas (ROUGEMAILLE et al., 2009). Como consequência, os perfis baseados na versão anterior da UML foram depreciados. Assim uma nova versão chamada ADELFE 2.0 foi desenvolvida, alterando principalmente a fase de projeto e implementação e criando uma linguagem de modelagem própria chamada AMAS-ML.

2.2.2 Gaia

Gaia foi a primeira metodologia proposta para guiar o processo de desenvolvimento de sistemas multiagentes (STERLING; TAVETER, 2009). Segundo (CASTRO; ALENCAR; SILVA, 2006), ela é aplicável a um grande conjunto de SMA, lidando com características de nível macro (sociedade) e de nível micro (agente) dos sistemas. Ela cobre desde as fases de

requisitos até o projeto suficientemente detalhado, a ponto de ser implementado diretamente. Esta metodologia foca no desenvolvimento de sistemas de tamanho médio a grande e situados em ambientes abertos e dinâmicos,

A metodologia Gaia usa o conceito de organizações computacionais no seu processo de desenvolvimento. Uma organização computacional define várias abstrações que são utilizadas nas fases de análise e projeto da metodologia, como o ambiente, os papéis e suas interações, as regras organizacionais e a estrutura organizacional. O processo de desenvolvimento de Gaia é dividido em quatro fases como mostra a Figura 2.4.

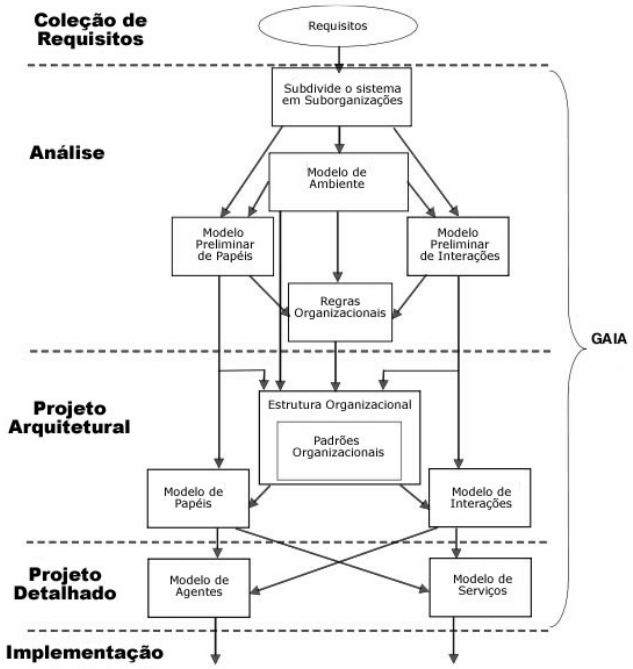

Figura 2.4: Processo de modelagem de Gaia

A fase de requisitos não é definida na metodologia podendo ser usada qualquer técnica da engenharia de requisitos para alcançar os requisitos iniciais do sistema. A fase de análise busca dar uma visão inicial do sistema dividindo-o em suborganizações que tem o mesmo objetivo. Em seguida deve-se definir o modelo de ambiente demonstrando todas as suas características para cada suborganização. O modelo preliminar de papéis[2] dá uma visão inicial dos papéis existentes em cada organização, demonstrando suas características e funcionalidades, enquanto o modelo preliminar de interações demonstra as interações entre esses papéis através de *templates* textuais. Por fim são definidas as regras organizacionais que definem as responsabilidades de cada papel. Elas são divididas em: vitais e de segurança. As vitais descrevem as

[2]Um papel é uma abstração que define as tarefas que um agente deve realizar dentro de uma organização.

situações que um agente deve buscar em determinadas condições ambientais. As de segurança são propriedades invariáveis que estabelecem que uma situação aceitável seja mantida durante todos os estados da execução.

A fase de projeto é divida em projeto arquitetural e projeto detalhado. O projeto arquitetural tem como objetivo capturar os artefatos gerados na fase de análise e transformá-los em abstrações de baixo nível. Nesta fase são definidos a estrutura organizacional e os modelos completos de papéis e interações. Na fase de projeto detalhado são definidos os seguintes modelos:

- Modelo de agente: define os agentes pertencentes ao sistema e realiza o mapeamento entre papéis e agentes, demonstrando quais papéis são de responsabilidade de cada agente;
- Modelo de serviços: identifica os serviços oferecidos por cada agente e as propriedades destes serviços (entradas, saídas, pré-condições e pós condições).

A fase de codificação não é definida na metodologia pois, de acordo com (HUANG; EL-DARZI; JIN, 2007), o projeto é suficientemente detalhado ao ponto de ser facilmente codificado com qualquer framework de programação de agentes.

MASDK (GORODETSKY et al., 2005) é uma ferramenta que suporta a modelagem da metodologia Gaia auxiliando o analista através de um ambiente visual de fácil utilização e que suporta todo o processo de desenvolvimento aplicado na metodologia. A Figura 2.5 demonstra a interface da ferramenta MASDK.

Rodriguez et al. (2009) propõe uma especificação para modelagem da fase de requisitos através da união de duas metodologias de captura de requisitos chamadas RETO (INSFRáN, 2003) e GBRAM (ANWER; IKRAM, 2006). Os autores criam o modelo de requisitos que possibilita a definição de objetivos, cenários, autores e restrições, que serão usados para gerar o modelo de ambiente, os papéis e as interações preliminares da fase de análise, oferecendo mais qualidade aos modelos gerados.

Huang, El-Darzi e Jin (2007) propõe uma extensão à metodologia utilizando AUML para a modelagem dos agentes. Basicamente é proposta uma melhor definição dos modelos de agentes, interação e serviços, utilizando os diagramas da AUML.

Juan e Sterling. (2002) propõe uma nova metodologia chamada ROADMAP que estende Gaia definindo novos modelos que buscam diminuir deficiências, como formalização dos

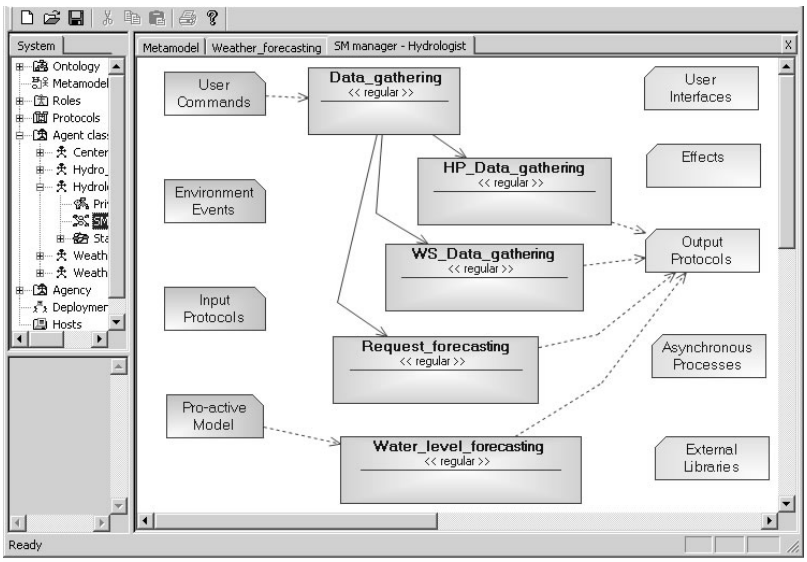

Figura 2.5: Ferramenta MASDK (GORODETSKY et al., 2005)

modelos de conhecimento e do ambiente, além de incorporar trocas dinâmicas nesse ambiente. Os modelos definidos em ROADMAP são:

- Modelo de casos de uso: este modelo tem como objetivo possibilitar uma definição eficiente dos requisitos do sistema;
- Modelo do ambiente: tem como objetivo descrever o ambiente que os agentes irão interagir;
- Modelo de conhecimento: propõe a descrição do conhecimento do domínio do sistema. Ele consiste de uma hierarquia de componentes de conhecimento e de uma descrição de cada componente;
- Modelo de papéis: tem a mesma função do modelo de papéis de Gaia, onde são definidas as características do papel;
- Modelo de protocolos e interação: o modelo de protocolos é o mesmo modelo de interação definido para Gaia enquanto o modelo de iteração utiliza AUML para definir as interações;
- Modelo de agentes e serviços: agentes e serviços são definidos da mesma forma da metodologia Gaia.

2.2.3 MAS-CommonKADS

É uma extensão do CommonKADS (SCHREIBER et al., 2000), uma metodologia da engenharia do conhecimento bastante conhecida e usada com ótimos resultados em uma grande quantidade de projetos. CommonKADS é a principal metodologia estruturada de suporte à engenharia do conhecimento. Segundo (SCHREIBER et al., 2000), são definidos vários modelos, sendo o Modelo de Experiência o principal modelo da metodologia CommonKADS. Este modelo tem como objetivo modelar o conhecimento empregado por um agente para realizar uma tarefa de resolução de problemas. Para a modelagem de SMA, Mas-CommonKADS inclui técnicas de orientação a objetos, projeto dirigido a responsabilidades e engenharia de protocolos como Message Sequence Charts (MSC) e Specification and Description Language (SDL).

O processo de desenvolvimento de software em MAS-CommonKADS é composto de cinco fases, que vão desde da contextualização do sistema até o desenvolvimento e manutenção. As fases do processo são:

- **Contextualização:** Nessa fase é dada uma visão preliminar do sistema através de diagramas de casos de uso e cenários. Nesta fase a metodologia propõe uma técnica chamada User-Environment-Responsibility (UER) que tem como objetivo capturar os requisitos através de três tipos de análise: (i) análise dos usuários, que identifica os atores do sistema e suas possíveis tarefas e funções; (ii) análise do ambiente, que identifica as interações do sistema com o ambiente; (iii) análise das responsabilidades ou metas, que identifica as metas que o sistema deve atingir;
- **Análise:** determina os requisitos d sistema partindo da contextualização do sistema;
- **Projeto:** define-se como os requisitos da fase de análise serão transformados em especificações para serem implementados. Nesta fase determinam-se as arquiteturas tanto da rede multiagentes como de cada agente;
- **Desenvolvimento e Teste:** Nesta fase cada agente é implementado e testado;
- **Operação e Manutenção:** o sistema é instalado e colocado em operação buscando sempre, em suas manutenções, corrigir erros não detectados nas fases anteriores e evoluir o sistema através de novas funcionalidades.

MAS-CommonKADS possui sete modelos que buscam descrever as características de um agente e seus comportamentos sociais no SMA. A Figura 2.6 demonstra os modelos da metodologia.

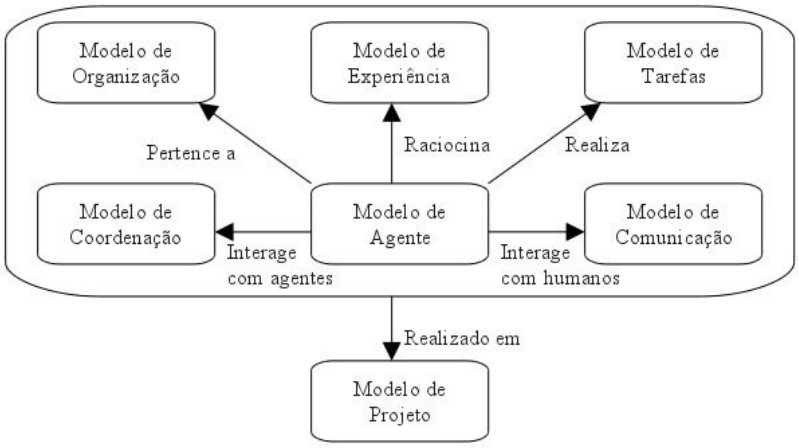

Figura 2.6: Modelos da metodologia MAS-CommonKADS (Adaptado de (IGLESIAS; GARIJO, 2005))

O modelo do agente é o principal modelo da metodologia. Neste modelo os agentes são identificados, através da análise dos casos de uso gerados na fase contextualização e descritos em forma de *templates* textuais que expressam características do agente, como papéis, serviços, metas, planos e conhecimento.

O modelo de tarefas descreve todas as atividades que devem ser realizadas para alcançar uma determinada meta. Todas as atividades são demonstradas em diagramas compostos de suas respectivas subtarefas, que serão realizadas para completar o objetivo, e através de *templates* textuais que descrevem suas entradas, saídas, precondições, pós condições e objetivo. A Figura 2.7 demonstra a notação gráfica utilizada para a decomposição das tarefas.

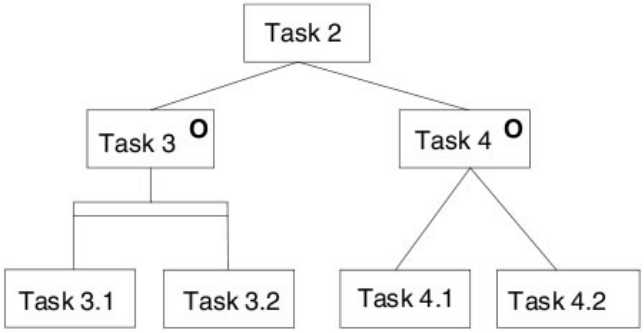

Figura 2.7: Decomposição de tarefas do modelo de tarefas (IGLESIAS; GARIJO, 2005)

O modelo de coordenação descreve a interação entre os agentes do sistema multiagente. Este modelo utiliza várias técnicas orientada a objetos para demonstrar a interação entre os agentes. As principais são diagramas de sequência de mensagem ou de comunicação para modelar a comunicação, entre agentes, e diagramas de transição de estados, para modelar o processamento de transições.

O modelo de comunicação descreve as interações entre um agente humano e um agente de software. Ele foca em fatores humanos para uma dada interação.

O modelo de conhecimento especifica o conhecimento que cada agente possui para atingir sua meta. Neste modelo, o conhecimento é descrito através: (i) do domínio de conhecimento (geralmente descrito através de uma ontologia); (ii) o conhecimento inferido, que descreve os passos para solucionar uma tarefa; (iii) o conhecimento da tarefa, que especifica a ordem de inferências para solucionar a tarefa; e (iv) dos métodos para solução de problemas que carregam as inferências.

O modelo de organização descreve a estrutura estática da organização dos agentes e seus relacionamentos. O modelo de projeto especifica três principais atividades: (i) o projeto da rede de agentes, que define a infraestrutura que envolve o conhecimento (servidores de ontologia), a coordenação (facilidades para gerência de grupos) e a rede de comunicação (páginas amarelas/brancas ou *agent name service*); (ii) o projeto do agente, que especifica a arquitetura de cada agente; e (iii) o projeto da plataforma que determina o software e o hardware necessários para o funcionamento do sistema.

Atualmente não existe nenhuma ferramenta que suporte a modelagem da metodologia MAS-Commonkads.

Em sua versão original, apresentada em (FERNÁNDEZ, 1998), MAS-CommonKADS contava com um modelo de comunicação para tratar possíveis interações entre agentes e seres humanos. Em publicações mais recentes, como (IGLESIAS; GARIJO, 2005), esse modelo não é mais descrito, deixando esse tipo de modelagem provavelmente para o modelo de coordenação. Contudo não está especificado exatamente onde deve ser tratado esse tipo de interação.

Uma nova metodologia que estende MAS-CommonKADS foi proposta por Aguilar, Vizcarrondo e Perozo (2008). Essa metodologia é chamada de *MultiAgent Systems in Automation* (MASINA). Ela modifica e adiciona novas características para a modelagem do sistema. Em MASINA é utilizado apenas cinco modelos para descrição do sistema multiagente: (i) o modelo de tarefas, que descreve as tarefas a serem realizadas no sistema. Nesse modelo algumas características do templates textuais são alteradas para melhor descrever as tarefas; (ii) o modelo de agentes, que adiciona novos atributos para descrever o sistema multiagente; (iii) o modelo de coordenação, que permite especificação das interações entre os agentes; (iv) o modelo de comunicação, que descreve as conversações dos agentes envolvidos; e (v) o modelo de inteligência, descreve todos os aspectos necessários para incorporar a noção de inteligência.

Neste modelo são descritos o domínio de conhecimento, as experiências, os mecanismos de aprendizado e os mecanismos de pensamento necessários para um agente completar seus objetivos.

2.2.4 Tropos

Tropos é uma metodologia orientada a agentes baseada no framework i* proposto por Yu (1995). Este framework propõe a modelagem baseada em conceitos como atores (podem ser agentes ou papéis), metas e dependências sociais entre atores para representar os requisitos, a arquitetura e o projeto detalhado do sistema.

O processo de desenvolvimento da Tropos consiste em quatro fases: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto Detalhado .

A fase de requisitos iniciais auxilia no entendimento do contexto organizacional onde o sistema irá funcionar. Nesta fase são descritos os atores do sistema e suas dependências para alcançar uma determinada meta. Nesta fase são gerados o modelo de dependência estratégica e o modelo de razão estratégica. O primeiro modelo fornece uma descrição dos relacionamentos de dependência entre os atores envolvidos. O segundo modelo apresenta uma descrição estratégica do processo, fornecendo uma análise de como as metas podem ser cumpridas através das contribuições dos demais atores (CAFARATE, 2008). A Figura 2.8 demonstra o modelo de dependência estratégica enquanto a Figura 2.9 demonstra o modelo de razão estratégica.

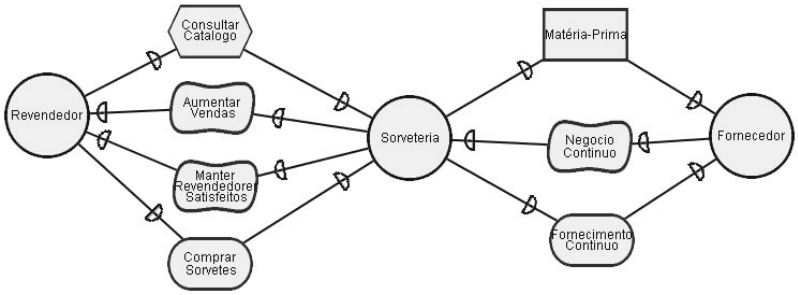

Figura 2.8: Modelo de dependência estratégica

A fase de requisitos finais estende os modelos gerados na fase de requisitos iniciais incluindo o sistema como um novo ator e descrevendo as novas dependências que irão dar origem aos requisitos funcionais e não funcionais do sistema.

O projeto arquitetural define a arquitetura global do sistema em termos de subsistemas, troca de dados e dependências. A fase de projeto detalhado especifica as estrutura interna e as interações do agente. As interações entre agentes são modeladas através de diagramas de

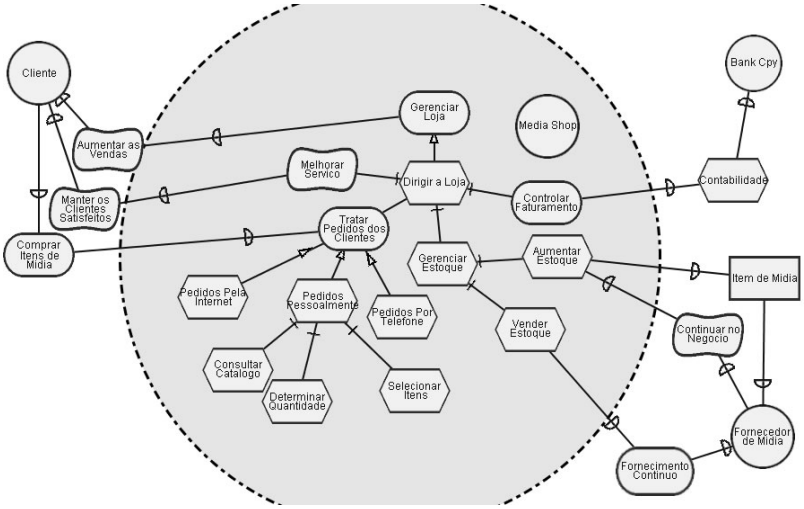

Figura 2.9: Modelo de razão estratégica

seqüência e de colaboração da UML, enquanto a estrutura interna do agente é modelada de acordo com a arquitetura BDI (Belief, Desire, Intention)(RAO; GEORGEFF, 1995). Nesta fase são gerados os diagramas de classes e diagramas de planos de cada agente para descrever a estrutura interna e os planos.

A codificação desta metodologia é baseada na definição do projeto detalhado. Para implementar o SMA, Tropos propõe utilizar a plataforma JACK (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002).

A metodologia Tropos utiliza algumas ferramentas que auxiliam o projetista no processo de desenvolvimento. A plataforma JACK Intelligent Agents é um ambiente de desenvolvimento orientado a agentes projetado para estender Java com o modelo de agente BDI. Os agentes codificados em JACK apresentam capacidades, eventos e planos. Outra ferramenta utilizada é a TAOM4E, que é um plugin para o ambiente eclipse que possibilita toda a modelagem da metodologia e gera códigos de agentes para o framework JADEX. A Figura 2.10 ilustra a ferramenta TAOM4E.

2.2.5 MaSE - Multi-agent Systems Engineering

Multi-agent Systems Engineering (MaSE) é uma metodologia para a análise e projeto de sistemas multiagentes originalmente desenvolvida para projetar sistemas multiagentes fechados, heterogêneos e de propósito geral (DELOACH; KUMAR, 2005). MASE trata agentes como especializações dos modelos de objetos aplicando várias técnicas do paradigma orientado a objetos para especificação e projeto do sistema multiagente.

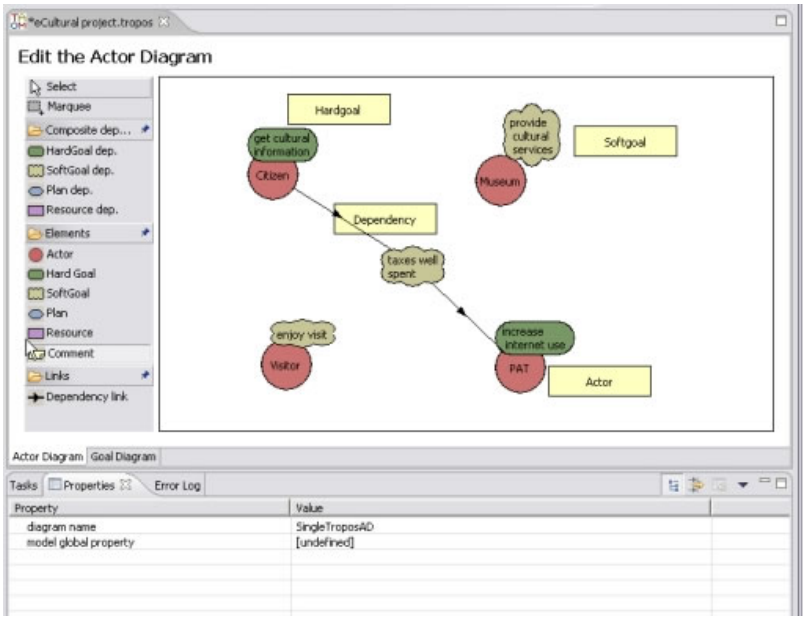

Figura 2.10: Ferramenta TAOM4E (TAOM4E, 2010)

O processo de desenvolvimento de MASE é composto das fases de análise e projeto, como pode ser visualizado na Figura 2.11.

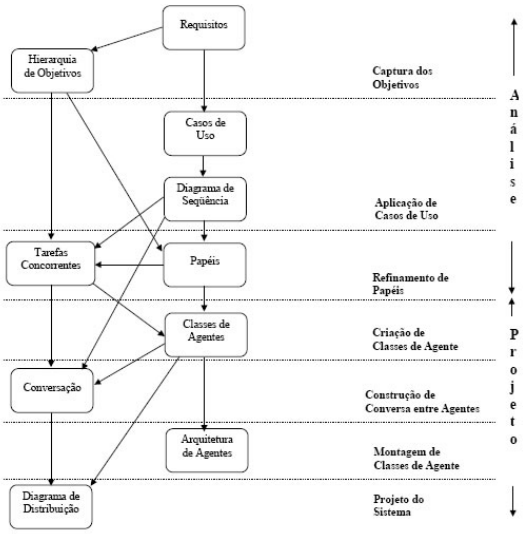

Figura 2.11: Processo de desenvolvimento de MaSE (DELOACH, 2001)

A fase de análise tem como objetivo definir um conjunto de papéis que irão alcançar as metas do sistema. Essa fase envolve três passos:

- Captura de metas: o objetivo desse passo é identificar as metas e submetas do sistema, extraídas de um conjunto de requisitos, e organizá-las em um diagrama hierárquico de metas;

- Aplicar casos de uso: neste passo as metas e submetas são traduzidas em casos de uso, que capturam os cenários previamente identificados com uma descrição detalhada e um conjunto de diagramas de seqüência. Estes casos de uso representam os comportamentos desejados do sistema e a seqüência de eventos;

- Refinamento de papéis: neste passo são definidos os papéis do sistema e a comunicação entre eles, associando com tarefas específicas que representam a comunicação e o conhecimento interno do agente. As tarefas são documentadas em um diagrama de tarefas concorrentes.

A fase de projeto tem como objetivo transformar a fase de análise em uma forma mais sugestiva para implementação do sistema multiagente. Esta fase é dividida em quatro passos:

- Criação de classes de agentes: o objetivo desse passo é identificar classes de agentes e relacionar com os papéis gerados na fase de análise. Um agente pode ser responsável por múltiplos papéis. Ao final deste passo é gerado um diagrama de classes de agente composto pelos agentes identificados e seus correspondentes papéis e as conversações entre agentes;

- Construção de conversações: conversações modelam a comunicação entre duas classes de agentes usando um par de autômatos de estado finito semelhantes, na forma e na função, a tarefas concorrentes. Cada tarefa gera múltiplos diálogos, quando solicita comunicação com mais de uma classe de agente;

- Montagem dos agentes: neste passo é definida a arquitetura interna do agente. MASE não define nenhuma arquitetura específica, podendo ser utilizada qualquer arquitetura desejada;

- Projeto do sistema: nesse ultimo passo são tomadas decisões com relação à plataforma, às linguagens de programação e à comunicação dos agentes. Um diagrama de implatação (*deployment*) é gerado para documentar numero, tipos, localizações e comunicações entre agentes.

A metodologia MASE possui um ambiente gráfico, chamado AgentTool, que auxilia ao desenvolvedor na modelagem de todas as fases da metodologia. Essa ferramenta, além de permitir a modelagem gráfica dos diagramas de cada fase, auxilia na verificação de erros entre as conversações, apresentando uma descrição textual dos erros, e na conversão semiautomática

dos modelos da fase de análise para o projeto. Em sua atual versão, AgentTool III funciona como um plugin para o Eclipse[3]. A Figura 2.12 mostra a ferramenta AgentTool III.

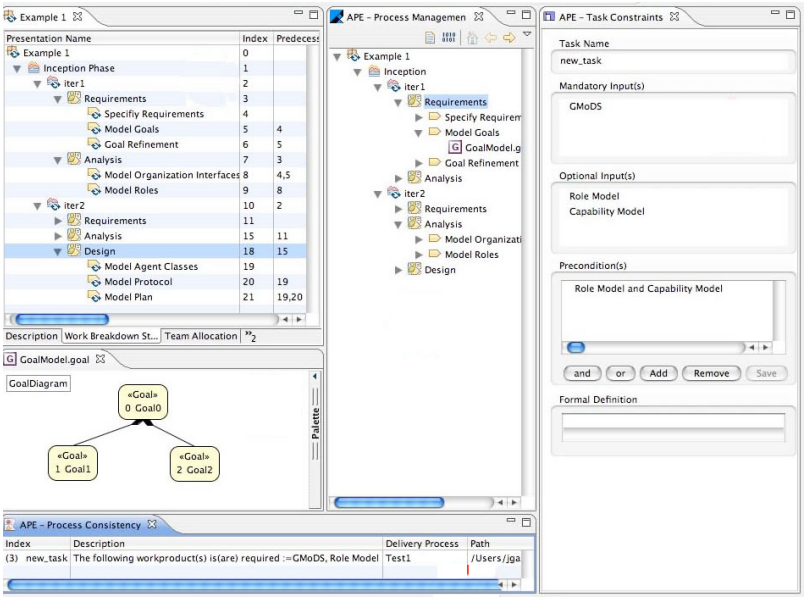

Figura 2.12: Ferramenta AgentTool (GARCIA-OJEDA; DELOACH; ROBBY, 2009)

DiLeo, Jacobs e DeLoach (2002) demonstra uma extensão ao modelo de análise da metodologia MASE adicionando um passo chamado construção de ontologias, que tem como objetivo definir ontologias que capturam o conhecimento do domínio do sistema.

Em Badr, Mubarak e Göhner (2008) é apresentada uma extensão à metodologia MASE para o suporte a sistemas embarcados de tempo real modificando e adicionando novos modelos a MASE. Alguns dos modelos foram adicionados na fase de requisitos, onde são incluídas algumas formas para capturar os requisitos do usuário, tendo em vista restrições impostas pelo hardware, e os requisitos temporais do sistema. Outro modelo adicionado visa descrever o ambiente e suas características comportamentais. Os modelos de papéis, metas e organização foram modificados com o objetivo de adicionar restrições temporais e o suporte à concorrência.

Um framework chamado Organization-Based Multiagent Systems Engineering (O-MASE) (GARCIA-OJEDA et al., 2008) foi proposto com o objetivo de possibilitar a personalização do processo de desenvolvimento da metodologia MASE. Esse framework é baseado no OPEN Process Framework, proposto por Henderson-Sellers e Giorgino (HENDERSON-SELLERS, 2005). O-MASE é definido em forma de um metamodelo composto de conceitos e restrições que guiam o desenvolvedor a criar um processo válido para a metodologia.

Em DeLoach (2009), é apresentada uma extensão chamada *Organization Model for*

[3]Ambiente de programação que pode ser encontrado no endereço http://www.eclipse.org.

Adaptive Computational Systems (OMACS) que permite a modelagem de sistemas adaptativos em MASE através da O-MASE.

2.2.6 Prometheus

Prometheus é uma metodologia de desenvolvimento que proporciona mecanismos para a análise e projeto de SMA baseados em arquiteturas BDI. Prometheus possui mecanismos de estruturação hierárquica que o permite ser executado em diferentes níveis de abstração. Ele utiliza o conceito de capacidade, a qual pode ser composta por planos, eventos, crenças e outras capacidades que dão habilidades específicas ao agente.

O processo de desenvolvimento é dividido em três fases: especificação do sistema, projeto arquitetural e projeto detalhado, como pode ser observado na Figura 2.13.

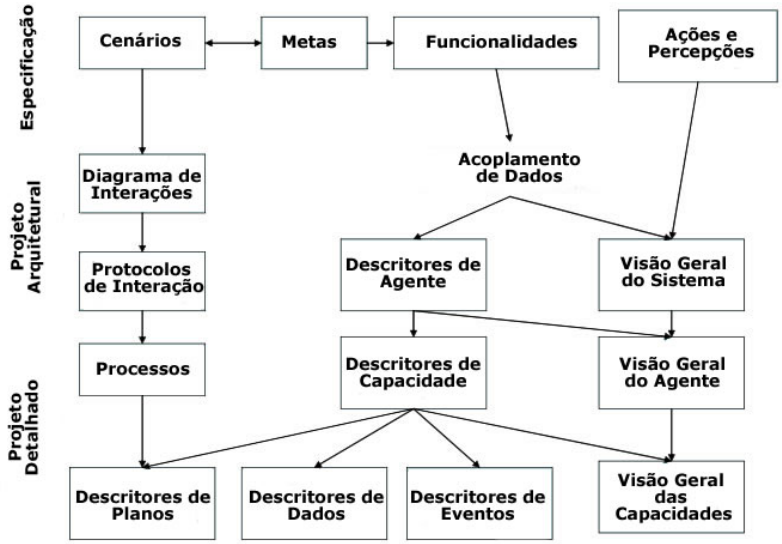

Figura 2.13: Fases da metodologia Prometheus (PADGHAM; WINIKOFF, 2005)

A fase de especificação do sistema busca definir os requisitos do sistema através da descrição das metas do sistema, dos cenários, das funcionalidades e do ambiente, que é descrito através de suas percepções, ações e dados externos.

O projeto arquitetural analisa os artefatos gerados na fase anterior para identificar os agentes e suas interações, os eventos e os dados compartilhados. A identificação dos agentes é realizada através do acoplamento das funcionalidades. Este é demonstrado no diagrama de acoplamento e de conhecimento. Em seguida, após a escolha dos grupos, os agentes são definidos e descritos em *templates* textuais chamados de descritores de agentes. As interações entre agentes são derivadas dos cenários e descritas nos diagramas de interação e nos protocolos de

interação. Os eventos são identificados através das percepções do ambiente e das interações do agente. Os dados compartilhados entre agentes são identificados através do diagrama geral do sistema.

A fase de projeto detalhado tem como objetivo descrever o projeto interno do agente através de suas capacidades, planos, eventos e dados. Uma capacidade pode conter planos, eventos e dados. Na identificação de capacidades, cada funcionalidade definida na fase de especificação pode gerar uma capacidade. Os agentes e suas capacidades são definidos no diagrama geral do agente e através dos descritores de capacidade, enquanto os detalhes de baixo nível são demonstrados usando o diagrama de capacidades e seus vários descritores (planos, eventos e dados).

Prometheus Design Tool (PDT) (SUN; THANGARAJAH; PADGHAM, 2010) é uma ferramenta desenvolvida para auxiliar os analistas no desenvolvimento dos modelos da metodologia Prometheus. Em sua atual versão, PDT foi desenvolvido como um plugin para o ambiente Eclipse (ver Figura 2.14), PDT inclui várias características para auxiliar na modelagem, como:

- Editor gráfico: A ferramenta guia o analista por todas as fases da metodologia oferecendo um grande numero de diagramas para modelar o projeto;
- Descritores de entidade: Através de abas, podem ser editadas e visualizadas várias propriedades das entidades;
- Propagação da informação: As informações são propagadas entre as fases da metodologia oferecendo uma manutenção de consistência quando ocorrem alterações no projeto, através de verificação cruzada entre os vários diagramas;
- Geração de código: A PDT cria estruturas que podem ser usadas na plataforma JACK para a geração dos agentes;
- Geração de relatórios: A geração de relatórios é feita através de um documento HTML que pode conter todo o projeto ou apenas alguns diagramas específicos.

2.2.7 MESSAGE

MESSAGE é uma metodologia orientada a agentes baseada no modelo de desenvolvimento de software do RUP. Ela estende a UML adicionando novas notações para a modelagem

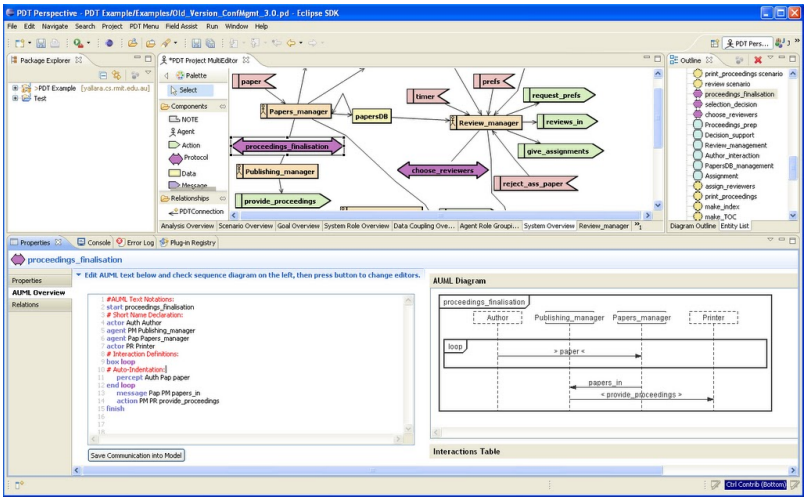

Figura 2.14: Ferramenta Prometheus Design Tool (PLUGIN, 2010)

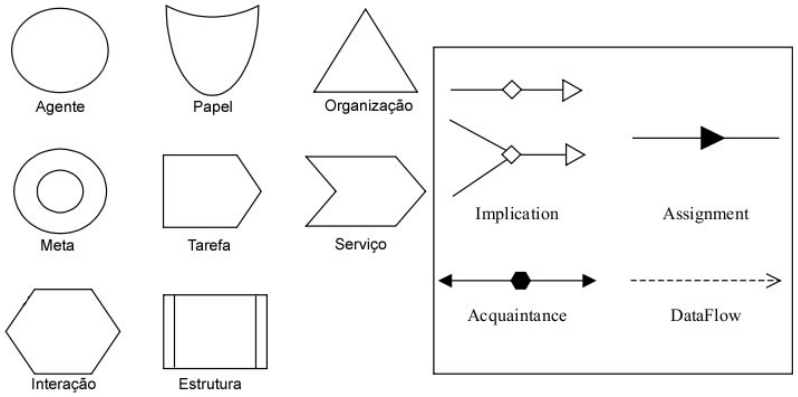

Figura 2.15: Notações estendidas para representar novos conceitos (GARIJO; GóMEZ-SANZ; MASSONET, 2005)

de conceitos relacionados a agentes, como papéis, tarefas, metas, organização e serviços. A Figura 2.15 demonstra as novas notações adicionadas.

O processo de desenvolvimento de MESSAGE consiste nas fases análise e projeto. A fase de análise tem como objetivo gerar uma especificação do sistema que descreve o problema a ser solucionado, gerando um modelo abstrato que auxilia na validação e no desenvolvimento do projeto. Essa fase gera cinco modelos: (i) modelo de organização; (ii) modelo de agentes/papéis; (iii) modelo de metas/tarefas; (iv) modelo de domínio; e (v) modelo de interação.

O modelo de organização descreve a estrutura geral do sistema multiagente demonstrando as interações entre entidades concretas (papéis, agentes, organização e recursos) e seu ambiente. O modelo de metas/tarefas especifica as metas do sistema, gerando um diagrama com a estrutura de decomposição das tarefas para alcançar a meta. Neste modelo é descrito quais papéis são responsáveis por cada tarefa. O modelo de agentes/papéis contem uma descrição de-

talhada das características do agente, descrevendo suas metas, recursos e tarefas. O modelo de domínio é responsável pela especificação das entidades e relações do domínio da aplicação. O modelo de interação concentra-se na interação entre agentes, definindo iniciador, colaboradores, pré-condições e pós-condições.

Os cinco modelos da fase de análise são definidos através de refinamentos sucessivos, onde cada novo ciclo gera um refinamento das fases anteriores. A Figura 2.16 demonstra os sucessivos refinamentos da fase de análise e projeto.

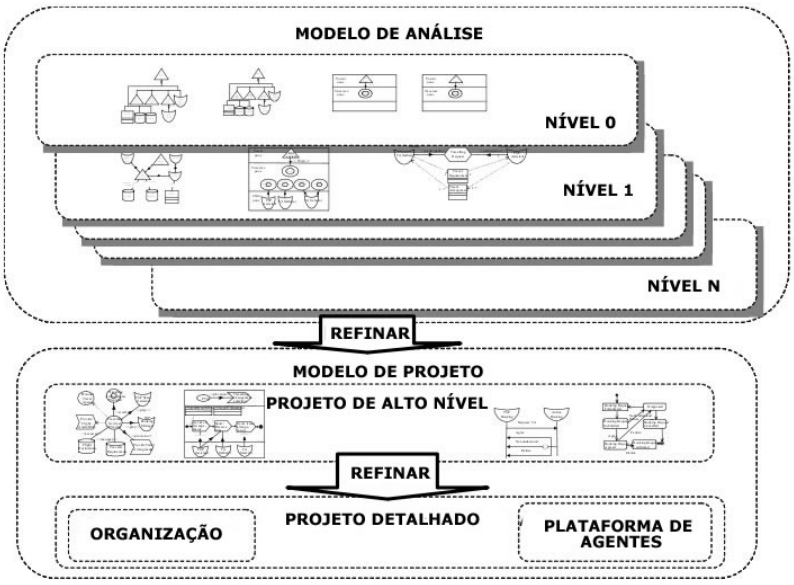

Figura 2.16: Descrição do processo de MESSAGE (GARIJO; GóMEZ-SANZ; MASSONET, 2005)

A fase de projeto transforma os modelos gerados na análise em entidades que podem ser codificadas em uma plataforma de agentes. Esta fase é dividida em projeto de alto nível e projeto detalhado.

O projeto de alto nível especifica os agentes do sistema e quais papéis são de sua responsabilidade. Nesta fase os agentes da análise são reavaliados com o objetivo de verificar se são realmente necessários ou podem ser substituídos por objetos. As interações entre agentes definidas na análise são especificadas através de diagramas da UML. O projeto detalhado busca mapear os modelos de alto nível para conceitos computacionais específicos de plataformas.

2.2.8 INGENIAS

INGENIAS é uma metodologia criada a partir da MESSAGE, com diversas modificações, como: i) inclusão de novos modelos (modelo de ambiente), ii) suporte a novas arquiteturas de

agentes, iii) ferramentas de suporte para documentação do sistema e iv) geração automática de código. INGENIAS é baseada no processo de desenvolvimento do RUP e utiliza o conceito de visões com o objetivo de guiar o analista pelas fases de análise e projeto, com as indicações de quais artefatos de software devem ser produzidos em cada fase.

O processo de INGENIAS é composto das fases de análise, projeto e implementação, contendo cerca de setenta passos que guiam o processo de desenvolvimento (TRAN; LOW, 2008). As fases de análise e projeto têm como objetivo gerar cinco modelos: (i) modelo do agente; (ii) modelo de interação; (iii) modelo de metas/tarefas; (iv) modelo de organização; e (v) modelo de ambiente. Na fase de projeto, todos os modelos são refinados com o objetivo de tornar os modelos mais próximos da fase de codificação. Todos os modelos usam notações especificas da metodologia.

O objetivo do modelo de agentes é definir o agente em termos de metas, papéis e tarefas, além das características comportamentais, como o estado mental. O modelo de interação trata da interação e comunicação dos agentes.

O modelo de metas/tarefas define quais as metas iniciais do sistema, quais as tarefas realizadas para cumprir as metas, a decomposição de metas e tarefas, quais as condições de sucesso e falha das metas e as precondições e pós condições das tarefas.

No modelo de organização, toda a estrutura do SMA é especificada através de grupos (agentes, papéis e recursos) e workflows. Nesse modelo também são descritas as dependências sociais entre os grupos. O modelo de ambiente demonstra os mecanismos de percepções usados pelos agentes para perceber as saídas do ambiente. Neste modelo também são descritos os recursos e aplicações que existem no ambiente.

A fase de codificação é a ultima fase do processo, gerando códigos de agente através da ferramenta INGENIAS Development Kit.

A ferramenta INGENIAS Development Kit, também conhecida como IDK, é uma ferramenta gratuita que auxilia o analista no processo de desenvolvimento da metodologia INGENIAS. Esta ferramenta oferece suporte visual a todas as fases da metodologia. Ao final da especificação, o sistema multiagente pode ser gerado exportando o código para a plataforma JADE. A Figura 2.17 demonstra a interface da ferramenta.

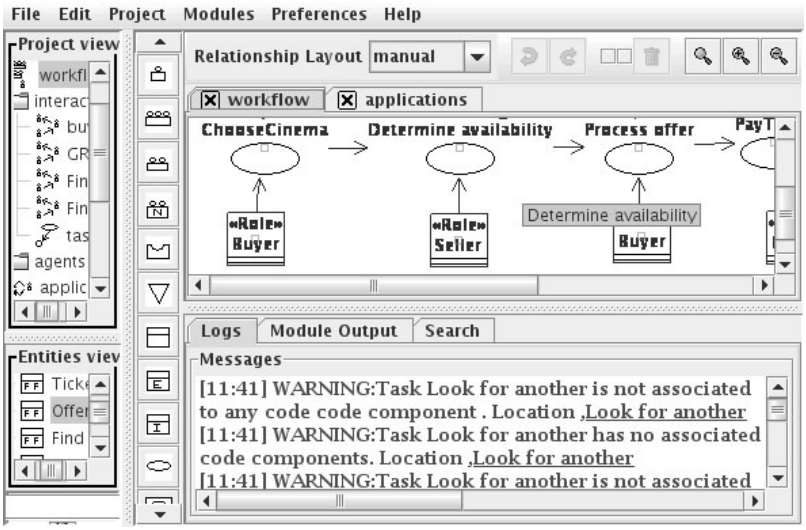

Figura 2.17: Editor visual do INGENIAS (GOMEZ-SANZ et al., 2008)

2.2.9 PASSI

PASSI (*Process for Agent Societies Specification and Implementation*) é uma metodologia para desenvolvimento e implementação de sistemas multiagentes composta de modelos da engenharia de software orientada a objetos, de abordagens da inteligência artificial e de notações UML que auxiliam o projetista na especificação do sistema, desde a fase de requisitos até a implementação do sistema (CHELLA; COSSENTINO; SEIDITA, 2009).

O processo de desenvolvimento de PASSI consiste em cinco modelos que são incrementados em doze passos seqüenciais, onde os artefatos gerados em um passo são requisitos para o próximo passo. A Figura 2.18 demonstra os modelos e os passos da metodologia.

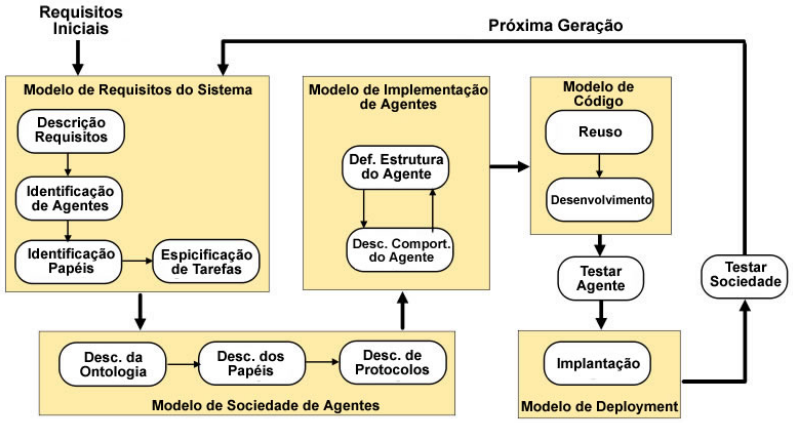

Figura 2.18: Modelos da metodologia PASSI (COSSENTINO, 2005)

O primeiro modelo a ser gerado trata-se do modelo de requisitos do sistema, que tem como objetivo capturar os requisitos do sistema necessários para a especificação do SMA. Ele

é composto por quatro passos:

- Descrição do domínio: tem como objetivo descrever as funcionalidades do sistema através de uma série de diagramas de casos de uso;
- Identificação dos agentes: nesta fase são identificadas as responsabilidades dos agentes, que são modeladas através do empacotamento dos casos de uso do passo anterior;
- Identificação dos papéis: identifica os papéis de cada agente através de cenários;
- Especificação de tarefas: especifica as capacidades de cada agente através de diagramas de atividades, que demonstram estímulos e comportamentos.

O modelo da sociedade de agentes descreve a interação entre os agentes envolvidos no sistema. Esse modelo possui três passos:

- Descrição de ontologias de domínio: utiliza diagramas de classe e OCL (*Object Constraints Language*) para definir conceitos e entidades que definem o conhecimento do domínio;
- Descrição de ontologias de comunicação: cada agente é descrito através de sua ontologia e suas comunicações;
- Descrição de papéis: descreve todos os papéis de responsabilidades dos agentes e todas as atividades envolvidas com cada papel.

O modelo de implementação possui dois passos (definição da estrutura e definição de comportamentos) que visam descrever a arquitetura interna dos agentes e seus comportamentos. Também são representadas a invocação de métodos e as trocas de mensagens entre agentes. No modelo de codificação, o sistema é codificado (passo de produção de código) sempre buscando reutilizar padrões predefinidos de agentes e tarefas. No último modelo são definidas restrições para mobilidade de agentes e como serão alocadas as unidades de agentes para uma posterior distribuição.

A metodologia PASSI apresenta duas ferramentas que oferecem suporte a sua modelagem, a PASSI Tool Kit e a AgentFactory. A PASSI Tool Kit é um plugin para a ferramenta Rational Rose, que auxilia a modelagem dos diagramas UML no suporte a sistemas multiagentes enquanto a AgentFactory dá suporte à fase de geração de código dos agentes.

Algumas extensões foram propostas à metodologia PASSI. Chella et al. (2006) apresentam a Agile PASSI, que é uma extensão que tem como objetivo agilizar o desenvolvimento de software através de processos mais simples e rápidos, mantendo sempre um certo nível de qualidade e documentação. Em Cossentino et al. (2008), os autores apresentam uma outra extensão, chamada PASSIM, que utiliza um processo de desenvolvimento baseado em simulação de agentes. Essa extensão adiciona uma fase de simulação à metodologia PASSI com o objetivo de realizar validações funcionais e não funcionais do sistema. Em Nunes et al. (2009), são propostas algumas modificações ao modelo de requisitos do sistema usando uma abordagem baseada em linhas de produtos de software.

2.2.10 Análise comparativa entre as metodologias

O estudo comparativo foi baseado no framework proposto por Tran e Low (2005) e nas avaliações realizadas por Ibrahim, Far e Eberlein (2009), Far, Mudigonda e Elamy (2009), LIN et al. (2007) e Ibrahim, Far e Eberlein (2009). Este framework é composto por um método de avaliação que divide os conceitos relacionados aos SMA em quatro grupos (Conceitos, Processos, Modelos e Notações, Aplicações e Suporte), permitindo assim uma melhor identificação de suas similaridades e diferenças, bem como os pontos fortes e fracos de cada metodologia.

O primeiro grupo permite uma avaliação de quais conceitos da orientação a agentes a metodologia é capaz de representar. Os conceitos que se deseja que a metodologia represente são:

- Metas do sistema: avalia se a metodologia é capaz de especificar as metas do sistema e em qual modelo;
- Casos de uso e cenários: descreve se a metodologia utiliza casos de uso;
- Papéis: define se o conceito de papéis esta presente na metodologia;
- Conceituação do domínio: avalia se é possível descrever conceitos do domínio;
- Objetivos do agente: avalia se é possível representar os objetivos do agente;
- Crenças: define se a metodologia representa as crenças de um agente;
- Serviços: avalia se existe suporte para modelar serviços;
- Percepções e ações: define se é possível modelar as percepções e ações dos agentes;

- Arquitetura do agente: avalia se as arquiteturas de agente podem ser modeladas;
- Protocolos de interação: avalia se a metodologia é capaz de representar protocolos de interação;
- Ambiente: defini se é possível representar o ambiente;
- Estrutura organizacional e relacionamento entre agentes: avalia se a estrutura organizacional e os relacionamentos entre agentes podem ser definidos;
- *Deployment* (implantação): Avalia se a metodologia oferece suporte a diagramas de deployment (implantação).

O segundo grupo avalia os aspectos de desenvolvimento relacionados ao processo que a metodologia é capaz de tratar. Os critérios usados são:

- Domínio da aplicação: avalia se a metodologia é aplicável a qualquer aplicação;
- Abrangência das fases do desenvolvimento: descreve quais as fases do desenvolvimento do projeto são suportadas pela metodologia;
- Ciclo de desenvolvimento: descreve qual o melhor ciclo de desenvolvimento para a metodologia;
- Verificação e validação: avalia se a metodologia oferece maneiras de verificação e validação dos modelos gerados;
- Natureza dos agentes: avalia a natureza dos agentes (BDI, adaptativo e heterogêneos).

A terceira categoria avalia a viabilidade e usabilidade das técnicas usadas para execução dos passos dos processos e do desenvolvimento de modelos e componentes notacionais. Os critérios usados para essa categoria são:

- Expressividade: avalia o nível em que os modelos da metodologia são capazes de representar diferentes perspectivas do sistema;
- Modularidade: avalia se a metodologia promove modularidade no projeto dos agentes e do sistema;
- Facilidade de entendimento: avalia se os modelos gerados pela metodologia são fáceis de entender;

- Reuso: avalia se os modelos gerados podem ser reusados;
- Capacidade de inferências: avalia se os modelos gerados conseguem representar a capacidade de realizar inferências dos agentes;
- Concorrência: avalia se a metodologia consegue representar a concorrência entre agentes, como, por exemplo, objetivos concorrentes ou participação em interações concorrente;
- Adaptação: avalia se os modelos são capazes de representar características de adaptação do agente, ou seja, a habilidade de aprender e agir com experiências anteriores;
- Autonomia: avalia se a metodologia oferece suporte a representação da característica de autonomia dos agentes, ou seja, a habilidade de agir sem intervenção direta do ser humano;
- Reatividade: avalia se a metodologia é capaz de representar agentes reativos, ou seja, agentes que percebem e reagem a mudanças no ambiente;
- Proatividade: avalia se a metodologia é capaz de representar a característica de proatividade dos agentes, ou seja, a habilidade do agente tomar iniciativas para realizar seus objetivos;
- Colaboração: avalia se a metodologia oferece modelos para representar a colaboração entre agentes, ou seja, a habilidade de trabalhar em conjunto com outros agentes para alcançar um objetivo comum;
- Comunicação: avalia se a metodologia é capaz de representar a comunicação entre os agentes.

O quarto e último grupo trata de características de aplicações e suporte da metodologia, como:

- Recursos disponíveis: avalia se existem recursos disponíveis, como: ferramentas para modelagem e implementação do sistema;
- Sistemas Abertos: avalia se a metodologia é capaz de modelar SMA em sistemas abertos como, por exemplo, a internet;
- Suporte a objetos: avalia se a metodologia oferece suporte à modelagem de objetos convencionais;

- Suporte a agentes móveis: avalia se a metodologia suporta a modelagem de agentes móveis;
- Suporte a ontologias: avalia se a metodologia oferece suporte ao uso e especificação de ontologias.

A seguir, apresentaremos a avaliação das metodologias, dividindo-as nos 4 (quatro) grupos apresentados.

Conceitos									
Critérios	**Gaia**	**Adelfe**	**Message**	**Ingenias**	**Passi**	**Tropos**	**Prometheus**	**Mas-Commonkads**	**MaSE**
Metas do sistema			Modelo de objetivos/tarefas	Modelo de objetivos e tarefas		Diagrama de Atores	Diagrama de objetivos	Goal Cases	Diagrama de hierarquia de objetivos
Casos de uso		Modelo de caso de usos		Diagrama de casos de uso	Modelo de requisitos do sistema		Descrição de casos de uso	Diagrama de casos de uso	Diagrama de casos de uso
Papéis	Modelo de papéis		Modelo de papéis	Modelo de organização	Modelo da sociedade de agentes				Diagramas de papéis
Conceituação do domínio			Modelo de Domínio		Modelo da sociedade de agentes			Modelo de Conhecimento	
Objetivos do agente		Documento de arquitetura detalhada		Modelo do agente	Modelo de requisitos do sistema	Diagrama de atores	Descrição da classe de agentes	Modelo do agente	
Crenças		Documento de arquitetura detalhada			Modelo de implementação do agente	Diagrama de classe de agentes	Descritor de dados	Modelo de conhecimento	
Serviços	Modelo de Serviços	Documento de arquitetura detalhada			Modelo da sociedade de agentes			Modelo do agente	
Percepções e ações							Descritor de percepções	Modelo de coordenação	
Arquiteturas do agente		Documento de arquitetura detalhada			Modelo de implementação do agente	Arquitetura BDI	Arquitetura BDI	Modelo de projeto	Diagrama de arquitetura do agente
Protocolos de interação	Modelo de interação	Documento de linguagem de interação	Modelo de interação	Modelo de interação	Modelo da sociedade do agente	Diagrama de seqüência	Protocolo de interação	Modelo de Coordenação	Diagrama de comunicação
Ambiente	Modelo de ambiente	Documento de definição do ambiente	Modelo de organização	Modelo do ambiente			Diagrama de visão do sistema		
Estrutura organizacional e relacionamentos entre agentes	Modelo de estrutura organizacional		Modelo de Organização	Modelo de organização		Requisitos não funcionais		Modelo de organização	
Deployment					Modelo de deployment				Diagrama de deployment

TABELA 2.2: Avaliação das metodologias baseada nos conceitos

Processos									
Critérios	**Gaia**	**Adelfe**	**Message**	**Ingenias**	**Passi**	**Tropos**	**Prometheus**	**Mas-Commonkads**	**MaSE**
Natureza do agente	Heterogêneos	Adaptativos	Heterogêneos	Agentes com estados e metas	Heterogêneos	Agentes BDI	Agentes BDI	Heterogêneos	Heterogêneos
Domínio da aplicação	Independente		Independente	Independente	Independente	Independente	Independente	Independente	Independente
Abrangência das fases do desenvolvimento	Análise e Projeto	Análise, Projeto e implementação	Análise e Projeto	Análise, Projeto e implementação	Análise, Projeto e implementação	Análise e Projeto	Análise e Projeto	Análise e Projeto	Análise e Projeto
Ciclo de desenvolvimento	Cascata	RUP	RUP	RUP	Iterativo	Interativo e incremental	Iterativo	Espiral	Iterativo
Verificação e validação	Não	Sim	Não	Sim	Sim	Sim	Sim	Sim	Sim

TABELA 2.3: Avaliação das metodologias baseada nos processos

Modelos									
Critérios	**Gaia**	**Adelfe**	**Message**	**Ingenias**	**Passi**	**Tropos**	**Prometheus**	**Mas-Commonkads**	**MaSE**
Expressividade	M	A	M	A	A	A	A	A	A
Modularidade	S	S	S	S	S	S	S	S	S
Facilidade de entendimento	A	A	A	M	A	A	A	A	A
Reuso	S	P	P	P	S	P	P	S	S
Capacidade de inferências	N	S	S	S	S	S	S	S	P
Autonomia	S	S	S	S	S	S	S	S	S
Reatividade	P	S	S	S	S	S	S	S	S
Pró-atividade	S	S	S	S	S	S	S	S	S
Concorrência	N	N	N	N	S	N	N	S	S
Adaptação	P	S	P	P	N	N	N	N	N
Colaboração	S	S	S	S	S	S	S	S	S
Comunicação	N	S	S	S	S	S	S	S	S

A= Alta, M= Média, B= Baixa, S = Sim, N = Não e P = Possivelmente

TABELA 2.4: Avaliação das metodologias baseada nos modelos

Suporte									
Critérios	**Gaia**	**Adelfe**	**Message**	**Ingenias**	**Passi**	**Tropos**	**Prometheus**	**Mas-Commonkads**	**MaSE**
Recursos disponíveis	S	S	S	S	S	S	S	N	S
Sistemas abertos	S	S	N	N	N	N	N	N	N
Suporte a objetos	N	N	S	S	N	N	N	N	N
Suporte a agentes móveis	N	N	N	N	S	N	N	N	N
Suporte a ontologias	N	N	N	N	S	N	N	S	N

Tabela 2.5: Avaliação das metodologias baseada no suporte

2.2.11 Limitações das Metodologias

ADELFE é uma metodologia exclusiva para o desenvolvimento de agentes adaptativos, sendo inapropriada para o uso em outros tipos de agentes, por exemplo, em agentes puramente reativos ou em agentes com planejamento, pois a estrutura interna do agente não tem planos.

Prometheus apresenta uma modelagem para o desenvolvimento de agentes com arquitetura BDI, limitando assim a especificação de outro tipo de agente, como os puramente reativos ou agentes híbridos. Outra falha desta metodologia trata-se da organização do SMA, onde o seu projeto não deixa bem especificado os relacionamentos entre papéis e agentes do sistema. Não é capaz também de representar a relação entre agentes e objetos.

Gaia e MESSAGE apresentam deficiências no suporte ao projeto interno do agente. Em nenhum dos seus modelos é definida a arquitetura interna do agente, nem como deve ser descrito o conhecimento dos agentes. Construções mentais, como planos, metas e ações, também não são definidas na metodologia. Em Gaia, devido à falta de diagramas (em sua versão original), a comunicação não apresenta um projeto bem detalhado das interações. **MASE** , apesar de possuir uma fase para a escolha da arquitetura interna do agente, não demonstra como podem ser representadas essas arquiteturas. Sendo assim, ela apresenta as mesmas deficiências de Gaia e MESSAGE. Gaia e MASE não são capazes de representar a relação entre agentes e objetos.

INGENIAS possui uma grande quantidade de passos no processo de desenvolvimento dos modelos, o que dificulta a sua usabilidade, por exemplo, o modelo de organização captura os conceitos de agente, de grupo, de workflow, de tarefas, de papel, de meta, de aplicação e de recurso. INGENIAS utiliza uma linguagem de modelagem própria, com uma notação não

comum que também pode dificultar sua adoção. A definição da arquitetura dos agentes não é definida em nenhum dos modelos da metodologia.

PASSI, apesar de apresentar uma fase para especificação do comportamento interno do agente, sugere a definição do comportamento através de eventos e invocação de métodos, que são técnicas mais adequadas para comportamentos de objetos. Ela também não é capaz de representar a relação entre agentes e objetos.

Em **MAS-CommonKADS**, devido a grande quantidade de tecnologias para a geração dos seus modelos, o suporte ao desenvolvimento do modelo do agente torna-se difícil. Devido à grande quantidade de conceitos relacionado ao agente, como papéis, serviços oferecidos, metas, capacidades e normas. Esta metodologia também não é capaz de representar a relação entre agentes e objetos.

Tropos apresenta uma modelagem para o desenvolvimento de agentes com arquitetura BDI, limitando assim a especificação de outro tipo de agente como os puramente reativos ou agentes híbridos. Além disso, não é capaz de representar mobilidade dos agentes, nem a modelagem de objetos.

Com base nas limitações apresentadas pode ser percebido que um dos grandes problemas das metodologias orientadas a agentes é a falta de especificação dos comportamentos internos e da arquitetura de agente escolhida para realizar esses comportamentos. A maioria das metodologias foca principalmente nos aspectos de interação e organização dos agentes e pouco tem sido observado em relação a como o agente irá solucionar um determinado objetivo. Isso implica em outra limitação importante que merece ser citada que é a impossibilidade da modelagem de técnicas de IA na resolução dos problemas.

2.3 Linguagens de Modelagem

O desenvolvimento de SMAs é uma tarefa complexa dada as várias propriedades de um agente de software. As linguagens de modelagem são o meio pelo qual se expressam os modelos, tendo como principal objetivo a descrição dos sistemas. Estas linguagens buscam demonstrar, através de diagramas semi-formais, todas as propriedades de um agente, como, por exemplo, papéis, organizações, ambientes e relacionamentos entre os agentes.

A UML (*Unified Modeling Language*) é a linguagem de modelagem padrão do paradigma de programação orientada a objetos e apesar dessa linguagem não ter sido desenvolvida especialmente para o paradigma de orientação a agentes, vários estudos foram realizados para a

sua adequação a este paradigma. Através de extensões da linguagem UML, muitas linguagens de modelagem foram desenvolvidas incorporando os conceitos de SMA. A seguir são apresentadas algumas linguagens de modelagem para modelar agentes.

2.3.1 AUML

Segundo Padilha e Jácome (2002), AUML foi criada a partir de um projeto desenvolvido por Odell, Parunak e Bauer (2001), visando adaptar a linguagem *Unified Modeling Language* (UML) às características inerentes ao paradigma de Orientação a Agentes (AO). O objetivo da AUML é fornecer uma semântica semi-formal e intuitiva através de uma notação gráfica amigável para o desenvolvimento de protocolos de interação entre agentes em sistemas orientados a agentes. Na AUML, a interação entre agentes pode ser definida em um modelo com três camadas: a primeira camada contendo definições gerais de protocolos de interação (pacotes e *templates*); a segunda camada representando as interações particulares entre agentes (diagramas de seqüência e comunicação); e, finalmente, a terceira camada detalhando o comportamento interno de cada agente (diagramas de atividades e estados) (DÁRIO, 2005).

Na primeira camada, a AUML define a modelagem de protocolos como um todo através de artefatos disponíveis, chamados pacotes e *templates*. A AUML estende a UML utilizando o conceito de pacote não somente para agregar diagramas de classes, mas também para agregar diagramas genéricos. Na AUML, pacotes podem representar abstrações de diagramas quaisquer. De uma maneira mais geral, quando os agentes são identificados pelos papéis que representam e não diretamente por seus nomes, pode-se utilizar *templates*. Nominando-se os agentes que realizam os papéis indicados no *template* obtemos os pacotes que descrevem os protocolos de interação (BASTOS; OLIVEIRA, 2000b). A Figura 2.19 demonstra como é composto um *template*.

Na segunda camada, é possível representar as interações particulares entre agentes. Para isso, utilizam-se diagramas de interação da UML (diagramas de seqüência e de comunicação). O diagrama de seqüência é utilizado para representar a comunicação entre agentes, dispondo os elementos gráficos de uma forma que enfatize a seqüência cronológica de comunicações. No diagrama de comunicação são enfatizadas as associações entre agentes, onde a seqüência das interações é representada através da numeração das mensagens (SILVA, 2004).

Na última camada da AUML, são utilizados o diagrama de atividades e o diagrama de estados para detalhar o comportamento interno de um único agente, quando este estiver executando protocolos. Os diagramas de atividades representam as operações e os eventos que ati-

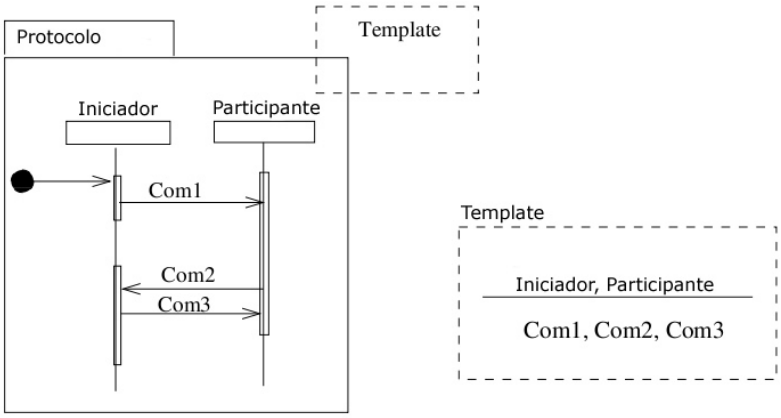

Figura 2.19: Template de protocolos AUML

vam estas. Diagramas de estados não são normalmente utilizados para expressar protocolos de interação, pois têm uma visão centrada nos estados, ao invés de uma visão centrada no agente ou no processo. Este tipo de diagrama é melhor aplicável como um mecanismo de restrições para o protocolo, sendo incorporado aos agentes para que as restrições sejam conhecidas (ODELL; PARUNAK; BAUER, 2001).

2.3.2 MAS-ML

MAS-ML (*Multi-Agent Systems Modeling Language*) (SILVA; CHOREN; LUCENA, 2004) é uma linguagem de modelagem que estende UML através da introdução de novas meta-classes ao metamodelo de UML. Estas descrevem as novas entidades que compõem um SMA, assim como os novos relacionamentos definidos entre estas entidades, e entre estas entidades e objetos (BRANDAO, 2005).

A MAS-ML é baseada no framework conceitual TAO (*Taming Agents and Objects*), que é um metamodelo que define aspectos estáticos e dinâmicos de SMA. Esta linguagem incorpora estes aspectos através da extensão do diagrama de classes da UML e da inclusão de dois novos diagramas estáticos (diagramas de organização e de papéis). Com o objetivo de descrever a interação entre as várias entidades que participam de um SMA e a estrutura de protocolos e planos, também foram estendidos os diagramas dinâmicos de seqüência de mensagens e atividades da UML.

Devido à inclusão de novos diagramas, o diagrama de classes da UML foi estendido com o objetivo de demonstrar os novos relacionamentos entre as classes e os ambientes, entre as classes e os agentes e entre as classes e as organizações (FARIAS et al., 2009).

O diagrama de organização modela a organização do SMA, demonstrando os rela-

cionamentos e as regras entre agentes, objetos, ambientes e suborganizações. A Figura 2.20 demonstra o diagrama de organização de MAS-ML.

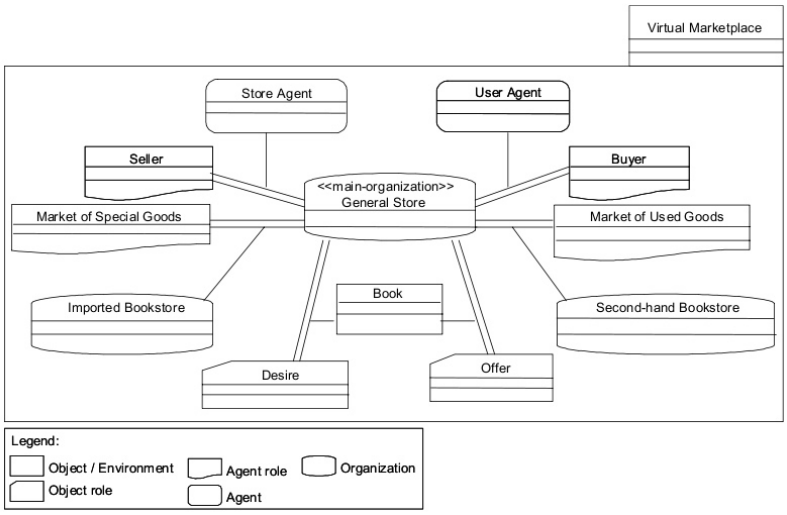

Figura 2.20: Diagrama de organização MAS-ML

O diagrama de papéis é responsável por determinar os relacionamentos entre os papéis de agente e os papéis de objeto (GONçALVES et al., 2010). A Figura 2.21 demonstra o diagrama de papéis de MAS-ML.

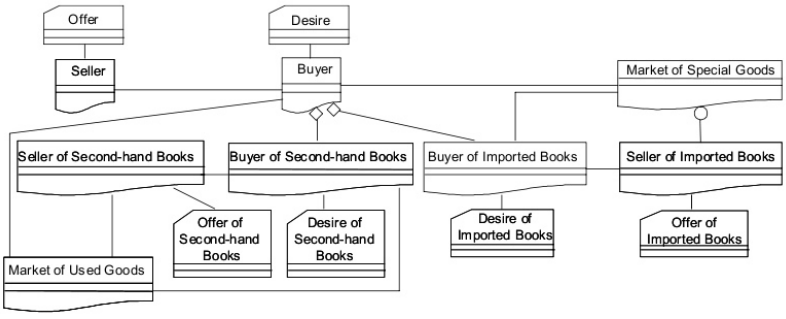

Figura 2.21: Diagrama de papéis MAS-ML

Os aspectos dinâmicos de SMAs modelam as interações entre agentes, organizações, ambientes e objetos. Os diagramas de seqüências de mensagens foram estendidos de forma a modelar as interações entre as novas entidades criadas através do conceito de troca de mensagens. Também foram incluídas representações para modelar planos e ações. O diagrama de atividades tem como objetivo modelar comportamentos internos do agente, como planos e ações.

Uma nova extensão à linguagem foi proposta por Gonçalves et al. (2010). Essa extensão tem como objetivo a modelagem das arquiteturas internas de agente propostas por Russel

e Norvig (1995), possibilitando a modelagem de agentes reativos, agentes baseados em metas e agentes baseado em utilidade.

2.3.3 AORML

A linguagem AORML (*Agent Oriented Rule Markup Language*) é baseada no meta-modelo AOR (*Agent-Object-Relationship*) (WAGNER, 2002) e representa os comportamentos dos agentes, as ações e a troca de mensagens que cada agente realiza. Ela também determina diferenças entre os agentes e objetos.

A modelagem em AORML pode ser composta por dois tipos de modelos: um modelo externo e um modelo interno. O modelo externo corresponde a um modelo conceitual, voltado para a análise do domínio da aplicação. O modelo interno corresponde a um modelo de projeto da aplicação (BRANDAO, 2005).

No modelo externo, os seguintes diagramas são definidos (NERY; OLIVEIRA; FURTADO, 2004):

- Diagrama de agentes: definem os agentes, os objetos e as relações entre eles.
- Diagrama de interação: destaca as classe de eventos e as solicitações que determinam possíveis interações entre dois tipos de agentes. Este diagrama possui quatro tipos de relacionamento:
  - Relacionamento entre agentes com ações e eventos comunicativos: enviar e receber mensagens são eventos ou ações do tipo que relata ao agente um tipo de relacionamento que gera comunicação.
  - Relacionamento entre agentes com ações e eventos não – comunicativos: fazer e perceber os agentes são eventos que não geram comunicação entre os agentes.
  - Relacionamento entre agentes e compromisso e solicitação: Este relacionamento é visualizado com um conector particular que acompanha uma ação ou evento.
  - Relacionamento entre evento sem ação: este relacionamento é percebido pelo agente no qual ele não executa nenhuma ação, ou seja, apenas uma chamada a um método.
- Diagrama de seqüência: destacando a interação entre processos. Interação entre processos é uma seqüência de ações e eventos feitos e recebidos pelos agentes que seguem um conjunto de regras.

A Figura 2.22 contém um modelo de um diagrama externo AORML, onde são descritas as ações que este realiza, os eventos a que reage e outros agentes ou objetos com quem interage.

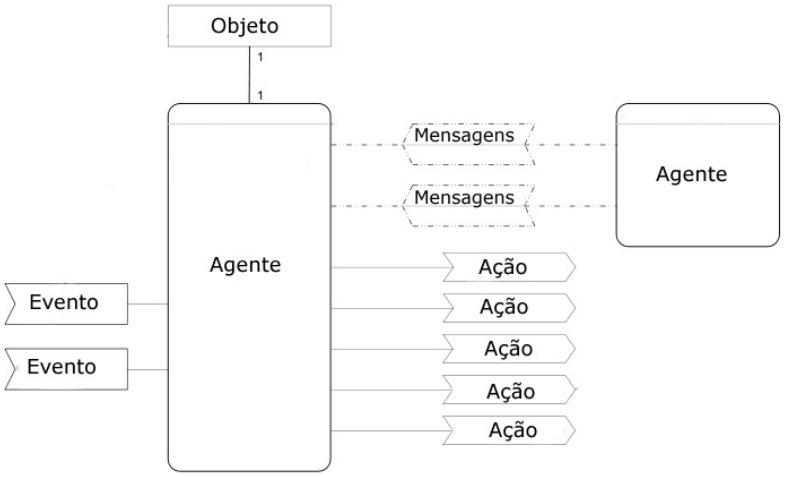

Figura 2.22: Modelo de um diagrama externo AORML

No modelo interno, os seguintes diagramas são definidos (BRANDAO, 2005):

- Diagrama de Estruturas de Reação: descreve classes de agentes, ações, eventos, solicitações e compromissos que determinam possíveis interações entre os agentes.
- Diagrama de Seqüência de Reação: define instâncias de interações de processos numa perspectiva interna ao agente.
- Diagrama de Padrão de Reação: descreve os padrões de reação dos agentes, os quais são descritos através de regras de reação.

2.3.4 AML

A AML é uma linguagem de modelagem semiformal, especificada como uma extensão da UML 2.0, com o objetivo de especificar, modelar e documentar sistemas que incorporem os conceitos de SMA (CERVENKA; TRENCANSKY, 2007). Essa linguagem é formada por um conjunto de elementos criados para representar diferentes aspectos de sistemas multiagentes. Esta linguagem de modelagem pode ser aplicada a qualquer sistema que: (i) contenha entidades autônomas, concorrentes, assíncronas e pró-ativas; (ii) necessite perceber e interagir com o ambiente; (iii) use interações complexas agregadas a serviços; (iv) empregue estruturas sociais; e (v) capture estados mentais (TRENCANSKY; CERVENKA; GREENWOOD, 2006).

A AML é construída sobre a superestrutura da UML 2.0, adicionando novos conceitos apropriados para as características de sistemas multiagentes. As vantagens dessa abordagem

são: reutilizar os conceitos da UML, que são bem definidos, fundamentados e conhecidos; utilizar mecanismos já existentes para especificação e extensão de linguagens baseadas em UML (extensões de metamodelo e perfis UML); e incorporar a AML em ferramentas CASE baseadas em UML. As entidades fundamentais que compõem a AML e podem ser utilizadas para modelar entidades em tipo e instancia são (DANC, 2008):

- Agente: é utilizado para especificar o tipo de agente, isto é, entidades auto contidas que são capazes de interagir, observar e possuem comportamento autônomo em relação ao ambiente em que estão inseridos;
- Recurso: é utilizado para modelar o tipo de recursos de um sistema, isto é, entidades físicas ou de informação;
- Ambiente: é utilizado para modelar o tipo um ambiente interno a um sistema, isto é, entidades lógicas ou físicas que cercam o agente e oferecem condições para que as outras entidades existam e realizem suas tarefas.

A AML oferece um rico conjunto de construções para modelagem de aplicações que possuem ou exibem características de SMA. Os modelos da AML são divididos em pacotes com modelos para estados mentais, comportamentos e de arquitetura, como mostra a Figura 2.23:

O pacote com os modelos para a arquitetura define metaclasses para definir aspectos arquiteturais do SMA. Dentro desse pacote estão definidos pacotes com as metaclasses que definem agentes, recursos, ambientes, ontologias e aspectos sociais.

O pacote de aspectos sociais de AML define características da estrutura social das entidades e aspectos do comportamento social. Neste pacote são encontrados diagramas para a modelagem de papéis e suas interações e de uma unidade de organização que demonstra a organização multiagente em duas perspectivas: (i) externa, que representa um grupo de entidades que interage com o ambiente e com outros agentes; e (ii) interna, que representa a estrutura dos agentes, demonstrando interações entre papéis, restrições e normas.

O pacote de comportamentos apresenta metaclasses que modelam os comportamentos dos agentes, as suas percepções e os seus atuadores, as interações através de comunicações, a mobilidade dos agentes e os serviços oferecidos. A Figura 2.24 demonstra um exemplo de um agente com seus sensores e atuadores. Segundo Cervenka e Trencansky (2004), o pacote com aspectos mentais oferece suporte à modelagem de problemas complexos, como modelagem de

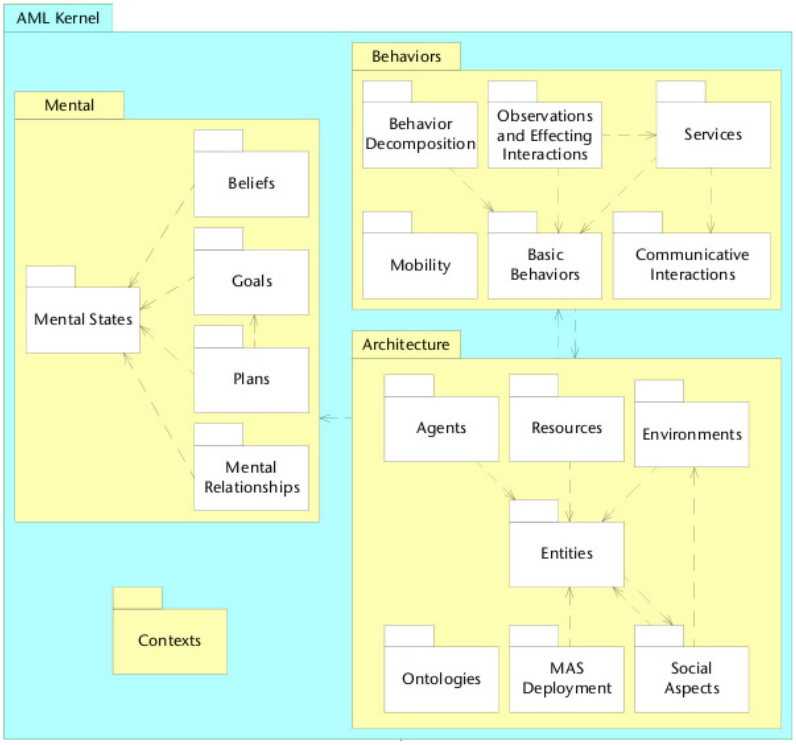

Figura 2.23: Pacotes da linguagem AML

requisitos baseados em metas, modelagem de intencionalidade em casos de uso e decomposição de problemas. Neste pacote existem diagramas para a modelagem de planos, metas e crenças.

2.3.5 Comparação entre as Linguagens de Modelagem

Muitos estudos têm sido realizados para avaliação de metodologias orientadas a agentes, como pode ser encontrado em Tran e Low (2005), Far, Mudigonda e Elamy (2009), LIN et al. (2007) e Ibrahim, Far e Eberlein (2009). Em relação às linguagens de modelagem, pouco tem sido feito nesse sentido, dificultando o trabalho de projetistas que se vêem obrigados a estudar uma linguagem de modelagem específica para saber se ela aborda os conceitos desejados no SMA. Buscando auxiliar pesquisadores e programadores a escolher a linguagem de modelagem que mais se adapte à sua aplicação, realizamos uma avaliação entre elas. Essa comparação é baseada apenas nos conceitos que cada linguagem é capaz de demonstrar.

Os conceitos esperados em uma linguagem de modelagem são:

- Metas: avalia se a linguagem é capaz de descrever os objetivos do agente;
- Crenças: avalia se a linguagem é capaz de descrever as crenças do agente;
- Ambientes: avalia se é possível modelar o ambiente;

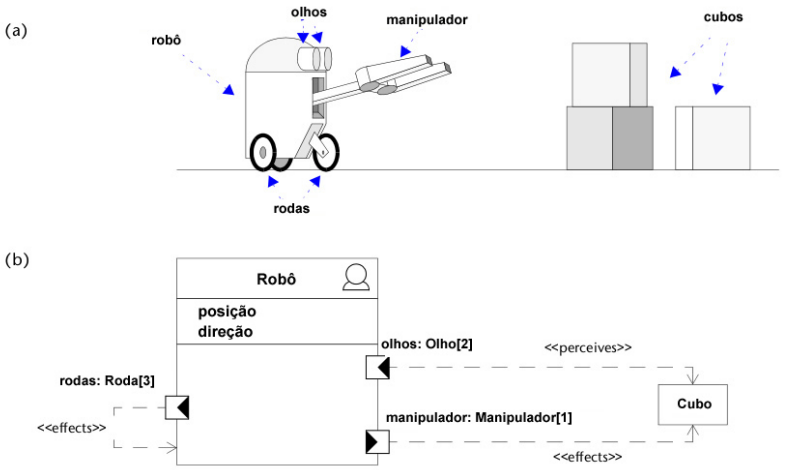

Figura 2.24: a) Mundo dos blocos b) Classe Agente com sensores e atuadores (CERVENKA; TRENCANSKY, 2004)

- Organização: define se é possível demonstrar a organização;
- Planos: avalia a possibilidade de descrever planos;
- Recursos: define se o conceito de recursos pode ser modelado;
- Papéis: avalia se é possível descrever os papéis;
- Ontologias: avalia se é possível descrever ontologias com a linguagem;
- Serviços: define se existem notações que representem os serviços oferecidos pelos agentes;
- Sensores e Atuadores: avalia se existem notações capazes de descrever os sensores e atuadores do agente;
- Restrições/Normas: avalia a possibilidade de descrever restrições em uma organização de agentes;
- Mobilidade: define se é possível modelar a mobilidade dos agentes;
- Protocolos de Interação: capacidade de definir protocolos de interação entre agentes;
- Técnicas de IA: define se é possível descrever as técnicas de IA que o agente irá utilizar;
- Arquiteturas Internas: capacidade de modelar a arquitetura interna de um agente.

Conceitos	AUML	MAS-ML	AORML	AML
Metas	Não	Sim	Não	Sim
Crenças	Sim	Sim	Sim	Sim
Ambientes	Não	Sim	Não	Sim
Organização	Sim	Sim	Sim	Sim
Planos	Não	Sim	Não	Sim
Recursos	Não	Não	Não	Sim
Papéis	Sim	Sim	Sim	Sim
Ontologias	Não	Sim	Não	Sim
Serviços	Não	Não	Não	Sim
Sensores e Atuadores	Não	Sim	Sim	Sim
Normas	Não	Não	Sim	Sim
Mobilidade	Não	Não	Não	Sim
Protocolos de interação	Sim	Sim	Sim	Sim
Técnicas de I.A	Não	Não	Não	Não
Arquiteturas Internas	Não	Sim	Não	Não

Tabela 2.6: Análise das linguagens de modelagem

3 Extensão Proposta para a Metodologia MAS-CommonKADS

Com o que foi apresentado no capítulo anterior, percebe-se que uma das grandes falhas das metodologias existentes ocorre na representação do comportamento racional dos agentes. Nenhuma das metodologias existentes tem capacidade de demonstrar quais técnicas de IA serão utilizadas e quais comportamentos serão realizados a partir de suas percepções. Além disso, elas não apresentam uma definição mais detalhada dos agentes, demonstrando qual arquitetura deve ser utilizada e como as informações recebidas serão processadas pelo agente. Este capítulo propõe uma extensão a metodologia MAS-CommonKADS com o objetivo de viabilizar o projeto de sistemas multiagentes racionais descrevendo todos os passos necessários a sua concepção. Para facilitar a descrição do projeto, a linguagem de modelagem AML foi escolhida para ser integrada a metodologia devido a grande quantidade de conceitos que ela é capaz de modelar e por possuir um metamodelo facilmente extensível para contemplar novos conceitos.

A abordagem proposta para a modelagem de sistemas multiagente racionais neste capítulo fundamenta-se em uma síntese que foi gerada a partir da integração dos trabalhos de (WOOLDRIDGE, 2002) e (RUSSEL; NORVIG, 1995) a respeito de agentes inteligentes, e de Kasabov a respeito da adequação de algumas das principais técnicas de Inteligência Computacional (IC/IA) utilizadas atualmente ao tipo de informação que está disponível para a tomada de decisão.

Assim, o capítulo foi organizado em quatro seções principais. As Seções 3.1 e 3.2 apresentam uma primeira contribuição deste trabalho de dissertação, ou seja, a descrição das arquiteturas de agentes racionais e alguns meios adequados de concretizá-las empregando-se técnicas de IA. Essas seções compõem a síntese identificada no parágrafo anterior. As Seções 3.3 e 3.4 apresentam às extensões a AML e a MAS-CommonKADS considerando a síntese. Essas seções compõem a extensão proposta nesta dissertação, a qual foi denominada MAS-COMMONKADS+.

3.1 Arquiteturas Abstratas de Agentes Racionais

Como visto no capitulo anterior, um agente interage com o ambiente através de seus sensores e age no mesmo através de atuadores. Mais especificamente, o agente (i) recebe, através dos seus sensores, informações a respeito do estado do ambiente, (ii) processa essa informação perceptiva (P), conforme especifica sua função de agente, com o objetivo de escolher uma ação (A) e, finalmente, (iii) executa a ação escolhida através dos seus atuadores. A Figura 3.1 demonstra essa interação.

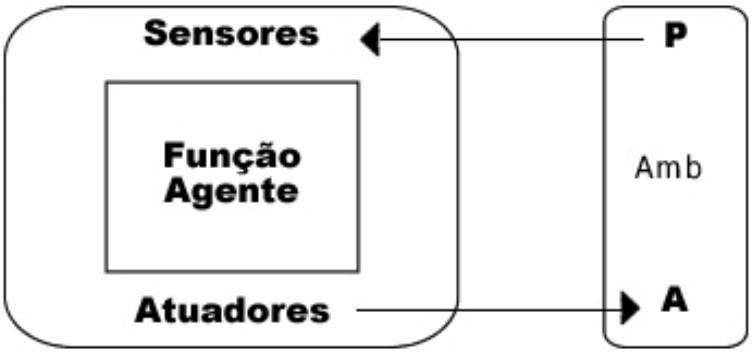

Figura 3.1: Interação do Agente com o Ambiente

Por agentes racionais entende-se um agente capaz de selecionar ações para serem executadas em seu ambiente por seus atuadores, que maximize a medida de avaliação de desempenho preestabelecida pelo projetista do sistema, conforme o mesmo espera dadas as informações perceptivas que chegam do ambiente pelos seus sensores, e todo e qualquer conhecimento embutido previamente no sistema e/ou aprendido com o decorrer de sua experiência, é importante que as metodologias tenham a capacidade de modelar o comportamento interno dos agentes para que se possa expressar esse comportamento racional.

Conforme mencionado, a abordagem proposta fundamenta-se em uma síntese que foi gerada a partir da integração dos trabalhos de Wooldridge (2002), Russel e Norvig (1995) e Kasabov (1996). Mais especificamente, relacionada aos dois primeiros trabalhos, esta síntese considera a formalização da visão abstrata do agente padrão, os refinamentos na função de tomada de decisão do agente, descrita por (WOOLDRIDGE, 2002), e os quatro tipos de programas de agentes que incorporam os princípios subjacentes a quase todos os sistemas inteligentes, descritos por (RUSSEL; NORVIG, 1995).

Assim, mais especificamente, as quatro arquiteturas abstratas de agentes descritas nesta seção, por um lado, empregam a notação funcional utilizada na formalização da visão abs-

trata do agente padrão (WOOLDRIDGE, 2002), para representar os módulos (subsistemas) de processamento de informação responsáveis por cada etapa do processo de tomada de decisão nos quatro tipos de programas agentes (RUSSEL; NORVIG, 1995) e, por outro, acrescenta à noção de estado (do ambiente) interno mantido pelo agente descrito na formalização (WOOLDRIDGE, 2002), outros tipos de informações que são utilizadas em cada uma das etapas nos programas agentes (RUSSEL; NORVIG, 1995).

A Figura 3.2 apresenta a estrutura do Agente Reativo Simples de acordo com a síntese. Neste caso, foram considerados o primeiro refinamento na função de tomada de decisão do agente padrão realizado por (WOOLDRIDGE, 2002) e as sugestões de (RUSSEL; NORVIG, 1995) sobre a utilização de informações no formato de regras condição-ação para auxiliar na tomada de decisão. Neste caso, em vez de regras, na síntese realizada, esta informação foi denominada *associações condição-ação* buscando considerar também a situação em que as informações sobre o domínio do problema estão disponíveis no formado de dados.

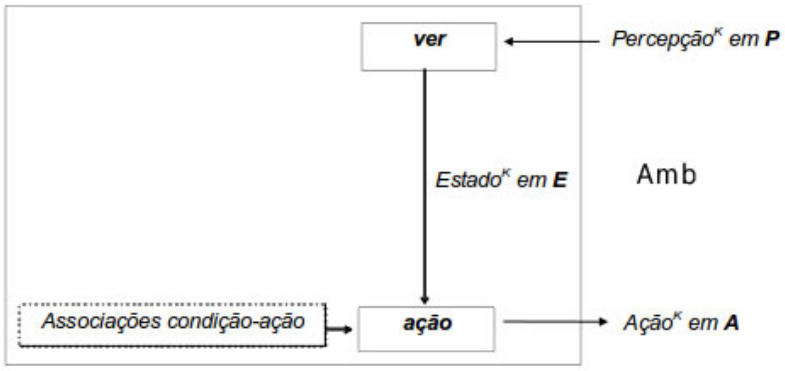

Figura 3.2: Agente Reatvo Simples

A arquitetura do agente reativo inteligente supõe que em qualquer instante k: (1) por meio de sensores o agente recebe informações do ambiente, $Percepção^K$, definidas em um conjunto $\boldsymbol{P} = \{Percepçaõ_1, ..., Percepção_n\}$, de n percepções possíveis para o ambiente; (2) o subsistema de percepção, ***ver:*** $\boldsymbol{P} \rightarrow \boldsymbol{E}$, processa cada percepção e mapeia em um de m estados possíveis, $Estado^K$, definidos em um conjunto $E = \{Estado_1, ..., Estado_m\}$, que são representações de aspectos do estado do ambiente que estão acessíveis ao agente para a tomada de decisão; (3) o subsistema de tomada de decisão, ***ação:*** $\boldsymbol{E} \rightarrow \boldsymbol{A}$, processa as informações sobre o estado em E, e, considerando um conjunto de associações entre determinadas condições que se estabelecem no ambiente e ações que se presumem serem adequadas às condições, *INFOassociação*, seleciona uma de l ações, $Ação^K$, definidas em um conjunto de ações possíveis para o agente, $A = \{Ação_1, ..., Ação_l\}$; (4) por meio de atuadores, o agente envia a ação selecionada para o ambiente.

função agente reativo (Percepção) **retorna** uma ação
entrada Percepção em **P,** uma percepção
var estáticas: INFOassociaçao, informação suplementar para a tomada de decisão
Estado em E, uma descrição do estado atual
Ação em A, uma ação mais recente, inicialmente nenhuma
Estado ← ver(percepção)
Ação ← ação(Estado, INFOassociação)
retorna Ação

Figura 3.3: Programa Agente Reativo

Mais especificamente, as associações condição-ação (INFOassociação) são informações suplementares para a tomada de decisão compostas por um conjunto de associações comuns observadas entre certas condições estabelecidas a partir das descrições de estados do ambiente definidos em E e certas ações definidas em A. Conforme destacado, essas informações podem estar disponível no formato de uma teoria (conjunto de proposições como, por exemplo, regras) e/ou de anotações em planilhas (banco) de dados. O pseudocódigo do programa agente reativo simples pode ser visualizado na Figura 3.3.

Conforme pode ser observado, em cada instante k, o agente reativo simples realiza um ciclo composto de três passos principais: (i) percebe o estado do ambiente $Percepção^K$ e gera uma descrição do estado ambiente $Estado^K = ver(Percepção^K)$; (ii) seleciona uma ação por meio de $Ação^K = ação(Estado^K, INFOassociação)$ e; (iii) executa a ação selecionada.

O agente reativo baseado em modelos realiza um número de passos maior que o agente reativo simples, já que ele mantém e atualiza uma descrição de estado interno do ambiente. Esta nova arquitetura consiste em um refinamento do subsistema de tomada de decisão do agente reativo. Na síntese, ela foi gerada considerando uma lógica semelhante a empregada para o caso reativo simples. A Figura 3.4 apresenta a estrutura do agente reativo baseado em modelos.

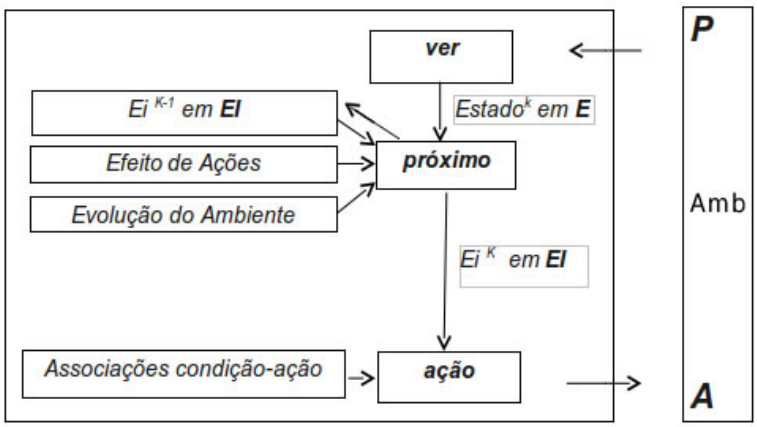

Figura 3.4: Agente Baseado em Modelos

função agente-reativo-modelos(Percepção) retorna uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
INFOefeito, informação suplementar para a atualização de estado interno
INFOevolução, informação suplementar para a atualização de estado interno
INFassociação, informação suplementar para a tomada de decisão
Estado em E, uma descrição de estado atual do ambiente
Ei em EI, uma descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Estado ← ver(percepção)
Ei ← próximo(Estado, Ei, INFOefeito,INFOevolução)
Ação ← ação(Ei,INFOassociação)
retorna Ação

Figura 3.5: Programa Agente Baseado em Modelos

No agente baseado em modelos, o subsistema de processamento de informação relacionado à tomada de decisão sofreu um novo refinamento dando origem ao subsistema de atualização de estado interno, ou seja, ***próximo***. Esta decomposição altera o funcionamento do agente reativo simples no terceiro passo **(3)** de sua operação. Mais especificamente, depois do subsistema de percepção ***ver*** mapear uma percepção do ambiente, $Percepção^K$ em P, em uma descrição de estado, $Estado^K$ em E, **(3.1)** o subsistema de atualização de estado interno, ***próximo: EIxE*** $\rightarrow$ ***EI***, empregando informações sobre o efeito das ações possíveis, *INFOefeito*, e sobre a maneira como o mundo evolui independente das ações do agente, *INFOevolução*, mapeia este estado e o estado interno corrente Ei^{K-1}em um novo estado interno Ei^K, definidos em um conjunto de p estados internos possíveis $\boldsymbol{EI} = \{Ei_1, ..., Ei_p\}$; que, por sua vez, **(3.2)** é processado pelo subsistema de tomada de decisão, ***ação: EI*** $\rightarrow$ ***A***, para selecionar uma ação possível $Ação^K$ em A baseado nas informações sobre associações condição-ação, INFOassociação.

As informações sobre efeito de ações (INFOefeito) e evolução do ambiente (INFOevolução) referem-se às informações suplementares disponíveis para atualização de estado interno e, assim como no caso de INFOassociação, podem estar disponíveis no formato de uma teoria (conjunto de proposições) e/ou de anotações em planilhas (banco) de dados. O pseudocódigo do programa agente reativo baseado em modelos pode ser visualizado na Figura 3.5.

Conforme pode ser observado, em cada instante k, o agente reativo baseado em modelos realiza um ciclo composto de quatro passos principais: (i) percebe o estado do ambiente $Percepção^K$, e gera uma descrição do estado ambiente, $Estado^K = ver(Percepção^K)$; (ii) atualiza o estado interno do ambiente, $Ei^K = próximo(Ei^{K-1}, Estado^K, INFOefeito, INFOevolução)$; (iii)

seleciona uma ação por meio de $Ação^K = ação(Ei^K, INFOassociação)$; e (iv) executa a ação.

Em vez de informações a respeito de associações condição-ação, os agentes baseados em objetivos tomam suas decisões buscando realizar seus objetivos. Nesta nova arquitetura, os objetivos constituem uma nova informação suplementar, *Objetivos*, que estará disponível para a tomada de decisão do agente e que lhe servirá como condição de parada. A Figura 3.6 apresenta a estrutura do agente baseado em objetivos de acordo com a síntese realizada nos pontos de vista de (WOOLDRIDGE, 2002) e (RUSSEL; NORVIG, 1995).

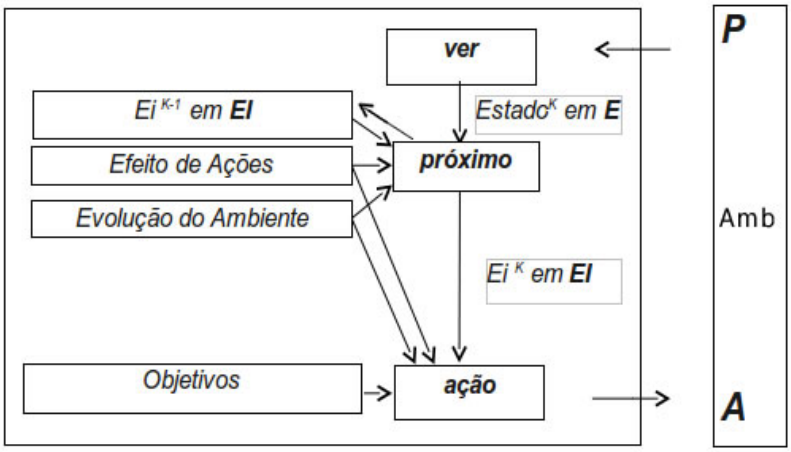

Figura 3.6: Agente Baseado em Objetivos

Conforme a figura indica, o funcionamento dos agentes baseados em objetivos pode ser visto como uma extensão do caso anterior. Apesar de não estar explícito um novo refinamento da função de tomada de decisão, neste novo tipo de programa agente, conforme Russel e Norvig (1995) indicam, é necessário que se especifique um novo subsistema de processamento de informação que seja capaz de considerar as informações em *INFOefeito* e *INFOevolução*, e de calcular o efeito de cada uma das ações em A, sobre a descrição de estado interno corrente Ei^K.

Os objetivos podem ser descritos a partir de um conjunto de descrições de estados desejados ou por meio de alguma descrição do que vem a ser um estado desejado no ambiente como, por exemplo, por meio da especificação de um teste objetivo. ***Teste_objetivo: EI ({V,F}***. Neste último caso, as informações em *Objetivos*, para a concepção de *Teste_objetivo*, também podem estar disponíveis no formato de uma teoria (conjunto de proposições) e/ou de anotações em planilhas (banco) de dados. O pseudocódigo do programa agente baseado em objetivos pode ser visualizado na Figura 3.7.

A seleção da ação baseada em objetivo pode ser direta, quando o resultado de uma única ação realiza o objetivo, ou de uma maneira mais complexa, quando for necessário con-

função agente-baseado-objetivo(Percepção) retorna uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
INFOefeito, informação suplementar para a atualização de estado interno e tomada de decisão
INFOevolução, informação suplementar para a atualização de estado interno e tomada de decisão
Estado em E, uma descrição de estado atual do ambiente
Ei em EI, uma descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Objetivo, um conjunto de descrições de estados desejados ou alguma descrição do que vem a ser um estado desejado no ambiente
Solução, ação ou seqüência de ações que alcança/realiza o objetivo
Estado $\leftarrow$ ver(Percepção)
Ei $\leftarrow$ próximo(Estado, Ei,INFOefeito,INFOevolução)
Ação $\leftarrow$ ação(Ei, INFOefeito,INFOevolução,Objetivo,Solução)
retorna Ação

Figura 3.7: Programa do Agente Baseado em Objetivos

ceber longas seqüências de ações para realizar o objetivo. Neste caso, conforme indicado por Russel e Norvig (1995), para encontrar seqüências de ações que alcançam os objetivos, a função tomada de decisão pode ser concebida a partir de algoritmos de busca e planejamento.

Em alguns problemas enfrentados por agentes baseados em objetivo pode acontecer de existirem várias seqüências de ações distintas capazes de realizar o objetivo. Estes agentes não diferenciam estas sequências em termos de melhor ou pior, simplesmente se satisfazem com qualquer sequência que realize o objetivo. Se for imposta ao agente alguma medida de avaliação que considere outro critério, além da realização do objetivo, pode ser necessário que se especifique para o agente uma função, Utilidade, que consiga medir o grau de satisfação do proporcionado por uma ação, ou sequência de ações, na realização do objetivo. A Figura 3.8 apresenta a estrutura do agente baseado em utilidade de acordo com a síntese realizada nos pontos de vista de (WOOLDRIDGE, 2002) e (RUSSEL; NORVIG, 1995).

Mais especificamente, a função serve para orientar o agente em relação a utilidade de um estado realizado (previsto) por uma ação no ambiente. Se um estado do mundo é mais desejável que outro, então ele terá maior **utilidade** para o agente. A **Utilidade** é uma função que mapeia um estado para um número real que representa o grau de satisfação com este estado, **Utilidade: EI** $\rightarrow$ **Reais**.

Conforme a Figura 3.9 indica, na síntese o comportamento do agente baseado em utilidade é semelhante ao do agente baseado em modelos.

Assim como as outras informações suplementares empregadas nas outras etapas do

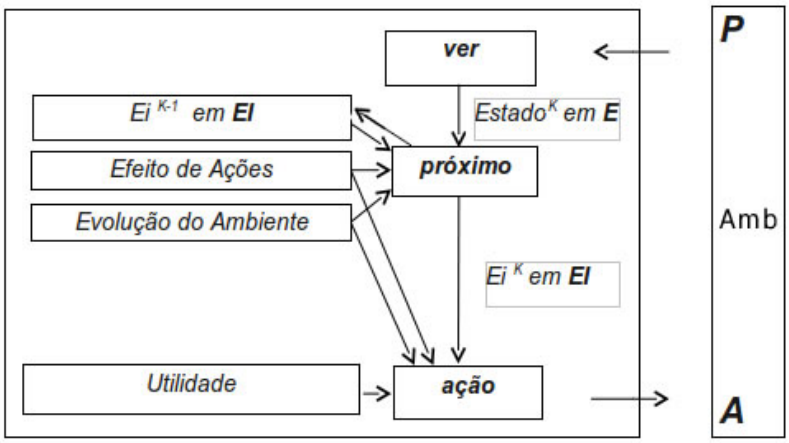

Figura 3.8: Agente Baseado em Utilidade

função agente-baseado-utilidade(Percepção) retorna uma ação
Entradas: Percepção em P, uma percepção
Variáveis estáticas:
INFOefeito, informação suplementar para a atualização de estado interno e tomada de decisão
INFOevolução, informação suplementar para a atualização de estado interno e tomada de decisão
Estado em E, descrição de estado atual do ambiente Ei em EInterno, descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma Utilidade, função que mede a utilidade de um estado
Solução, ação ou ações que satisfazem algum critério de utilidade
Estado ← ver(Percepção)
Ei ← próximo(Estado, Ei,INFOefeito,INFOevolução)
Ação ← ação(Ei,INFOefeito,INFOevolução,Utilidade,Solução)
Retorna Ação

Figura 3.9: Programa do Agente Baseado em Utilidade

processo de tomada de decisão, informações para gerar a função utilidade podem estar disponíveis no formato de uma teoria (conjunto de proposições) e/ou de anotações em planilhas (banco) de dados.

3.2 Concretização dos agentes com Inteligência Computacional

A concretização adequada de um programa agente racional vai depender da especificação de uma função agente adequada, o que, por sua vez, dependerá da forma em que as informações empregadas nas diversas etapas do processo de tomada de decisão (INFOAssociação, INFO-Efeito, INFOEvolução, Objetivo e Utilidade) estão disponíveis no domínio do problema. A

síntese iniciada na seção anterior considera que as informações podem estar disponíveis no formato de uma teoria (conjunto de proposições) ou de anotações em planilhas (banco) de dados.

Dentro do contexto da Inteligência Computacional existem diversas técnicas que podem ser utilizadas para representar os diversos tipos de proposições e de aprender a partir de dados que foram anotados previamente. A síntese iniciada na seção anterior, considera o ponto de vista de Kasabov (1996) a respeito da adequação das técnicas ao tipo de informação disponível no domínio do problema. A Figura 3.10 indica como essas informações podem ser tratadas empregando-se principalmente: Redes Neurais, Sistemas Fuzzy, IA Simbólica, Raciocínio Probabilístico e Busca.

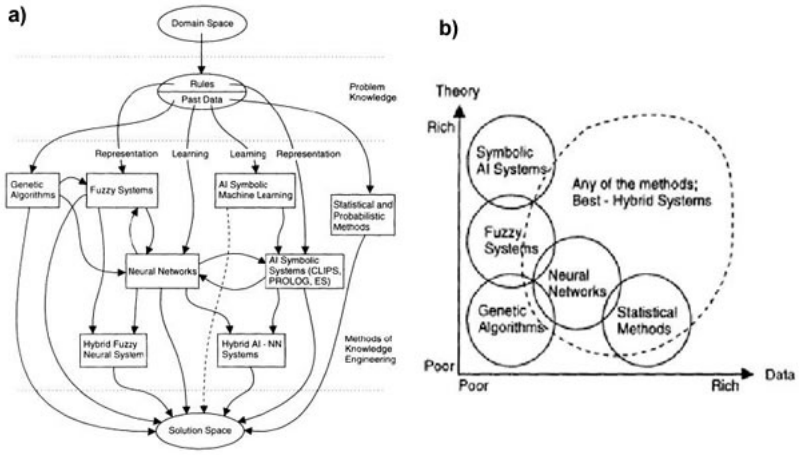

Figura 3.10: Possibilidade de mapeamento das informações (KASABOV, 1996)

A Figura 3.10(a) passa uma ideia em torno das possibilidades de se empregar as técnicas de IC para se representar e/ou aprender as informações conforme elas estejam disponíveis e, consequentemente, de se concretizar as quatro arquiteturas abstratas de agentes sintetizadas na seção anterior. A Figura 3.10(b) destaca a adequação das técnicas em função da quantidade de informação disponível. A seguir, é exemplificado quatro das possibilidades de funções agentes que podem ser especificadas levando-se em consideração a figura 3.10 e que foram consideradas em parte da avaliação realizada na extensão proposta nesta dissertação.

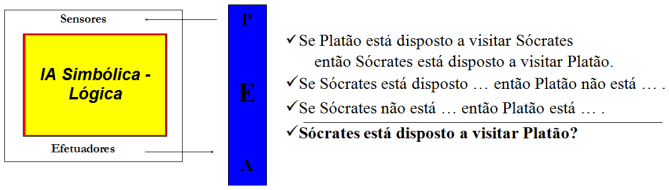

Figura 3.11: Exemplos de funções agentes para lógica

A figura 3.11 identifica uma função agente concretizada empregando-se técnicas da

IA simbólica e, mais especificamente, os sistemas formais lógicos para representar as teorias disponíveis e conceber os módulos de processamento da informação. Por exemplo, considerando que é possível representar adequadamente em lógica de predicados de primeira ordem o conjunto de proposições que compõe a teoria disponível, o quadro indica que os módulos podem ser concebidos, fundamentalmente, como provadores automáticos de teoremas e outros programas auxiliares que, integrados, devem ser capazes de raciocinar com as representações em lógica e selecionar as ações para o agente.

A figura 3.12 considera funções agentes para resolver problemas bem definidos empregando estratégias de busca. Neste caso, as informações disponíveis a respeito dos estados corrente e desejado do problema são representadas por meio de estruturas simbólicas, e as informações a respeito das ações possíveis para o agente são representadas por meio de uma função sucessor de estados, que especifica, para cada ação possível a partir de um estado corrente, a mudança na estrutura simbólica que representa o estado corrente, ou seja, o efeito da execução de uma ação. Estratégias de busca para resolver o tipo de problema em destaque exploram alternativas de ações e encontram uma sequência que transforme um estado corrente em um estado desejado. A exploração sistemática do espaço de estados do problema pode ocorrer cegamente como, por exemplo, empregando busca em profundidade e largura, ou baseada em funções avaliação de estados que medem: o custo de ações ($g(n)$) como, por exemplo, a estratégia custo uniforme; e/ou uma estimativa do custo de se chegar a uma solução a partir de um determinado estado ($h(n)$) como, por exemplo, a busca gulosa.

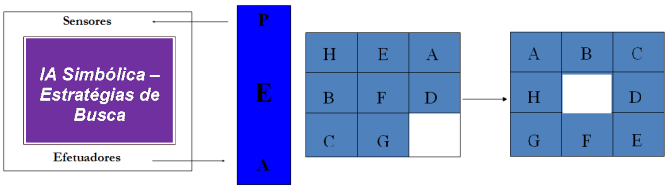

Figura 3.12: Exemplo de função agente para buscas

Na figura 3.13 é considerado o problema de reconhecimento de caracteres escritos à mão. Pressupõe-se que a informação para resolver o problema está disponível em uma certa quantidade de exemplos que podem ser utilizados para aprendizagem em uma rede neural. Uma rede treinada de maneira razoável tem a habilidade de generalizar quando são apresentadas entradas que não estão presentes nos exemplos de treinamento. A especificação de uma rede neural para um problema envolve a determinação de aspectos estruturais, como é o caso do número de camadas, número de neurônios por camada e tipos de conexões entre neurônios; e a escolha de um algoritmo capaz de aprender a partir de exemplos significativos como, por exemplo, o algoritmo back-propagation para o treinamento de redes multicamadas diretamente

alimentadas.

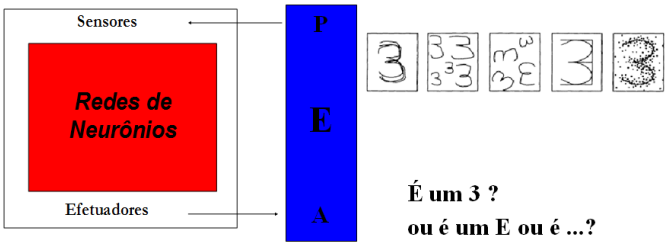

Figura 3.13: Exemplo de função agente para redes neurais

A figura 3.14 considera uma situação semelhante a da figura 3.11. Entretanto, diferentemente, as proposições que compõem as teorias estão sobrecarregadas de imprecisão, devido aos termos lingüísticos subjetivos, ambíguos e fuzzy presentes no discurso ordinário humano. Neste novo tipo de situação, os conjuntos fuzzy e, mais especificamente, os sistemas fuzzy são empregados para representar este tipo de informação e raciocinar com as mesmas. A especificação de uma função agente fundamentada na teoria fuzzy envolve a quantização matemática dos termos lingüísticos envolvidos e das relações definidas nas proposições; bem como a escolha de um método de inferência fuzzy que seja adequado ao problema sobre consideração como, por exemplo, os métodos Zadeh-Mamdani, Mizumoto e Sugeno.

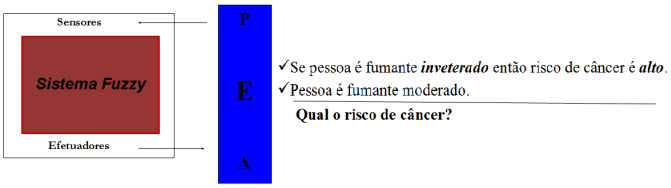

Figura 3.14: Exemplo de função agente para lógica fuzzy

Os exemplos demonstrados não consideram a situação em que a função agente é concretizada como um sistema híbrido, ou seja, um sistema que combine várias técnicas de IA para a resolução de um problema. Entretanto, a princípio, não existem restrições quanto à combinação de técnicas, as arquiteturas abstratas/estruturas de programa agente propostas, na síntese descrita na seção anterior, podem ser concretizadas considerando que as informações para tomada de decisão (INFOAssociação, INFOEfeito, INFOEvolução, Objetivo e Utilidade) estão disponíveis em planilhas de dados e conjuntos de proposições, inclusive, proposições fuzzy.

3.3 Extensões à Linguagem AML

As modificações realizadas na linguagem AML tem como objetivo o suporte a agentes racionais conforme as arquiteturas descritas anteriormente. A linguagem de modelagem que foi escolhida para integrar a metodologia MAS-CommonKADS oferece mecanismos que são capazes de descrever o comportamento interno de um agente através de um pacote chamado *behavior*. Com as entidades que compõe este pacote, conseguimos modelar alguns comportamentos internos dos principais tipos de agentes racionais, ou seja:

- *behavior fragment*: é responsável por decompor o comportamento em fragmentos que serão executados de acordo com a percepção;
- *perceptor*: é uma entidade capaz de perceber modificações no ambiente;
- *perceptAction*: é uma entidade responsável por tratar percepções específicas capturadas e acionar o comportamento interno do agente;
- *effector*: é responsável pelas ações efetuadas;
- *effectorAction*: é uma entidade responsável por executar ações acionadas por um comportamento.

Apesar de ser possível modelar as principais arquiteturas de agentes racionais, estes comportamentos não foram origionalmente criados com este propósito. Por isso, foi necessário estender a linguagem com o objetivo de englobar aspectos importantes para o modelo racional dos agentes.

Foi incorporado ao pacote *Behaviors* um novo pacote chamado *Rational Behaviors*. Este possui entidades que servem para descrever comportamentos especializados que auxiliam no modelo de desenvolvimento de agentes racionais. A figura 3.15 demonstra o novo pacote proposto.

Com duas novas entidades contidas neste pacote, é possível para o projetista modelar de forma correta o comportamento dos agentes reativos, baseados em modelos, baseados em objetivos e baseados em utilidade. As subseções a seguir descrevem as duas entidades contidas dentro do novo pacote e demonstram como as extensões são utilizadas.

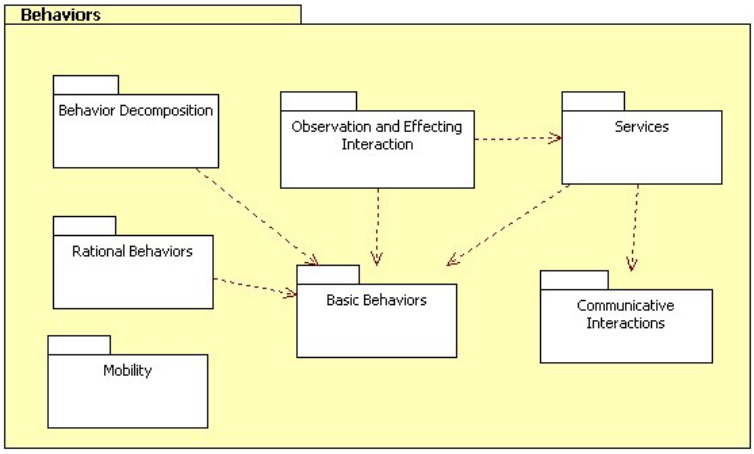

Figura 3.15: Pacotes de Behaviors

3.3.1 Entidade *Skill*

A entidade skill é uma extensão da AML com o objetivo de possibilitar a linguagem representar os comportamentos internos do agente através de técnicas de IA. Ela também é utilizada para auxiliar a modelagem dos agentes racionais. Ela é uma extensão da superclasse *behaviorSemiEntity* capaz de representar lógica, busca, redes neurais e os sistemas *fuzzy*. A Figura 3.16 demonstra o metamodelo de *skill*.

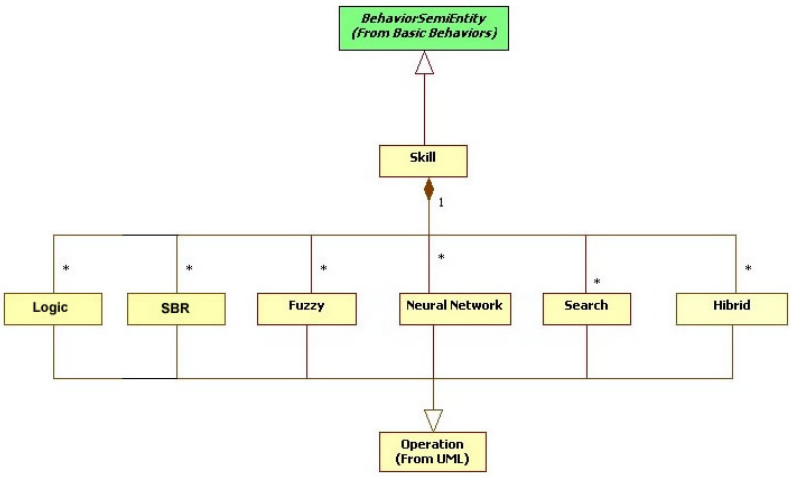

Figura 3.16: Entidade *skill*

As técnicas que skill é capaz de demonstrar são:

- *logic*: é utilizado para a modelagem de agentes que tomam suas decisões baseados em lógica;

- *SBR*: é utilizado para a modelagem de agentes que tomam suas decisões através de sistemas baseados em regras;
- *fuzzy*: é utilizada para auxiliar na construção de agentes que utilizam lógica fuzzy;
- *neural network*: possibilita a modelagem de agentes que utilizam redes neurais;
- *search*: é utilizada para modelar os mais diferentes tipos de busca como, por exemplo, busca em extensão, busca em profundidade, metaheuristicas, entre outras;
- *hibrid*: é utilizado quando a modelagem necessita da união entre duas técnicas.

Esta nova entidade é muito importante para a modelagem do comportamento racional do agente, pois através dela conseguimos representar como as informações serão tratadas e como a função agente é especificada e, consequentemente, é realizada a tomada de decisão dos agentes racionais. A Figura 3.17 demonstra como pode ser modelado um agente que possui um *skill* e como são definidas as operações deste comportamento.

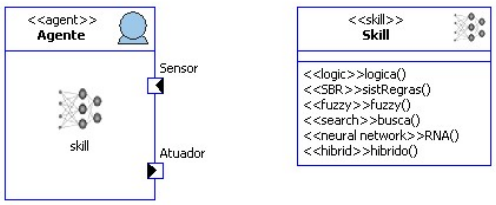

Figura 3.17: Agente com um *skill*

Relacionando o diagrama apresentado na figura 3.17 com as arquiteturas demonstradas na seção 3.1 e na seção 3.2, podemos observar na figura os sensores e atuadores do agente, este diagrama também contempla uma entidade *skill*, que é uma entidade especializada para a definição de um comportamento interno do agente como, por exemplo, a escolha de uma determinada ação através de lógica. Esta entidade define quais as técnicas que serão usadas pelo agente para a tomada de uma decisão ou para a atualização do seu estado interno. Então podemos observar que o diagrama poderia ser utilizado para representar um agente reativo simples, pois ele contém sensores e atuadores e um comportamento interno para a tomada de decisão.

3.3.2 Entidade Rational Behavior Formulation

A *rational behavior formulation* é uma entidade que descreve um comportamento especializado do novo pacote que é responsável por modelar características específicas dos agen-

tes baseados em objetivos e em utilidade. Esta entidade foi estendida a partir da classe abstrata *BehavioredSemiEntityType* da AML que serve como superclasse para todas as entidades que descrevem comportamentos.

O *rational behavior formulation* é uma entidade especializada para auxiliar na definição dos comportamentos internos dos agentes baseados em objetivos e em utilidade, ela engloba as características responsáveis por formular o objetivo do agente dado um determinado estado corrente e selecionar uma ação de acordo com esse objetivo. A Figura 3.18 demonstra o metamodelo de *rational behavior formulation*.

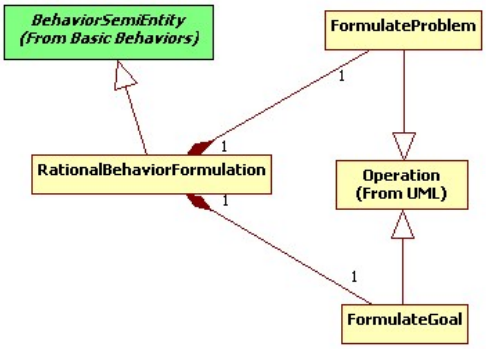

Figura 3.18: Entidade *rational formulation behavior*

Como visto na figura, *rational behavior formulation* apresenta *formulateGoal* e *formulateProblem*. O primeiro possibilita a formulação dos objetivos desejados (GONçALVES et al., 2010), enquanto o segundo é responsável por analisar os possíveis estados e ações a serem considerados dado um determinado objetivo.

Esta entidade, em conjunto com uma entidade *skill* para modelar atualização do estado interno do agente, permite a modelagem de diversas opções de agentes racionais baseados em objetivo ou em utilidade. A Figura 3.19 demonstra como é modelada a entidade *rational behavior formulation*.

Considerando a extensão da linguagem AML para o suporte correto das arquiteturas de agentes racionais, a seção a seguir define o processo de modelagem para a metodologia MAS-CommonKADS representar estes tipos de agentes.

3.4 Extensões à metodologia MAS-CommonKADS

A metodologia MAS-CommonKADS foi selecionada pelas seguintes razões (MENDES; II, 2010):

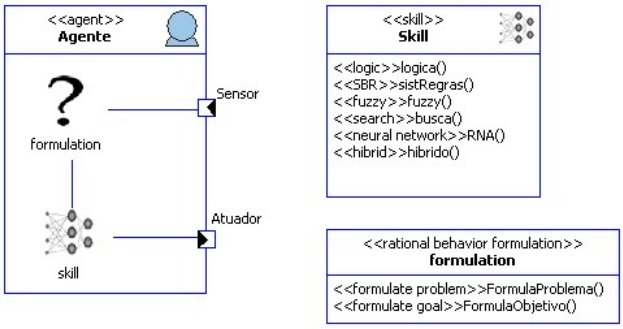

Figura 3.19: Agente com um *rational behavior formulation*

- Explica detalhadamente todos os modelos empregados e demonstra como desenvolver seus respectivos diagramas, além de possuir uma fase inicial de conceituação na qual se pode abordar o problema a tratar com facilidade;
- Emprega técnicas de metodologias orientadas a objetos que permitem fácil compreensão por parte dos desenvolvedores de software;
- Utiliza o modelo em espiral que avalia os riscos existentes permitindo assim o planejamento de estratégias alternativas para o projeto com o objetivo de não ultrapassar o cronograma e os custos;
- Deixa aberta a possibilidade de empregar uma arquitetura independente para cada agente, promovendo mais flexibilidade ao desenvolvimento da aplicação;
- A metodologia é uma extensão do CommonKADS que é uma metodologia que já foi aplicada com êxito em diversos projetos.

Além do mais, MAS-CommonKADS é uma metodologia capaz de representar com grande quantidade de detalhes os aspectos relacionados ao conhecimento dos agentes sendo uma das poucas a apresentar a possibilidade de utilizar ontologias e também é a única apresentar algumas características relacionadas aos agentes como, por exemplo, serviços, planos e condições de ativação e parada.

Assim, a extensão proposta contribuirá para a evolução de MAS-CommonKADS, permitindo a modelagem das arquiteturas de agentes racionais propostas nas duas primeiras seções, e, além disso, um ponto importante que as metodologias orientadas a agentes deixam de lado, a integração de agentes e objetos. Na maioria das metodologias essa questão não é tratada, o que obriga o projetista a modelar o sistema completo como um SMA ou a descrever os objetos através de uma outra metodologia. Visando solucionar esses problemas, esta seção propõe uma

nova arquitetura para a metodologia MAS-CommonKADS mantendo muitos dos modelos já propostos, mas realizando algumas modificações e adicionando novos conceitos. A Figura 3.20 demonstra o novo esquema proposto.

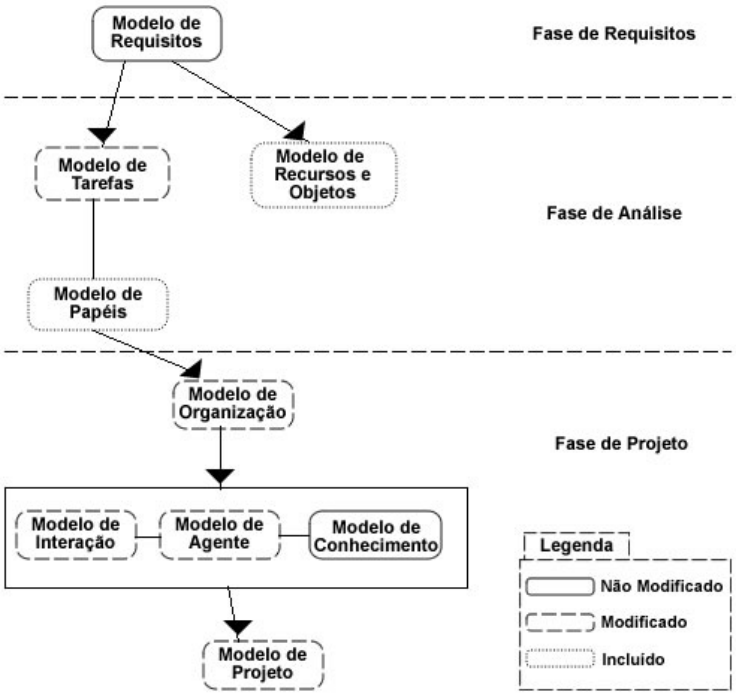

Figura 3.20: Nova arquitetura proposta para MAS-CommonKADS

Na nova arquitetura, a fase de contextualização do sistema, o modelo de tarefas e o modelo de conhecimento continuam sendo definidos da mesma maneira que era anteriormente, porém a fase de contextualização foi renomeada para fase de Requisitos, onde existe um modelo de requisitos responsável por capturar todos os requisitos do sistema. A metodologia, ao invés de definir sete modelos, agora contém nove modelos com vários passos responsáveis pela modelagem do sistema. Durante todo o projeto, o projetista irá incrementar os modelos até finalizar todas as fases de especificação do sistema. Todos os modelos são representados através de notações da AML, que auxiliam no processo de desenvolvimento, considerando as extensões propostas para esta linguagem.

Os modelos necessários para o processo de desenvolvimento do sistema multiagente são:

- **Modelo de Requisitos:** É responsável por descrever os requisitos do sistema, sendo dividido em análise por casos de uso e cenários, análise por objetivos e análise do ambiente;

- **Modelo de Tarefas:** Especifica as tarefas do sistema através da decomposição de diagramas e de *templates* textuais que auxiliam na descrição das tarefas;
- **Modelo de Papéis:** Descreve os papéis da organização e quais tarefas são de sua responsabilidade;
- **Modelo de Agentes:** Especifica os agentes e quais papéis eles são responsáveis, as percepções, os atuadores, as condições de ativação e de parada e a arquitetura do agente;
- **Modelo de Organização:** Descreve a organização entre papéis;
- **Modelo de Conhecimento:** Descreve o conhecimento do agente e a aquisição de novos conhecimentos;
- **Modelo de Interação:** Especifica regras para a coordenação e a comunicação entre agentes e objetos;
- **Modelo de Recursos e Objetos:** Descreve os recursos e os objetos do sistema;
- **Modelo de Projeto:** Descreve as características onde o sistema será instalado, os diagramas de *deployment* e a mobilidade dos agentes.

Os modelos de papéis e de recursos e objetos foram adicionados a metodologia com o objetivo de auxiliar na descrição de agentes e objetos, enquanto que os modelos de organização, interação e de projeto foram alterados de forma a contemplar a especificação dos diagramas da AML. O modelo que sofreu mais alterações foi o do agente, este agora é capaz de demonstrar como o agente irá perceber e atuar no ambiente de acordo com seus comportamentos e planejamentos. As próximas seções apresentarão as modificações realizadas na metodologia, visando uma melhor especificação dos agentes racionais do SMA e objetos do sistema.

3.4.1 Modelo de Papéis

Um papel é uma abstração que define as tarefas que um agente deve realizar dentro de uma organização. Um único agente pode ser responsável por vários papéis em um sistema. Este modelo tem como objetivo a identificação dos papéis do sistema e a representação de papéis que realizam as tarefas descritas no Modelo de Tarefas. Em seu modelo original MAS-CommonKADS não é capaz de descrever papéis limitando o agente a exercer sempre o mesmo papel. A inclusão deste modelo é importante visto que um agente pode realizar vários papéis em um sistema, com isso o modelo de papéis permitirá a modelagem de forma correta destes

papéis. É importante observar que se um agente possui mais de um papel a arquitetura interna do agente será modelado através de seus papéis.

A identificação dos papéis pode ser realizada examinando o modelo de tarefas e definindo quais os papéis que podem realizar as tarefas. Outra maneira é construindo uma estrutura preliminar da organização do SMA e verificá-la em busca de novos papéis. O segundo passo desse modelo é realizar o mapeamento das tarefas que devem ser de responsabilidade de cada papel. Segundo Tran e Low (2008), esse passo deve ser realizado diversas vezes pois na prática a modelagem de um papel pode descobrir outras tarefas que não foram descritas no Modelo de Tarefas.

Um papel em AML é representado pela entidade *EntityRoleType* e é utilizado para representar comportamentos, tarefas, participação em interações e serviços oferecidos ou necessários a outras entidades. A Figura 3.21 demonstra a estrutura de um papel.

Figura 3.21: Estrutura de um papel

3.4.2 Modelo de Recursos e Objetos

Sabendo que em um sistema pode existir tanto entidades ativas como entidades passivas, percebe-se que é necessária uma maneira de poder modelar essas entidades passivas em uma metodologia. Em sua versão original MAS-CommonKADS não permite ao projetista modelar os objetos do sistema e sua interação com os agentes. Este modelo foi adicionado à metodologia com o objetivo de possibilitar a modelagem de objetos e recursos permitindo uma melhor definição do sistema.

Especificar recursos e objetos é necessário quando temos um sistema heterogêneo no qual temos agentes e recursos e/ou objetos que fornecem informação, serviços ou executam algum objetivo que não necessite de autonomia. Este modelo tem como objetivo identificar recursos e objetos de um SMA. Os recursos podem ser identificados a partir da análise do Modelo de Tarefas, do Modelo de Papéis ou através da análise dos requisitos. Um recurso é uma aplicação específica que fornece informação e/ou serviço ao SMA, como, por exemplo, um servidor de banco de dados, um servidor web ou serviços web. Após a identificação dos

recursos, o Modelo de Papéis deve ser novamente revisado em busca de encontrar novos papéis.

A AML possui uma entidade específica responsável por modelar recursos enquanto um objeto é modelado como uma classe comum. A Figura 3.22 demonstra o diagrama responsável por modelar um recurso.

Figura 3.22: Estrutura de um recurso

Neste modelo também devem ser definidos os outros diagramas relativos aos objetos como diagramas de classes, diagramas de seqüência de mensagens, diagramas de atividades e diagramas de colaboração.

3.4.3 Modelo de Organização

A estrutura organizacional do SMA é modelada através de papéis e seus relacionamentos. Grupos de papéis podem estar contidos dentro de uma unidade de organização e se relacionar com outros papéis. Este modelo, diferentemente de sua versão original, que representava a organização de agentes, tem como objetivo demonstrar a estrutura organizacional de papéis do sistema. Esta diferença é importante porque, enquanto a estrutura organizacional de papéis é uma estrutura estática e pode ser definida em tempo de projeto, a estrutura organizacional de agentes pode ser dinâmica, visto que um agente pode trocar de papel em tempo de execução (TRAN; LOW, 2008).

A estrutura organizacional do SMA pode ser baseada em qualquer dos estilos organizacionais a seguir (PARROTT; LACROIX; WADE., 2003):

- Ponto a ponto: nesta estrutura, todos os papéis trabalham como pares. Cada papel assume um *status* de autoridade igual aos outros;
- Hierárquico: esta estrutura organiza os papéis em uma hierarquia onde alguns papéis em uma camada mais baixa são subordinados a outros;
- Federado: esta estrutura organiza papéis em grupos ponto a ponto, onde em cada grupo existe um mediador que subordina os outros papéis;
- Híbrida: esta estrutura integra qualquer um dos estilos acima.

A AML modela uma organização de papéis através da unidade *OrganizationUnitType,* que pode demonstrar duas visões: uma externa, que apresenta a interação de grupos de papéis, e a visão interna, que apresenta a estrutura dos papéis, seus relacionamento e normas. Os relacionamentos existentes para esse modelo são: i) *peer*, usado para papéis com a mesma autoridade, sendo representado por uma seta parcialmente cheia; ii) *superordinate*, usado para expressar papéis com autoridade sobre seus subordinados, sendo representado por uma seta cheia; e, por fim, iii) o *subordinate*, representado por uma seta vazia. A Figura 3.23 demonstra a unidade *OrganizationUnitType* e os relacionamentos.

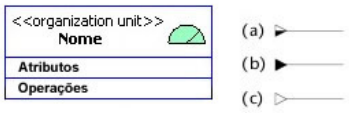

Figura 3.23: Estrutura de uma organização e seus relacionamentos

Uma Organização em AML possui atributos e operações como pode ser visualizado na figura, os relacionamentos que podem existir entre papéis de uma organização e entre organizações são a) peer, b) superordinate e c) subordinate (CERVENKA; TRENCANSKY, 2007)

3.4.4 Modelo de Agentes

Em seu projeto inicial, este modelo era responsável pelas diversas características dos agentes, como planos, serviços, definição de colaboradores, definição dos objetivos, entre outras. Todas estas características eram definidas por *templates* textuais e cartões CRC (Classes-Responsabilidades-Colaboradores). Era perceptível que, além de utilizar várias tecnologias em um só modelo, ele também utilizava algumas definições que seriam melhor definidos em outros modelos.

Na extensão proposta, o primeiro passo do Modelo de Agentes é descrever os agentes do sistema identificando-os e definindo os papéis realizados por eles. A identificação é feita através dos próprios papéis. Cada agente pode realizar um ou mais papéis, podendo, durante sua execução, trocar de papel de forma dinâmica.

Após a identificação dos agentes, é realizada a identificação dos comportamentos internos do agente de acordo com sua arquitetura. Um agente é composto por percepções, que constitui a função ver, atualização de estados, que representa a função próximo, tomada de decisão, que representa a função ação e por atuadores, que executam a ação do agente. Essas características são demonstradas neste modelo através da AML utilizando a entidade *agent* e as

entidades *skill* e *rational behavior formulation* para representar os comportamentos internos do agente, alêm destes é utilizado *templates* textuais que descrevem o tipo de informação presente em cada função do agente.

Para a definição correta das arquiteturas abstratas de agentes racionais propostas, foi necessário adicionar outras informações aos *templates* textuais do MAS-CommonKADS. A seguir demonstramos a planilha textual de um agente. O final da planilha apresenta as novas informações adicionadas.

Parâmetros de entrada texto. Dados recebidos através dos sensores. **Condição de ativação** texto. Condições para ativação do agente. **Condição de finalização** texto. Condições de parada do agente. **Condição de êxito** texto. Condições para que o agente tenha êxito na sua tarefa. **Condição de fracasso** texto. Condições para fracasso do agente. **Linguagem de representação do conhecimento (Opcional)** texto. Nome da Linguagem de representação do conhecimento empregada. **Ontologia (Opcional)** texto. Descreve as ontologias associadas ao agente.
CONTEÚDOS ADICIONADOS
Informação da Evolução [INFOEvolução] (Opcional) texto. Descreve o formato da informação recebida, sobre a evolução, para a atualização do estado interno. **Informação do Efeito [INFOEfeito] (Opcional)**texto. Descreve o formato da informação recebida, sobre o efeito das ações, para a atualização do estado interno. **Informação Associada [INFOAssociação]**texto. Descreve o formato da informação associada para a tomada de decisão. **Descrição da Técnica de IA**texto. Descreve qual a técnica de IA utilizada para tratar as informações recebidas.

3.4.5 Modelo de Interação

Este modelo consiste na junção dos modelos de coordenação e comunicação do MAS-CommonKADS, os dois modelos citados anteriormente tinham um único objetivo que era modelar as interações entre agentes. Por esse motivo renomeamos o modelo para modelo de interação, onde o projetista pode descrever todas as interações entre agentes através da AML. Neste modelo são identificadas as interações entre os agentes, especificadas como diagramas de seqüência de mensagem e de colaboração, utilizando os diagramas da AML. Cada interação

deve ser governada por um protocolo de interação, o qual define padrões de comunicação permitidos entre uma interação entre agentes.

A Figura 3.24 demonstra um exemplo de diagrama de seqüência de mensagem da AML.

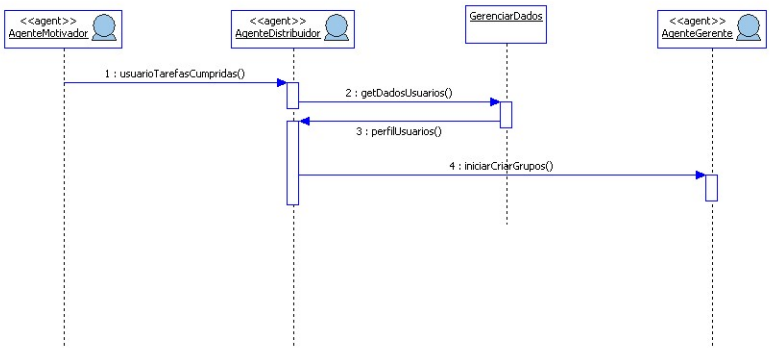

Figura 3.24: Interação entre agentes

Na figura 3.24 podemos observar uma interação entre agentes modelados com AML, este diagrama tem como objetivo modelar a troca de dados entre agentes para iniciar a criação de um grupo. O agente motivador inicia a interação comunicando ao agente distribuidor que os usuários já cumpriram as tarefas desempenhadas, sendo assim o agente distribuidor consulta o objeto gerenciar dados para buscar o perfil dos usuários, em seguida ele se comunica com o agente gerente para dar inicio a criação dos grupos.

Neste modelo também devem ser descritas as normas necessárias para que os agentes possam resolver seus conflitos (quando for necessário). Uma boa definição da coordenação dos agentes é importante para o SMA, pois permite ao projetista perceber problemas na interação entre agentes em tempo de projeto. As normas são descritas nas interações através da linguagem OCL (Object Constraints Language). Após a descrição das normas que regem a coordenação dos agentes, é necessário identificar quais agentes oferecem serviços que podem ser acessados por outros agentes e, em seguida, descrever essas novas interações através do diagrama específico para a modelagem de serviços e de diagramas de seqüência de mensagem e de colaboração. As interações entre agentes e objetos devem ser representadas nesse modelo. A Figura 3.25 apresenta um diagrama onde agente medidador acessa o serviço “perfil do aluno“ oferecido pelo agente aluno. Este agente é responsável por gerar o perfil do aluno e depois disponibiliza-lo para o uso de outros agentes. Como pode ser visualizado um serviço em AML é representado por um hexágono.

Figura 3.25: Agente comunicando com serviços

3.4.6 Modelo de Projeto

Este modelo já existia no em MAS-CommonKADS porém ele sofreu algumas altereções para possibilitar a modelagem de agentes móveis e o projeto de classes e agentes através da AML. Os modelos anteriores apresentam toda a parte lógica do SMA. Através de diagramas podemos visualizar os agentes com suas crenças e objetivos, alêm dos relacionamentos existentes, das inferências e dos recursos e objetos. Porém em nenhum momento é especificado onde esse SMA irá ser implantado. O Modelo de Projeto tem como objetivo descrever algumas facilidades de infraestrutura, que os agentes podem utilizar para movimentações dos agentes entre plataformas, e os diagramas de *deployment* do SMA, que descrevem como a arquitetura lógica definida anteriormente pode ser acionada em um ambiente operacional, ou seja, como componentes do SMA podem ser localizados, distribuídos e conectados.

As facilidades de infraestrutura são definidas da mesma maneira como descrito pela versão original de MAS-CommonKADS, ou seja, através de descrições textuais. As facilidades que devem ser descritas são (FERNÁNDEZ, 1998):

- facilidades de rede: podem ser serviços de nome de agentes, serviços de migração de agentes e/ou serviços de segurança;
- facilidades para coordenação: podem ser utilizados serviços de páginas amarelas ou brancas, serviços de transporte de mensagem e servidores de protocolo;
- facilidades para o conhecimento: podem ser utilizados servidores de ontologias, servidores de *problem solver method* (métodos solucionadores de problemas) e serviços de tradução de linguagens.

Após essas definições, deve ser cnfeccionado um diagrama detalhado das classes de agentes e os relacionamentos entre objetos. Os diagramas de *deployment* são expressos através da AML, como pode ser visualizado na Figura 3.26. A Figura apresenta um agente que se encontra no computador do cliente e pode se movimentar entre o computador do cliente e do servidor para ter acesso ao recurso conta.

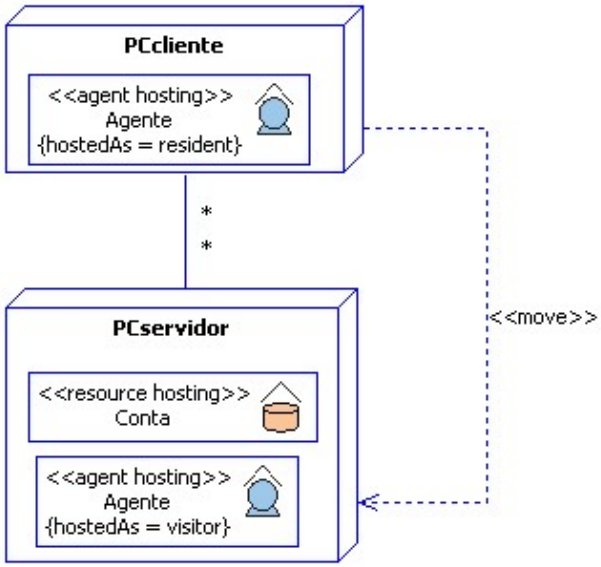

Figura 3.26: Diagrama de Deployment com agentes moveis

3.4.7 Extensão da Ferramenta de Suporte

Devido à ausência de ferramenta de suporte para a modelagem de MAS-CommonKADS, a metodologia tornava-se muito difícil de ser aplicada. Com o objetivo de possibilitar a modelagem de MAS-CommonKADS de uma forma simples e com todas as características adicionadas sobre agentes racionais, foi estendido um *plugin* desenvolvido para a linguagem de modelagem AML e que funciona sobre a plataforma StarUML.

Atualmente existem três ferramentas para modelagem de agentes com a linguagem AML: Enterprise Architect (ENTERPRISEARCHITECT,), StarUML (BOUQUET et al., 2008) e IBM Rational Rose (ROSE,). As ferramentas Enterprise Architect e IBM Rational Rose são plataformas proprietárias, o que inviabilizava a extensão da ferramenta. Por isso foi escolhida a plataforma StarUML, que é disponibilizada gratuitamente e oferece uma maneira simples para extensão, utilizando linguagens como javascript e XML. Além disso, StarUML possibilita a geração de código a partir dos modelos desenvolvidos e a verificação dos modelos desenvolvidos. Outro ponto importante que a ferramenta trata é a possibilidade de definir *templates* textuais para a geração do documento com todas as especificações do projeto. Este documento é gerado para um arquivo de texto com a extensão .doc. A StarUML possui um *plugin*, desenvolvido pela empresa Whitestein (CERVENKA; TRENCANSKY, 2004), onde trabalha o desenvolvedor da linguagem AML. O *plugin* é regido por uma licença gratuita e é disponível para modificações, o que permitem a inclusão das extensões. A Figura 3.27 demonstra a ferra-

menta StarUML utilizando o *plugin* da linguagem AML.

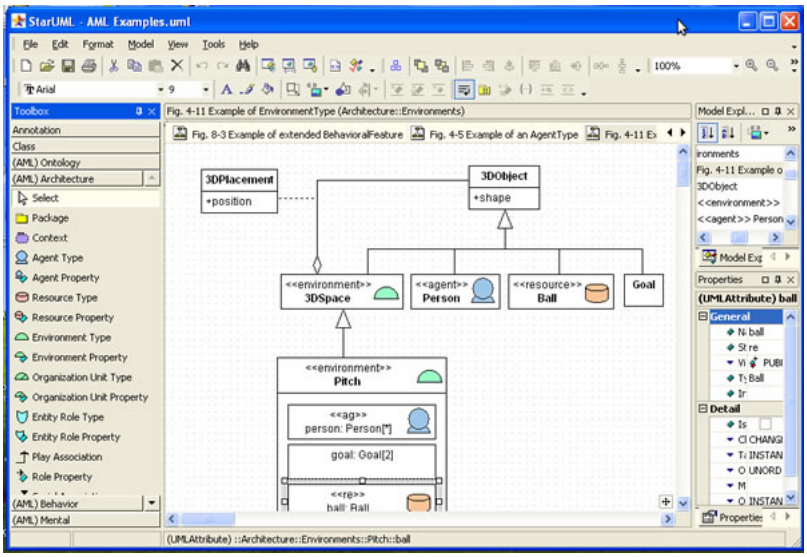

Figura 3.27: Ferramenta StarUML

Para realizar a extensão da ferramenta para a descrição da metodologia, foram realizadas algumas modificações. Para poder representar o modelo de tarefas, que não faz parte da AML, foi gerado uma nova notação chamada *task*, criada para a modelagem do diagrama de tarefas. Para gerar esta notação, a StarUML oferece uma espécie de linguagem própria com funções predefinidas que permitem gerar novas notações com o formato desejado. Além da criação da notação, foram definidos os campos para a descrição textual das tarefas. Após a criação da notação, foi necessário a inclusão desta em um diagrama e, em seguida, na paleta de notações, permitindo sua disponibilidade apenas para o modelo de tarefas.

Para poder definir a extensão para agentes racionais proposta nesta dissertação, foram incluídas duas novas entidades ao *plugin* da AML, a entidade *skill* e a *rational formulation problem*. Para alcançar este objetivo, foi necessário criar novos estereótipos baseados no diagrama de classes da UML. Estes foram adicionados à paleta de comportamentos chamada *behavior*. Após a inclusão das novas entidades, também foi necessária a inclusão de novos campos para a descrição textual dos agentes, permitindo tratar o tipo de informação de entrada dos agentes. Todas essas modificações foram realizadas utilizando XML.

Para a definição da sequência de passos da metodologia, foi gerada uma nova abordagem definida em StarUML através do elemento <*approach*> da XML. Com esse elemento foi definido uma estrutura que serve de passo a passo para a concepção da metodologia MAS-CommonKADS+, proposta nesta dissertação.

Conceitos		
Critérios	**Mas-CommonKADS+**	**MAS-CommonKADS**
Metas do sistema	Modelo de requisitos	Goal Cases
Requisitos	Modelo de requisitos	Diagrama de casos de uso
Papéis	Modelo de papéis	
Conceituação do domínio	Modelo de conhecimento	Modelo de Conhecimento
Objetivos do agente	Modelo de agentes	Modelo de agentes
Crenças	Modelo de conhecimento	Modelo de conhecimento
Serviços	Modelo de interação	Modelo de agentes
Percepções e ações	Modelo de agentes	Modelo de coordenação
Arquiteturas do agente	Modelo de agentes	Modelo de projeto
Protocolos de interação	Modelo de interação	Modelo de coordenação
Ambiente	Modelo de requisitos, modelo de organização e modelo de agentes.	
Estrutura organizacional e relacionamentos entre agentes	Modelo de organização	Modelo de organização
Deployment	Modelo de projeto	

Tabela 3.1: Análise entre as versões de MAS-CommonKADS baseada nos conceitos

Suporte		
Critérios	**Mas-Commonkads+**	**MAS-CommonKADS Original**
Recursos disponíveis	Sim	Não
Sistemas abertos	Sim	Não
Suporte a objetos	Sim	Não
Suporte a agentes móveis	Sim	Não
Suporte a ontologias	Sim	Sim

Tabela 3.2: Análise entre as versões de MAS-CommonKADS baseada no suporte

3.4.8 Análise comparativa entre as versões de MAS-CommonKADS

Esta seção tem como objetivo realizar um estudo comparativo usando os mesmos critérios da seção 2.2.10, porém utilizando as versões original e a estendida de MAS-CommonKADS. Este estudo comparativo possibilitará uma melhor visão das modificações realizadas na metodologia. As tabelas com as avaliações são descritas a seguir.

Nesta avaliação foram omitidos as tabelas de processos e modelos por não apresentarem modificações em relação ao modelo original. Observando a avaliação descrita podemos observar que a nossa extensão possibilitou a modelagem de conceitos que não eram tratados anteriormente. Os novos conceitos adicionados foram: (i) papéis, que são importantes para a descrição dos agentes; (ii) ambiente, que possibilita definir o ambiente onde os agentes estão atuando; e (iii) Diagramas de *deployment* (implantação), que auxilia na implantação e descrição

de agentes móveis. Em relação a avaliação de suporte a nossa extensão possibilitou uma grande melhoria pois possibilitou a modelagem de recursos e objetos do sistema e de agentes móveis, estes modelos são descritos respectivamente nos modelos de recursos e objetos e no modelo de deployment.

4 Validação dos Modelos de Agentes Racionais e exemplo de uso

Este capítulo tem como objetivo demonstrar a validação dos modelos propostos para as arquiteturas de agentes racionais demonstrando o seu uso para problemas que utilizam lógica e sistemas baseados em regras, redes neurais, busca e lógica fuzzy. Neste capitulo também é demonstrado um pequeno exemplo de uso da metodologia após as modificações realizadas, para isso foi utilizado como base um sistema multiagente para suporte a gerência do conhecimento no processo de software proposto por Mendes e II (2010).

4.1 Validação dos Modelos de Agentes Racionais

Para realizar a validação dos novos modelos propostos foram utilizados dois problemas bastante conhecidos da IA, o mundo do aspirador de pó e o mundo de wumpus. O mundo do aspirador de pó é mundo muito simples, mas que possui muitas variações. Esse mundo possui duas salas A e B. O agente aspirador de pó está localizado em uma dessas salas e tem a capacidade de perceber em que sala está e o estado da sala, sujo ou limpa. As ações que ele pode realizar neste mundo são mover para direita, para esquerda, aspirar e não operar. O objetivo do agente consiste em limpar as duas salas. A Figura 4.1 demonstra o mundo do aspirador de pó e seus possíveis estados.

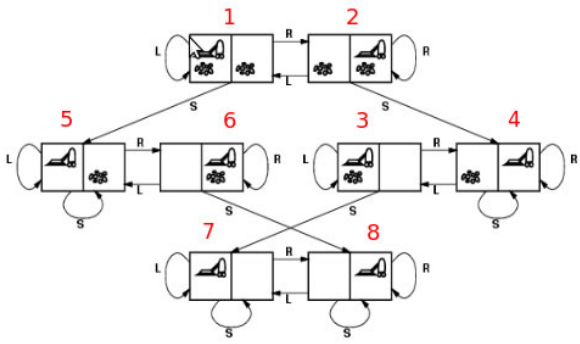

Figura 4.1: O mundo do aspirador de pó

O mundo de wumpus é um ambiente que simula uma caverna nele é encontrado um monstro chamado wumpus, vários buracos e um pote de ouro. O objetivo de um agente nesse mundo é conseguir pegar o ouro sem cair em um buraco nem ser capturado pelo wumpus. Este mundo possui regras que devem ser seguidas pelo agente para alcançar o pote de ouro. A Figura 4.2 demonstra o mundo de wumpus.

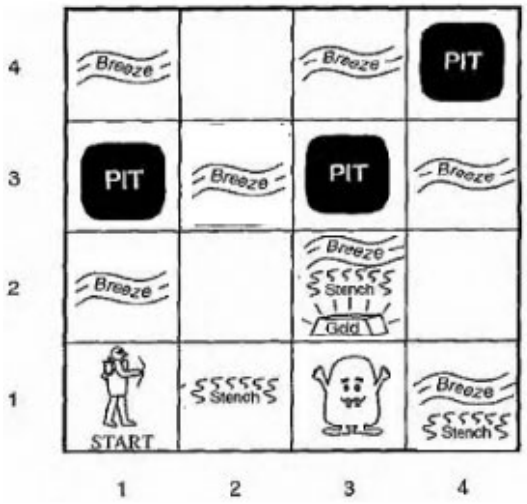

Figura 4.2: O mundo de wumpus

O agente começa na posição [1,1], deve achar o encontrar o ouro, retornar para [1,1] e sair da caverna, sem ser morto pelo Wumpus. No quadrado que contém o Wumpus e nos quadrados adjacentes (não diagonal) o agente percebe o Wumpus pelo cheiro. Nos quadrados adjacentes ao buraco o agente percebe um vento (brisa). No quadrado que tem ouro o agente percebe um brilho. Quando o agente vai em direção ao muro ele percebe a batida e quando o Wumpus é morto o agente percebe o grito do monstro. O que é percebido pelo agente em um dado momento é dado por: [Cheiro, Vento, Brilho, Parede, Grito]. O agente não percebe a posição atual no tabuleiro (ela deve ser calculada). As ações de movimento do agente são: virar 90 graus para direita ou para esquerda, e ir para frente ou para trás. As outras ações são: pegar o ouro, atirar (só uma flecha disponível, ele atira e ela caminha até a parede em linha reta) e sair da caverna. A localização do Wumpus e do Ouro é aleatória (em aproximadamente 21% dos ambientes não tem como o agente pegar o ouro, o agente deve saber disso e desistir de pegar o ouro). O ouro pode estar em uma posição vazia e em uma posição com o Wumpus e não em uma posição com um buraco.

4.1.1 Validação de Agentes Reativos

Para a validação de um agente reativo que utiliza regras condição-ação foi usado o mundo do aspirador de pó tendo as seguintes características:

- Percepção em P - informação sobre a sala em que o agente está (A ou B) e o estado da sala (Suja ou Limpa);
- Ação em A = asp, dir, esq;
- Medida de desempenho em G: um ponto por quadro limpo em um passo, menos um ponto por passos sem limpar;
- Propriedades do Ambiente: parcialmente observável, determinístico, seqüencial, discreto, estático, agente único;
- Estados possíveis em Estado = [a, suja], [a, limpa], [b, suja], [b, limpa].

Para este problema o Conjunto de regras condição ação é demonstrado a seguir:

se Estado = [a, suja] então Ação=asp,

se Estado = [a, limpa] então Ação=dir,

se Estado = [b, suja] então Ação=asp,

se Estado = [b, limpa] então Ação=esq

A validação dos modelos é realizada apresentando as suas descrições na linguagem de modelagem AML que foi estendida, é importante frisar que características como representação de conhecimento, condições de ativação e alguns parâmetros de entrada do agente são modelados através da metodologia usando templates textuais ou outros diagramas.

A arquitetura concreta de um agente reativo que utiliza regras condição-ação é apresentada na figura 4.3, ou seja, nela pode ser observada que a informação denominada INFOassociação no programa agente reativo da Seção 3.1, está disponível no formato de um conjunto de regras "se condição então ação" e que, em função disso, a função ação é concebida a partir de técnicas de IA Simbólica, ou seja, empregando representação do conhecimento e um sistema baseado em regras (SBR).

O programa agente reativo regras condição-ação é apresentado na figura 4.1.1. Conforme mencionado, esse programa consiste em uma especialização do programa agente reativo na Seção 3.1.

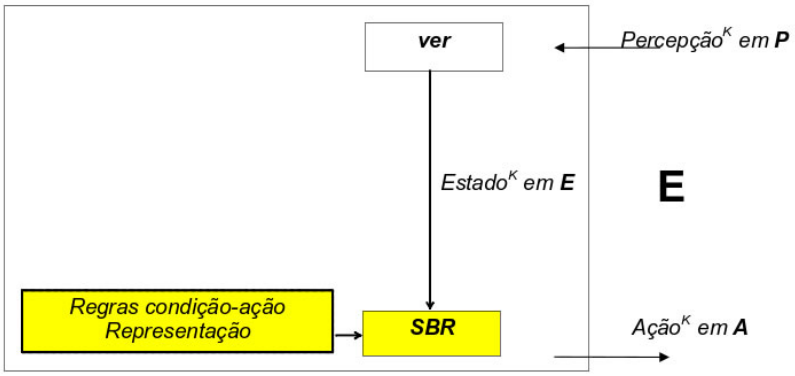

Figura 4.3: Agente reativo com regras

função agente-reativo-REGRAS(Percepção) retorna uma ação
entradas: Percepção em P, uma percepção
var estáticas:
Regras, um conjunto de regras no formato se condição então ação
SBR, um tipo específico de sistema baseado em regras
Estado em E, uma descrição de estado atual do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Estado ← ver(Percepção)
Ação ← SBR(Estado,Regras)
retornar Ação

Figura 4.4: Programa agente reativo com regras

Para modelar um agente desse tipo com as extensões propostas deve-se adicionar os sensores e atuadores do agente e incluir um *skill* como comportamento interno do agente, esse possui um estereotipo <<SBR>> associado a ele que permite identificar que as informações recebidas irão ser tratadas através de um sistema baseado em regras. Nesse caso é importante perceber que o *skill* SBR representa a função ação para a tomada de decisão do agente. A Figura 4.5 demonstra o diagrama para o tipo de agente apresentado.

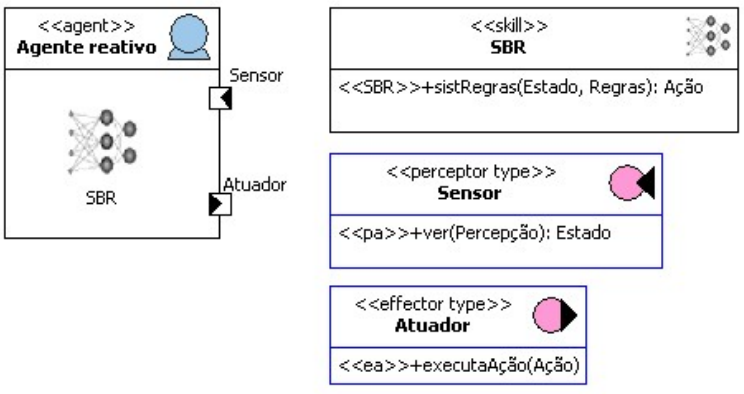

Figura 4.5: Agente reativo baseado em regras condição-ação usando AML

Por outro lado, quando as informações disponíveis para o agente estão disponíveis em um formato de conjunto de exemplos é interessante que o agente possa trabalhar com técnicas adequadas para esse tipo de informação. Por exemplo, considerando o que foi descrito na Seção 3.2, a função ação do aspirador de pó definido nesta subseção, retirando-se as regras e disponibilizando-se um conjunto de exemplos singinificativos condição-ação, pode ser concretizada empregando-se uma rede neural para a tomada de decisão do agente como mecanismo de seleção de ações. O conjunto de exemplo para o aspirador de pó pode ser dado como a seguir.

([posição,estado], ação)$^{L}{}_{L=1,...,4}$ = ([a, suja], asp), ([a, limpa], dir), ([b, suja], asp), ([b, limpa],esq)

A arquitetura concreta de um agente reativo que utiliza um conjunto de exemplos de condição ação é demonstrada na figura 4.6, nela pode ser observada que INFOassociação está disponível no formato de um conjunto de exemplos no formato de pares [condição, ação] e que para isso a Função ação da IA Conexionista é dada através de uma rede neural associada a um algoritmo de aprendizagem.

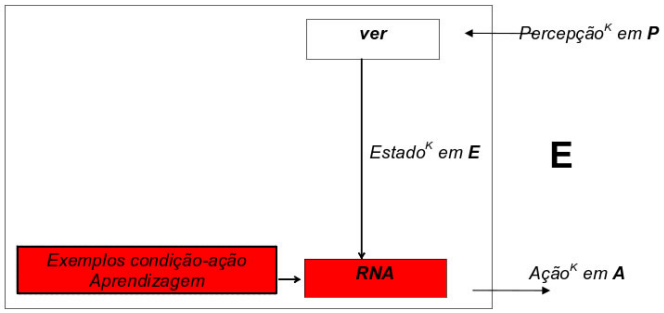

Figura 4.6: Agente reativo com conjunto de exemplos

O programa agente reativo é apresentado na figura 4.1.1. Esse programa consiste em um outro tipo de especialização possível para a função tomada de decisão (ação), destacada na seção 3.2, do programa agente reativo descrito na Seção 3.1.

Neste caso, a informação associada (INFOassociação) é dada por uma matriz de pesos, W_ação, que são obtidos a partir de um conjunto de treinamento $\{$(Estado em E, Ação em A)^{L} $\}_{L=1,...,M}$ e de um algoritmo de aprendizagem associado ao modelo de RNA como, por exemplo:

- Se o problema for de aproximação de funções (p.ex: previsão, reconhecimento de padrões, ...) e a RNA for multicamadas diretamente alimentada, então o algoritmo de aprendizagem pode ser o back-propagation;
- Se o problema for de classificação ou clusterização e a rede for um mapa auto-organizável de Kohonen, então o algoritmo de aprendizagem é o algoritmo de kohonen.

função agente-reativo-RNA(*Percepção)* **retorna** uma ação
entradas: *Percepção* em *P*, uma percepção
variáveis estáticas:
W_ação, informação suplementar para a tomada de decisão no formato de uma matriz de valores reais (pesos)
Estado em *E,* uma descrição de estado atual do ambiente
Ação em*A*, ação mais recente, inicialmente nenhuma

Estado ← ver(*Percepção*)
Ação ← RNA*(Estado,W_ação*)
retornar *Ação*

Figura 4.7: Programa agente reativo com uma rede neural

Conforme aconteceu para o caso anterior, para modelar esse agente com as extensões propostas, deve-se adicionar os sensores e atuadores do agente e incluir um *skill* como comportamento do agente, esse possui um estereotipo *<<neural network>>* associado a ele que permite identificar que as informações recebidas irão ser tratadas através de uma rede neural. A Figura 4.8 demonstra o diagrama para o tipo de agente apresentado.

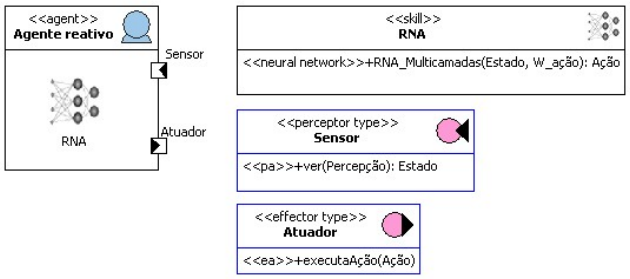

Figura 4.8: Agente reativo com redes neurais em AML

4.1.2 Validação de Agentes Reativos Baseados em Modelos

A validação do agente reativo baseado em modelos emprega o exemplo do mundo de wumpus. A arquitetura concreta do agente utiliza como informação um conjunto de proposições a respeito de como o ambiente evolui independente das ações do agente, INFOEvolução. Mais especificamente, o agente emprega a Lógica para representar estas proposições e um provador automático de teoremas na concretização da função próximo. Para este exemplo, o conjunto de proposições INFOEvolução foi representado em lógica proposicional conforme destaca a Figura 4.9.

Por exemplo, a primeira proposição na figura diz que “se uma caverna não está fe-

$R1: \neg f1\text{-}1 \Rightarrow \neg W1\text{-}1 \wedge \neg W1\text{-}2 \wedge \neg W2\text{-}1$
$R2: \neg f2\text{-}1 \Rightarrow \neg W1\text{-}1 \wedge \neg W2\text{-}1 \wedge \neg W2\text{-}2 \wedge \neg W3\text{-}1$
$R3: \neg f1\text{-}2 \Rightarrow \neg W1\text{-}1 \wedge \neg W1\text{-}2 \wedge \neg W2\text{-}2 \wedge \neg W1\text{-}3$

Figura 4.9: Representação em lógica

dendo, então o Wumpus não está nessa caverna, nem está em uma caverna adjacente". A Figura 4.10 destaca este novo tipo de especialização na arquitetura do agente reativo baseado em modelos descrito nas Seções 3.1 e 3.2.

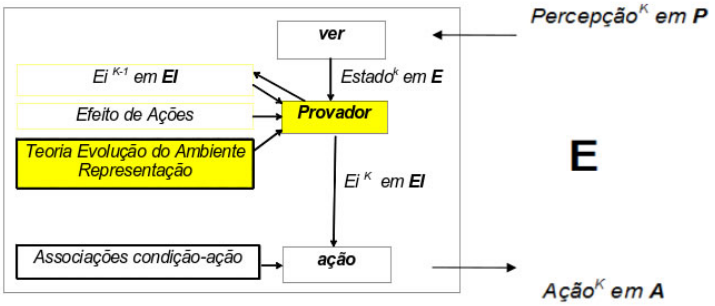

Figura 4.10: Agente baseado em modelos com lógica

A especialização no programa agente reativo baseado em modelos lógicos pode ser visualizado na figura 4.1.2.

Um exemplo de atualização do estado interno através da função próximo para mostrar que o Wumpus não está em nenhuma outra caverna, e então concluir, por eliminação, que ele está em (1,3), pode ser descrito através dos seguintes passos:

1.Aplicando Modus Ponens a ¬ f1-1 e R1, obtemos:

função agente-reativo-modelos-LÓGICA(Percepção) **retorna** uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
INFOevolução, informação suplementar para a atualização de estado interno
INFOassociação, informação suplementar para a tomada de decisão
Estado em E, uma descrição de estado atual do ambiente
Ei em EI, uma descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Estado ← ver(percepção)
Ei ← Provador(Estado, Ei,INFOevolução)
Ação ← ação(Ei,INFOassociação)
retornar Ação

Figura 4.11: Programa agente reativo baseado em modelos

R1:¬f1-1⇒ ¬ W1-1 ∧ ¬W1-2 ∧ ¬W2-1 ¬ W1-1 ∧ ¬W1-2 ∧ ¬W2-1

Modus Ponens $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$

2.Aplicando E-eliminação a (1), obtemos três sentenças isoladas:

E-Eliminação: $\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_1}$ ¬ W1-1 ¬W1-2 ¬W2-1

3.Aplicando Modus Ponens a ¬f2-1 e R2, e em seguida aplicando eliminação obtemos:

R2:¬f2-1⇒ ¬ W1-1 ∧ ¬W2-1 ∧ ¬W2-2∧ ¬W3-1 ¬ W1-1 ¬W2-1 ¬W2-2 ¬W3-1

4.Aplicando Modus Ponens a f1-2 e R4:

R4: f1-2⇒ W1-3 ∨ W1-2 ∨ W2-2∨ W1-1 W1-3 ∨ W1-2 ∨ W2-2∨ W1-1

Modus Ponens $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$

5.Aplicando Resolução a ¬w1-1 e (4):

Resolução Unitária: $\frac{\alpha \vee \beta, \neg\beta}{\alpha}$ W1-3 ∨ W1-2 ∨ W2-2

6.Aplicando Resolução a ¬w2-2 e (5):

W1-3 ∨ W1-2

7.Aplicando Resolução ¬w1-2 e (6):

W1-3

Para modelar esse agente com as extensões propostas deve-se adicionar os sensores e atuadores do agente e incluir um skill que será responsável pela atualização do estado interno (provador), esse possui um estereotipo <<logic>> associado a ele que permite identificar que as informações recebidas irão ser tratadas através de lógica e um skill para a tomada de decisão este pode possuir qualquer um dos estereótipos existentes para skill para tratar o tipo de informação de saída. A Figura 4.12 demonstra o diagrama para o tipo de agente apresentado.

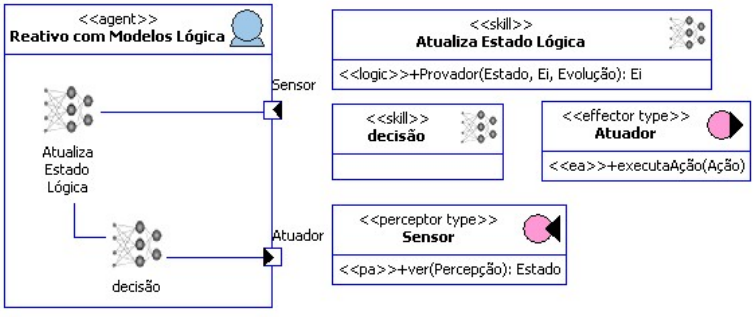

Figura 4.12: Agente reativo baseado em modelos em AML

Quando as informações sobre efeito de ações e evolução estão disponíveis em um formato de conjunto de exemplos é interessante que o agente possa utilizar uma rede neural para atualizar seu estado interno. A arquitetura concreta demonstrada na figura 4.13 exibe um agente baseado em modelos que utiliza uma rede neural para atualizar seu estado interno, este estado recebe como informação de efeito um conjunto de exemplos (Dados) no formato de pares ($[E_i^{K-1}$, Ação], E_i^K).

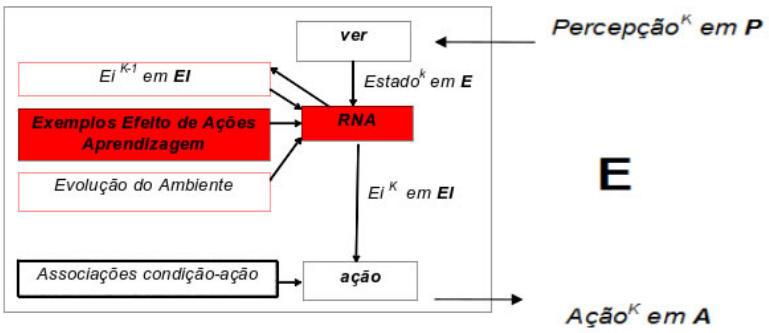

Figura 4.13: Agente reativo baseado em modelos com uma RNA

Para este tipo de agente o algoritmo de construção é demonstrado na figura 4.14.

função agente-reativo-modelos-RNA(Percepção) **retorna** uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
W_efeito, informação suplementar para a tomada de decisão no formato de uma matriz de valores reais (pesos)
INFOevolução, informação suplementar para a atualização de estado interno
INFOassociação, informação suplementar para a tomada de decisão
Estado em E, uma descrição de estado atual do ambiente
Ei em EI, uma descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Estado ← ver(percepção)
Ei ← RNA(Estado, Ei, W_efeito, INFOevolução)
Ação ← ação(Ei, INFOassociação)
retornar Ação

Figura 4.14: Programa agente reativo baseado em modelos com RNA

Conforme mencionado, a matriz de pesos *W_efeito* é obtido a partir de um conjunto de treinamento $\{([E_i^{K-1}$ em *EI, Ação* em *A*], E_i^K em $EI)^L\}_{L=1,\ldots,M}$ e de um algoritmo de aprendizagem associado ao modelo de RNA.

Para modelar esse agente com as extensões propostas deve-se adicionar os sensores e atuadores do agente e incluir um *skill* que será responsável pela atualização do estado interno

(RNA), esse possui um estereotipo <<*neural network*>> associado a ele que permite identificar que as informações recebidas irão ser tratadas através de uma rede neural e um *skill* para a tomada de decisão este deve possuir o estereotipo de acordo com o tipo de informação de saída do *skill* anterior. A Figura 4.15 demonstra o diagrama para o tipo de agente apresentado.

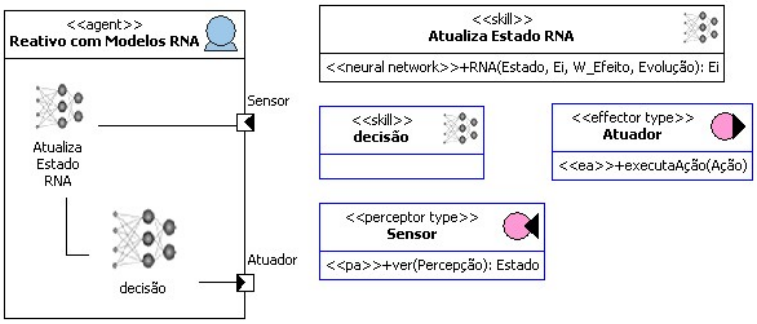

Figura 4.15: Agente reativo baseado em modelos em AML

4.1.3 Validação de Agentes Baseados em Objetivos

A validação do agente baseado em objetivos utilizou um agente de resolução de problemas que é um tipo de agente baseado em objetivos. Segundo Russel e Norvig (1995), o agente de resolução de problemas decidem o que fazer encontrando uma seqüência de ações que levam a estados desejáveis. Para demonstrar o uso deste tipo de agente foi utilizado como exemplo do aspirador de pó. Os estados possíveis para o mundo do aspirador de pó são:

E = *[a, suja, suja], [a, suja, limpa], [a,limpa,suja], [a, limpa, limpa], [b, suja, suja], [b, suja, limpa], [b,limpa,suja], [b, limpa, limpa]*

Assumindo que o estado inicial seja [*a, limpo, sujo*] e considerando que o custo do caminho seja 1 ponto por ação realizar o objetivo dado por [*a, limpo, limpo*], [*b, limpo, limpo*] com o menor custo possível.

A arquitetura concreta para o agente baseado em objetivos é demonstrado na figura 4.16. Nesta a arquitetura o efeito é informação suplementar disponível para atualização de estado interno e tomada de decisão, que está disponível no formato de uma teoria, ou seja, a descrição de uma função sucessor de estados no formato: **successor : EI** $\rightarrow$ **P(AxEI)**. A função ação desta arquitetura é definida através da representação simbólica do conhecimento mais um método de busca.

A Informação sobre o efeito das ações é da através da função sucessor definida a seguir:

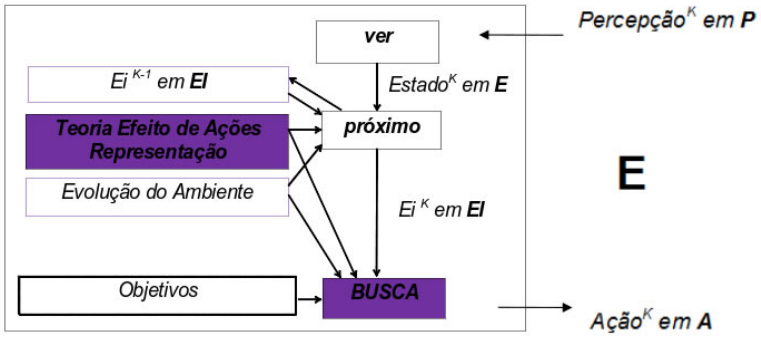

Figura 4.16: Agente baseado em objetivos com busca

Sucessor(1) = {(asp, 5), (dir, 2), (esq, 1)},

Sucessor(2) = {(asp, 4), (dir, 2), (esq, 1)},

Sucessor(3) = {(asp, 7), (dir, 4), (esq, 3)},

Sucessor(4) = {(asp, 4), (dir, 4), (esq, 3)},

Sucessor(5) = {(asp, 5), (dir, 6), (esq, 5)},

Sucessor(6) = {(asp, 8), (dir, 6), (esq, 5)},

Sucessor(7) = {(asp,7), (dir, 8), (esq, 7)},

Sucessor(8) = {(asp, 8), (dir, 8), (esq, 7)}

O algoritmo para a construção de um agente baseado em objetivo é demonstrado na figura 4.17.

Para modelar esse agente com as extensões propostas deve-se adicionar os sensores e atuadores do agente e incluir um skill que será responsável pela atualização do estado interno, esse possui um estereotipo <<logic>> associado a ele que permite identificar que as informações recebidas irão ser tratadas através de lógica, em seguida temos um rational behavior formulation que é responsável pela formulação de objetivos e de problemas do agente e um skill para a tomada de decisão este deve possuir o estereotipo do tipo <<search>> para realizar a busca de ações que satisfaçam o objetivo retornando uma seqüência de ações possíveis, neste caso de modelagem é extremamente importante a utilização do diagrama de atividades para demonstrar os passos realizados internamente para a escolha da ação. A figura 4.18 demonstra o diagrama para o tipo de agente apresentado.

função agente-baseado-objetivo-BUSCA(Percepção) **retorna** uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
Sucessor, informação de efeito de ações suplementar para a atualização de estado interno e tomada de decisão
INFOevolução, informação suplementar para a atualização de estado interno e tomada de decisão
Estado em E, uma descrição de estado atual do ambiente
Ei em EI, uma descrição de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
Objetivo, um conjunto de descrições de estados desejados ou alguma descrição do que vem a ser um estado desejado no ambiente
Sequência, uma sequência de ações, inicialmente vazia
Custo, uma função para medir o custo de uma sequência
Problema, uma formulação de problema
Estado ← ver(Percepção)
Ei ← próximo(Estado, Ei,Sucessor,INFOevolução)
se Sequência vazia **então faça**
Objetivo ← formular_objetivo(Ei)
Problema ← formular_problema(Ei,Sucessor,Objetivo,Custo)
Sequência ← BUSCA(Problema)
Ação ← primeira(Sequência)
Seqüência ← resto(Sequência)
retornar Ação

Figura 4.17: Programa agente baseado em objetivos com busca

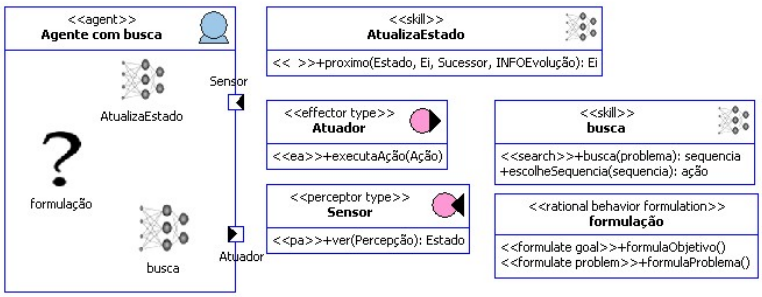

Figura 4.18: Agente baseado em objetivos com busca

4.1.4 Validação de Agentes Baseados em Utilidade

Por fim é apresentado o agente baseado em utilidade, esse agente tem uma informação associada que representa o quão satisfeito o agente estará se realizar determinada ação. A arquitetura concreta demonstrada na figura 4.19 mostra um agente baseado em utilidade que utiliza lógica *fuzzy*.

Nesta arquitetura o efeito das ações é dado como uma informação suplementar dis-

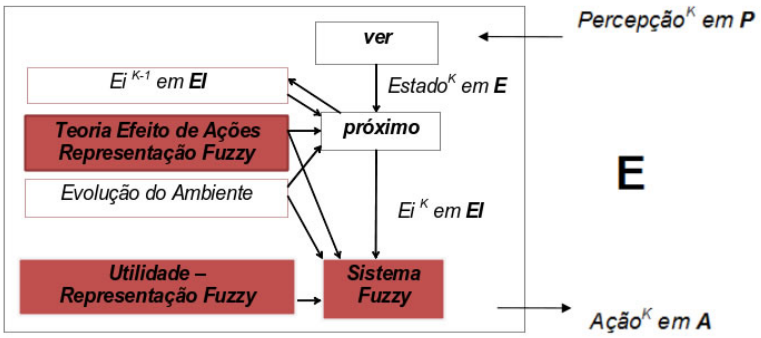

Figura 4.19: Agente baseado em utilidade

ponível para atualização de estado interno e tomada de decisão, disponível no formato de uma teoria formada por proposições *fuzzy*, ou seja, a descrição de uma função sucessor de estados no formato: **successor : EI** *rightarrow* **P(AxEI)**, onde as descrições de estado interno corrente E_i^{K-1} em EI e de estado interno realizado E_i^K (previsto) por uma Ação em A são termos lingüísticos representando os valores dos atributos (características) que representam (compõem) uma descrição de estado interno em EI.

A Utilidade é dada como informações sobre a utilidade de um estado e está disponível também no formato de uma teoria formada por um conjunto de M proposições f*uzzy* no formato: {**SE** descrição $fuzzy^L$ de estado realizado E_i^K **ENTÃO** descrição $fuzzy^L$ de valor de Utilidade$\}_{L=1,...,M}$. A Função ação é uma representação *Fuzzy* das informações sobre a utilidade em conjunto com sistemas *Fuzzy* para a seleção de ações.

Para o mundo do aspirador de pó os estados possíveis em *fuzzy* para esse agente são:

E = {[*a, muito suja*], [*a, mais ou menos suja*], [*a,suja*], [*a, limpa*], [*a, mais ou menos limpa*], [*a, muito limpa*], ..., [*b, muito suja*], [*b, mais ou menos suja*], [*b,suja*], [*b, limpa*], [*b, mais ou menos limpa*], [*b, muito limpa*], ...}

Como exemplo de uma descrição de estado internos *fuzzy* em EI temos E_i = [*A, muito limpo, mais ou menos sujo*] as informações sobre o efeito de ações é dada pela função sucessor dada por:

SucessorFuzzy([A,muito suja, muito suja]) = {(asp, [A, suja, muito suja]), (asp muito, [A, mais ou menos limpa, muito suja]), (asp pouco, [A, mais ou menos suja, muito suja]), ...,

(dir, [B, muito suja, muito suja]),

(esq, [A, muito suja, muito suja])},

SucessorFuzzy([B, muito suja, muito suja]) = {*(asp, [B, suja, muito suja]),*

(asp muito, [B, mais ou menos limpa, muito suja]),

(asp pouco, [B, mais ou menos suja, muito suja]),

...,

(dir, [B, muito suja, muito suja]),

(esq, [A, muito suja, muito suja])},

. . .

A informação sobre *Utilidade* é dada por um conjunto contendo *M* proposições fuzzy:

{*SE Ei K = [A, muito suja, muito suja] ENTÃO Utilidade = baixíssima,*

...,

SE Ei K = [A, suja, muito suja] ENTÃO Utilidade = muito baixa,

...,

SE Ei K = [A, limpa, mais ou menos limpa] ENTÃO Utilidade = considerável,

...,

SE Ei K = [B, muito suja, muito suja] ENTÃO Utilidade = baixíssima,

...,

SE Ei K = [B, muito suja, suja] ENTÃO Utilidade = muito baixa,

...,

SE Ei K = [B, mais ou menos limpa, limpa] ENTÃO Utilidade = considerável,

... }

Por ultimo a função *ação é dada por método de Inferência Fuzzy Mizummoto/Sugeno.* O algoritmo para a construção do agente baseado em utilidade é descrito na figura 4.20.

Para modelar esse agente com as extensões propostas deve-se adicionar os sensores e atuadores do agente e incluir um *skill* que será responsável pela atualização do estado interno, esse possui um estereotipo <<*fuzzy*>> associado a ele que permite identificar que as informações recebidas irão ser tratadas através de lógica fuzzy e um *skill* para a tomada de decisão este deve possuir o estereotipo do tipo <<*fuzzy*>>, para esse skill existe uma operação para a função utilidade que é gerada através de regras fuzzy esta possui um estereótipo do tipo <<fuzzy>>. A Figura 4.21 demonstra o diagrama para o tipo de agente apresentado.

função agente-baseado-utilidade(Percepção) **retorna** uma ação
entradas: Percepção em P, uma percepção
variáveis estáticas:
SucessorFuzzy, informação fuzzy de efeito de ações suplementar para a atualização de estado interno e tomada de decisão
INFOevolução, informação fuzzy suplementar para a atualização de estado interno e tomada de decisão
Estado em E, descrição fuzzy de estado atual do ambiente
Ei em EI, descrição fuzzy de estado interno do ambiente
Ação em A, ação mais recente, inicialmente nenhuma
RegrasFuzzyUtilidade, conjunto de M proposições fuzzy SE-ENTÃO
Ação, ação que produz um estado que tenha uma utilidade máxima
Estado ← ver(Percepção)
Ei ← próximo_fuzzy(Estado, Ei,SucessorFuzzy,INFOevolução)
Ação ← SISTEMAFUZZY(Ei,SucessorFuzzy,INFOevolução,RegrasFuzzyUtilidade)
retornar Ação

Figura 4.20: Programa do agente baseado em utilidade

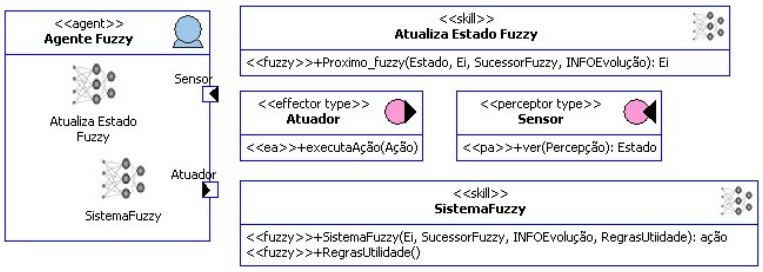

Figura 4.21: Agente baseado em utilidade

4.2 Exemplo de Uso da Metodologia

Para demonstrar o uso da metodologia foi utilizado a modelagem proposta por Mendes e II (2010), que demonstra um SMA que auxilia os gerentes de projeto a gerenciar o processo de desenvolvimento do sistema a ser desenvolvido. Este SMA é composto por agentes com capacidade de descobrir riscos na fase de projeto baseados em conhecimentos anteriores, gerar grupos de desenvolvedores baseados em suas experiências em projetos e preparar e gerenciar o cronograma de atividades na fase de desenvolvimento. Nesta modelagem não será contemplada a fase de requisitos por ser realizada da mesma maneira que a modelagem já apresentada no capitulo de livro em que o sistema foi proposto. Todos os modelos foram gerados através da ferramenta que foi estendida nesta dissertação.

Após a fase de requisitos o passo seguinte da metodologia é a geração do modelo de tarefas através dos diagramas e templates textuais referentes às tarefas do sistema, neles são

descritos funcionalidades que o sistema deve realizar, para isso é gerado um diagrama de tarefas que apresentam todas as funcionalidades possíveis do sistema, esses diagramas podem ser decompostos em várias sub-tarefas cada tarefa ou sub-tarefa apresentam um template textual contendo os objetivos e suas descrições. O diagrama com as tarefas do sistema pode ser observado na figura 4.22, nesta mesma figura também pode ser observada uma janela que possibilita a descrição dos objetivos e descrição das tarefas.

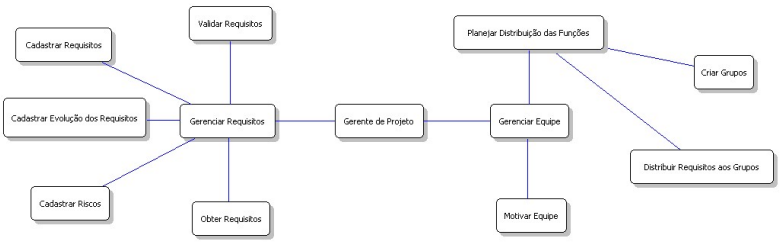

Figura 4.22: Modelo de tarefas

Seguindo os passos propostos para a metodologia deve-se realizar a captura dos objetos do sistema esses são capturados visualizando os requisitos e as tarefas do sistema. Para este sistema algumas tarefas que não necessitam de autonomia como, por exemplo, tarefas de cadastramento as quais foram incluídas no diagrama de classes. Neste modelo são descritos também os recursos como, por exemplo, um banco de dados. O diagrama de classe encontrado para esse modelo é demonstrado na figura 4.23.

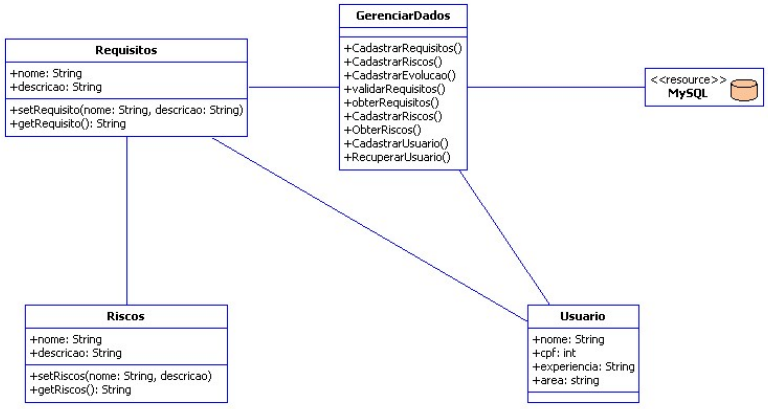

Figura 4.23: Modelo de Recursos e Objetos

O próximo passo da metodologia é descobrir os papéis que cada agente pode desempenhar no sistema, esses papéis são responsáveis por cumprir as tarefas existentes no modelo de tarefas, após a criação do modelo de papéis o modelo de tarefas deve ser revisado, pois novas tarefas podem surgir após a definição dos papéis, se novas tarefas forem descobertas o modelo

de objetos e o modelo de papéis deve ser revisado. Na Figura 4.24 podemos observar os papeis encontrados e as tarefas que eles realizam.

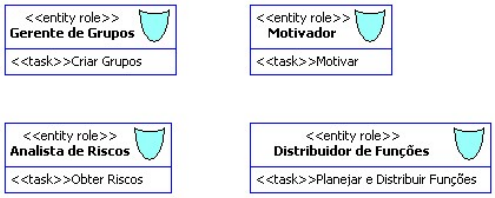

Figura 4.24: Modelo de Papéis

Depois de definir os papéis que compõem o SMA deve ser desenvolvido um diagrama contendo a organização do SMA, neste diagrama estarão contidas todas as hierarquias entre os papéis e grupos de papéis, este modelo deve ser dividido em organização interna e externa. Na organização interna serão apresentados os relacionamentos entre papéis do sistema, enquanto que no externo será demonstrado o relacionamento entre os grupos de agentes. O modelo de organização pode ser visualizado na figura 4.25.

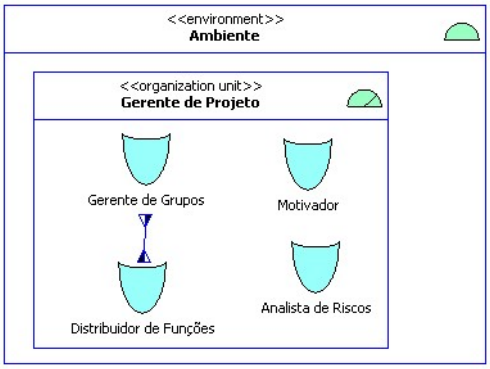

Figura 4.25: Modelo de Organização

O modelo de agentes é o responsável por definir quais os agentes serão responsáveis por cada papel apresentado, por definir a arquitetura dos agentes, seus objetivos e características como entrada de dados, condições de ativação do agente e tipos de informação disponível. Os *templates* textuais com objetivos e características são demonstrados a seguir.

Agente: AgenteMotivador

Descrição

Este agente é responsável pela comunicação direta com o usuário tendo o objetivo de motivá-lo e de capturar algumas características do usuário.

Objetivo: Motivar Grupo ou Profissional.

Parâmetros de Entrada

Dados sobre prazos para o projeto.

Parâmetros de Saída

Motivação.

Condição de Ativação

Quando o agente percebe que o grupo ou profissional vai ou não atingir a meta.

Condição de Finalização

Quando se envia alguma imagem ou frase de motivação para o usuário.

Informação Associada

Toma as decisões para exibir as motivações através de um sistema fuzzy.

Descrição

Este objetivo prover mecanismo para a motivação do grupo ou profissional. Esta motivação pode ser realizada de várias formas podendo ser utilizado imagens ou frases.

Agente: AgenteGerente

Descrição

Este agente é responsável por auxiliar o gerente de projeto na formação de grupos.

Objetivo: Criar grupos

Parâmetros de Entrada

Perfil dos profissionais envolvidos no projeto.

Parâmetros de Saída

Grupos criados.

Condição de Ativação

Quando o gerente de projeto manda o sistema escalonar tarefas para um grupo de profissionais.

Condição de Finalização

Quando os grupos são gerados.

Condição de fracasso

O Gerente de Projeto não aceitar o grupo.

Informação Associada

Utiliza um algoritmo de busca para montar as equipes.

Descrição

Este objetivo visa criar grupos baseado no perfil profissional dos usuários, para gerar o grupo é analisado o perfil de cada um com base nos requisitos a serem desenvolvidos e em seguida gerados o grupo, se o gerente de projeto recusar o grupo o agente deve ser capaz de guardar as informações para saber que para determinados requisitos não devem ser criados esses grupos.

Agente: AgenteAnalista

Descrição

Este agente é responsável por auxiliar o gerente de projeto na reutilização de requisitos já definidos e implementados anteriormente e verificar requisitos que estão pendentes durante o projeto. Este agente tem uma importância significativa, pois com a reutilização de requisitos ocorre uma diminuição do tempo do projeto alem de se trabalhar com riscos já conhecidos ele também colabora para auxiliar a negociação de novos prazos para o projeto.

Objetivo: Informar riscos

Parâmetros de Entrada

Requisitos já desenvolvidos.

Parâmetros de Saída

Riscos conhecidos.

Condição de Ativação

Quando um requisito semelhante é encontrado na base de requisitos desenvolvidos.

Condição de Finalização

Quando os riscos são informados.

Condição de fracasso

Não existir nenhum requisito semelhante já desenvolvido.

Informação Associada

Utiliza um sistema baseado em regras que verifica os riscos.

Descrição

Este objetivo visa informar os riscos que ocorrerão quando o requisito foi desenvolvido possibilitando assim o gerente de projeto ter uma visão prévia do que pode acontecer.

Os objetivos que um agente pode ter podem ser definidos neste modelo através de um diagrama de objetivos, estes objetivos podem ser decompostos em vários sub-objetivos com o objetivo de solucioná-lo. Nesse modelo não foi necessário a decomposição.

Após os objetivos, deve ser realizada a escolha da arquitetura interna dos agentes, demonstrando os comportamentos que o agente irá realizar quando receber as informações perceptivas. Na Figura 4.26 podemos observar a modelagem do AgenteGerente.

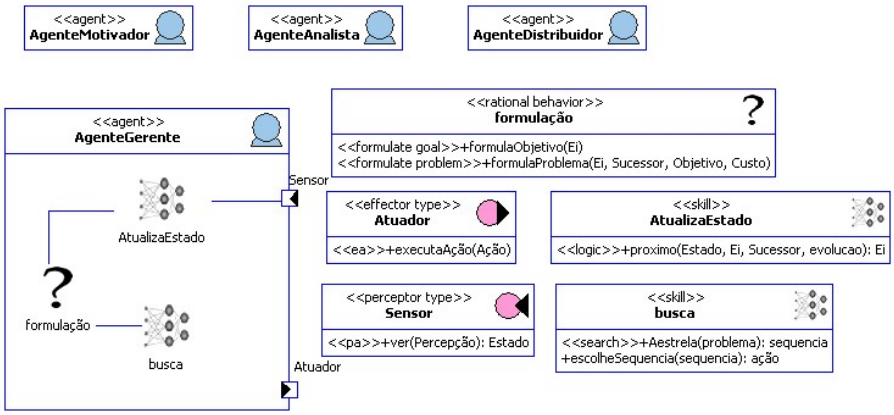

Figura 4.26: Modelo de Agentes

Para cada agente desenvolvido deve-se desenvolver um diagrama de atividades da AML descrevendo as atividades realizadas após a ativação do agente. Nesta etapa devem ser demonstrados também os planos do agente se for necessário. O diagrama de atividades do AgenteGerente que realiza a formação de grupos pode ser visualizado na figura 4.27.

Após a identificação dos agentes são criados os modelos de conhecimento e interação, no modelo de conhecimento são descritos todos os conhecimentos que os agentes possuem do ambiente e de outros agentes, este conhecimento pode ser apresentado através de ontologias, lógica ou qualquer linguagem que represente o conhecimento de um agente. Neste modelo também deve ser modelado a forma de inferência dos agentes para solucionar cada objetivo que

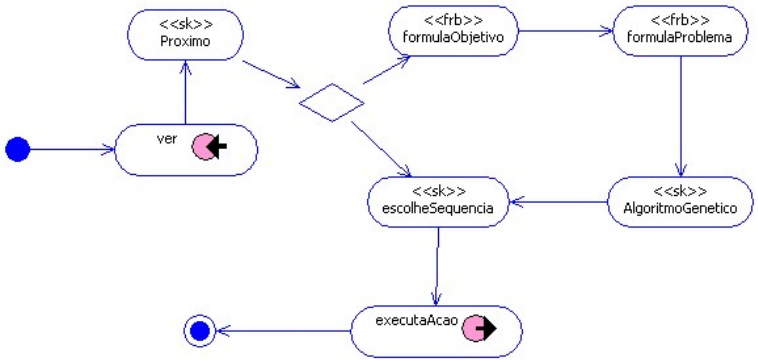

Figura 4.27: Diagrama de Atividades do AgenteGerente

ele possui. No modelo de interação são criados os diagramas de seqüência de mensagens que descrevem a comunicação entre os agentes. A Figura 4.28 demonstra o modelo de interação para o problema proposto.

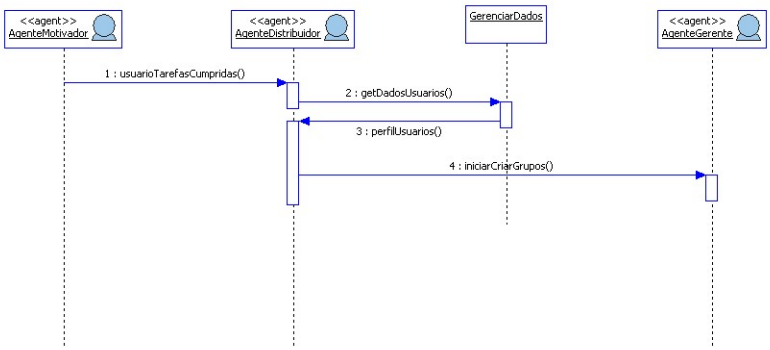

Figura 4.28: Modelo de interação para uma das comunicações entre os agentes

O ultimo modelo a ser definido na metodologia trata-se do modelo de projeto, neste modelo deve ser criado um diagrama detalhado demonstrando a integração entre os agentes encontrados e as classes de objetos do sistema, isto permitira aos programadores uma visão detalhada do sistema. Neste modelo também são descritos os diagramas de deployment do SMA e através deste diagramas pode ser visualizado a movimentação de agentes entre servidores diferentes quando necessário.

5 Considerações Finais e Trabalhos Futuros

A extensão proposta busca possibilitar a representação dos conceitos que compõem as características de racionalidade de um agente de software e assim permitir ao projetista do SMA modelar a arquitetura interna de um agente com base nos modelos propostos por Wooldridge (2002) e Russel e Norvig (1995).

Atualmente, existem diversas metodologias que podem ser utilizadas no desenvolvimento de sistemas multiagentes. Neste trabalho, foram estudadas e avaliadas nove metodologias, demonstrando suas potencialidades e suas fraquezas. Dessas metodologias, MAS-CommmonKADS foi escolhida para ser estendida, pois, além de ser capaz de representar uma grande quantidade de conceitos, é uma das metodologias que apresentam um maior nível de detalhamento em cada modelo gerado.

Além disso, também foi estudada a linguagem de modelagem AML, apresentada na seção 2.3.4. A AML é importantíssima para o trabalho aqui proposto, pois ela define modelos para o desenvolvimento de sistemas multiagentes. Dessa forma, a AML possibilita a modelagem dos principais aspectos estruturais e dinâmicos de um SMA. A partir da extensão desta linguagem, foi possível permitir que a metodologia escolhida tratasse com detalhes o desenvolvimento de agentes racionais.

Na extensão proposta, são consideradas todas as características necessárias para a definição completa de um SMA. Com a integração da linguagem AML estendida e a metodologia MAS-CommonKADS estendida, é possível não só descrever as capacidades externas de um agente, como suas interações e organização, como também é possível descrever com detalhes a capacidade de raciocínio dos agentes do SMA. Este é o ponto chave deste trabalho, pois as outras metodologias orientadas a agentes não possuem esta capacidade de representação. Outro ponto importante é que, com a integração com a AML, é possível modelar de forma simples e de maneira intuitiva o SMA, além de possibilitar a aplicação da metodologia através de uma ferramenta de modelagem que facilmente pode ser estendida para gerar códigos a partir do modelo.

Um trabalho fortemente relacionado é proposto em Gonçalves et al. (2010). Nesse trabalho, é descrito uma extensão à linguagem MAS-ML para a representação das arquiteturas internas dos agentes. Porém, o mesmo não possui nenhum detalhamento de como as informações perceptivas, recebidas através dos sensores, podem ser tratadas usando as várias técnicas de IA. Além disso, o trabalho se limita à extensão da linguagem e não trabalha com nenhuma metodologia específica para o processo de desenvolvimento do SMA.

Do estudo realizado, as maiores contribuições são: (i) avaliação das metodologias e linguagens de agentes, que auxiliará os projetistas na escolha da que melhor se adapte ao seu projeto; (ii) padronização dos modelos da metodologia MAS-CommonKADS através da AML, possibilitando assim uma modelagem mais simples e clara do SMA; (iii) adição de conceitos relativos à racionalidade do agente e a possibilidade de tratamento de informações internas ao agente com o uso de técnicas de IA; (iv) possibilidade de modelar agentes e objetos em uma única metodologia, e, finalmente, (v) uma ferramenta de modelagem para a metodologia.

Apesar das melhorias realizadas, surgem diversos trabalhos futuros para o trabalho desenvolvido como:

- Geração automática de código através da ferramenta StarUML;
- Ferramenta web para a modelagem colaborativa de projetos, essa ferramenta está fase de implementação por alunos bolsistas do IFRN Campus Caicó;
- Aplicação da metodologia em um estudo de caso real, este estudo está em fase de desenvolvimento sendo aplicado a um projeto da Petrobrás;
- Gerar modelos que possam representar corretamente os modelos de aprendizagem propostos por Russel e Norvig (1995).

Referências Bibliográficas

AGUILAR, J.; VIZCARRONDO, J.; PEROZO, N. Definition of a verification method for the masina methodology. 2008. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary10.1.1.103.2990>.

ALHASHEL, E.; BALACHANDRAN, B. M.; SHARMA, D. Extending prometheus with agent cooperation. IEEE Computer Society, Washington, DC, USA, p. 912–918, 2008.

ANWER, S.; IKRAM, N. Goal oriented requirement engineering: A critical study of techniques. In: *APSEC '06: Proceedings of the XIII Asia Pacific Software Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2006. p. 121–130. ISBN 0-7695-2685-3.

BADR, I.; MUBARAK, H.; GöHNER, P. Extending the mase methodology for the development of embedded real-time systems. Springer-Verlag, Berlin, Heidelberg, p. 106–122, 2008.

BASSANI, P. S. et al. Em busca de uma proposta metodológica para o desenvolvimento de software educativo colaborativo. In: *Revista Novas Tecnologias na Educação*. [S.l.: s.n.], 2006. v. 4, n. 1.

BASTOS, R. M.; OLIVEIRA, J. P. M. A conceptual modeling framework for multi-agent information systems. *19th International Conference on Conceptual Modeling*, n. 9-12, Outubro 2000.

BASTOS, R. M.; OLIVEIRA, J. P. M. A conceptual modeling framework for multi-agent information systems. *19th International Conference on Conceptual Modeling*, n. 9-12, Outubro 2000.

BITTENCOURT, G. Inteligência artificial distribuída. *I workshop de computação do ITA*, 1998.

BOUQUET, F. et al. A test generation solution to automate software testing. In: *Proceedings of the 3rd international workshop on Automation of software test*. New York, NY, USA: ACM, 2008. (AST '08), p. 45–48. ISBN 978-1-60558-030-2. Disponível em: <http://doi.acm.org/10.1145/1370042.1370052>.

BRANDAO, A. A. F. *Um método para estruturação e análise de modelos de sistemas multiagentes baseado em ontologias*. Tese (Doutorado) — PUC-Rio, 2005.

CAFARATE, L. S. *Utilização da Engenharia de Software Orientada a Agentes na Modelagem de um Sistema de Seleção de Pessoas* . Dissertação (Mestrado) — Universidade Federal de Santa Maria - UFSM, 2008.

CARVALHO, R. W. d. *Um Ambiente de Suporte para uma Linguagem de Modelagem de Sistemas Multi-Agentes*. Dissertação (Mestrado) — PUC-Rio, 2005.

CASTRO, J.; ALENCAR, F.; SILVA, C. T. L. L. Engenharia de software orientada a agentes. In: BREITMAN, K.; ANIDO, R. (Ed.). *Atualizações em Informática*. Rio de Janeiro: Editora PUC-Rio, 2006. p. 245–282.

CERVENKA, R.; TRENCANSKY, I. *Agent Modeling Language: Language specification. Version 0.9*. Whitestein Technologies, 2004.

CERVENKA, R.; TRENCANSKY, I. *The Agent Modeling Language - AML: A Comprehensive Approach to Modeling Multi-Agent Systems (Whitestein Series in Software Agent Technologies and Autonomic Computing)*. [S.l.]: Birkhäuser Basel, 2007. ISBN 376438395X.

CHELLA, A. et al. Agile passi: An agile process for designing agents. *INTERNATIONAL JOURNAL OF COMPUTER SYSTEMS SCIENCE & ENGINEERING. SPECIAL ISSUE ON "SOFTWARE*, 2006.

CHELLA, A.; COSSENTINO, M.; SEIDITA, V. Towards a methodology for designing artificial conscious robotic systems. 2009. Disponível em: <http://www.aaai.org/ocs/index.php/FSS/FSS09/paper/view/882/1274>.

COSSENTINO, M. From requirements to code with the passi methodology. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005.

COSSENTINO, M. et al. Passim: a simulation-based process for the development of multi-agent systems. *Int. J. Agent-Oriented Softw. Eng.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 2, p. 132–170, 2008. ISSN 1746-1375.

DANC, J. *Formal Specification of AML*. Dissertação (Mestrado) — Comenius University, 2008.

DÁRIO, C. F. B. *Uma Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2005.

DELOACH, S. Analysis and design using mase and agenttoll. In: *Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*. [S.l.: s.n.], 2001.

DELOACH, S. A. Omacs a framework for adaptive, complex systems. In: DIGNUM, V. (Ed.). *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. [S.l.]: IGI Global, 2009.

DELOACH, S. A.; KUMAR, M. Multi-agent systems engineering: An overview and case study. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005. p. 317–340.

DILEO, J.; JACOBS, T.; DELOACH, S. Integrating ontologies into multiagent systems engineering. In: *Fourth International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002) at AAMAS '02*. [S.l.: s.n.], 2002.

ENTERPRISEARCHITECT. *Enterprise Architect - UML Design Tools and UML CASE tools for software development*. http://www.sparxsystems.com.au/products/ea/index.html: [s.n.].

FAR, B. H.; MUDIGONDA, V.; ELAMY, A. H. A general purpose software evaluation system. *Informatica (Slovenia)*, v. 33, n. 3, p. 261–270, 2009.

FARIAS, K. et al. Mas-ml tool: Um ambiente de modelagem de sistemas multi-agentes. *Fifth Workshop on Software Engineering for Agent-oriented Systems*, 2009.

FERNÁNDEZ, C. A. I. *Definición de uma metodología para el desarrollo de sistemas multiagente*. Tese (Doutorado) — Universidad Politécnica de Madrid, 1998.

FRANKLIN, S.; GRAESSE, A. Is it an agent, or just a program? a taxonomy for autonomous agents. In: *third international workshop on agents theories*. [S.l.]: Springer-Verlag, 1996.

GARCIA-OJEDA, J. C.; DELOACH, S. A.; ROBBY. agenttool process editor: supporting the design of tailored agent-based processes. ACM, New York, NY, USA, p. 707–714, 2009.

GARCIA-OJEDA, J. C. et al. O-mase: a customizable approach to developing multiagent development processes. In: *AOSE'07: Proceedings of the 8th international conference on Agent-oriented software engineering VIII*. Berlin, Heidelberg: Springer-Verlag, 2008. p. 1–15. ISBN 3-540-79487-5, 978-3-540-79487-5.

GARIJO, F. J.; GóMEZ-SANZ, J. J.; MASSONET, P. The message methodology for agent-oriented analysis and design. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005.

GIORGINI, P. et al. Tropos: A requirements-driven methodology for agent-oriented software. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005. p. 20–45.

GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. The tropos software development methodology: Processes, models and diagrams. In: GIUNCHIGLIA, F.; ODELL, J.; WEIß, G. (Ed.). *International Workshop on Agent-Oriented Software Engineering*. Bologna: [s.n.], 2002. p. 162–173.

GOMEZ-SANZ, J. J. et al. Ingenias development kit: a visual multi-agent system development environment. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008. p. 1675–1676.

GONçALVES, E. J. T. et al. Towards the modeling reactive and proactive agents by using mas-ml. In: *SAC 10: Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010. p. 936–937. ISBN 978-1-60558-639-7.

GORODETSKY, V. et al. Masdk software tool and technology supported. In: *Integration of Knowledge Intensive Multi-Agent Systems, 2005. International Conference on*. [S.l.: s.n.], 2005. p. 528 – 533.

HENDERSON-SELLERS, B. Creating a comprehensive agent-oriented methodology - using method engineering and the open metamodel. In: *the OPEN metamodel, Chapter 13 in Agent-Oriented Methodologies (eds. B. Henderson-Sellers and P. Giorgini), Idea Group*. [S.l.: s.n.], 2005. p. 11.

HUANG, W.; EL-DARZI, E.; JIN, L. Extending the gaia methodology for the design and development of agent-based software systems. IEEE Computer Society, Washington, DC, USA, p. 159–168, 2007.

IBRAHIM, H.; FAR, B. H.; EBERLEIN, A. Scalability improvement in software evaluation methodologies. In: *IRI'09: Proceedings of the 10th IEEE international conference on Information Reuse & Integration*. Piscataway, NJ, USA: IEEE Press, 2009. p. 236–241. ISBN 978-1-4244-4114-3.

IGLESIAS, C. A.; GARIJO, M. The agent-oriented methodology mas-commonkads. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005. p. 46–78.

INSFRáN, E. *A Requirements Engineering Approach for Object-Oriented Conceptual Modeling*. Tese (Doutorado) — Universidad Politécnica de Valencia, 2003.

JENNINGS, N. R. Coordination techniques for dai. In: O'HARE, G.; JENNINGS, N. R. (Ed.). *Foundations of distributed artificial intelligence*. [S.l.]: John Wiley and Sons, 1996.

JUAN, A. P. T.; STERLING., L. Roadmap: Extending the gaia methodology for complex open systems. 2002.

JUNG, J.; LIU, C.-C. Multi-agent system technologies and an application for power system vulnerability. *IEEE*, p. 52–55, 2003.

KASABOV, N. K. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. Cambridge, MA, USA: MIT Press, 1996. ISBN 0262112124.

LIN, C.-E. et al. A methodology to evaluate agent oriented software engineering techniques. In: *40th Annual Hawaii International Conference on System Sciences*. [S.l.]: IEEE Computer Society, 2007.

MENDES, F. M.; II, M. J. de O. M. Multiagent system for supporting the knowledge management in the software process. In: RAMACHANDRAN, D. M. (Ed.). *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*. [S.l.]: IGI-Global, 2010.

NERY, H. d. A.; OLIVEIRA, D. G. D.; FURTADO, V. Aorml uma linguagem para modelagem de uma aplicação multi-agentes: Uma aplicação no sistema expertcop. In: *X Encontro de Iniciação à Pesquisa*. Fortaleza: UNIFOR, 2004. v. 10, p. 184–184.

NUNES, I. et al. Extending passi to model multi-agent systems product lines. In: *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*. New York, NY, USA: ACM, 2009. p. 729–730. ISBN 978-1-60558-166-8.

ODELL, J. D.; PARUNAK, H. V. D.; BAUER, B. S. V. Representing agent interaction protocols in uml. *22nd International Conference on Software Engineering (ISCE)*, p. 121–140, 2001.

PADGHAM, L.; WINIKOFF, M. Prometheus: A practical agent-oriented methodology. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005. p. 107–135.

PADILHA, T. P. P.; JáCOME, T. O uso de técnicas de modelagem de agentes em ambientes educacionais. *VI Congresso Iberoamericano de Informática Educativa*, 2002.

PARROTT, L.; LACROIX, R.; WADE., K. M. Design considerations for the implementation of multi-agent systems in the dairy industry. *Computers and Electronics in Agriculture*, v. 38 (2), p. 79–98, 2003.

PLUGIN, P. *Quick Start Manual*. Junho 2010. Disponível em: <http://code.google.com/p/pdt-plugin/wiki/QuickStartManual>.

RAO, A.; GEORGEFF, M. Bdi agents: from theory to practice. In: *Technical Note 56*. [S.l.]: Australian Artificial Intelligence Institute, 1995.

REZENDE, S. O. *Sistemas inteligentes: fundamentos e aplicações*. Barueri, SP: Manole, 2003.

RODRIGUEZ, L. et al. Improving the quality of agent-based systems: Integration of requirements modeling into gaia. In: *QSIC '09: Proceedings of the 2009 Ninth International Conference on Quality Software*. Washington, DC, USA: IEEE Computer Society, 2009. p. 278–283. ISBN 978-0-7695-3828-0.

ROSE. *IBM Rational Rose*. http://www-01.ibm.com/software/awdtools/developer/rose/: [s.n.].

ROUGEMAILLE, S. et al. Adelfe design, amas-ml in action. Springer-Verlag, Berlin, Heidelberg, p. 105–120, 2009.

RUSSEL, S.; NORVIG, P. *Artificial Intelligence: a modern approach*. New Jersey: Prentice-Hall, 1995.

SCHREIBER, G. et al. *Knowledge Engineering and Management: The CommonKADS Methodology*. [S.l.]: MIT Press, 2000.

SILVA, V. T. da. *Uma Linguagem de Modelagem para Sistemas Multi-agentes Baseada em um Framework Conceitual para Agentes e Objetos*. Tese (Doutorado) — PUC-Rio, 2004.

SILVA, V. T. da; CHOREN, R.; LUCENA, C. J. P. de. A uml based approach for modeling and implementing multi-agent systems. In: *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. Washington, DC, USA: IEEE Computer Society, 2004. p. 914–921. ISBN 1-58113-864-4.

STERLING, L.; TAVETER, K. *The Art of Agent-Oriented Modeling*. [S.l.]: The MIT Press, 2009. ISBN 0262013118, 9780262013116.

SUN, H.; THANGARAJAH, J.; PADGHAM, L. Eclipse-based prometheus design tool. *9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

TAOM4E. *TAOM4E*. Junho 2010. Disponível em: <http://www.inf.unioeste.br/ les/materiais/screen_taom4e.jpg>.

TRAN, Q.-N. N.; LOW, G. Mobmas: A methodology for ontology-based multi-agent systems development. *Inf. Softw. Technol.*, Butterworth-Heinemann, Newton, MA, USA, v. 50, n. 7-8, 2008. ISSN 0950-5849.

TRAN, Q.-N. N.; LOW, G. C. Comparison of ten agent-oriented methodologies. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005.

TRENCANSKY, I.; CERVENKA, R.; GREENWOOD, D. Applying a uml-based agent modeling language to the autonomic computing domain. In: *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 2006. p. 521–529. ISBN 1-59593-491-X.

TVEIT, A. *A Survey of Agent-Oriented Software Engineering*. May 2001. Proc. of the First NTNU CSGS Conference (http://www.amundt.org). Disponível em: <http://www.abiody.com/jfipa/publications/AgentOrientedSoftwareEngineering/>.

WAGNER, G. A uml profile for external aor models. *Third Workshop on Agent-Oriented Software Engineering*, 2002.

WEISS, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 2000. ISBN 0262731312.

WOOLDRIDGE, M. Intelligent agents. In: WEISS, G. (Ed.). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. [S.l.]: The MIT Press, 1999.

WOOLDRIDGE, M. Intelligent agents. In: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. [S.l.: s.n.], 2002.

YU, E. *Modelling strategic relationships for process reengineering*. Tese (Doutorado) — University of Toronto, 1995.

ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. Multi-agent systems as computational organizations: The gaia methodology. In: GIORGINI, B. H.-S. e P. (Ed.). *Agented-Oriented Methodologies*. [S.l.]: IDEA Group Publishing, 2005. p. 136–171.