UNIVERSIDADE ESTADUAL DO CEARÁ
CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT
MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

MODELAGEM DE ARQUITETURAS INTERNAS DE AGENTES DE SOFTWARE UTILIZANDO A LINGUAGEM MAS-ML 2.0

ENYO JOSE TAVARES GONCALVES

FORTALEZA - CEARÁ
2009

ENYO JOSE TAVARES GONCALVES

MODELAGEM DE ARQUITETURAS INTERNAS DE AGENTES DE SOFTWARE UTILIZANDO A LINGUAGEM MAS-ML 2.0

Dissertação submetida à comissão examinadora do Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará – UECE, como requisito parcial para obtenção de título de Mestre em Ciência da Computação.

Orientadora: Profª. Dra. Mariela Inés Cortés

Co-Orientadora: Profª. Dra. Viviane Torres da Silva

FORTALEZA – CEARÁ
2009

G635m Gonçalves, Enyo J. T.
Modelagem de arquiteturas internas de agentes de software utilizando a linguagem MAS-ML 2.0/ Enyo José Tavares Gonçalves. __ Fortaleza, 2009.
106 p. ; il.
Orientadora: Profª. Dra. Mariela Inés Cortés
Dissertação (Mestrado Acadêmico em Ciência da Computação) – Universidade Estadual do Ceará. Centro de Ciência e Tecnologia.
1. Sistemas Multi-agentes. 2. Framework Conceitual. 3. Linguagem de Modelagem. 4. Metamodelo. 5. Arquitetura interna de agente. I. Universidade Estadual do Ceará. Centro de Ciência e Tecnologia.
CDD: 001.6

ENYO JOSÉ TAVARES GONÇALVES

MODELAGEM DE ARQUITETURAS INTERNAS DE AGENTES DE SOFTWARE UTILIZANDO A LINGUAGEM MAS-ML 2.0

Dissertação de Mestrado submetida à comissão examinadora do Mestrado Acadêmico em Ciência da Computação, da Universidade Estadual do Ceará – UECE, como requisito parcial para obtenção de título de Mestre em Ciência da Computação. Aprovada pela comissão examinadora abaixo assinada

Dissertação aprovada em: 11/12/2009

BANCA EXAMINADORA

_____

Profª. Dra. Mariela Inés Cortés (Orientadora)
Universidade Estadual do Ceará - UECE

_____

Profª. Dra. Viviane Torres da Silva (Co-Orientadora)
Universidade Federal Fluminense - UFF

_____

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

_____

Prof. Dr. Francisco Milton Mendes Neto
Universidade Federal Rural do Semi-Árido – UFERSA

FORTALEZA – CEARÁ
2009

Dedico este trabalho aos meus pais, Erardo e Margarida, meus exemplos...

"Lembro-me do teu olhar, do teu sorriso, dos teus passos acompanhando o meu andar, da tua mão amparando-me quando eu caía, das tuas lagrimas lamentando quando eu perdia, mas me lembro principalmente do teu sorriso, do brilho de satisfação que percorria tua face, e lembro-me ainda do som das tuas palmas aclamando-me quando eu vencia".
(Joyce Maria Worschech Gabrielli - adaptado).

... e à minha namorada Denise, minha fortaleza em todos os momentos.

Venci... Por mim e por vocês!

Agradecimentos

Inicialmente a Deus, por conduzir meus passos pelos caminhos certos na vida. Ao espírito santo por ter iluminado meus pensamentos em meio às dúvidas e angustias, dando-me sabedoria, discernimento e perseverança.

A minha orientadora Profª. Mariela Cortés e minha co-orientadora Profª. Viviane Torres, por terem direcionado meus estudos e acompanhado o andamento da pesquisa de maneira incansável e sempre presente. Obrigado pela disponibilidade, colaboração e paciência nos momentos de dúvidas e correções de artigos, da proposta de dissertação e da dissertação. Parabéns pela competência na condução desta.

Ao Prof. Gustavo, agradeço a colaboração desde o inicio do mestrado e a disponibilidade que sempre encontrei para tirar dúvidas e mostrar os resultados.

Aos colegas de mestrado Gilzamir, Tales, Marçal, Marcelo e Rafael Carmo, pelo excelente convívio durante este período, principalmente a Gilzamir pela parceria relacionada às nossas pesquisas.

Aos meus familiares e amigos, pela torcida e apoio no percurso deste estudo.

Aos pesquisadores da PUC-RIO, Prof. Lucena, Andrew, Baldoino, Ingrid, Camila e Kleinner, pela colaboração durante o intercâmbio lá realizado, especialmente ao Kleinner pelos esclarecimentos em relação à ferramenta de modelagem MAS-ML tool.

A todos os professores e funcionários do Mestrado Acadêmico em Ciência da Computação da UECE, pelo suporte dado durante estes dois anos.

Ao Instituto Federal de Educação Ciência e Tecnologia.

À Fundação Cearense de Amparo a Pesquisa (Funcap) e a CAPES, pelo apoio financeiro parcial.

"Determinação, coragem e autoconfiança são fatores decisivos para o sucesso. Não importa quais sejam os obstáculos e as dificuldades. Se estamos possuídos de uma inabalável determinação, conseguiremos superá-los.Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho."

(Dalai Lama, do livro O Caminho da Tranquilidade)

Resumo

Engenharia de Software para Sistemas Multi-Agentes (SMAs) vem se desenvolvendo significativamente nos últimos anos. O principio básico de modelagem aplicado a SMAs requer que elementos específicos do domínio descritos através das suas características estáticas e comportamentais possam ser modelados através de artefatos de modelo gerados para dar apoio ao processo de desenvolvimento. A teoria de agentes define um conjunto de propriedades, atributos, componentes mentais e comportamento, a partir dos quais a arquitetura interna de um agente pode ser estabelecida. Tais propriedades precisam ser modeladas corretamente para posteriormente serem mapeadas para uma implementação especifica em cada caso.

Esta dissertação aborda o estudo das principais arquiteturas internas de agente no intuito de determinar as características (atributos e comportamento) inerentes a cada uma delas, as quais precisam ser mapeadas para elementos de modelagem através de uma linguagem de modelagem para SMAs. Neste cenário destacamos MAS-ML, uma linguagem de modelagem para SMAs baseada em um *framework* conceitual que define conceitos relacionados a agentes com arquitetura pró-ativa baseada em objetivo e guiada por plano. O presente trabalho de dissertação envolve a evolução do *framework* conceitual TAO e da linguagem MAS-ML de forma a possibilitar a modelagem de agentes com diferentes arquiteturas internas.

Palavras-Chave

Sistemas Multi-agentes; *Framework* Conceitual; Linguagem de Modelagem; Metamodelo; Arquitetura interna de agente.

Abstract

Software Engineering for multi-agent systems (MASs) has evolved significantly in the last years. The modeling of such systems requires the explicit modeling of domain-dependent elements together with their static and behavioral properties by the use of modeling artifacts generated to support the development process. The agent theory defines a set of properties, attributes, mental components and behavior, from which the internal architecture of an agent can be established. These properties must be correctly modeled to facilitate the mapping to a specific implementation in each case.

This work involves the study of main internal agent architectures in order to determine the related characteristics (attributes and behavior), which must be mapped to modeling elements using a modeling language for MASs. In this context, we highlight MAS-ML, a MASs modeling language based on a conceptual framework that defines concepts related with agents based on goal-oriented architecture guided by plan. This work involves the extension of the conceptual framework TAO and MAS-ML in order to allow the modeling of agents with different internal architectures.

Keywords

Multi-Agent System; Conceptual Framework; Modeling Language; Metamodel; Internal agent architecture.

Sumário

Lista de Figuras

Lista de Siglas e Abreviações

AML	*Agent Modeling Language*
AOR	*Agent-Object-Relationship*
AORML	*Agent-Object-Relationship Modeling Language*
AUML	*Agent Unified Modeling Language*
EMOF	*Essential Meta-Object Facility*
GMF	*Graphical Modeling Framework*
IA	Inteligência Artificial
KAoS	*Knowledgeable Agent-oriented System*
MAS-ML	*Multi-Agent System Modeling Language*
MAS-ML 2.0	*Multi-Agent System Modeling Language* estendida
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
SMA	Sistema Multi-Agentes
SMAs	Sistemas Multi-Agentes
TAC	*Trading Agent Competition*
TAC-SCM	*Trading Agent Competition - Supply Chain Management*
TAO	*Taming Agents and Objects*
UML	*Unified Modeling Language*

1. Introdução

Com a necessidade de desenvolvimento de sistemas complexos, a tecnologia de agentes vem, a cada dia, sendo mais utilizada na resolução de problemas do mundo real. De acordo com Russell e Norvig (2004), um agente é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Diferente dos objetos, agentes são entidades (i) autônomas e não passivas; e (ii) capazes de interagir através de troca de mensagens e não pela invocação explícita de uma tarefa, como no caso de objetos [Wagner 2003]. O termo Sistema Multi-Agente (SMA) refere-se à sub-área de Inteligência Artificial 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 [Jennings 1996].

É inevitável comparar agentes e objetos, mas apesar de existirem semelhanças, as diferenças são claras. Wagner (2003) define duas grandes diferenças entre agentes e objetos: (i) enquanto um objeto possui uma estrutura genérica, o estado de um agente tem uma estrutura mental que consiste de componentes mentais como crenças e compromissos / reinvidicações, e (ii) enquanto em programação orientada a objetos as mensagens são codificadas em aplicações específicas de maneira ad-hoc, uma mensagem em programação orientada a agentes é codificada como um ato de comunicação oral de acordo com um padrão de linguagem de comunicação de agentes que é independente de aplicação.

Devido à diferença ontológica entre agentes e objetos, surge a necessidade de adequar as técnicas da Engenharia de Software tradicional aos problemas específicos do desenvolvimento orientado a agentes. Engenharia de Software orientada a agentes representa uma nova maneira de analisar, projetar, construir, gerenciar e manter software complexo envolvendo o paradigma de desenvolvimento orientado a agentes [Castro, Alencar e Silva 2006].

Wooldridge e Jennings (1994) dividem o estudo relacionado ao projeto e construção de agentes em três temas: Teoria de Agente, Arquiteturas de Agente e Linguagens de Agente. Entende-se *Arquitetura de Agente* como a aplicação dos conceitos da engenharia de software a modelos de agentes; pesquisas relacionadas à modelagem de arquiteturas de agentes concentram-se no problema da construção de

sistemas de software, de modo a satisfazer as propriedades estabelecidas na teoria de agente.

A arquitetura interna do agente define suas propriedades (componentes mentais e comportamento), determinando, consequentemente, uma implementação diferenciada para cada caso. As principais arquiteturas internas de agente definidas por Russell e Norvig (2004) são as seguintes: agente reativo simples, agente reativo baseado em conhecimento, agente baseado em objetivo e agente baseado em utilidade. Por exemplo, a atribuição de *crenças* e a utilização de uma *função-próximo* para auxiliar na escolha da próxima ação a ser executada são propriedades inerentes ao agente reativo baseado em conhecimento [Weiss 1999].

Um único SMA pode conter agentes com diferentes arquiteturas internas [Wooldridge e Jennings 1994]. Neste contexto, Weiss (1999) afirma que é possível balancear o comportamento dos agentes de um SMA em relação à sua pró-atividade e reatividade.

Analogamente é possível combinar diferentes arquiteturas pró-ativas em um SMA. Como exemplo, um SMA para leilões virtuais de computadores pessoais com atividades de compra de peças, montagem, venda e entrega pode ser modelado através de um agente baseado em utilidade, que irá se concentrar no gerenciamento, e um agente baseado em objetivo guiado por planejamento para controlar a produção (Seção 6.3).

Por outro lado, Sommerville (2007) cita que Engenharia de Software (ES) é uma disciplina da Engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção. Os fundamentos científicos para Engenharia de Software envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantido sua qualidade.

Neste contexto, a existência de uma linguagem que possibilite a modelagem de diferentes arquiteturas de agente torna-se essencial no desenvolvimento de SMAs.

1.1. Problema

Como será demonstrado no Capítulo 3, embora várias linguagens de modelagem para SMAs tenham sido propostas, ainda há a necessidade de uma linguagem de

modelagem que consiga modelar as diversas arquiteturas internas dos agentes inteligentes[1] (agente reativo simples, agente reativo baseado em conhecimento, agente baseado em objetivo e agente baseado em utilidade). A modelagem destas diversas arquiteturas se faz importante dada a necessidade de desenvolver diferentes SMAs onde agentes com diferentes características e propriedades possam interagir.

Várias linguagens de modelagem para SMAs foram propostas, algumas delas estão diretamente associadas a uma metodologia e outras não. Metodologias como Gaia, Message, Ingenias e Tropos[2] oferecem um fluxo a ser seguido para realizar a modelagem de SMA. Algumas delas oferecem suporte a arquiteturas pró-ativas e reativas, mas nenhuma consegue modelar as quatro arquiteturas propostas para agentes inteligentes definidas na literatura por Russell e Norvig [2004].

O escopo deste trabalho está relacionado à modelagem de agentes inteligentes (ou agentes racionais) através de uma linguagem de modelagem independente de metodologia, possibilitando a utilização da linguagem de acordo com a metodologia mais adequada ao projeto do SMA.

A UML é um padrão na área de modelagem para sistemas orientados a objetos e tem enorme aceitação tanto na indústria quanto na academia. Deste modo, tanto as fábricas de software, e consequentemente os projetistas, quanto a academia e os pesquisadores já estão familiarizados com a UML. Portanto a curva de aprendizado de um profissional que vai utilizar uma linguagem de modelagem que estende a UML é bem menor. Além disto, A UML vem evoluindo desde seu surgimento na década de 90, tornando-se uma linguagem de modelagem que disponibiliza cada vez mais recursos e diferentes visões da modelagem.

Apesar de tantas linguagens de modelagem existentes, ainda há a necessidade de uma linguagem de modelagem que (i) consiga modelar corretamente as arquiteturas internas dos agentes inteligentes; (ii) seja independente de metodologia; (iii) possua uma ontologia que descreva os elementos utilizados pela linguagem de modelagem e (iv) viabilize a utilização de padrões na notação adotada, tal como UML.

[1] Um agente inteligente (ou agente racional) é aquele que executa a ação correta [Russell e Norvig 2004].

[2] Estas e outras metodologias são descritas em Henderson-Sellers e Giorgini [2005].

1.2. Objetivos

A adequada modelagem de aplicações multi-agentes torna necessário o estabelecimento de técnicas que abordem as características principais das diferentes arquiteturas internas de agente disponíveis, onde cada arquitetura possui atributos e funcionalidades específicas.

As linguagens de modelagem para SMAs não contemplam os componentes de modelagem das principais arquiteturas internas existentes na literatura. MAS-ML é um destes casos, pois oferece suporte apenas a agentes baseados em objetivo guiados por plano. No entanto, MAS-ML destaca-se em relação às demais linguagens de modelagem por: (i) modelar agentes com objetivo, (ii) possuir uma ontologia adequada, (iii) oferecer suporte a objetos convencionais, (iv) identificar papéis, e (v) modelar adequadamente ambientes e a interação entre agente e ambiente.

Este trabalho de dissertação visa um estudo aprofundado de tais arquiteturas com o objetivo geral de determinar os elementos a serem contemplados na geração de artefatos de modelagem por uma linguagem de modelagem orientada a agentes. Este trabalho contempla a modelagem dos agentes inteligentes (ou agentes racionais): reativo simples, reativo baseado em conhecimento, pró-ativo baseado em objetivo guiado por planejamento e pró-ativo baseado em utilidade, definidos por Russell e Norvig (2004).

Os objetivos específicos do presente trabalho envolvem a evolução de MAS-ML e do o *framework* TAO, de modo que a linguagem seja capaz de modelar adequadamente agentes inteligentes.

As ferramentas existentes para modelagem de SMAs utilizando MAS-ML foram analisadas, com o intuito de selecionar uma para implementar a extensão proposta neste trabalho. Diversos estudos de caso foram realizados com o intuito de validar a evolução proposta.

Ao atingirmos nossos objetivos, MAS-ML deixa de ser uma linguagem capaz de modelar apenas agentes baseados em objetivos guiados por planos, e torna-se uma linguagem de modelagem que contempla a modelagem das principais arquiteturas internas de agente, além de manter as suas características próprias. O resultado trará benefícios aos profissionais envolvidos no desenvolvimento de SMAs, promovendo uma maior adequação entre a modelagem dos sistemas e sua implementação.

1.3. Justificativa

O processo de analisar um problema, encontrar uma solução, e expressá-la em uma linguagem de programação de alto nível pode ser visto como uma forma implícita de modelagem [France e Rumpe 2007]. Com isso em mente, é possível argumentar que o desenvolvimento de software é essencialmente uma atividade consistente em resolver problemas baseando-se em modelos. Consequentemente, a fim de se desenvolver aplicações seguindo o paradigma de SMAs, é necessário realizar uma atividade essencial: a modelagem de SMAs. Desse modo, as linguagens de modelagem para SMAs desempenham um papel central dentro do processo de desenvolvimento.

O conceito central do paradigma orientado a agentes é o próprio agente. Dependendo da arquitetura interna do agente a ser implementado, sua modelagem deve ser diferente. Por isto é importante a existência de uma linguagem de modelagem que consiga modelar as diferentes arquiteturas de agentes inteligentes.

Através da extensão de MAS-ML relacionada às arquiteturas internas, o projetista pode especificar a arquitetura interna a ser utilizada pelo desenvolvedor para cada um dos agentes inteligentes do SMA a ser desenvolvido. Deste modo, a modelagem terá a capacidade de representar a implementação de maneira correta.

1.4. Metodologia do trabalho

O embasamento teórico deste trabalho está fundamentado em livros, artigos de periódicos e conferências, dissertações de mestrado e teses de doutorado relacionadas ao tema em questão, e informações obtidas com pesquisadores de instituições de ensino com competência na área.

Com base no levantamento teórico realizado, foi estabelecida uma teoria dos elementos relacionados à modelagem dos agentes inteligentes. Com a teoria pronta, foi realizado um estudo das linguagens de modelagem existentes e a escolha de MAS-ML foi feita.

A linguagem escolhida foi estendida através da criação de novas metaclasses e de novos estereótipos para representar graficamente os agentes reativos simples, reativos baseados em conhecimento, baseados em objetivo e baseados em utilidade e, consequentemente, o *framework* TAO foi adaptado de modo a manter a consistência com MAS-ML. Em seguida, foram modeladas diversas aplicações fazendo uso de

diferentes arquiteturas de agentes inteligentes utilizando a linguagem MAS-ML estendida.

Um estudo sobre as ferramentas existentes para modelagem utilizando MAS-ML foi realizado com o intuito de selecionar a mais adequada. Em seguida, foi projetada e implementada a extensão na ferramenta escolhida, seguindo os conceitos de extensão para MAS-ML definidos neste trabalho.

A validação da proposta envolveu a realização de estudos de caso que comprovaram a eficiência da extensão proposta bem como a modelagem de exemplos utilizando a ferramenta de modelagem estendida.

Diversos artigos foram criados a partir dos resultados obtidos, gerando algumas publicações [Gonçalves et al. 2009], [Gonçalves et al. 2010a] e [Gonçalves et al. 2010b], e esta dissertação é o resultado final deste trabalho.

1.5. Estrutura da Dissertação

Esta dissertação está organizada da seguinte maneira:

- O Capítulo 2 apresenta o referencial teórico, sendo que o mesmo aborda Ambientes, Arquiteturas internas de agente, o *framework* TAO, MAS-ML e MAS-ML tool;
- O Capítulo 3 apresenta os trabalhos relacionados;
- O Capítulo 4 descreve a extensão proposta à MAS-ML e ao TAO;
- O Capítulo 5 aborda a extensão na ferramenta de modelagem de MAS-ML;
- O Capítulo 6 apresenta três estudos de caso que utilizam a linguagem de modelagem MAS-ML.
- Finalmente, o Capítulo 7 contém as considerações finais e os trabalhos futuros.

2. Referencial Teórico

Este capítulo apresenta alguns referenciais teóricos que fundamentaram a extensão proposta. Inicialmente, os ambientes e as arquiteturas de agente são descritos; em seguida, MAS-ML e o TAO são descritos de maneira resumida. Finalmente, a ferramenta MAS-ML Tool é apresentada.

2.1. Ambientes

Os ambientes podem ser classificados em categorias, onde cada categoria contribui na definição da arquitetura de agente mais apropriada, e posteriormente, na aplicabilidade de uma técnica para a implementação de agentes [Russell e Norvig 2004]. Os tipos de ambiente definidos por Russell e Norvig (2004) são:

- **Totalmente Observável ou Parcialmente Observável:** se o sensor do agente tem acesso completo ao estado do ambiente o tempo todo, conseguindo observar todos os aspectos relevantes para escolher uma ação a executar, neste caso o ambiente é dito de totalmente observável. Um ambiente pode ser parcialmente observável devido a ruído ou sensores não acurados, ou quando somente parte do estado do ambiente pode ser acessado pelo agente - por exemplo, o agente aspirador de pó com apenas um sensor de sujeira local não pode dizer se há sujeira em outros quadrados. Similarmente, um táxi automatizado não pode saber o que outros motoristas estão pensando.

- **Determinístico ou Estocástico:** se o próximo estado do ambiente é completamente determinado pelo estado atual e pelas ações executadas pelo agente, então podemos dizer que o ambiente é determinístico, ou seja, previsível; caso contrario, é estocástico. Se o ambiente é determinístico, exceto pelas ações de outros agentes, dizemos que o ambiente é estratégico.

- **Episódico ou Sequencial:** em um ambiente episódico, a experiência do agente é dividida em episódios atômicos. Cada episódio consiste da percepção do agente e da realização de uma única ação. Crucialmente, o próximo episódio não depende de ações realizadas em episódios anteriores. Um ambiente episódico, a escolha de uma ação depende somente do próprio episódio. Em um sistema onde o agente para apontar peças defeituosas em uma linha de montagem baseia cada decisão somente na peça atual, sem se preocupar com

decisões anteriores e a decisão atual não afeta decisões futuras, possui um ambiente episódico. Em ambientes sequenciais, a decisão atual pode afetar todas as decisões futuras. O jogo de xadrez e a simulação de um motorista de táxi são sequenciais: em ambos os casos, ações de curto prazo podem ter consequências de longo prazo. Ambientes episódicos são mais simples porque o agente não tem que pensar a frente, ou seja, não tem que pensar nas consequências futuras de suas ações, apenas nas consequências da ação atual.

- **Estático ou Dinâmico:** Se o ambiente pode mudar enquanto o agente está deliberando, podemos dizer que ele é dinâmico, senão, é estático. Ambientes estáticos são mais fáceis de lidar, pois o agente não precisa ficar observando o mundo enquanto está decidindo uma ação a executar, nem precisa se preocupar com a passagem do tempo. Ambientes dinâmicos, por outro lado, estão continuamente perguntando ao agente o que ele quer fazer; se ele não decidiu ainda, isto conta como se tivesse decidido por fazer nada. Se o ambiente não muda com a passagem do tempo, mas o desempenho do agente sim, então dizemos que ele é semi-dinâmico. Dirigir um táxi é claramente dinâmico. O jogo de xadrez, quando jogado com um relógio, é semi-dinâmico. Jogos de palavras cruzadas são estáticos.
- **Discreto ou Contínuo:** Um ambiente discreto, como o jogo de xadrez, tem um conjunto finito de estados, além de um discreto (finito) conjunto de percepções e ações. Dirigir um táxi é um problema de estados, ações e tempo contínuos.

Dependendo do tipo de ambiente em que o agente a ser implementado irá atuar, algumas arquiteturas internas de agente podem ser mais adequadas. O tipo de arquitetura interna de um agente também pode estar relacionada ao sub-problema que o mesmo resolverá no contexto do SMA que está inserido.

Segundo Russell e Norvig (2004) e Weiss (1999), uma solução interessante para ambientes complexos (parcialmente observáveis, estocástico e dinâmico, p. e.) é a criação de SMAs com diferentes arquiteturas internas.

2.2. Arquiteturas internas de agente

Existem diversas maneiras de classificar agentes, porém o critério mais aceito classifica agentes de acordo com sua arquitetura. A arquitetura de um agente específica como se dá o processo de deliberação e a escolha da ação a ser tomada, de acordo com as percepções obtidas [Wooldridge e Jennings 1995].

Russell e Norvig (2004) destacam quatro arquiteturas relacionadas aos agentes inteligentes: agentes reativos simples, agentes reativos baseados em conhecimento, agentes baseados em objetivo e agentes baseados em utilidade.

A Figura 1 representa esquematicamente um agente e seu ambiente, cuja interação é dada através dos respectivos sensores e atuadores. É possível observar na Figura 1 um símbolo de '?' situado entre as percepções do agente e suas ações, este símbolo representa a função agente que é diferente para cada arquitetura interna.

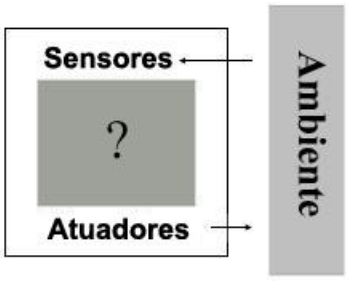

Figura 1 – Agente com seus sensores e atuadores interagindo com o ambiente (Adaptado) [Russell e Norvig 2004].

A concepção da função agente que faz o mapeamento entre os canais de percepção e de ação consiste em uma das principais tarefas da Inteligência Artificial. Dependendo do ambiente de tarefa, a concepção do agente pode não ser uma tarefa trivial. É necessário que as propriedades do ambiente interno do agente sejam adequadas às propriedades do ambiente externo onde os objetivos devem ser realizados.

O termo arquitetura interna foi definido na Introdução (Capítulo 1) como: A arquitetura interna do agente determina suas propriedades, atributos, componentes mentais e comportamento, determinando consequentemente, uma implementação diferenciada para cada caso.

As arquiteturas internas são definidas basicamente a partir dos princípios pró-ativo e reativo.

2.2.1. Pró-atividade e Reatividade

Um agente pró-ativo é capaz de tomar a iniciativa e selecionar ações de ***A*** (conjunto de possíveis ações atômicas ou planos), e/ou conceber sequências de ações que transformem o estado corrente em um dos estados meta possíveis para o ambiente de tarefa do agente. Supondo que as crenças do agente e suas pré-condições (definem a seleção de uma ação ou uma sequência de ações), não mudar durante a tomada de decisão e a execução da ação. Nesse caso, um estado desejado em **S** deverá ser produzido como efeito, isto é, as pós-condições estabelecidas nos objetivos do projeto do agente. Assim, o comportamento puramente pró-ativo é orientado por objetivos, podendo utilizar na tomada de decisão as informações sobre os estados desejados do ambiente.

O comportamento pró-ativo é principalmente adequado a problemas que ocorrem em ambientes de tarefa bem definidos, ou seja, que são: (a) determinísticos, isto é, o agente sabe exatamente a mudança de estado proporcionada no estado corrente, as pós-condições, por cada uma das ações possíveis; (b) observável, isto é, os sensores do agente têm acesso a toda a informação sobre o estado corrente do ambiente que é necessária para a seleção de ações; e (c) estático, isto é, o estado do ambiente e as pré-condições não mudam enquanto o agente está deliberando e/ou executando uma sequência de ações. Conceber agentes pró-ativos em ambientes estocásticos e parcialmente observáveis pode ser uma tarefa bastante complexa.

Além do mais, em ambientes dinâmicos, cujas pré-condições podem mudar enquanto o agente estiver deliberando, o comportamento puramente pró-ativo pode produzir efeitos indesejados. Nestes casos, o agente deve ser capaz de interromper o processo de deliberação, ou de execução de um plano que foi deliberado, e responder de maneira adequada às novas condições estabelecidas no ambiente, ou seja, selecionar em tempo aceitável ações que sejam capazes de produzir os efeitos desejados. Assim, nestes ambientes onde o tempo de resposta é crucial, a reatividade é uma propriedade necessária que, combinada à pró-atividade, busca colaborar para a realização dos objetivos do projeto do agente.

A literatura sobre agentes disponibiliza informações sobre diversas arquiteturas abstratas e concretas reativas, pró-ativas e híbridas [Weiss 1999] e [Russell e Norvig 2004]. No que diz respeito à pró-atividade, por serem amplamente utilizadas merecem

destaque as arquiteturas abstratas dos agentes orientados por objetivos e por utilidade [Russell e Norvig 2004] e [Weiss 1999]. No que diz respeito à reatividade, merecem destaque as arquiteturas do agente reativo (simples) e do agente reativo com estado interno (modelo) [Russell e Norvig 2004] e [Weiss 1999].

2.2.2. Agente reativo simples

Os agentes puramente reativos devem responder continuamente às mudanças que ocorrem em seu ambiente. Dependendo do ambiente de tarefa, o projeto deste tipo de agente pode ser muito simples. Por exemplo, para o caso em que o ambiente é observável e determinístico, a arquitetura abstrata do agente padrão pode ser refinada dando origem aos agentes reativos [Weiss 1999] e [Wooldridge 2002]. A Figura 2 ilustra algumas das informações e os subsistemas (módulos) processadores de informação propostos em um primeiro refinamento. O módulo processador ***Ver*** identifica os sensores componentes da arquitetura física do agente. O módulo processador ***Ação*** identifica a função tomada de decisão intrínseca no programa agente que realiza o mapeamento *percepção-ação*.

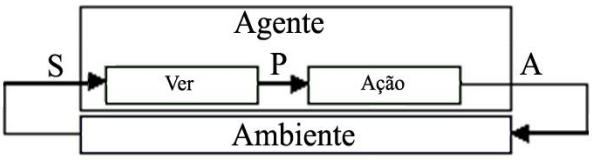

Figura 2 – Agente padrão [Weiss 1999].

Mais especificamente, o funcionamento da arquitetura do agente padrão envolve quatro passos, ou seja: (1) por meio dos sensores, o agente recebe informações do ambiente que são sequências de estados definidos em $\mathbf{S} = \{s_1, ..., s_n\}$; (2) um subsistema de percepção, **Ver: S** $\rightarrow$ **P**, processa cada estado de uma sequência **S*** e mapeia em uma de m percepções, $\boldsymbol{P} = \{p_1, ..., p_m\}$, que são representações de aspectos dos estados de S que estão acessíveis ao agente para a tomada de decisão; (3) um subsistema de tomada de decisão, Ação: P* $\rightarrow$ A, processa as sequências perceptivas P*, resultantes de S*, e seleciona uma de **l** ações em $\boldsymbol{A} = \{a_1, ..., a_l\}$; e (4) por meio de atuadores o agente envia a ação selecionada para o ambiente.

Os agentes puramente reativos não armazenam sequências perceptivas **P***, mas tomam decisões estritamente baseados nas percepções correntes, **P**, sem levar em consideração qualquer tipo de informação histórica. Isto implica em uma especialização

do subsistema de tomada de decisão do agente, ou seja, **Ação: P → A**. É mais simples conceber agentes reativos para ambientes estáticos e observáveis. Por exemplo, pode ser suficiente a concretização de um programa agente a partir de um comportamento formalizado em máquinas de estados finitos com saída [Menezes 2005] e [Hopcroft e Ullman 2002]. De maneira equivalente, [Russell e Norvig 2004] incorporaram um conjunto de regras condição-ação no subsistema de tomada de decisão do agente reativo.

As regras condição-ação são formalizações de associações comuns observadas entre certas percepções e ações possíveis para o agente. Assim, o agente tem pré-computado um conjunto de ações para várias situações previstas, o que simplifica a implementação do mapeamento *percepção-ação*. Por exemplo, vale destacar o projeto do agente reativo aspirador de pó em um mundo simplificado [Russell e Norvig 2004]. O aspirador deve ser capaz de limpar um ambiente estático formado por duas salas (*A* e *B*) que podem conter sujeira. O ambiente é parcialmente observável, ou seja, o sensor do aspirador disponibiliza apenas informações locais a respeito da sala (*A* ou *B*) e do estado da sala (*L*-limpa ou *S*-suja) em que o agente está. Os atuadores do aspirador permitem que ele *Aspire*, vá para a *Esquerda* ou para *Direita* de uma sala. A Figura 3 ilustra os oito estados possíveis no mundo do aspirador de pó simplificado, o comportamento especificado em uma máquina de Moore e um conjunto de regras *condição-ação* correspondente.

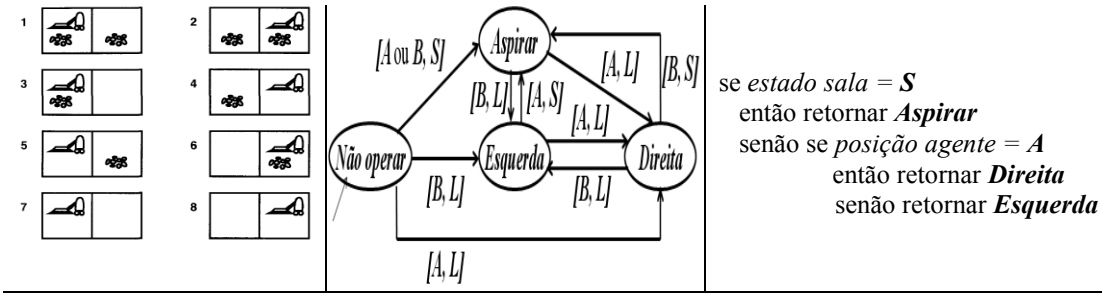

Figura 3 – Aspirador de pó simplificado [Russell e Norvig 2004].

Considerando o ambiente simplificado (estático e parcialmente observável) e o objetivo do agente (limpar as duas salas), pode-se afirmar que, independentemente da configuração inicial do mundo, o agente reativo aspirador de pó simplificado esboçado na Figura 3 é racional. Para este mundo, não existe outro mapeamento capaz de produzir um desempenho melhor. Mesmo que a medida de avaliação de desempenho do agente seja alterada, penalizando-o com um ponto por se locomover de uma sala para

outra, é possível conceber um agente racional puramente reativo, capaz de parar quando as salas estiverem limpas. Entretanto, em mundos que tenham uma geografia mais complexa (extensão, limites e obstáculos) uma abordagem puramente reativa pode não ser adequada. Nestes mundos, pode ser mais interessante concretizar a arquitetura do agente com estado interno (modelo do mundo) e regras *condição-ação*, um refinamento da arquitetura do agente reativo simples.

Um agente reativo simples pode ser mais adequado para ambientes completamente observáveis, dado que este tipo de agente deve tomar decisões com base apenas na percepção atual e, consequentemente, não mantém o histórico de percepções passadas. Nesse caso, é comum dizer que o agente esquece seu próprio passado.

2.2.3. Agente reativo baseado em conhecimento (ou modelo)

O agente reativo baseado em modelo (conhecimento a respeito do estado do ambiente) é uma arquitetura adequada às situações de projeto em que o ambiente é parcialmente observável e sua geografia é desconhecida. Por exemplo, o mundo do aspirador descrito acima, em que os sensores do agente não oferecem acesso ao estado completo do ambiente (percebe apenas a sala e o estado da sala em que está). Para lidar com a falta de acessibilidade, o agente mantém em uma estrutura de dados interna uma descrição de estado do ambiente (por exemplo: o AgenteComprador do SMA para o TAC-SCM, Capítulo 6, guarda as ofertas dos fornecedores para decidir de quem irá comprar os componentes), que é atualizada à medida que os episódios vão acontecendo. Esta nova arquitetura consiste em um refinamento do subsistema de tomada de decisão do agente puramente reativo [Weiss 1999]. A Figura 4 ilustra este refinamento.

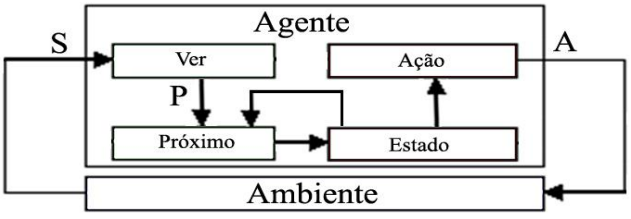

Figura 4 – Agente reativo baseado em conhecimento [Weiss 1999]

Comparando com o agente reativo simples ilustrado na Figura 2, o agente baseado em modelo possui um subsistema de processamento de informação a mais: a função ***Próximo***. Isto implica em realizar uma nova decomposição no subsistema de tomada de decisão do agente padrão. Em termos de funcionamento esta decomposição equivale à decomposição do passo **(3)** do ciclo de operação do agente puramente reativo descrito

na seção anterior. Mais especificamente, depois do subsistema de percepção ***Ver*** mapear um estado do ambiente em ***S*** em uma percepção em ***P***, em **(3.1)** um subsistema de atualização de estado interno, ***Próximo : IxP → I***, mapeia a percepção em ***P*** e o estado interno corrente em ***I*** = $\{i_1, ..., i_m\}$, em um novo estado interno; que, por sua vez, e em **(3.2)** é processado pelo subsistema de tomada de decisão, ***Ação: I → A***, para selecionar uma ação possível em ***A***.

Por exemplo, voltando ao mundo simplificado do aspirador descrito na Figura 3, considerando que na configuração inicial, em $t = 0$, o mundo está no *estado 1* e que o estado interno inicial do agente é "vazio" (não sabe a sala em que está, e nem os estados das salas *A* e *B*), o funcionamento desta nova arquitetura pode ser resumido em ciclos compostos de cinco passos principais: (0) o *agente sai de um estado interno inicial* $i^{(0)} = [\,];$ (1) observa o estado do ambiente $s^{(0)} = [sujo, sala\ A];$ (2) e gera uma percepção $p^{(0)} = Ver(s^{(0)}) = [S, A];$ (3.1) seu estado interno é então atualizado por meio da função próximo, tornando-se, $i^{(1)} = Próximo(i^{(0)},ver(s^{(0)})) = [A, S\ em\ A];$ (3.2) seleciona uma ação por meio de $a^{(0)} = Ação(próximo(i^{(0)},ver(s^{(0)}))) = Aspirar;$ e (4) a ação é então executada no ambiente. Em seguida, visando realizar o objetivo o agente inicia outro ciclo de operação, em $t = t+1$, ou seja, percebendo o mundo por meio de *Ver*, adaptando o estado interno por meio de *Próximo*, escolhendo uma nova ação por meio de *Ação* e executando a mesma no ambiente.

Vale ressaltar, conforme proposto para o agente puramente reativo, as regras *condição-ação* podem ser incorporadas no módulo de tomada de decisão, desta vez levando em consideração o estado interno mantido a respeito de seu ambiente, ao invés da informação que chega diretamente dos sensores do agente. Para problemas mais complexos que o mundo do aspirador simplificado, Russell e Norvig (2004) propõem que, além da descrição de estado interno corrente, a função *Próximo* incorpore informações a respeito do efeito das ações do agente em termos de mudanças proporcionadas nos estados do ambiente e da evolução dos estados do ambiente independentemente das ações do agente.

Um agente reativo baseado em conhecimento pode se sair melhor em uma gama maior de ambientes que um agente reativo simples, pois é capaz de manter uma representação interna de parte do ambiente que ele não consegue observar. Portanto,

normalmente agentes reativos baseados em modelos se saem melhor que agentes reativos simples em ambientes parcialmente observáveis.

Os agentes reativos são mais adequados a subproblemas que exigem respostas rápidas. Quando agem em conjunto, agentes reativos podem atingir bons resultados, como no caso das colônias de formigas [Dorigo e Stutzle 2004].

2.2.4. Agente baseado em objetivo

Muitas vezes apenas o estado atual do ambiente não é o bastante para executar a ação correta. Se um agente motorista estivesse em um entroncamento com capacidade de seguir em frente, virar à esquerda ou virar à direita, o objetivo para o motorista poderia ser o destino do cliente. Adicionalmente, o agente pode usar as informações existentes nas crenças para auxiliar a tomada de decisão, visando cumprir seu objetivo.

Algumas vezes a ação baseada em objetivos é direta, quando o objetivo é satisfeito com uma única ação. Porém, em situações mais complexas, uma sequência extensa de ações é necessária para atingir o objetivo.

Planejamento dedica-se a encontrar sequências de ações que alcançam os objetivos do agente [Russell e Norvig 2004]. Para Silva (2004), o plano é uma sequência de ações estabelecida previamente que leva o agente a atingir um objetivo.

A Figura 5 demonstra a estrutura interna de um agente baseado em objetivo.

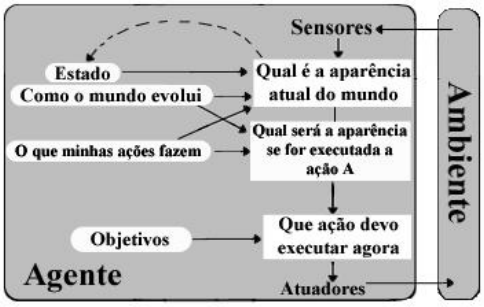

Figura 5 – Um agente baseado em Objetivo [Russell e Norvig 2004].

Deste modo o agente baseado em objetivo guiado por planejamento possuirá os seguintes elementos, conceituados segundo [Russell e Norvig 2004]:

- **Percepção:** Os agentes podem perceber seu ambiente através de sensores [Russell e Norvig 2004]. Com exceção dos agentes baseados em objetivo com plano [Silva 2004], a percepção do agente é importante para a decisão de que ação tomar em um determinado momento.

- **Função próximo:** Recebe o estado e a percepção atual e retorna um novo estado atualizado. A função próximo é introduzida para mapear as percepções e o estado interno atual para um novo estado interno, que será utilizado para selecionar a próxima ação.
- **Função de formulação de objetivo:** Recebe o estado e retorna o objetivo formulado.
    - Meta (ou objetivo) é um subconjunto de estados do mundo (são os estados que satisfazem a meta);
    - Por exemplo: Para um turista visitando o Brasil:

      Os estados do mundo são os pontos turísticos Brasileiros;

      A meta é um subconjunto dos pontos turísticos, aqueles que o turista tenciona visitar.
- **Função de formulação de problema:** Recebe o estado e o objetivo e retorna o problema.
    - Deve ser executada após a formulação da meta;
    - A saída da função de formulação do problema retorna uma representação para os quatro elementos:

      O **estado inicial** em que o agente começa;

      Uma descrição das ações possíveis disponíveis para o agente. Formulação mais comum: uso de um **Sucessor** (o sucessor retornará os possíveis estados seguintes a partir do estado atual);

      O **teste de objetivo** – determina se um dado estado é um estado objetivo;

      Custo de caminho – atribui um custo numérico a cada passo.
- **Planejamento:** Recebe o problema e utiliza busca e/ou lógica para encontrar uma sequência de ações para atingir um objetivo.
- **Ação:** Representada com suas pré-condições e pós-condições.

Enquanto planejam sobre a sequência de ações que levarão aos seus objetivos, é esperado que o ambiente não seja modificado. Caso isto ocorra, a sequência de ações

encontrada pode deixar de ser racional, pois foram planejadas em um estado do mundo que foi modificado após o inicio do planejamento. Observe também que há uma suposição de que as ações realmente produzirão os resultados desejados no ambiente. Portanto, um agente orientado a objetivos é adequado para ambientes parcialmente observáveis, estáticos e determinísticos.

2.2.5.Agente baseado em utilidade

De forma geral, os objetivos não são suficientes para gerar um comportamento ótimo por parte do agente, uma vez que ele deve apenas cumprir o objetivo independente do caminho seguido ser o melhor caminho possível ou não.

Em busca de melhorar o comportamento do agente, podemos utilizar uma função responsável por mapear um estado (ou conjunto de estados) possível com um grau de utilidade associado. Esta função é chamada *Função Utilidade*.

A *Função Utilidade* é importante em dois casos: a) quando objetivos contraditórios estão associados ao agente (velocidade e segurança para um agente motorista, por exemplo), a função utilidade especifica o compromisso de cada um, e b) quando existem vários objetivos que o agente deseja alcançar e nenhum deles pode ser atingido com certeza. A utilidade fornece um meio pelo qual a probabilidade de sucesso pode ser ponderada em relação à importância dos objetivos [Russell e Norvig 2004].

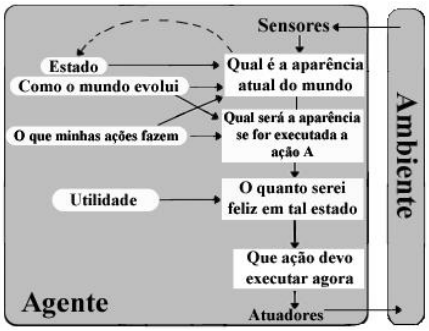

Figura 6 – Um agente baseado em modelo e orientado a utilidade [Russell e Norvig 2004]

O agente baseado em utilidade estende o conceito de agente baseado em objetivos, isto é, ele inclui os seguintes elementos: Percepção, Função próximo, Função de formulação de objetivo, Função de formulação de problema, Planejamento, Ações, e adiciona a Função utilidade. Esta função recebe um estado e retorna o grau de utilidade de tal estado de acordo com os objetivos atuais.

Um agente orientado a utilidade é mais eficaz em duas situações em que um agente baseado em objetivos pode falhar ou ficar sem saber o que fazer: 1) quando há incerteza no resultado das ações e 2) quando há mais de um objetivo a ser alcançado. No primeiro caso, um agente orientado a utilidade pode selecionar aquela ação que terá mais chances de produzir um estado desejado que seja mais promissor. No segundo caso, um agente orientado a objetivos pode selecionar um estado com maior utilidade dos estados desejados. Portanto, um agente orientado a utilidade é mais adequado para ambientes parcialmente observáveis e não determinísticos.

2.3. TAO.

A principal função do *framework* TAO é oferecer uma ontologia que englobe os fundamentos de engenharia de software baseada em agentes e objetos, para entender as abstrações e seus relacionamentos, a fim de oferecer suporte ao desenvolvimento de SMAs de larga escala [Silva et al. 2003]. A seguir os aspectos estáticos e dinâmicos do TAO são descritos.

2.3.1. Aspectos Estáticos do TAO

As entidades definidas pelo metamodelo do TAO são:

- **Objeto**: É um elemento passivo que possui um estado e um comportamento. Um objeto não pode modificar seu comportamento e não tem controle sobre ele, ou seja, um objeto não é autônomo e então faz tudo que for solicitado, e somente quando for solicitado por outra entidade.
- **Agente**: É um elemento autônomo, adaptativo e interativo, que possui os seguintes componentes mentais: crenças (tudo que o agente conhece), objetivos (estados futuros que o agente deseja alcançar), planos (sequências de ações que alcançam um objetivo) e ações.
- **Organização**: É responsável por agrupar agentes, objetos e sub-organizações. Uma organização possui objetivos, crenças (como agentes) e axiomas, e está situada e um ambiente. Os axiomas caracterizam as restrições globais da organização às quais os agentes e as sub-organizações devem obedecer. Uma organização também define os papéis que devem ser exercidos pelos agentes e pelas sub-organizações dentro dela, e os papéis que devem ser exercidos pelos objetos.

- **Papel de Objeto**: Orienta e restringe o comportamento de um objeto limitando as informações e o comportamento que outras entidades podem acessar. Um papel de objeto pode adicionar informação, comportamento e relacionamentos à instância do objeto que o executa.
- **Papel de Agente**: Orienta e restringe o comportamento de um agente descrevendo seus objetivos ao exercer o papel, definindo as ações que deve exercer e as ações que pode executar ao exercer o papel. Um papel de agente define deveres, direitos e protocolos. Um dever define uma ação que deve ser executada por um agente; um direito define uma ação que pode ser executada por um agente; e um protocolo define uma interação entre um papel de agente e outros elementos.
- **Ambiente**: É um elemento no qual residem agentes, objetos e organizações.

O metamodelo TAO define ainda os relacionamentos a seguir:

- **Habita**: Este relacionamento especifica que o ambiente conhece todos os elementos que residem nele e que cada elemento conhece seu ambiente. Um elemento não pode estar em dois ambientes ao mesmo tempo. Aplica-se a ambientes e agentes, ambientes e objetos, e ambientes e organizações.
- **Posse**: Especifica que uma entidade – o membro – é definida no escopo de outra entidade – o proprietário – e que um membro deve obedecer a um conjunto de restrições globais definidas pelo proprietário. Esse relacionamento aplica-se a classes de papel – os membros – e às classes de organização – os proprietários.
- **Exerce**: Especifica que um elemento que está exercendo um papel assume suas propriedades e relacionamentos. Todos os agentes e sub-organizações exercem pelo menos um papel na organização. Objetos podem executar papéis de objetos.
- **Especialização/Generalização**: Define que a sub-entidade que especializa a super-entidade herda suas propriedades e relacionamentos. As propriedades herdadas também podem ser redefinidas pela sub-entidade. Ademais, a sub-entidade também pode definir novas propriedades e novos relacionamentos. O relacionamento de *Especialização* pode ser usado entre objetos, agentes, organizações, papel de objeto e papel de agente.

- **Controle**: O relacionamento controle define que a entidade controlada deve fazer tudo que a entidade controladora pedir. O relacionamento Controle pode ser usado entre dois papéis de agente. O relacionamento controle não é definido entre papéis de objeto porque os objetos sempre fazem tudo que outra entidade solicita.
- **Dependência**: Especifica que uma entidade (o cliente) pode ser definida dependendo de outra (o fornecedor) para realizar seu trabalho. Um papel de agente pode depender de outro papel de agente e um papel de objeto pode depender de outro papel de objeto.
- **Associação**: Define que um elemento interage com outro, indicando que ambos se conhecem. Pode ser usado entre: (i) papéis (de objeto e de agente), (ii) ambientes, (iii) agentes e objetos, (iv) organizações e objetos, e (v) papéis e objetos.
- **Agregação**: Define que um elemento é parte de um agregador. O agregador pode usar as funcionalidades existentes em suas partes. Este relacionamento pode ser aplicado entre papéis de objeto, papéis de agente, objetos, agentes e organizações.

Na Figura 7 são ilustrados as entidades e os relacionamentos definidos no TAO.

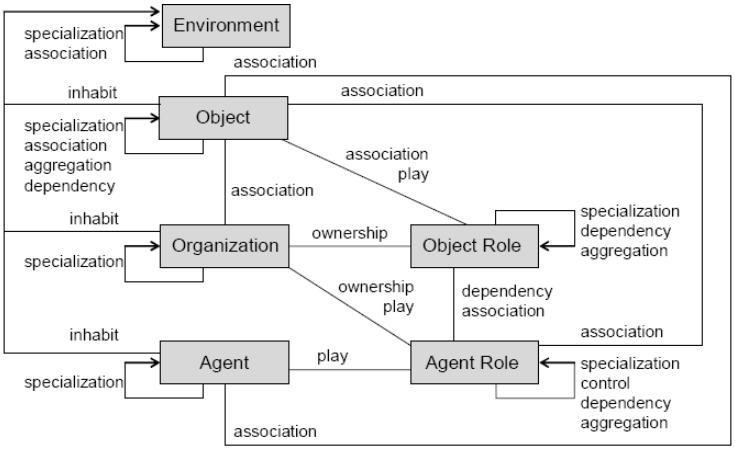

Figura 7 - Os relacionamentos e as entidades do TAO [Silva 2004].

2.3.2. Aspectos Dinâmicos do TAO

Os aspectos dinâmicos de uma entidade definem sua execução em uma aplicação. Eles podem ser definidos como processos dinâmicos primitivos e processos dinâmicos de alto-nível. As interações elementares ou básicas entre as entidades de um SMA são descritas como processos dinâmicos primitivos. Os processos de criação e destruição de elementos são caracterizados como processos primitivos.

Os processos dinâmicos de alto nível descrevem os padrões de comportamento derivados das características dos relacionamentos Habita, Posse e Exerce entre as entidades de SMAs. Eles abrangem processos para um agente entrando ou saindo da organização, uma organização entrando ou saindo de outra organização, e um agente ou organização entrando ou saindo de um ambiente.

2.4. MAS-ML

MAS-ML é uma linguagem de modelagem que estende a UML para permitir a modelagem de SMAs. A extensão proposta à UML está baseada nos conceitos definidos pelo *framework* conceitual TAO.

2.4.1. Aspectos estáticos de MAS-ML

A definição de objeto em MAS-ML reutiliza a definição já existente em UML descrita pela metaclasse Class. Além disso, a notação utilizada para representar classes (instancias da metaclasse Class) em MAS-ML é exatamente a mesma utilizada em UML. Entretanto, foi necessário criar novas metaclasses para definir os outros conceitos do TAO que não estavam presentes em UML. MAS-ML inclui as metaclasses AgentClass, OrganizationClass, EnvironmentClass, ObjectRoleClass e AgentRoleClass [Silva 2004]. Assim como no caso de objetos, a notação utilizada nos diagramas estáticos de MAS-ML para representar elementos instâncias destas metaclasses também define três compartimentos separados por linhas horizontais.

O compartimento superior guarda o nome da entidade que deve ser único em seu espaço de nomes. O compartimento intermediário agrupa as características estruturais, e o compartimento inferior agrupa as características comportamentais. Cada elemento tem uma notação diferente capaz de caracterizá-lo. Na Figura 8 é apresentada a notação utilizada em diagramas estáticos de MAS-ML para representar agentes que são instâncias da metaclasse AgentClass.

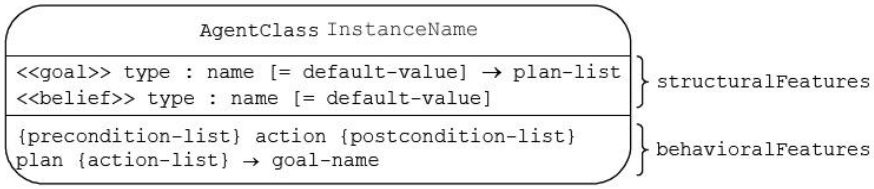

Figura 8 – Representação de uma instância da metaclasse AgentClass (adaptado) [Silva 2004].

Além das metaclasses para a definição das novas entidades, MAS-ML inclui as metaclasses que definem os novos relacionamentos identificados no TAO: Habita, Posse, Exerce e Controle.

UML possui um diagrama capaz de modelar a parte estática de um sistema: o diagrama de classes. Ele é um gráfico de entidades conectadas por seus relacionamentos estáticos. Já MAS-ML utiliza três diagramas estáticos para fazer esta modelagem: diagrama de classes, de organizações e de papéis.

MAS-ML estende o diagrama de classes de UML para incluir a modelagem de entidades de um SMA, utilizando os relacionamentos já definidos em UML e o relacionamento *habita*. O diagrama de organização modela as organizações do sistema, identificando seus ambientes e regras. Este diagrama mostra os relacionamentos Posse, Exerce e Habita. Já o diagrama de papéis é responsável por esclarecer os relacionamentos entre os papéis de agente e os papéis de objeto. Este diagrama mostra os relacionamentos Controle, Dependência, Associação, Agregação e Especialização.

2.4.2. Aspectos dinâmicos de MAS-ML

Os aspectos dinâmicos de MAS-ML são representados através de uma extensão dos diagramas de sequência e de atividade da UML para representar os aspectos dinâmicos de SMAs, ou seja, para representar as interações entre as instâncias do SMA e as ações de cada instância.

2.4.2.1. Diagrama de Sequência

O diagrama de sequência modela o comportamento de um sistema orientado à objetos, compreendendo a modelagem de *atores*, objetos, criação e destruição dos mesmos e troca de mensagens. A extensão do diagrama de sequência para representar os aspectos dinâmicos de SMAs, ou seja, para representar as interações entre as instâncias do SMA e as intra-ações (i.e., ações executadas internamente pelas entidades) definidas por cada instância envolve dois aspectos:

Primeiro, novos *pathnames*[4] e ícones foram definidos para as instâncias dos SMAs (agentes, organizações e ambiente) em diagramas de sequência.

[4] Na UML, estrutura de nomes para identificar uma entidade em um diagrama de sequência.

Segundo, para representar as interações entre agentes, organizações, ambientes e objetos, a definição do conceito *mensagem* usado em UML precisa ser estendida para representar entidades que estão enviando e recebendo mensagens, mas não através de chamadas a métodos de outras entidades. Agentes interagem enviando mensagens e não chamando métodos.

Adicionalmente, alguns estereótipos foram criados para representar a criação e destruição de instâncias de SMAs e para representar a interação entre agentes, organizações, objetos e seus papéis. Alguns estereótipos já existentes e associados a mensagens foram redefinidos.

O diagrama de sequência ainda foi estendido para representar a execução de planos e ações enquanto modela as intra-ações relacionadas a agentes, organização e ambientes.

2.4.2.2. Diagrama de Atividades

Um diagrama de atividade modela um fluxo de execução através de uma sequência de unidades subordinadas chamadas de ação. MAS-ML estende o diagrama de atividade da UML 2.1 para modelar planos e ações, e os demais conceitos definidos na linguagem [Silva, Choren e Lucena 2007]. Essa extensão é representada através dos estereótipos para representar agentes, ambientes, organizações, papéis, crenças, planos, ações, objetivos, mensagens, direitos e obrigações.

Mais detalhes sobre representações das entidades, relacionamentos e diagramas de MAS-ML são encontrados em [Silva, Choren e Lucena 2005] e [Silva, Choren e Lucena 2007].

3. Trabalhos Relacionados

Várias linguagens de modelagem e *frameworks* conceituais têm sido propostos em decorrência da necessidade de modelar SMAs adequadamente. Boa parte das propostas se inspira ou estende conceitos para o desenvolvimento orientado a objetos, enquanto outras utilizam conceitos de modelagem da Engenharia do conhecimento. A análise dos trabalhos relacionados a esta dissertação cobre *frameworks* conceituais e linguagens de modelagem para SMAs.

3.1. *Framework* Conceitual d'Inverno e Luck

O *framework* proposto por d'Inverno e Luck (2001) define uma hierarquia constituída de quatro camadas compreendendo entidades, objetos, agentes e agentes autônomos.

Os objetos estendem entidades ao adicionar ações primitivas que definem suas habilidades. Como os agentes estendem objetos, eles têm atributos e ações, mas também definem objetivos. Agentes autônomos estendem agentes definindo motivações. Um ambiente é definido pelo *framework* como um conjunto de entidades, estas entidades podem ser Objetos, Agentes ou Agentes Autônomos.

Alguns pontos fracos deste *framework* conceitual são:

- A entidade ambiente possui apenas características estruturais, não apresentando operações;
- O *framework* conceitual não define nenhum aspecto dinâmico associado às entidades propostas;
- Não oferece elementos para definir corretamente as diferentes arquiteturas internas de agente.

3.2. *Framework* Conceitual Yu e Schmid

Em Yu e Schimid (1999) ***apud*** Silva (2004), os autores propõem um *framework* conceitual para a definição de SMA com base em papéis e orientado a agentes. Eles se concentram na definição de papéis descrevendo suas propriedades e relacionamentos. Os papéis são definidos em termos de objetivos, qualificações, obrigações (ou deveres), permissões (ou direitos) e protocolos. Os agentes são apresentados como uma entidade que exerce papéis dentro de uma organização. Os papéis são atribuídos a agentes com base na avaliação da qualificação e das habilidades. Os agentes são selecionados para exercer os papéis de acordo com as políticas organizacionais e suas habilidades.

Alguns pontos fracos dessa proposta são:

- Apesar de os agentes serem definidos como uma entidade que exerce papéis, o *framework* conceitual não define as propriedades de agentes e os relacionamentos entre agentes e papéis.

- Esse *framework* não define a criação e a destruição de agentes e, portanto, não descreve a interdependência entre agentes e papéis;
- Apesar de os autores afirmarem que papéis são exercidos em organizações, a proposta não define as propriedades de organização e o relacionamento entre elas e papéis.
- Tampouco define a entidade ambiente em que estão os agentes e as organizações.

3.3. KAoS

Segundo Dardenne, Lamsweerde e Fickas (1993) ***apud*** Silva (2004), KAoS é um *framework* conceitual que define abstrações, como entidades, relacionamentos e agente, como extensões de objeto. Uma entidade é um objeto autônomo independente de outros objetos. Um relacionamento é um objeto subordinado. Um agente é um objeto que tem escolha e comportamento, e define crenças, objetivos e ações. Alguns pontos fracos do KAoS são:

- Ele não considera organizações, papéis e ambientes, que são abstrações importantes;
- O KAoS não explica de forma satisfatória a distinção entre uma entidade e um agente;
- Ele não descreve as características de um objeto ou explica como ele é estendido por outras abstrações;
- Também não descreve os relacionamentos entre as abstrações definidas e por que um relacionamento deve ser descrito como um objeto;
- O KAoS não descreve qual é a relação das propriedades de um agente, ou seja, não descreve como um agente alcança um objetivo com base em suas ações e crenças;
- Ele não define nenhum aspecto dinâmico associado às entidades descritas.

3.4. AUML

AUML foi criada a partir de um projeto desenvolvido por Odell, Parunnak e Bauer (2000), visando adaptar a *Unified Modeling Language* (UML) às características inerentes ao paradigma de Orientação a Agentes [Padilha e Jácome 2002].

O objetivo de 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.

Em 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 sequência e comunicação); e, finalmente, a terceira camada detalhando o comportamento interno de cada agente (diagramas de atividades e estados) [Dário 2005].

As entidades definidas em AUML são: agentes, papéis e organizações (ou grupos). Em Huget (2002), os relacionamentos *associação, generalização*, *agregação* e *dependência* foram definidos para serem aplicados a classes do agente. Além disso, o relacionamento chamado *ordem* foi descrito para indicar que um agente pode pedir a outro agente que execute algumas tarefas.

Alguns pontos fracos de AUML são:

- AUML não oferece suporte à modelagem de arquiteturas internas dos agentes inteligentes. Não oferece elementos para representar corretamente percepções, função próximo e planejamento, por exemplo.
- AUML não define ambiente como uma abstração, portanto não é possível modelar um agente que esteja se movendo de um ambiente para outro usando o diagrama de sequência proposto;
- A criação e a destruição de agentes e papéis não são descritas. Portanto, não está claro se a criação de um papel depende da criação de um agente para exercer o papel.

3.5. AORML

AORML [Wagner 2003] é uma linguagem de modelagem para SMAs baseada no metamodelo AOR. As entidades definidas em AOR são agente, evento, ação, reinvidicação, comprometimento e objeto comum. As organizações são representadas através do agente do tipo institucional.

Os relacionamentos definidos por AOR são: *does*, *perceives*, *sends*, *receives*, *hasClaim* e *hasCommitment*. Os relacionamentos de associação, agregação/composição e generalização definidos pela UML também podem ser utilizados.

Os diagramas de AORML dividem-se em diagrama externo e diagrama interno. No modelo externo temos os seguintes diagramas: Diagrama de agentes, Diagrama de interação de quadros, Diagrama de interação de sequência e Diagrama de interação de padrões. Enquanto no modelo interno temos o Diagrama de Quadros da Reação e o Diagrama de Sequência de Reação.

A AORML também descreve o processo de transformar uma modelagem do ponto de vista externo para modelos do ponto de vista interno chamada de internalização.

Alguns pontos fracos de AORML são:

- AORML não oferece suporte à modelagem de arquiteturas internas de agente. Portanto não é possível diferenciar agentes com arquiteturas reativas e proativas em AORML.
- Além disto, as duas linguagens citadas anteriormente não definem ambiente como uma abstração, portanto não é possível modelar a migração de um agente de um ambiente para outro. Representar a capacidade de migração de agentes é interessante na modelagem de agentes móveis [Silva P. S. e Mendes 2003], aplicados em redes de computadores, por exemplo.

3.6. AML

A linguagem de modelagem para agentes (AML) [Cervenka et al. 2005] é uma linguagem de modelagem visual semi-formal para especificação, modelagem e documentação de sistemas que incorporam conceitos da teoria de sistemas multi-agentes.

Esta linguagem de modelagem é baseada em um metamodelo e estende a UML. As entidades definidas por AML são agentes, recursos e ambientes. Devido à necessidade de modelar aspectos sociais, AML utiliza conceitos de unidades organizacionais, relacionamento social, papéis e propriedades de papéis.

Algumas desvantagens de AML são descritas a seguir:

- Não é capaz de modelar os diferentes tipos de agentes inteligentes, pois nem todo agente necessita de crenças, objetivos e de planos como é definido por esta linguagem. Por outro lado, outras arquiteturas de agentes precisam de componentes tais como função próximo e percepções, os quais não são modelados pela linguagem;
- Nem mesmo utilizando mecanismos de extensão fornecidos pela UML ou simplesmente mudando alguns diagramas, a linguagem consegue dar apoio suficiente à complexidade e dinâmica de SMAs;
- Os aspectos semânticos de comunicação são modelados como especializações de elementos existentes na UML, tal como invocação de métodos, uma vez que não foram definidas abstrações especificas para agentes [Silva, Choren e Lucena 2007]. Esta restrição pode dificultar a modelagem das diferentes formas de interação previstas para agentes.
- Não introduz explicitamente novos conceitos no metamodelo UML.

3.7. ANote

A finalidade básica do ANote [Choren e Lucena 2004] é fornecer aos usuários uma expressiva linguagem de modelagem visual para desenvolver modelos baseados nos conceitos de agentes. Para esta linguagem, o agente é a unidade básica da estrutura que encapsula o comportamento com as decisões e os planos.

O Anote utiliza os conceitos de objetivo, agente, recurso, cenário, ação, mensagem e organização e fornece um conjunto de modelos para a especificação de SMA, conhecidos como visões. Anote possui sete visões: visão de objetivo, visão de agentes, visão de ontologia, visão de cenários, visão de planejamento (ação), visão de interação (mensagem) e visão de organização.

Alguns pontos fracos de Anote são:

- Não oferece elementos para modelar as arquiteturas internas dos agentes inteligentes;
- Não oferece suporte a objetos convencionais;
- Não identifica papéis.

3.8. MAS-ML

Dentre as linguagens de modelagem citadas, destacamos MAS-ML (*Multi-Agent System Modeling Language* - Linguagem de Modelagem para Sistemas Multi-Agente) por modelar agentes baseados em objetivo guiados por um plano pré-estabelecido. Além disto, MAS-ML também agrupa as seguintes características que a diferencia das demais linguagens de modelagem orientadas a agentes: (i) modela agentes baseados em objetivo com plano, (ii) possui uma ontologia associada, (iii) oferece suporte a objetos convencionais, (iv) identifica papéis, (v) modela adequadamente ambientes e a interação entre agente e ambiente, e (vi) introduz explicitamente novos conceitos ao metamodelo UML relacionados a entidades orientadas a agentes, abstração e decomposição do comportamento, aspectos sociais e mentais e interações comunicativas.

MAS-ML é uma linguagem de modelagem que realiza uma extensão conservativa da UML para permitir a modelagem de SMAs baseando-se no *framework* conceitual TAO (*Taming Agents and Objects*) [Silva et al. 2003], [Silva, Choren e Lucena 2004], [Silva, Choren e Lucena 2005] e [Silva, Choren e Lucena 2007].

MAS-ML foi originalmente projetada para dar suporte à modelagem de agentes pró-ativos orientados a objetivos guiados por planos. Porém, nem todos os SMAs necessitam ou possibilitam que seus agentes sejam pró-ativos. Conceber agentes pró-ativos baseados em objetivo e guiados por planos em ambientes estocásticos e parcialmente observáveis, por exemplo, pode ser uma tarefa bastante complexa. Além do mais, em ambientes dinâmicos, cujas pré-condições podem mudar enquanto o agente estiver deliberando, o comportamento puramente pró-ativo baseado em objetivo guiado por plano pode produzir efeitos indesejados [Weiss 1999].

Neste contexto surge a necessidade da utilização de agentes reativos simples, reativos baseados em conhecimento, pró-ativos baseados em objetivo guiados por

planejamento e pró-ativos baseados em utilidade, os quais não podem ser modelados através de MAS-ML na sua versão atual.

Sendo assim, é proposta uma extensão à MAS-ML para a modelagem de agentes reativos simples, reativos baseados em conhecimento, pró-ativos baseados em objetivo e pró-ativos baseados em utilidade. Visando esta evolução, o *framework* TAO também precisa ser estendido de forma a contemplar os elementos necessários para a definição das arquiteturas de agentes que serão adicionados à MAS-ML.

4. Estendendo o *Framework* TAO e a linguagem MAS-ML

Neste capítulo é apresentada uma extensão ao TAO e a MAS-ML de modo que agente reativo simples, reativo baseado em conhecimento, pró-ativo baseado em objetivo e pró-ativo baseado em utilidade possam ser modelados corretamente.

Inicialmente, a definição do conceito de agente e de papel de agente no *framework* conceitual TAO foi estendida. Em relação a MAS-ML, a nível de metamodelo, duas metaclasses e seis estereótipos foram criados, assim como uma notação específica para cada elemento a ser utilizado pelo agente nos diagramas estáticos. Os diagramas de sequência e de atividades também foram alterados para representar agentes reativos, agentes baseados em objetivo e agentes baseados em utilidade. A representação do papel de agente também sofreu alteração em relação aos agentes reativos. Os termos MAS-ML após a extensão e MAS-ML 2.0 serão utilizados de maneira intercambiável.

4.1. Novas características no TAO

Segundo Silva (2004), o TAO descreve um agente como um elemento autônomo, adaptativo e interativo, onde as características estruturais de um agente são objetivo e crença e suas características comportamentais são ação e plano.

Dependendo da arquitetura interna utilizada, algumas destas características podem não ser modeladas e ficar implícitas no comportamento do agente. Neste contexto, uma nova definição para agente pode ser formulada da seguinte forma:

Um agente é um elemento autônomo, adaptativo e interativo. A característica comportamental básica definida por um agente é a ação. Aspectos estruturais, mentais e/ou comportamentais são adicionados conforme Quadro 1:

Quadro 1 – Novas características de agente no TAO

	Características Estruturais / Mentais	Características Comportamentais
Agente Reativo Simples	-	Percepção e Ação (guiada por Regras Condição-Ação)
Agente Reativo baseado em Conhecimento	Crença	Percepção, Função próximo e Ação (guiada por Regras Condição-Ação)
Agente baseado em objetivo com plano	Objetivo e Crença	Plano e Ação (baseados na seleção do plano de acordo com o objetivo)
Agente baseado em objetivo com planejamento	Objetivo e Crença	Percepção, Função próximo, Função de formulação de objetivo, Função de formulação do problema, Planejamento e Ação
Agente baseado em Utilidade	Objetivo e Crença	Percepção, Função próximo, Função de formulação de objetivo, Função de formulação do problema, Planejamento, Função Utilidade e Ação

O conceito inicial do TAO para agente foi mantido através do item Agente baseado em objetivo com plano.

Ainda é necessário alterar o conceito de papel de agente no que diz respeito aos agentes reativos. As características estruturais do papel de agente são definidas no TAO como sendo compostas por suas crenças e objetivos [Silva 2004]. Podemos adicionar a este conceito o fato que no caso do agente reativo simples as crenças e os objetivos não existirão e no caso dos agentes reativos baseados em conhecimento, apenas as crenças existirão como característica estrutural.

4.2. Novas características em MAS-ML

O padrão UML (2009) incorpora os seguintes mecanismos de extensão: *tagged values*, estereótipos (*stereotypes*) e restrições (*constraints*). Adicionalmente, o metamodelo pode ser estendido através da adaptação de metaclasses existentes ou da definição de novas. MAS-ML realiza uma extensão conservativa da UML, fazendo uso de seus mecanismos de extensão. Seguindo a mesma estratégia, as novas características em MAS-ML foram incorporadas através de estereótipos e novas metaclasses.

As novas características definidas para um agente são: percepção, regras condição-ação, função próximo, função de formulação de objetivo, função de formulação do problema, planejamento e função utilidade, todas são características comportamentais. A seguir conceituaremos as novas características e apresentaremos nossa extensão.

Segundo Russell e Norvig (2004), o comportamento dos agentes inicia através de uma percepção. O intuito da percepção é apenas coletar informações do ambiente e / ou de outros agentes, sem realizar nenhuma modificação no ambiente, em outro agente ou

em si próprio. Agentes possuem uma definição do que pode ser percebido no ambiente. Estas propriedades fazem com que a percepção não possa ser representada com as metaclasses já existentes, tornando necessária a criação da metaclasse AgentPerceptionFunction. A metaclasse AgentPerceptionFunction tem uma restrição que é utilizada para definir as informações que podem ser percebidas pelos sensores do agente.

A tarefa de planejamento resulta em uma sequência ordenada de ações a serem seguidas para atingir um objetivo [Russell e Norvig 2004]. Adicionalmente as seguintes propriedades são observadas: (i) diferentemente de um plano (representado no metamodelo de MAS-ML pela metaclasse AgentPlan), a sequência de ações é criada em tempo de execução; e (ii) diferentemente de uma ação (representada no metamodelo de MAS-ML pela metaclasse AgentAction), o planejamento possui um objetivo associado. Levando em conta estas propriedades, uma nova metaclasse AgentPlanningStrategy foi criada para representar o planejamento. Um relacionamento de associação foi definido estabelecendo que uma ação de planejamento pode criar planos. Vale salientar que os planos são gerados em tempo de execução pelo planejamento, portanto não são representados graficamente na modelagem do agente que utiliza planejamento para guiar suas ações.

As metaclasses AgentPerceptionFunction e AgentPlanning estendem a metaclasse BehavioralFeature. A Figura 9 demonstra esta extensão.

Como poderá ser constatado em Gonçalves et al. (2010a), as percepções do agente também estão representadas no ambiente, uma vez que ele representa os elementos que o agente pode perceber e o alcance da percepção dos sensores do agente (parcial ou totalmente, por exemplo). Uma vez que o ambiente influencia a percepção dos agentes que nele se encontram, uma associação foi estabelecida entre as metaclasses AgentPerceptionFunction e EnvironmentClass. Essa associação não tem impacto na representação do ambiente nos diagramas de MAS-ML.

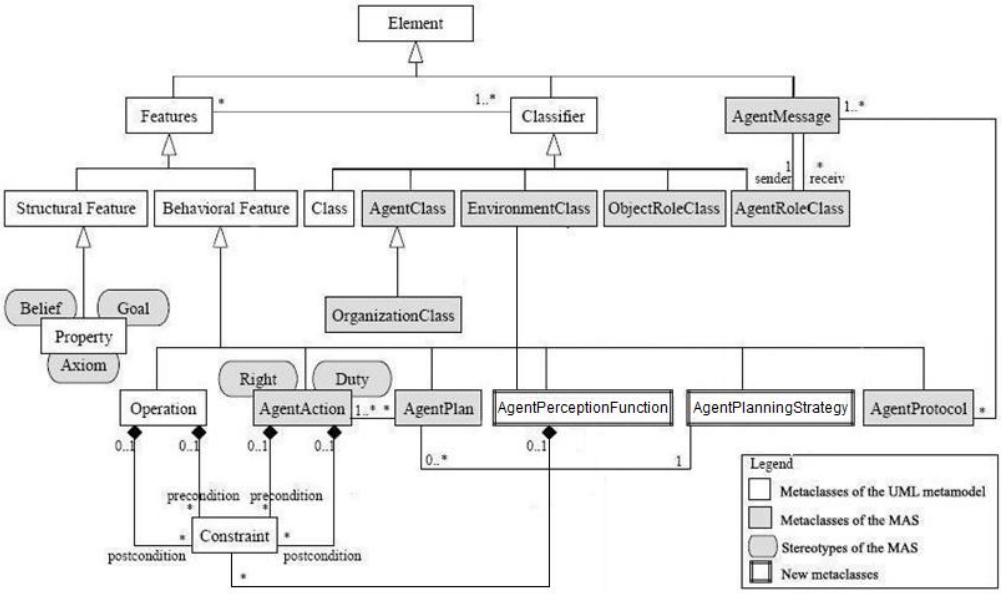

Figura 9 – Extensões propostas no metamodelo MAS-ML.

A função próximo, função de formulação de objetivo, função de formulação de problema e função utilidade (definidas no Capítulo 2) são ações especiais do agente utilizadas dependendo da arquitetura interna específica. Os seguintes estereótipos associados à metaclasse AgentAction foram respectivamente criados para representar estes elementos: <<next-function>>, <<formulate-goal-function>>, <<formulate-problem-function>> e <<utility-function>>. A Figura 10 ilustra os novos estereótipos associados à metaclasse AgentAction.

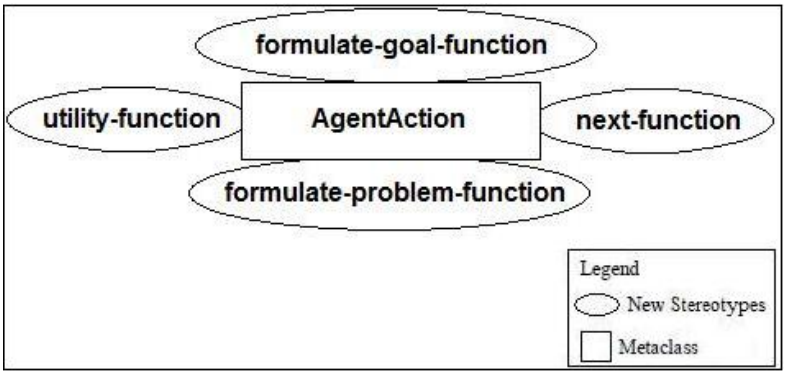

Figura 10 – Novos estereótipos da metaclasse AgentAction.

Finalmente, as regras condição-ação, que guiam as ações dos agentes reativos, podem ser representadas através da própria representação da ação do agente proposta por Silva (2004), a qual pode ter uma pré-condição associada. Deste modo temos uma condição associada a uma ação, que é o princípio das regras condição-ação.

Com a mudança na notação, os diagramas de classes e de organização foram consequentemente adaptados para dar suporte a esta nova representação de uma classe de agente.

Além da mudança nos diagramas estáticos foi necessário alterar os diagramas dinâmicos para representar a percepção, função próximo, função de formulação de problema, função de formulação de objetivo, planejamento, função utilidade e as ações baseadas em regras condição-ação.

Adicionalmente, a representação da entidade AgentRoleClass também necessita de alterações em relação aos agentes reativos.

4.3. Representação dos novos elementos nos diagramas estáticos

A representação nos diagramas estáticos dos elementos adicionados em MAS-ML 2.0 é apresentada nesta seção. Todos estes elementos estão relacionados com a representação do agente.

4.3.1. Representação das percepções do agente

Os agentes podem perceber seu ambiente através de sensores [Russell e Norvig 2004]. Com exceção dos agentes baseados em objetivo com plano [Silva 2004], a percepção do agente é importante para a decisão de qual ação tomar em um determinado momento. Neste contexto surge a necessidade de tornar explícito o elemento de percepção do agente, dada a importância para as arquiteturas reativa simples, reativa baseada em conhecimento, baseada em objetivo com planejamento e baseada em utilidade.

As percepções do agente são representadas através do estereótipo <<perceives>>, o qual está associado à metaclasse AgentPerceives. Após o estereótipo, o nome dado à percepção do agente é representado e, em seguida, uma lista de percepções deve ser representada como uma restrição. A percepção é ilustrada a seguir:

```
<<perceives>> perceivesName {p1,p2, p3,...}
```

Um exemplo de percepção para o agente aspirador de pó (Seção 6.2) seria *háSugeiraNaSala*. Para o agente Caçador do Mundo Wumpus (Seção 6.1), *brisa e brilho* são exemplos de elementos que o agente pode perceber.

4.3.2. Função próximo, Função de formulação de objetivo, Função de Formulação de problema e Função Utilidade.

Considerando que a função próximo, função de formulação de objetivo, função de formulação de problema e função utilidade são estereótipos de action, podem ser descritas como uma ação. Estas funções são representadas por suas pré-condições, seguidas pelo estereótipo <<next-function>>, <<formulate-goal-function>>, <<formulate-problem-function>> ou <<utility-function>>, respectivamente, sua assinatura (nome) e uma lista de pós-condições. A seguir, a sintaxe das funções é apresentada:

```
{ } <<next-function>> name { }
```

```
{ } <<formulate-goal-function>> name { }
```

```
{ } <<formulate-problem-function>> name { }
```

```
{ } <<utility-function>> name { }
```

4.3.3. Planejamento

A atividade de planejamento, na prática, pode ser implementada de diversas formas, através de algoritmos tais como busca cega, busca para frente, etc. No entanto, este detalhamento na hora da modelagem poderia tornar a linguagem de difícil entendimento e aplicabilidade. Além disto, MAS-ML 2.0 se tornaria obsoleta, no momento que uma nova maneira de realizar busca fosse proposta, o que poderia acontecer em um curto espaço de tempo. Por estes motivos, optou-se por representar o planejamento de maneira genérica tornando a modelagem desta propriedade mais abstrata.

Desta forma, o planejamento do agente é representado na estrutura do agente através do estereótipo <<planning>> seguido do nome do planejamento e associado a um objetivo. A seguir a representação do planejamento do agente:

```
<<planning>> planningName → goal-name
```

O planejamento é realizado com base nas ações disponíveis na lista de ações, no intuito de criar uma sequência de ações (plano). Observe que um plano é gerado somente em tempo de execução, portanto não há como representar o(s) plano(s) gerado(s) pelo planejamento em tempo de modelagem.

No caso em que o projetista deseje que o desenvolvedor implemente um tipo específico de planejamento no agente, pode ser utilizada uma nota textual como será apresentado na Seção 4.3.4.

4.3.4. Representações de AgentClass

Uma AgentClass continua sendo representada por um retângulo de bordas arredondadas com três compartimentos. O compartimento superior contém o nome da classe do agente que deve ser único em seu espaço de nome. O conteúdo do compartimento intermediário e inferior depende da arquitetura interna do agente.

A representação do agente proposto inicialmente por MAS-ML, que utiliza um plano pré-estabelecido foi mantida, onde seu compartimento intermediário contém a lista de objetivos e as crenças e o compartimento inferior contém a lista de ações e planos [Silva 2004]. Na Figura 8 é ilustrada a representação do AgentClass como proposto originalmente por MAS-ML.

4.3.4.1. Estrutura do agente reativo simples

O agente reativo simples não possui nenhum componente específico no compartimento intermediário, pois não tem seu(s) objetivo(s) explícito(s) em tempo de modelagem e não possui uma base de conhecimento. O compartimento inferior onde são representadas as propriedades comportamentais contém as percepções e ações guiadas por regras condição-ação, não por um plano específico [Gonçalves et al. 2009]. Na Figura 11 temos a representação gráfica do AgentClass para um agente reativo simples.

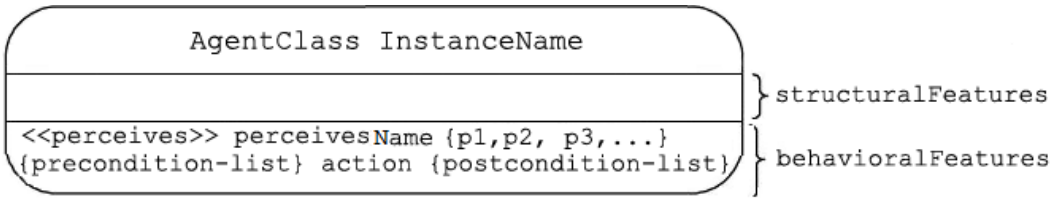

Figura 11 – Representação da AgentClass para um agente reativo simples.

4.3.4.2. Estrutura do agente reativo baseado em conhecimento

O agente reativo baseado em conhecimento possui suas crenças, as quais são representadas no compartimento intermediário, onde são representadas as propriedades estruturais; No compartimento inferior as percepções, as ações guiadas por regras-condição-ação e a função próximo são apresentadas [Gonçalves et al. 2009]. Na Figura

12 temos a representação gráfica do AgentClass para um agente reativo baseado em conhecimento.

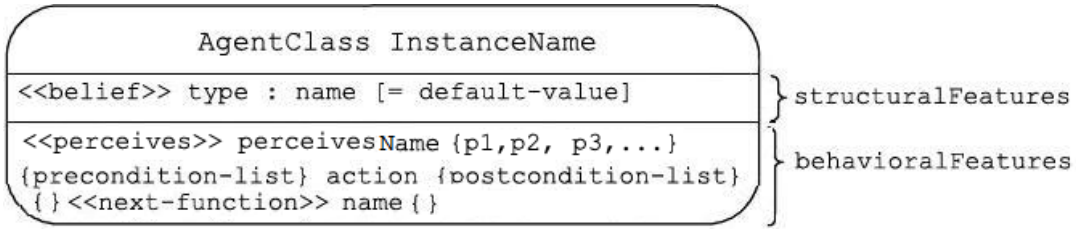

Figura 12 – Representação da AgentClass para um agente reativo baseado em conhecimento.

4.3.4.3. Estrutura do agente baseado em objetivo

O agente pró-ativo baseado em objetivo que utiliza planejamento, da mesma forma que a representação original do agente MAS-ML, representa a lista de objetivos e as crenças no compartimento intermediário. No caso das propriedades comportamentais do agente, representadas no compartimento inferior, foram incluídas percepções, função próximo, função de formulação do objetivo, função de formulação do problema, uma lista de ações e o planejamento [Gonçalves et al. 2010b]. Na Figura 13 é ilustrada a representação gráfica do AgentClass para um agente pró-ativo baseado em objetivo que utiliza planejamento.

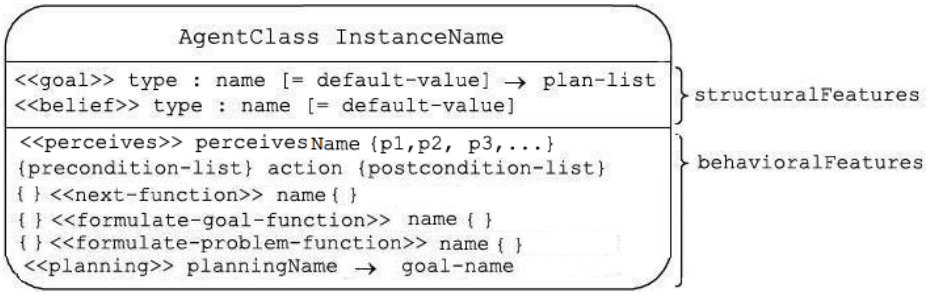

Figura 13 – Representação da AgentClass para um agente baseado em objetivo utilizando planejamento.

Segundo Russell e Norvig (2004), busca e lógica de primeira ordem são métodos que podem ser utilizados para realizar o planejamento. Uma nota textual informando o critério a ser utilizado pelo agente pode ser adicionada para identificar a estratégia que será utilizada pelo agente para fazer o planejamento. O projetista pode, ainda, omitir a nota se quiser permitir que o desenvolvedor decida o tipo de planejamento que irá codificar. A Figura 14 ilustra um agente com uma nota textual indicando que busca em profundidade deverá ser implementada como forma de planejamento.

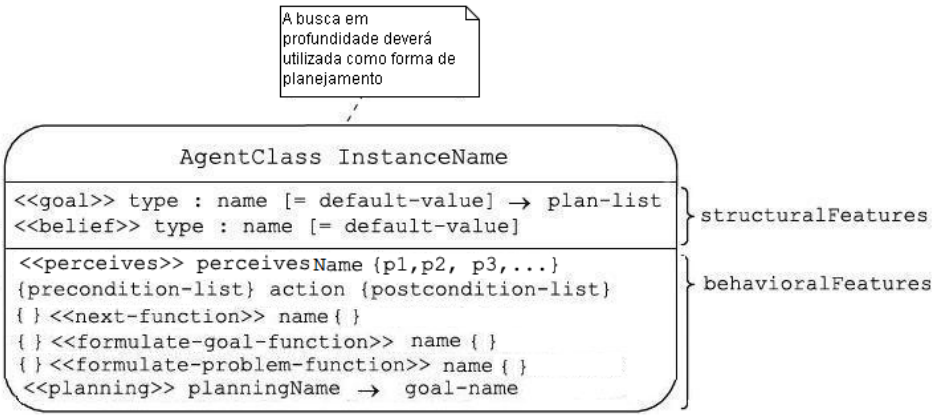

Figura 14 – Representação da AgentClass para um agente baseado em objetivo utilizando planejamento.

4.3.4.4. Estrutura do agente baseado em utilidade

O agente baseado em utilidade possui a mesma estrutura do agente baseado em objetivo, porém o elemento *função utilidade* é adicionado à estrutura, no intuito de guiar o comportamento do agente.

Desta forma, o agente pró-ativo baseado em utilidade continua representando a lista de objetivos e as crenças no compartimento intermediário. O compartimento inferior contém as percepções, função próximo, função de formulação do objetivo e função de formulação do problema, uma lista de ações, planejamento e a função utilidade [Gonçalves et al. 2010a]. Na Figura 15 é ilustrada a representação gráfica do AgentClass para um agente pró-ativo baseado em utilidade.

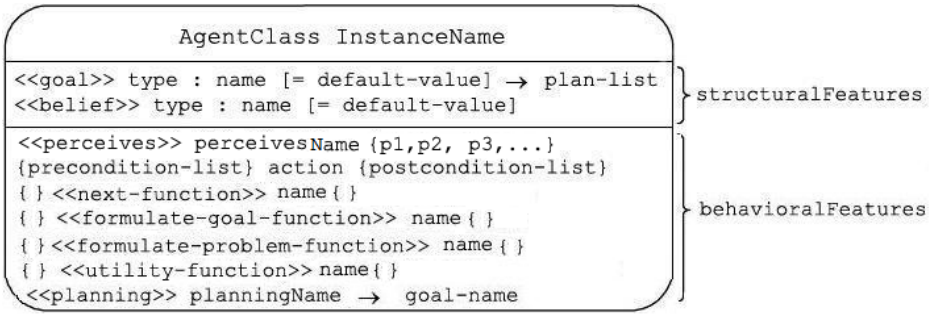

Figura 15 – Representação da AgentClass para um agente baseado em utilidade.

No caso do agente baseado em utilidade, seu planejamento pode estar ligado a mais de um objetivo a ser atingido, ou a situações em que nenhum objetivo pode ser atingido. Neste caso, os objetivos podem ser conflitantes ou, eventualmente, vários estados possíveis podem satisfazer os objetivos do agente. Neste contexto, a função utilidade é inserida na estrutura do agente, no intuito de determinar o grau de utilidade

em relação aos objetivos associados, com base em um estado (que será atingido através da tomada de uma ação).

Para identificar a maneira específica como o agente implementará o planejamento, uma nota com a identificação textual informando o critério a ser utilizado pode ser associada ao agente, de maneira análoga ao apresentado no caso do agente baseado em objetivo que utiliza planejamento (Figura 14).

4.3.5. Representações de AgentRoleClass

Silva (2004) representa uma AgentRoleClass por um retângulo sólido com uma curva na parte inferior. Ela possui três compartimentos separados por linhas horizontais. O compartimento superior contém o nome de papel do agente que deve ser único em seu espaço de nome incluído. O compartimento intermediário de lista contém a lista de objetivos e crenças associados ao papel, e o inferior, uma lista de deveres, direitos e protocolos. A Figura 16 representa um AgentRoleClass como proposto por Silva (2004).

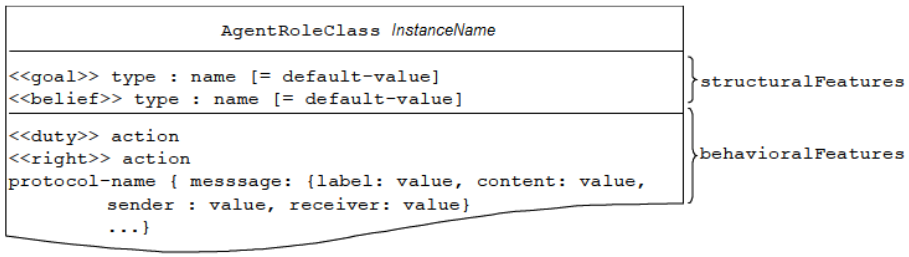

Figura 16 – Representação de uma instância da metaclasse AgentRoleClass.

Considerando que, de acordo com a literatura, os agentes reativos não possuem objetivos (*goals*) explícitos no código e, mais particularmente, os agentes reativos simples não possuem crenças (*beliefs*), se torna necessário alterar a representação do papel de agente de forma a manter a consistência com estas definições.

No caso dos agentes reativos simples, o papel de agente não inclui objetivos e crenças [Gonçalves et al. 2010a]. Desta maneira, a representação de um papel de agente para agentes reativos simples é representada como na Figura 17.

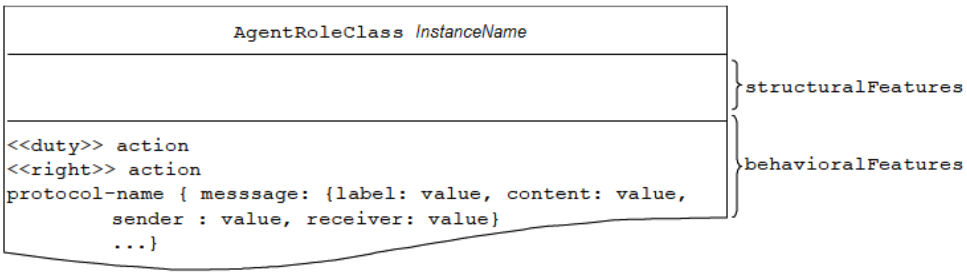

Figura 17 – Representação de uma instância da metaclasse AgentRoleClass para agentes reativos simples.

No caso dos agentes reativos baseados em conhecimento, o papel de agente não inclui objetivos, possuindo apenas crenças no compartimento intermediário [Gonçalves et al. 2010a]. Desta maneira, a representação de um papel de agente para agentes reativos baseados em conhecimento é ilustrada na Figura 18.

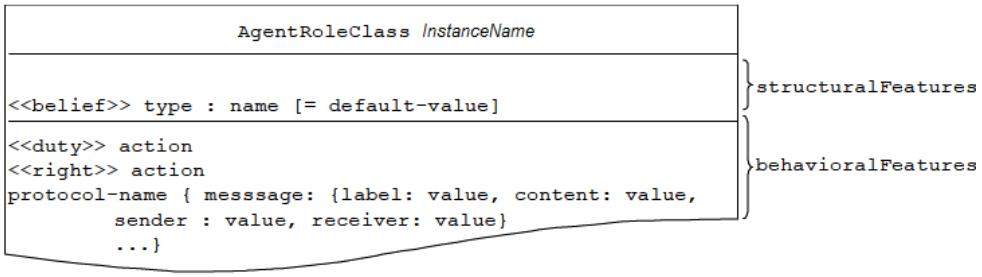

Figura 18 – Representação de uma instância da metaclasse AgentRoleClass para agentes reativos baseados em conhecimento.

A representação do papel de agente para os agentes pró-ativos não necessitou de alteração, pois os agentes pró-ativos possuem crenças e objetivos.

As alterações referentes à entidade AgentRoleClass influenciam nos diagramas de Organização e Papéis. Os diagramas dinâmicos não necessitam de alteração, tendo em vista que a entidade AgentRoleClass não foi alterada no compartimento inferior, que abriga as características comportamentais.

4.4. Representação do diagrama de sequência em MAS-ML 2.0

Nesta seção, a representação no diagrama de sequência das características das arquiteturas internas é demonstrada nesta seção.

4.4.1. Representação para agentes reativos

O conceito de percepção é representado no diagrama de sequência de MAS-ML 2.0 por uma seta de cabeça aberta partindo do agente para o ambiente (Figura 19), acompanhada do estereótipo <<perceives>> junto do nome da percepção e dos elementos que o agente consegue ver.

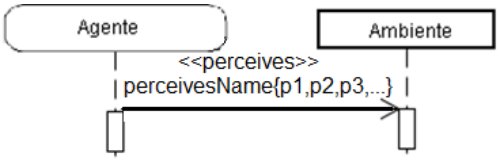

Figura 19 – Percepção do agente no diagrama de sequência de MAS-ML 2.0.

O comportamento do agente reativo em relação à sequência de ações que serão executadas não pode ser definido na fase de análise, no entanto é possível representar o conjunto de ações e a condição ou condições associada(s). A Figura 20 ilustra a ação tomada por um agente reativo, se a sala A ou a sala B tiverem sujeira, o agente executará a ação op.

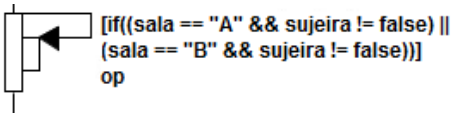

Figura 20 – Ação do agente reativo no diagrama de sequencia de MAS-ML 2.0.

A função próximo é representada no diagrama de sequência de MAS-ML 2.0 por uma seta de cabeça fechada, que começa no agente e termina no próprio agente, seguida do estereótipo <<next-function>> seguido pelo nome da função. A Figura 21 ilustra a função próximo no diagrama de sequência de MAS-ML 2.0.

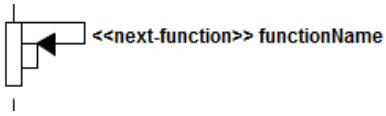

Figura 21 – Execução da função próximo no diagrama de sequência de MAS-ML 2.0.

Portanto, se um agente reativo simples for modelado, inicialmente teremos sua percepção e em seguida suas ações guiadas pelas regras condição-ação. No caso de um agente reativo baseado em conhecimento, inicialmente teremos a percepção, em seguida a função próximo e, por último, suas ações baseadas em regras condição-ação.

4.4.2. Representação de Agentes Pró-ativos

Inicialmente temos a percepção que é representada na Figura 19. O elemento função próximo é representado no diagrama de sequência por meio de uma seta de cabeça fechada, que começa no agente e termina nele mesmo, acompanhada do estereótipo <<next-function>> e do nome da função como mostrado na Figura 21. A função próximo é executada antes da função de formulação de objetivo e é utilizada pelos dois tipos de agentes pró-ativos abordados neste trabalho. Em seguida temos a

representação da função de formulação de objetivo e função de formulação do problema no diagrama de sequência. A representação é feita através de uma seta de cabeça fechada, que começa no agente e termina nele mesmo, acompanhada do respectivo estereótipo. As figuras 22 e 23 ilustram a função de formulação de objetivo e função de problema, respectivamente, no diagrama de sequência de MAS-ML 2.0.

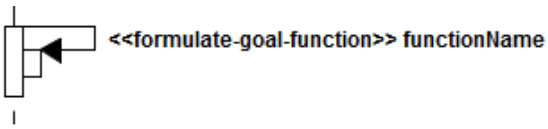

Figura 22 – função de formulação de objetivo.

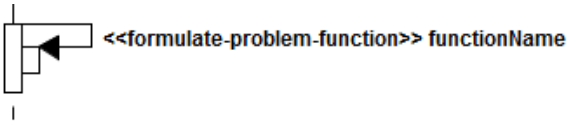

Figura 23 – função de formulação de problema.

No caso do agente que utiliza planejamento, é impossível prever a sequência de ações que o agente irá tomar para atingir o objetivo. O planejamento é representado por uma seta de cabeça fechada que começa no agente e termina nele mesmo acompanhado do estereótipo <<planning>>. As ações que podem ser utilizadas pelo planejamento para atingir o(s) objetivo(s) são representadas como definido inicialmente por Silva (2004). Opcionalmente, o critério ou algoritmo utilizado para realizar o planejamento pode ser especificado através de uma nota textual. A Figura 24 ilustra o planejamento no diagrama de sequência de MAS-ML 2.0.

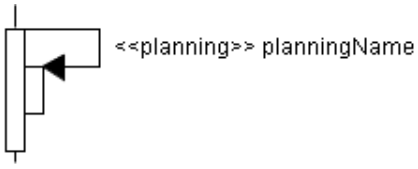

Figura 24 – Planejamento no diagrama de sequência de MAS-ML 2.0.

O elemento função-utilidade é representado no diagrama de sequência por meio de uma seta de cabeça fechada que começa no agente e termina nele mesmo, acompanhada do estereótipo <<utilityFunction>>. A Figura 25 ilustra a representação da função utilidade no diagrama de sequência de MAS-ML 2.0.

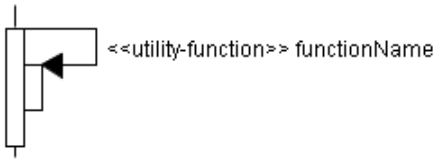

Figura 25 – Planejamento no diagrama de sequência de MAS-ML 2.0.

A sequência de ações (plano) para os agentes que utilizam planejamento é gerada em tempo de execução, portanto não é possível representar esta sequência na fase de análise. Desta maneira, para representar as ações do agente no diagrama de sequência de MAS-ML 2.0 são utilizados quadros de ocorrência de iteração e fragmento combinado, já existentes na UML. O uso dos elementos de iteração e fragmento combinado permite a representação de qualquer combinação de ações, tendo em vista que a combinação exata das ações não pode ser modelada antes da execução deste tipo de agente. Um exemplo é apresentado na Figura 26.

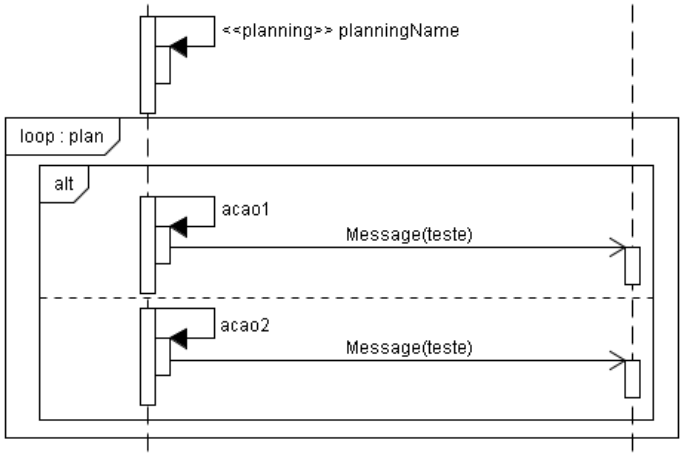

Figura 26 – Execução das ações do agente com planejamento no diagrama de sequência de MAS-ML 2.0.

O agente baseado em objetivo guiado por plano mantém a representação proposta por Silva (2004), assim como também o plano definido durante a fase de projeto.

No caso do agente baseado em objetivo guiado por planejamento, inicialmente é executada a percepção, função próximo, função de formulação de objetivo, função de formulação do problema e em seguida a execução de seu planejamento, que resulta na execução das ações possíveis associadas ao agente.

Finalmente, para o agente baseado em utilidade, necessitamos da percepção, função próximo, função de formulação de objetivo, função de formulação do problema, planejamento, função utilidade e resulta nas ações que são executadas, nesta ordem.

4.5. Representação do diagrama de atividades de MAS-ML 2.0

As características propostas por Silva, Choren e Lucena (2005) para o diagrama de atividades, foram mantidas. Desta maneira, cada atividade é representada por um retângulo de bordas arredondadas. As crenças do agente são representadas por um quadrado com a identificação das crenças utilizadas pelo agente e os objetivos são representados no canto superior direito através de uma descrição textual com o estereótipo <<goal>>.

4.5.1. Representação para agentes reativos

O diagrama de atividades do agente reativo simples representa o comportamento do mesmo, desde a percepção até a ação. O comportamento de um agente reativo simples é representado no diagrama de atividades de MAS-ML 2.0 da seguinte maneira: a atividade inicial é a percepção do agente, em seguida as regras condição ação selecionam uma das ações possíveis com base na percepção atual. Por último, a ação selecionada é executada. Após a execução da ação, o agente pode encerrar sua execução ou realizar uma nova percepção. A Figura 27 ilustra o diagrama de atividades para um agente reativo simples.

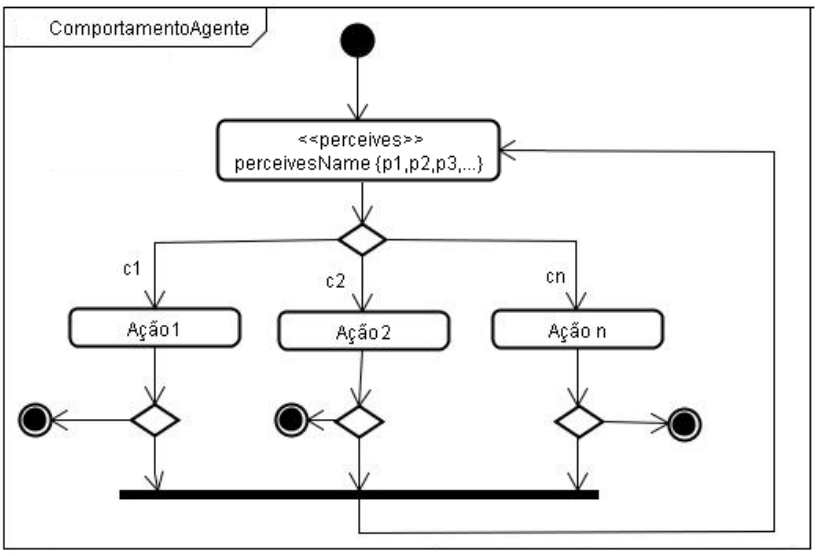

Figura 27 – Representação do diagrama de atividades para agentes reativos simples.

O diagrama de atividades do agente reativo baseado em conhecimento representa a execução do mesmo, desde a percepção até a ação. O comportamento de um agente reativo baseado em conhecimento é representado no diagrama de atividades de MAS-ML 2.0 da seguinte maneira: a atividade inicial é a percepção do agente, a partir da qual

a função próximo atualiza as crenças. As regras condição-ação selecionarão uma das ações possíveis com base nas crenças atualizadas. Por ultimo a ação selecionada é executada. A Figura 28 ilustra o diagrama de atividades para um agente reativo baseado em conhecimento.

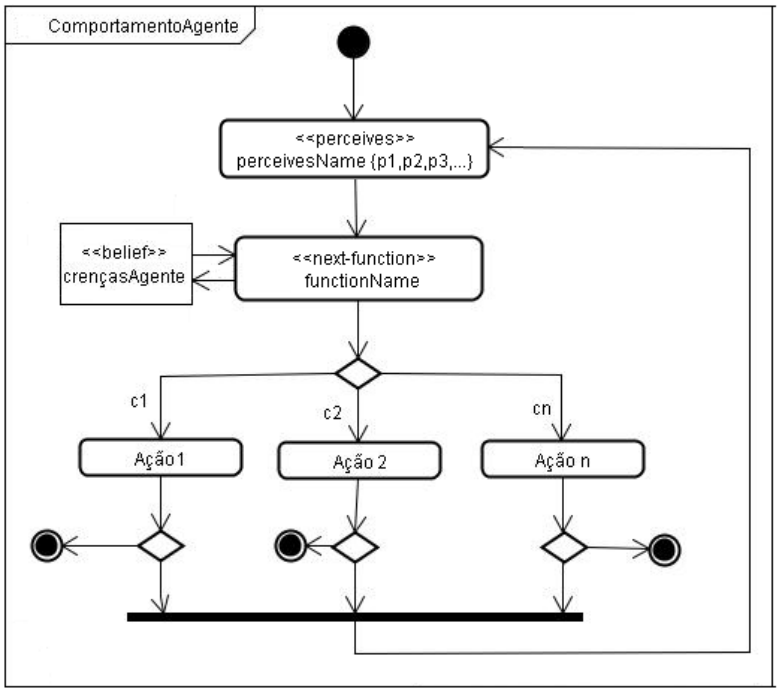

Figura 28 – Representação do diagrama de atividades para agentes reativos baseados em conhecimento.

4.5.2. Representação para agentes pró-ativos

O diagrama de atividades do agente baseado em objetivo com planejamento representa a execução do mesmo, desde a percepção até a ação. O comportamento de um agente baseado em objetivo é representado no diagrama de atividades de MAS-ML 2.0 da seguinte maneira: a atividade inicial é a percepção do agente, em seguida, a função próximo atualiza as crenças com base na percepção atual. A função de formulação de objetivo é executada e em seguida é executada a função de formulação do problema. O planejamento é executado para verificar qual ação (ou sequencia de ações) deve ser tomada. Por último a ação selecionada é executada. Após a execução de uma ação, o agente poderá ter chegado a um estado desejado e encerrar sua execução; poderá continuar executando outras ações da sequência de ações definida no planejamento ou realizar a execução desde a percepção. A Figura 29 ilustra o diagrama de atividades para um agente baseado em objetivo com planejamento.

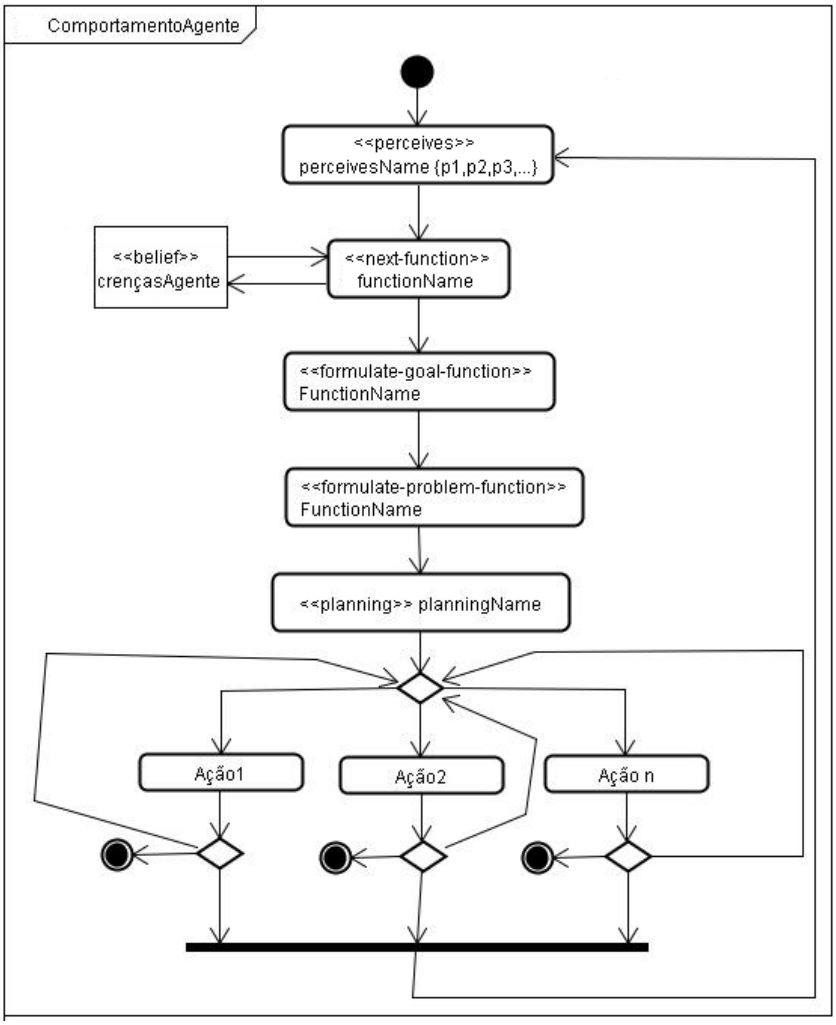

Figura 29 – Representação do diagrama de atividades para agentes baseados em objetivo com planejamento.

O comportamento do agente baseado em utilidade é representado no diagrama de atividades de MAS-ML 2.0 da seguinte maneira: a atividade inicial é a percepção do agente. Em seguida, a função próximo atualiza as crenças com base na percepção atual. A função de formulação de objetivo é executada e em seguida a função de formulação do problema. O planejamento é executado para verificar que ação deve ser tomada, neste caso a função utilidade participa da escolha da ação. Finalmente, a ação selecionada é executada. Após a execução de uma ação, o agente poderá ter chegado a um estado desejado e encerrar sua execução; poderá continuar executando outras ações da sequência de ações definida no planejamento ou realizar a execução desde a percepção. A Figura 30 ilustra o diagrama de atividades para um agente baseado em utilidade.

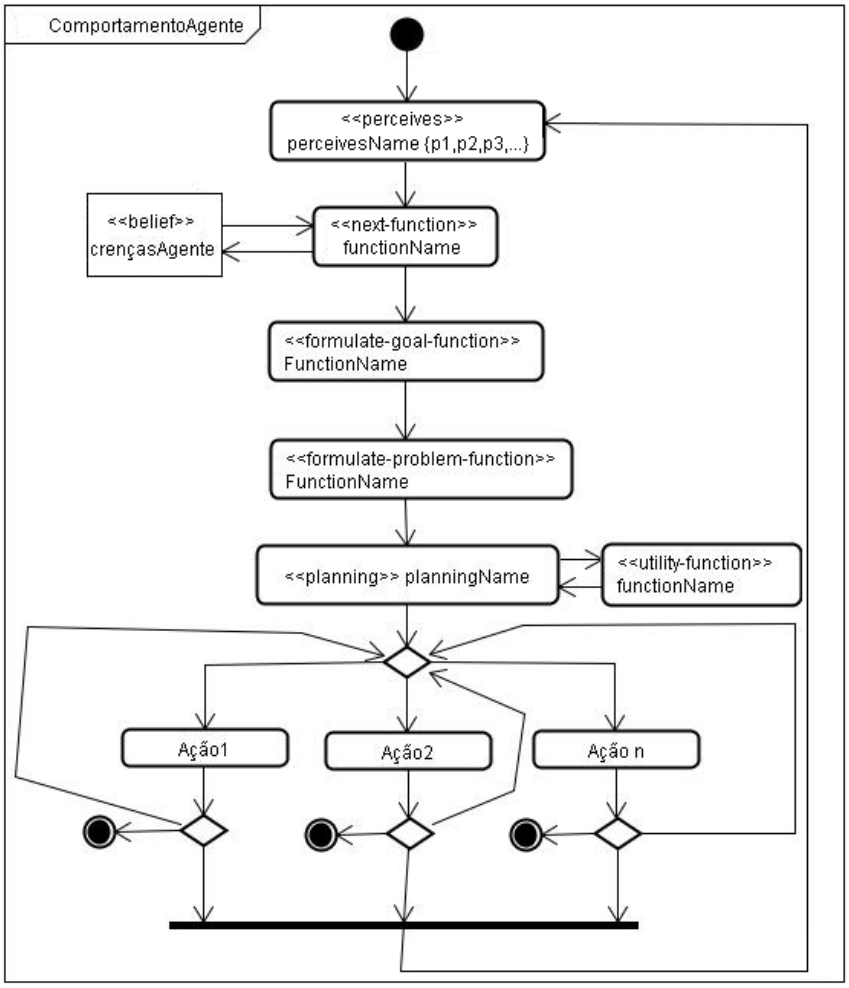

Figura 30 – Representação do diagrama de atividades para agentes baseados em utilidade.

Neste capítulo foi apresentada uma extensão de MAS-ML para modelar corretamente os agentes reativos simples, reativos baseados em modelo, baseados em objetivo guiado por planejamento e baseado em utilidade. Inicialmente, os conceitos do *framework* conceitual TAO foram adaptados. Em seguida, a extensão de MAS-ML foi apresentada a partir do metamodelo, passando pela representação de cada elemento adicionado e a representação dos agentes e papéis nos diagramas estáticos e dinâmicos.

5. Extensão na ferramenta de modelagem

Neste capítulo é apresentado o processo de extensão da ferramenta de modelagem utilizada para dar apoio à modelagem de SMAs com base em MAS-ML 2.0.

5.1. Escolha da ferramenta

Atualmente existem duas ferramentas para modelagem de SMAs utilizando MAS-ML: MAS-ML tool [Farias et al. 2009] e VisualAgent [De Maria et al. 2005]. VisualAgent é um ambiente de desenvolvimento de software que tem a finalidade de auxiliar desenvolvedores na especificação, projeto e implementação de SMAs. Ela também propõe uma abordagem dirigida por modelo para o desenvolvimento de SMAs fazendo uso do ASF (*Agent Society Framework*) [Silva, Cortés e Lucena 2004], o que representa um ponto relevante da abordagem.

Por outro lado, MAS-ML tool é um ambiente de modelagem específico de domínio que atende à modelagem de sistemas multi-agentes, de acordo com a especificação do metamodelo definido na concepção original de MAS-ML. O objetivo de MAS-ML Tool é fornecer um ambiente de modelagem no qual desenvolvedores possam trabalhar com os conceitos do domínio do problema, ao mesmo tempo que utilizam explicitamente conceitos definidos no domínio da solução, neste caso os conceitos e abstrações definidos no paradigma de SMAs.

Uma das principais vantagens da MAS-ML tool em relação à VisualAgent é a capacidade de realizar verificação do modelo em relação ao metamodelo da MAS-ML. A ausência desta funcionalidade na VisualAgent pode comprometer as sucessivas transformações dos modelos presentes na abordagem MDD [Mellor, Clark e Futagami 2003] proposta e levar a um mal entendimento dos modelos criados, além de que um modelo inconsistente pode gerar código com erros. Além disso, a falta de documentação e acesso ao código fonte levaram à escolha da ferramenta MAS-ML tool para hospedar a implementação da extensão proposta à MAS-ML.

5.2. MAS-ML tool

O ambiente foi desenvolvido como um *plug-in* da plataforma Eclipse [Eclipse 2009]. Isso implica que os usuários podem trabalhar com modelagem de SMAs ao mesmo tempo em que fazem uso dos recursos oferecidos pela plataforma Eclipse. Dado que muitas plataformas de agentes são implementadas em Java, tais como Jade

[Bellifemine, F. L.; Caire, G.; Greenwood], Jadex [Pokahr, Braubach e Lamersdorf 2003], Jason [Bordini, Wooldridge e Hübner 2007], o uso da plataforma Eclipse também facilita uma possível geração de código dentro do mesmo ambiente de desenvolvimento.

A ferramenta foi desenvolvida utilizando uma abordagem dirigida por modelos, onde o modelo central e de maior abstração é o próprio metamodelo da linguagem MAS-ML. A Figura 31 mostra uma visão geral da MAS-ML tool juntamente com alguns dos seus componentes principais. As letras representam os seguintes recursos:

(A) Package Explorer - Tem como funcionalidade central permitir a organização dos arquivos em uma estrutura de árvore a fim de que seja possível um melhor gerenciamento e manipulação dos arquivos;

(B) Modeling View - Os modelos que são criados precisam necessariamente ser visualizados com o objetivo de atender dois requisitos básicos quando se usa modelos: compreensibilidade e a comunicação. Diante dessa necessidade, a modeling view permite os desenvolvedores visualizar e editar os modelos de forma interativa;

(C) Nodes Palette - Os construtores que podem fazer parte dos diagramas encontram-se na nodes palette. Sendo assim, os desenvolvedores podem criar instâncias desses construtores, as quais são visualizadas na modeling view.

(D) Relationship Palette - Os relacionamentos que podem ser estabelecidos entre os construtores presentes na nodes palette são disponibilizados na relationship palette.

(E) Properties View - Permite manipular de forma precisa as propriedades dos modelos. Exibe as propriedades definidas no metamodelo da MAS-ML quando o modelo é exibido e selecionado na modeling view.

(F) Problems View - Caso exista alguma inconsistência no modelo, ela será mostrada em problems view. Esta funcionalidade é particularmente importante para viabilizar o uso de modelagem de SMAs dentro do contexto do desenvolvimento dirigido por modelos, na qual modelos são vistos como artefatos de primeira ordem.

(G) Outline View - Uma vez que um modelo tenha sido criado, um overview da distribuição dos elementos presentes no mesmo poderá ser visualizado através da outline view.

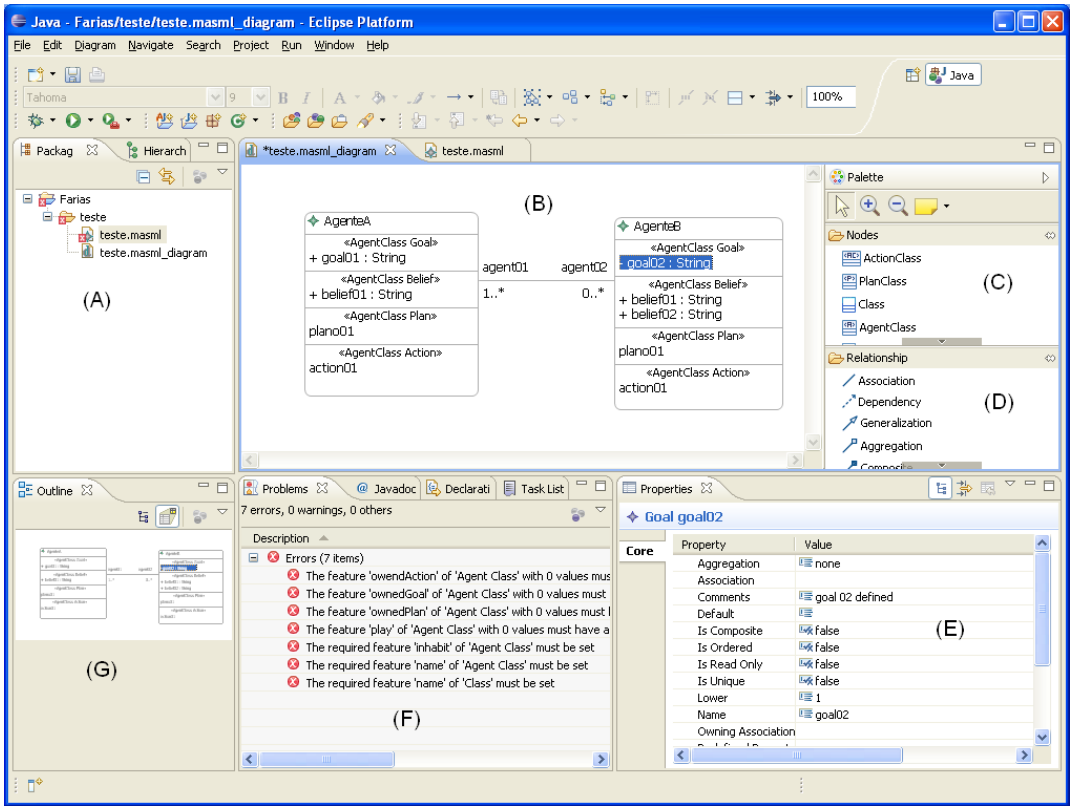

Figura 31 – Visão Geral da MAS-ML Tool [Farias et al. 2009].

5.3. Extensão em MAS-ML tool

A versão da ferramenta utilizada como base para a sua evolução disponibiliza exclusivamente o diagrama de classes. No contexto da extensão proposta, o diagrama de classes precisou ser modificado, e o diagrama de organização criado, pois os modelos criados através destes dois diagramas conseguem expressar as alterações realizadas nas entidades de MAS-ML. O diagrama de papéis e os diagramas dinâmicos serão desenvolvidos como continuação deste trabalho.

A estratégia adotada para implementar as extensões propostas segue a abordagem dirigida por modelos, que foi utilizada originalmente para desenvolver a ferramenta, utilizando, neste caso, o metamodelo da linguagem MAS-ML estendido. A seguir, são descritas, de forma resumida, as cinco etapas realizadas no processo de evolução da ferramenta:

1. **Extensão do Modelo de Domínio.** Originalmente, o metamodelo foi especificado usando o EMOF[5] (*Essencial Meta-Object Facility*), uma linguagem de definição de metamodelo usada para definição da UML. Em relação à evolução, os estereótipos apresentados na Seção 4.2 foram adicionados a ActionClass através de uma semântica chamada ActionSemantics. Esta semântica representa as opções 0- Sem estereótipo, 1- next-function, 2- utility-function, 3- formulate-problem-function e 4- formulate-goal-function. A Figura 32 ilustra a criação da semântica para ActionClass.

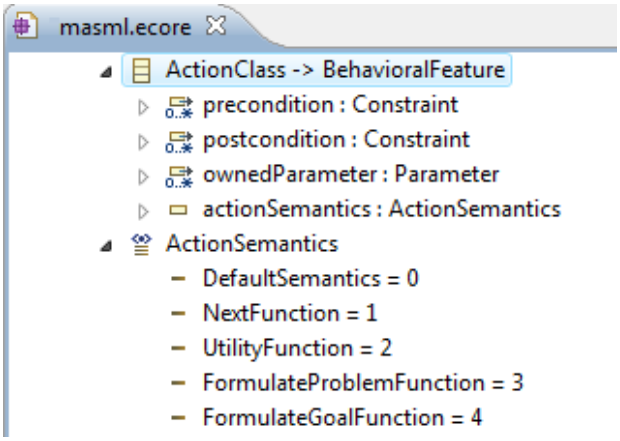

Figura 32 – Representação da semântica relacionada a ActionClass.

2. **Extensão do Modelo Gráfico.** O Modelo Gráfico contém a definição das entidades e de suas propriedades, assim como de seus relacionamentos com base no metamodelo da linguagem. No contexto da evolução, as novas metaclasses apresentadas na Seção 4.2 foram adicionadas na Definição do Modelo Gráfico da ferramenta e seus relacionamentos com outras entidades foram criados, como pode ser observado na Figura 32. A multiplicidade do relacionamento entre as entidades goal, belief e plan e a entidade AgentClass foi alterada de 1 para 1..* para 1 para 0..*, pois objetivos, crenças e planos não são mais obrigatórios para o agente. As entidades perception e planning também possuem um relacionamento

[5] O EMOF é parte integrante do plug-in EMF (Eclipse Metamodel Framework) que consiste em um framework de modelagem e de geração de código para construir aplicações baseadas em modelos [EMF 2009]

com multiplicidade 1 para 0..*. A Figura 33 ilustra as alterações realizadas.

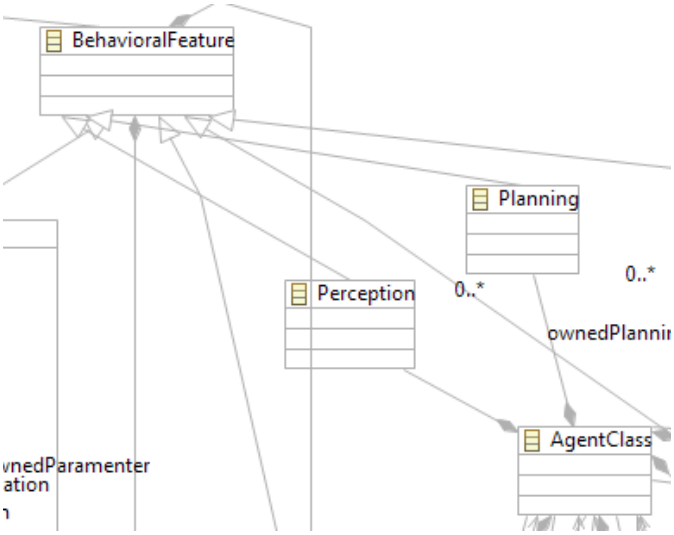

Figura 33 – Representação das novas metaclasses e seus relacionamentos.

3. **Extensão do Modelo de Ferramenta.** Esta etapa especifica quais elementos fazem parte da paleta da ferramenta. Para isso, essa etapa recebe como entrada o modelo de domínio e o modelo gráfico determinados anteriormente. Adicionalmente, para o diagrama de classes foram criados novos elementos na paleta da ferramenta para criar percepções e planejamento. A Figura 34 ilustra os novos elementos adicionados a paleta.

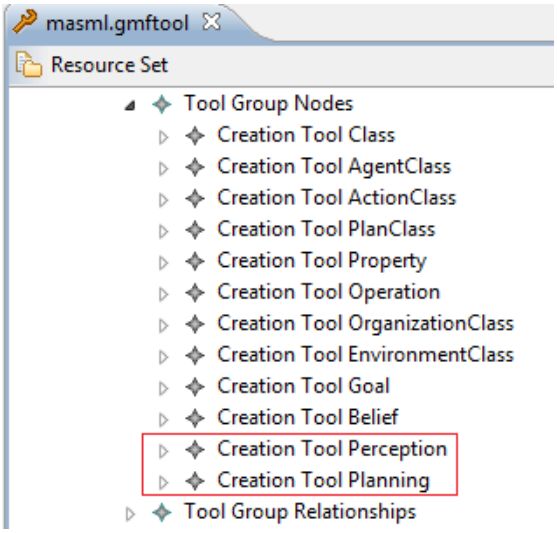

Figura 34 – Novos elementos da paleta no modelo de ferramenta.

Na representação do modelo de ferramenta do diagrama de Organização, foram adicionados os elementos Perception, Planning, AgentRoleClass, ObjectRoleClass, Duty, Right e Protocol e os relacionamentos inhabit, ownership e play. Na Figura 35, estes elementos podem ser observados na representação no modelo de ferramenta do diagrama de organização.

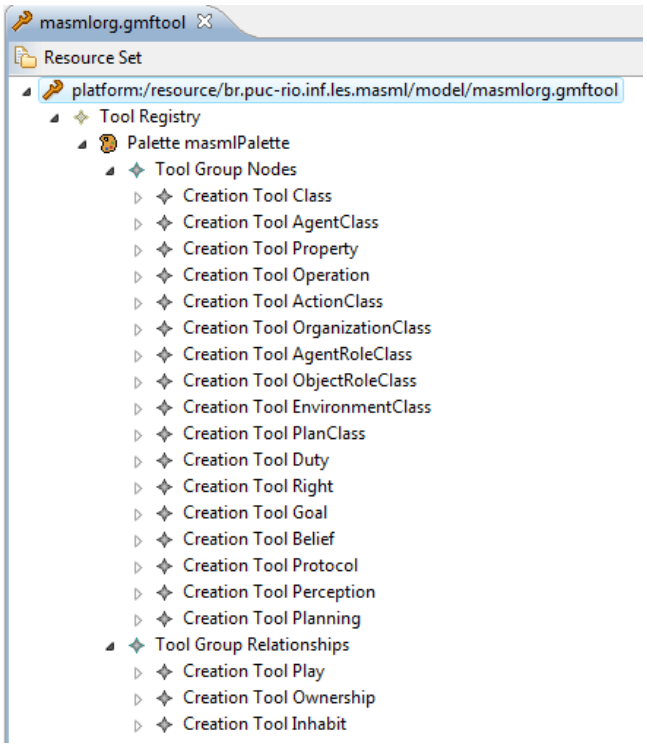

Figura 35 – Novos elementos do diagrama de organização na paleta.

4. **Extensão da Definição do Modelo Gráfico.** Esta etapa está relacionada com a representação dos elementos no respectivo diagrama. O modelo Gráfico, que tem como entrada o modelo de domínio, é utilizado na combinação para gerar o modelo de mapeamento. Para esta etapa, foram criados compartimentos para a representação das percepções e do planejamento, além da representação dos nós (*nodes*) provenientes da etapa anterior. Na Figura 36 podem ser vistos os novos nós e os novos compartimentos para o diagrama de classes.

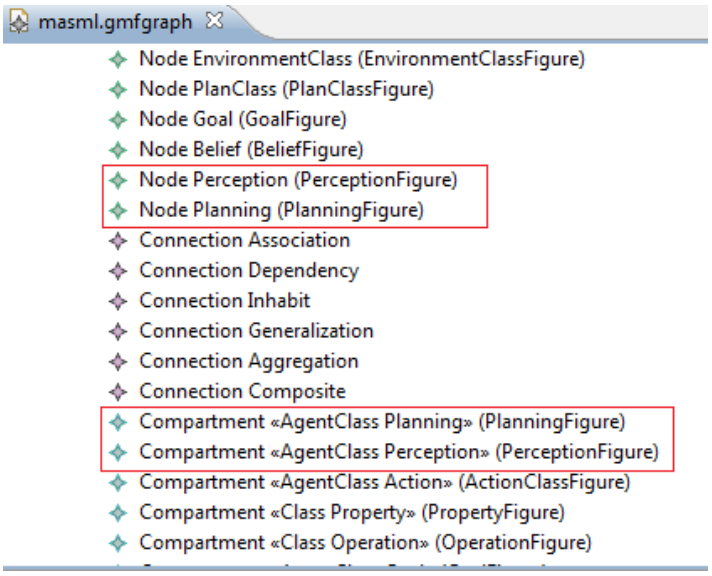

Figura 36 – Novos elementos na definição do modelo gráfico da ferramenta.

Na representação do diagrama de Organização, na definição do modelo gráfico, foram adicionados os elementos Planning, Perception, AgentRoleClass, ObjectRoleClass, Duty, Right e Protocol, compartimentos para Planning, Perception, Duty, Right e Protocol e os relacionamentos inhabit, ownership e play. Na Figura 37, estes elementos podem ser observados na representação no modelo gráfico do diagrama de organização.

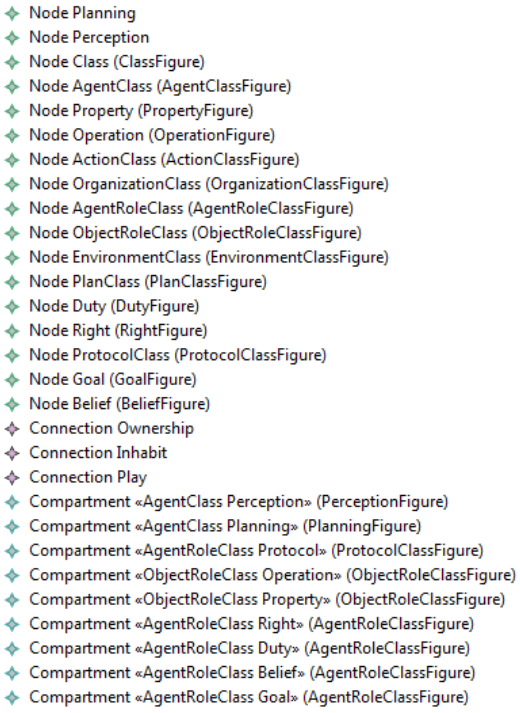

Figura 37 – Novos elementos do diagrama de organização no modelo gráfico.

5. **Extensão do Modelo de Mapeamento.** Este modelo é criado a partir de uma combinação dos modelos definidos nos itens 1, 2, 3 e 4 desta seção, ou seja, se concentra na construção de um mapeamento entre os três modelos: modelo de domínio, modelo gráfico, e o modelo de ferramenta. O mapeamento gerado é usado como entrada em um processo de transformação que tem como objetivo criar um modelo específico de plataforma.

   As regras de validação foram implementadas através da multiplicidade das metaclasses na Definição do Modelo Gráfico da ferramenta e através de restrições em OCL (*Object Constraint Language*). A representação das regras OCL no Eclipse é mostrada na Figura 38.

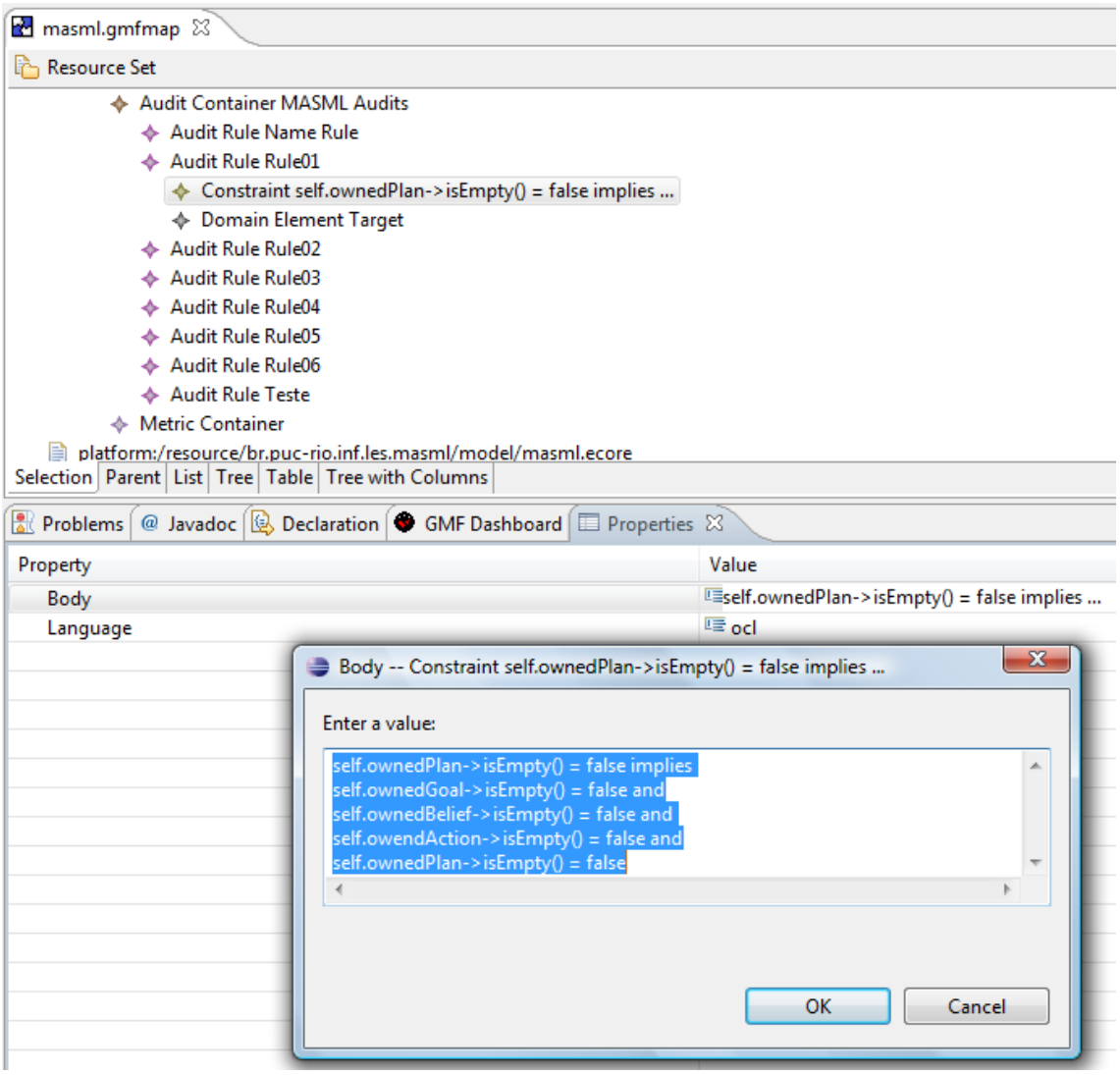

Figura 38 – Representação das regras OCL na ferramenta.

A seguir, cada regra e seu respectivo propósito são apresentados.

Regra 1: Se o agente possui plano, então ele possui objetivo, crença e ação.

Regra 2: Se o agente possui plano, então não possui percepção.

Regra 3: Se o agente possui um objetivo, então possui plano ou planejamento.

Regra 4: Se o agente possui planejamento, então ele possui crença, objetivo, percepção e ação.

Regra 5: Se o agente possui plano, então ele não possui planejamento.

Regra 6: Caso o agente possua planejamento, então ele não terá plano.

As regras criadas são descritas no Quadro 2, no qual pode ser vista a identificação da regra na ferramenta e sua respectiva definição em OCL.

Quadro 2 – Regras de validação dos modelos.

Regra	Definição em OCL
Regra 01	self.ownedPlan->isEmpty() = false implies self.ownedGoal->isEmpty() = false and self.ownedBelief->isEmpty() = false and self.owendAction->isEmpty() = false and self.ownedPlan->isEmpty() = false
Regra 02	self.ownedPlan->isEmpty() = false implies self.ownedPerception->isEmpty() = true and self.ownedPlan->isEmpty() = false
Regra 03	self.ownedGoal->isEmpty() = false implies ((self.ownedPlan->isEmpty() = false and self.ownedPlanning->isEmpty() = true) or (self.ownedPlan->isEmpty() = true and self.ownedPlanning->isEmpty() = false)) and self.ownedGoal->isEmpty() = false
Regra 04	self.ownedPlanning->isEmpty() = false implies (self.ownedBelief->isEmpty() = false and self.ownedGoal->isEmpty() = false and self.ownedPerception->isEmpty() = false and self.owendAction->isEmpty() = false) and self.ownedPlanning->isEmpty() = false
Regra 05	self.ownedPlan->isEmpty() = false implies self.ownedPlanning->isEmpty() = true and self.ownedPlan->isEmpty() = false
Regra 06	self.ownedPlanning->isEmpty() = false implies self.ownedPlan->isEmpty() = true and self.ownedPlanning->isEmpty() = false

6. **Geração da Ferramenta.** Finalmente, seguindo a abordagem generativa [Czarnecki e Eisenecker 2000], o próximo passo consiste na geração do

código da ferramenta levando em consideração o modelo específico de plataforma que foi estendido na etapa anterior.

Na Figura 39 é apresentado o Dashboard, recurso visual do Eclipse utilizado no desenvolvimento utilizando a abordagem generativa com GMF [GMF 2009], no qual as fases descritas anteriormente podem ser percebidas.

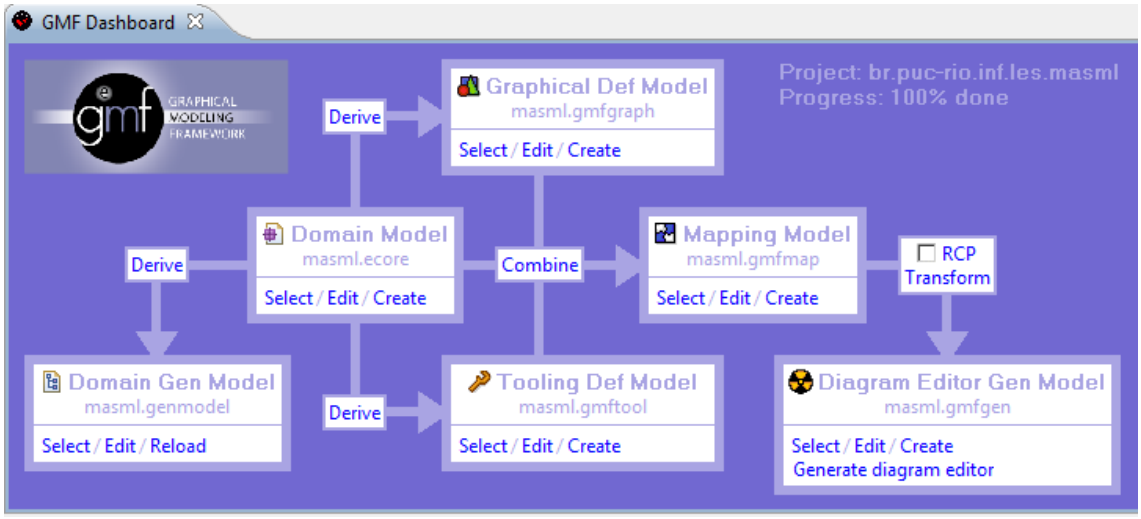

Figura 39 – Representação do Dashboard.

No diagrama de classes, o agente passou a ter dois compartimentos extras para abrigar as percepções do agente e o planejamento. Adicionalmente, uma ação pode ter um estereótipo associado para representar a função próximo, formulação de objetivo, formulação de problema e utilidade. A representação do agente na ferramenta pode ser vista na Figura 40, e o diagrama de classes do SMA para o TAC-SCM (Capítulo 6) é apresentado na Figura 41, ambos foram criados com a ferramenta MAS-ML tool após a extensão.

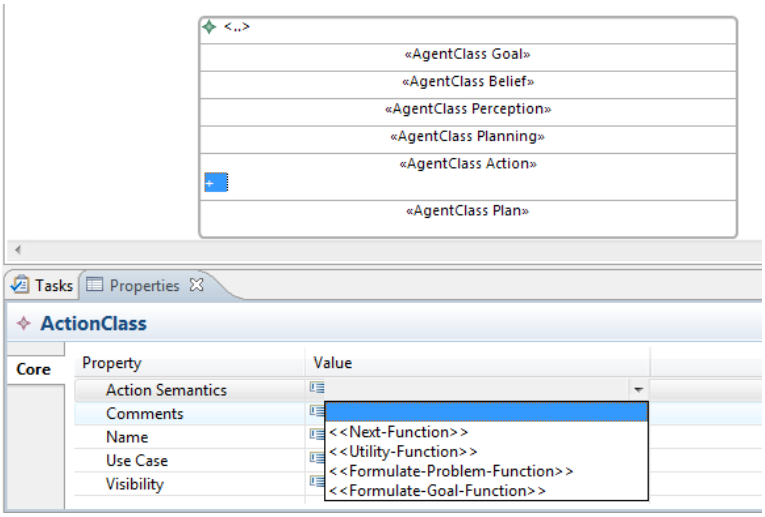

Figura 40 – Nova representação do agente na ferramenta MAS-ML tool.

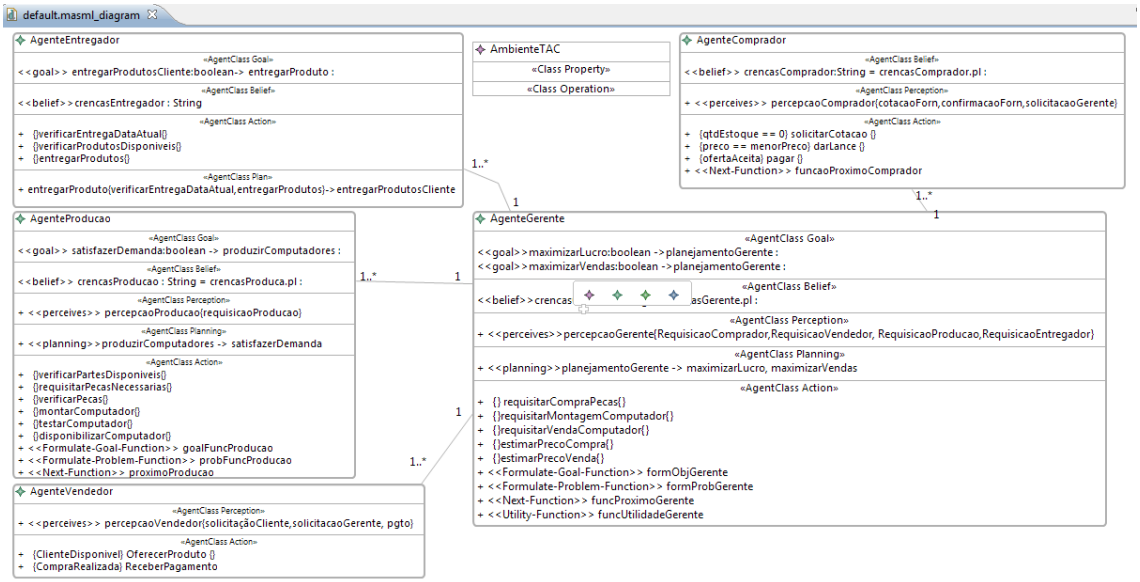

Figura 41 – Diagrama de classes para o estudo de caso do TAC-SCM.

A estrutura criada para o diagrama de classes foi aproveitada para a criação do diagrama de organização, pois grande parte das entidades é comum aos dois diagramas. Os papéis de agente foram representados de acordo com a extensão proposta neste trabalho (Seção 4.3.5). A Figura 42 ilustra um papel de agente no diagrama de organização criado na ferramenta MAS-ML tool.

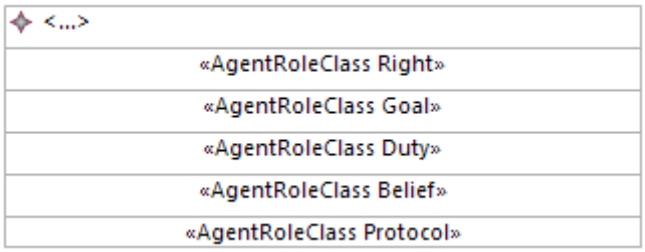

Figura 42 – Representação do papel de agente na ferramenta MAS-ML tool.

Os relacionamentos *posse* e *exerce* definidos em MAS-ML, os quais fazem parte do diagrama de organização, foram adicionados, bem como o relacionamento *habita* teve sua semântica alterada para permitir que agentes, papéis de agente e organizações possam participar deste relacionamento. Na Figura 43, o diagrama de organização do SMA para o TAC-SCM (Capítulo 6) pode ser visto.

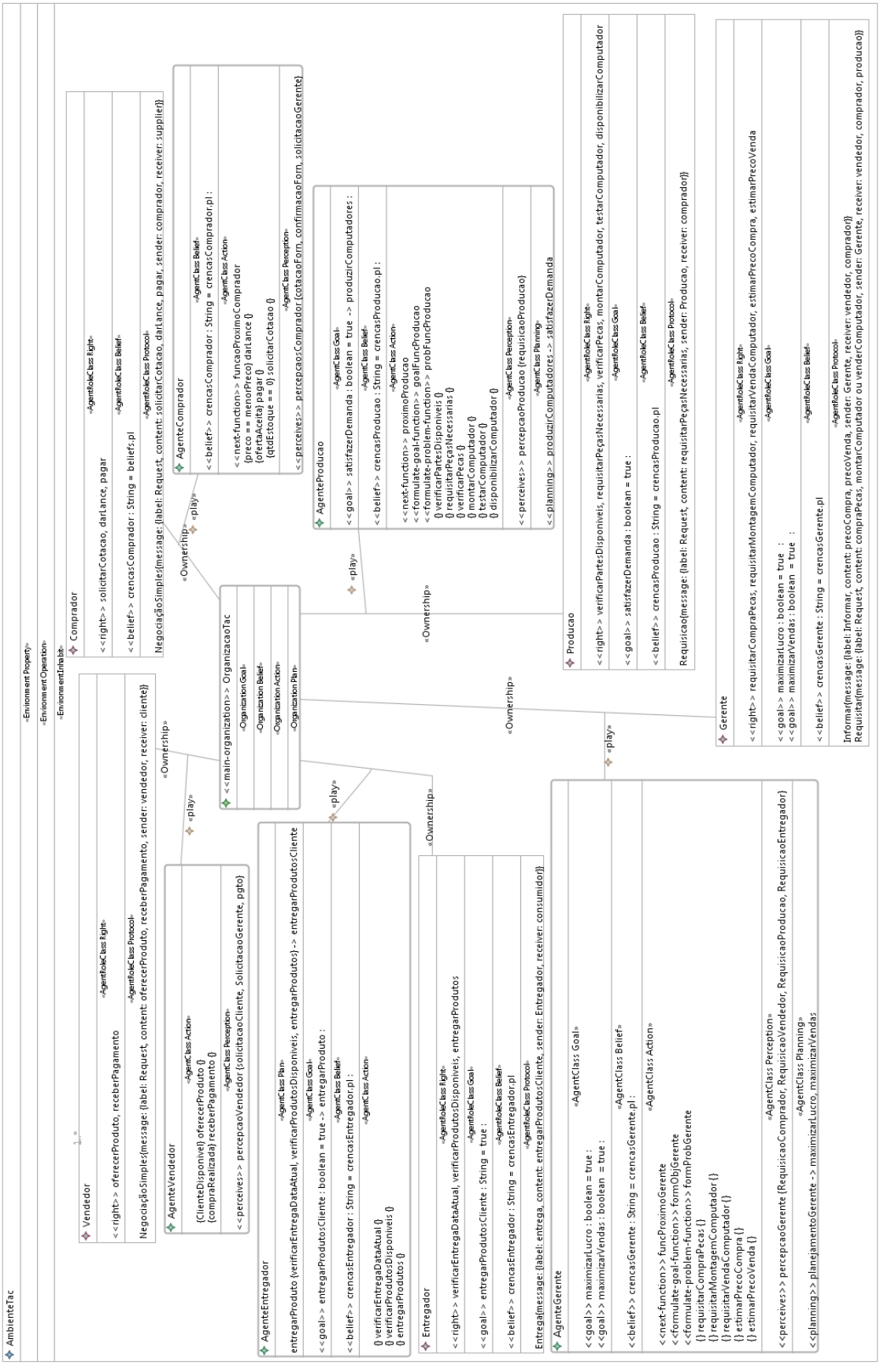

Figura 43 – Diagrama de organização para o estudo de caso do TAC-SCM.

Os modelos gerados a partir da ferramenta podem ser validados aplicando as restrições OCL implementadas, as quais podem ser acessadas através do menu *edit* e da opção *validate*. Caso alguma regra seja violada, o elemento que apresenta problema é assinalado e os problemas são apresentados através do recurso *problems* do eclipse. A Figura 44 apresenta o recurso de validação no Eclipse, o elemento *Teste* do diagrama que apresenta problemas detectados pela validação e o relatório dos problemas ocorridos.

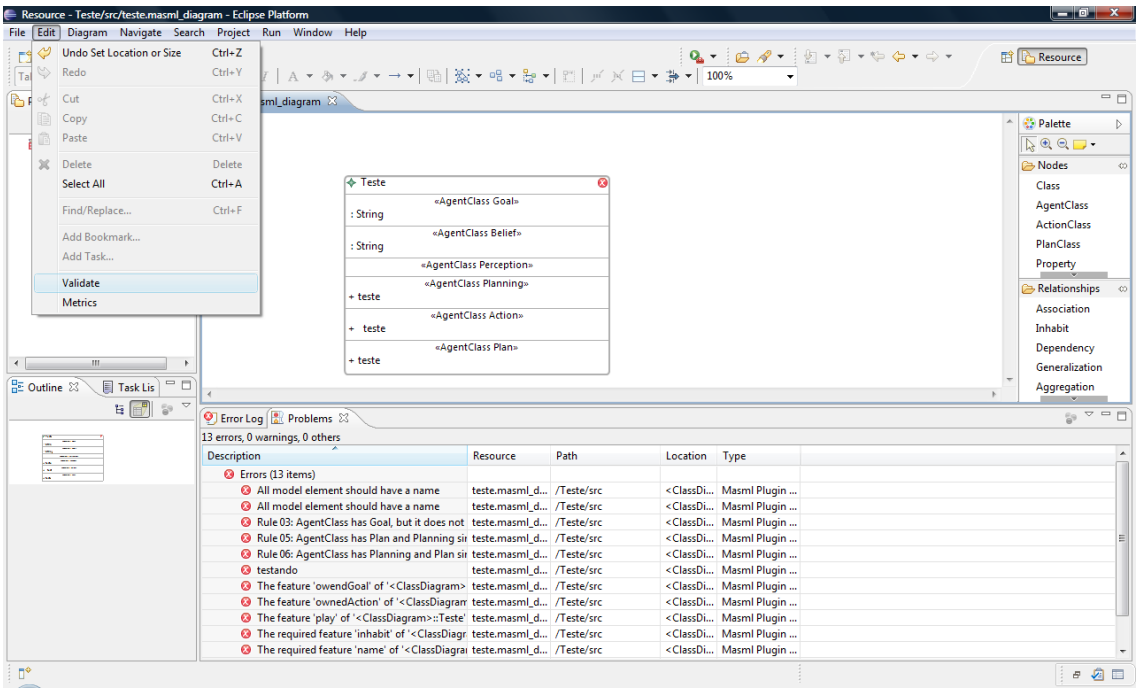

Figura 44 – Validação na ferramenta MAS-ML tool.

Neste capítulo, o processo de evolução da ferramenta MAS-ML tool em conformidade com MAS-ML 2.0 foi descrito. Neste contexto, a extensão do diagrama de classes da ferramenta MAS-ML tool envolveu: a criação de duas novas metaclasses, a alteração da multiplicidade de alguns relacionamentos, criação de novos compartimentos, bem como a representação das novas entidades na paleta e criação de novas restrições OCL.

O diagrama de organização foi criado, tomando por base as alterações implementadas para o diagrama de classes. Foi necessário remover a representação dos relacionamentos associação, dependência, generalização, agregação e composição. Os relacionamentos ownership e play foram adicionados na representação deste diagrama. Os papéis de agente e de objeto, que não poderiam ser utilizados no diagrama de classes, foram adicionados. Vale ressaltar que os elementos adicionados já estavam

presentes no modelo de domínio e no modelo gráfico, porém não eram utilizados na representação.

6. Estudos de caso

A seguir são apresentados três estudos de caso: o Mundo Wumpus, Mundo do aspirador de pó e o TAC-SCM. Os diagramas apresentados neste capítulo foram criados com o software JUDE community 5.3 (disponível em http://jude.change-vision.com/jude-web/download/index.html) que é baseado na UML, e adequados para MAS-ML 2.0 através de editores de imagens.

6.1. Estudo de caso 1: O Mundo Wumpus

Este estudo de caso foi incluído com o intuito de disponibilizar um comparativo entre modelagem de um agente realizada com MAS-ML antes da extensão e a modelagem do mesmo agente com MAS-ML 2.0. Inicialmente o mundo Wumpus é brevemente descrito e em seguida a modelagem do agente é apresentada.

O mundo Wumpus é um jogo descrito por Russell e Norvig (2004) da seguinte maneira: trata-se de uma caverna consistindo de salas conectadas por passagens. Escondido em algum lugar da caverna está o Wumpus, um monstro que devora qualquer agente que entrar em sua sala. O Wumpus pode ser atingido por um agente, mas este tem apenas uma flecha. Algumas salas têm alçapões sem fundo nas quais cairá qualquer um que vagar por elas. Em alguma sala há ouro que deixará rico qualquer um que o encontrar. Assim, o objetivo do agente é encontrar o ouro em algum local na caverna. Neste estudo de caso, o mundo Wumpus foi simplificado, supondo que o agente conhece os limites da caverna e não foi levada em conta a existência de flechas.

O agente caçador pode perceber cheiro do Wumpus e a brisa dos alçapões e é capaz de agir se movendo para a direita, esquerda, cima ou para baixo e pode capturar o ouro. A Figura 45 ilustra o funcionamento do mundo Wumpus.

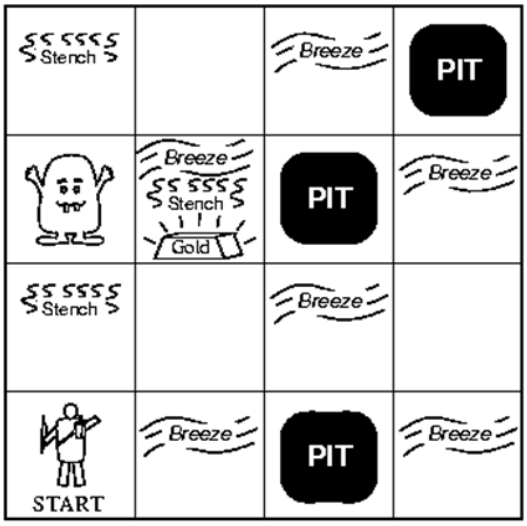

Figura 45 – Ilustração do Mundo Wumpus.

6.1.1. Modelagem do Mundo Wumpus

O ambiente no qual o agente caçador se encontra pode ser definido como estocástico e parcialmente observável, e inicialmente o agente não tem qualquer conhecimento sobre a configuração do ambiente. Considerando as características do ambiente e do agente, uma arquitetura reativa é a mais indicada. Além disso, o agente caçador precisa de uma base de conhecimento para que as informações acerca do ambiente possam ser inferidas e o risco de cair em alçapões ou ser pego pelo Wumpus diminua. Com isso, é estabelecida uma arquitetura reativa baseada em conhecimento como a mais adequada para a modelagem do agente caçador do mundo Wumpus.

A Figura 46 apresenta a modelagem do agente caçador nos diagramas estáticos, utilizando a notação MAS-ML antes da extensão proposta neste trabalho.

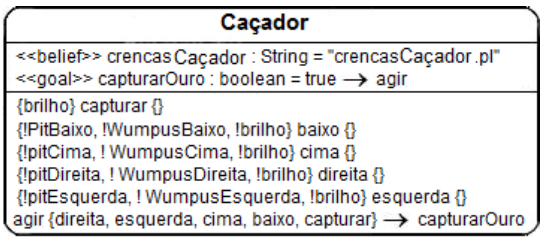

Figura 46 - Agente caçador nos diagramas estáticos de MAS-ML.

A Figura 47 apresenta a modelagem do agente caçador no diagrama de sequência, utilizando a notação de MAS-ML antes da extensão proposta neste trabalho. Os agentes reativos não possuem a sequência de suas ações definida em tempo de execução, não possuem um objetivo explícito e conseguem perceber o ambiente para tomar a decisão correta para o momento atual, deste modo a representação da Figura 47 não é adequada

para um agente com arquitetura reativa baseada em conhecimento. A função próximo, que está presente no agente reativo baseado em conhecimento, também não é representada através do agente em MAS-ML antes da extensão.

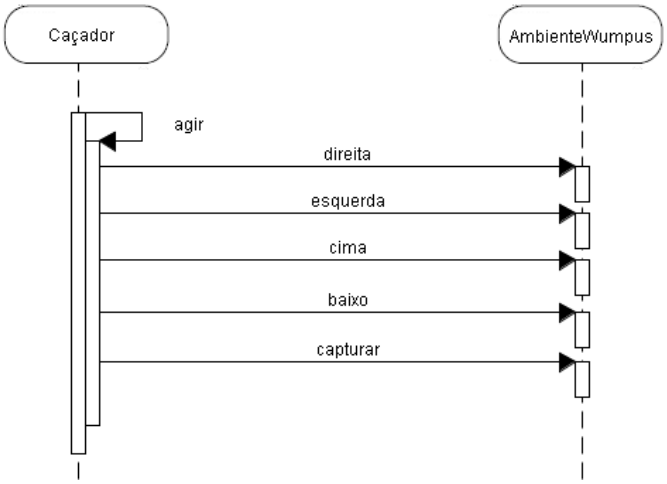

Figura 47 - Agente caçador no diagrama de sequência de MAS-ML.

Similarmente, a Figura 48 apresenta a modelagem do agente caçador no diagrama de atividades utilizando MAS-ML.

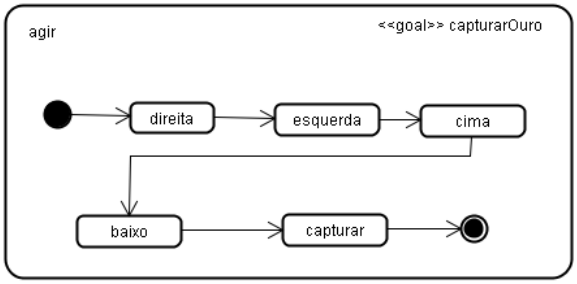

Figura 48 - Agente caçador no diagrama de atividades de MAS-ML.

Note que, dadas as características do ambiente do mundo Wumpus (estocástico e parcialmente observável), o agente dotado de um objetivo explícito e um plano composto por uma sequência de ações, de acordo com a concepção original de agente em MAS-ML, não seria uma opção de modelagem adequada. A Figura 49 apresenta a modelagem do agente caçador nos diagramas estáticos, utilizando a notação MAS-ML 2.0 prevista para o caso de arquiteturas reativas baseadas em conhecimento.

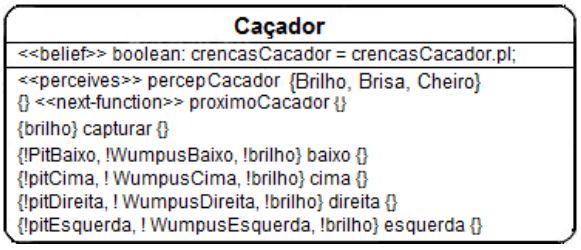

Figura 49 - Agente caçador nos diagramas estáticos de MAS-ML 2.0.

A Figura 50 apresenta a modelagem do agente caçador no diagrama de sequência, utilizando a notação de MAS-ML 2.0.

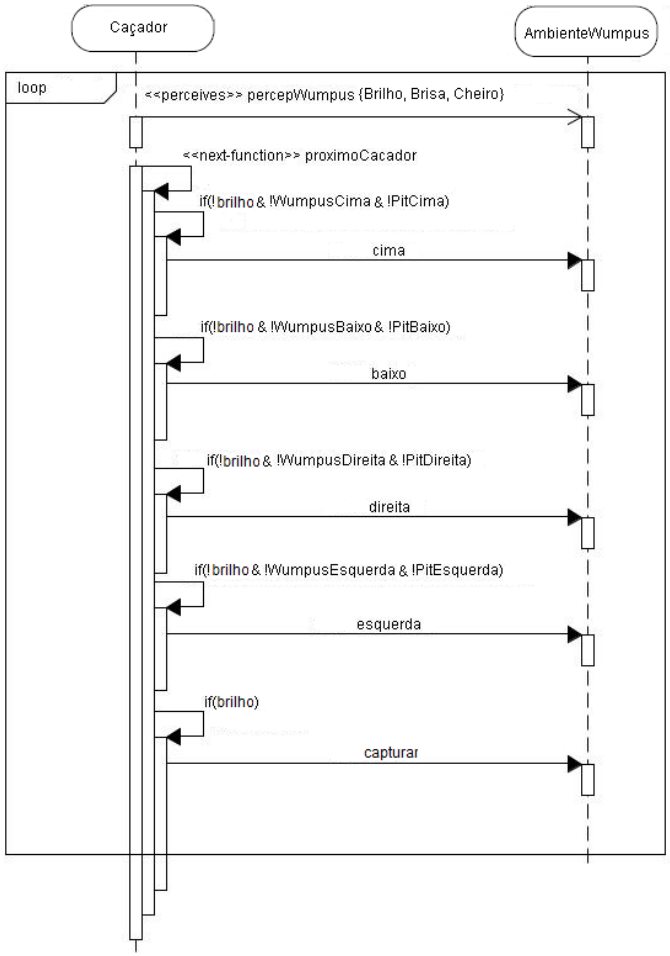

Figura 50 - Agente caçador no diagrama de sequência de MAS-ML 2.0.

A Figura 51 apresenta a modelagem do agente caçador no diagrama de atividades, utilizando a notação de MAS-ML 2.0.

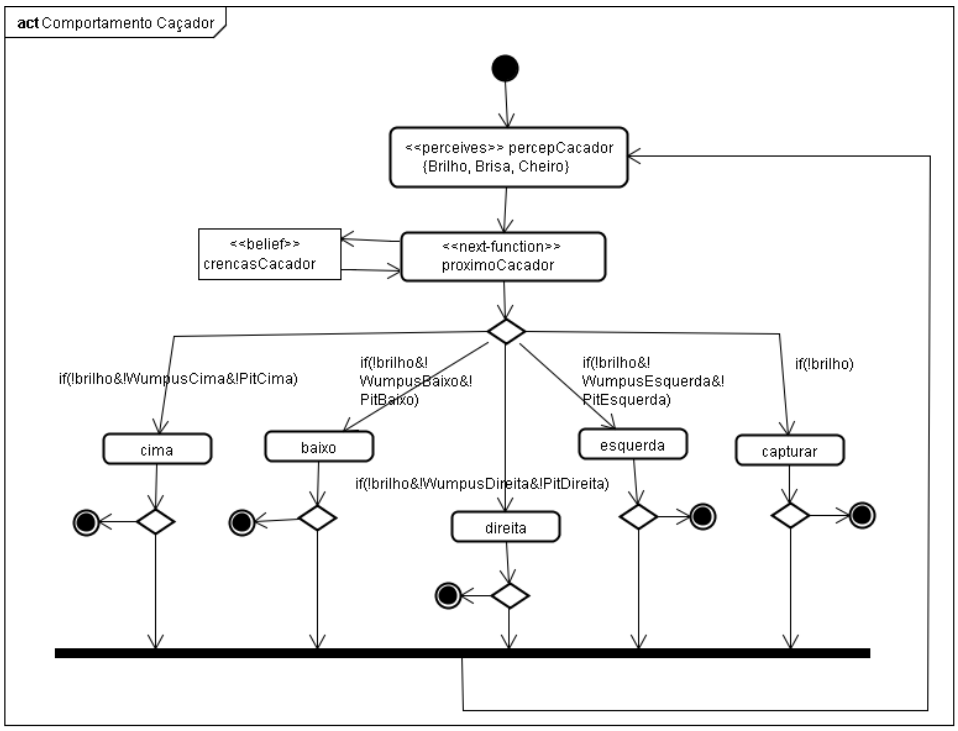

Figura 51 - Agente caçador no diagrama de atividades de MAS-ML 2.0.

Como os agentes reativos baseados em conhecimento não possuem um plano explícito, podemos concluir que MAS-ML 2.0, diferentemente de MAS-ML, representa corretamente agentes reativos. Analogamente, os agentes baseados em objetivo e agentes baseados em utilidade conseguem ser modelados corretamente em MAS-ML 2.0, o que não era possível com MAS-ML.

Em relação ao comportamento dos agentes que não possuem um plano pré-definido, não é possível determinar a sequência de ações durante a modelagem, uma vez que esta sequência é gerada durante a execução do agente. Dadas estas características as figuras 50 e 51 representam o comportamento do agente Caçador de maneira mais adequada, uma vez que o agente poderá combinar suas ações dependendo da percepção do ambiente e de suas crenças.

6.2. Estudo de caso 2: O Mundo de Aspirador de pó

O mundo do aspirador de pó é tão simples que podemos descrever tudo o que nele acontece. Este mundo particular possui somente duas localizações: Salas A e B. O agente aspirador de pó percebe em que sala ele está e se há sujeira na sala. Ele pode escolher se movimentar para a esquerda, para a direita, aspirar a sujeira ou não agir [Russell e Norvig 2004]. O comportamento do agente está descrito a seguir:

1. O agente aspirador de pó parte de um estado interno inicial i_0.
2. Ele observa o estado do ambiente s, e gera uma percepção *ver(s)*.
3. Seu estado interno é então atualizado por meio da função próximo.
4. Em seguida, o agente aspirador de pó seleciona uma ação por meio de regras condição-ação.
5. A ação a ser executada é então enviada ao ambiente.
6. O agente inicia outro ciclo, percebendo o mundo por meio de *ver*, adaptando seu estado por meio de próximo, e escolhendo uma ação para ser executada.

Na Figura 52 é apresentado o conjunto de estados possíveis para o aspirador de pó e as ações que podem ser executadas para cada caso.

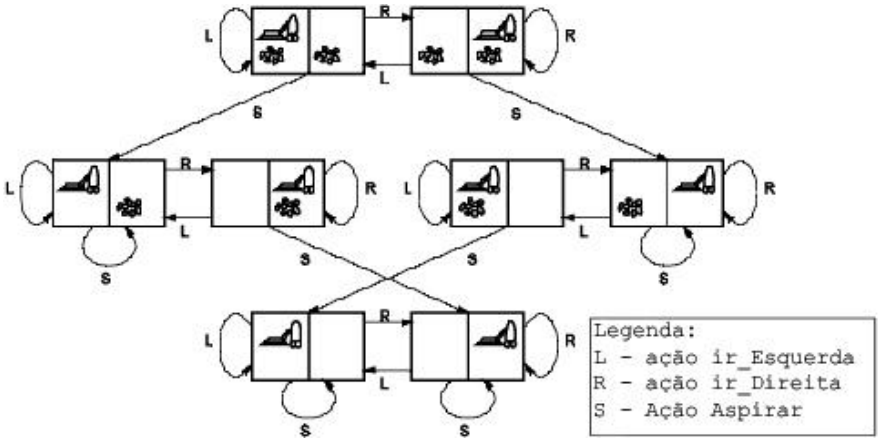

Figura 52 – Espaço de estados e ações possíveis para o aspirador de pó.

6.2.1. Modelagem do mundo do aspirador de pó usando MAS-ML 2.0

O SMA proposto é composto pelo agente aspirador de pó que possui arquitetura interna reativa baseada em conhecimento, e é encarregado de perceber o ambiente e agir. O agente aspirador é auxiliado pelo agente intermediário, que se encontra em contato direto com o ambiente, observando seu estado e recebendo mensagens do agente aspirador para executar ou não ações junto ao ambiente. De acordo com estas características, o agente intermediário é modelado através de uma arquitetura interna reativa simples.

Na Figura 53 é apresentado o diagrama de classes para o SMA do mundo aspirador de pó utilizando os conceitos da extensão propostos neste trabalho. Podemos observar que os agentes modelados não possuem planos e objetivos explícitos, pois foram modelados utilizando a arquitetura dos agentes reativos simples e agentes reativos baseados em conhecimento.

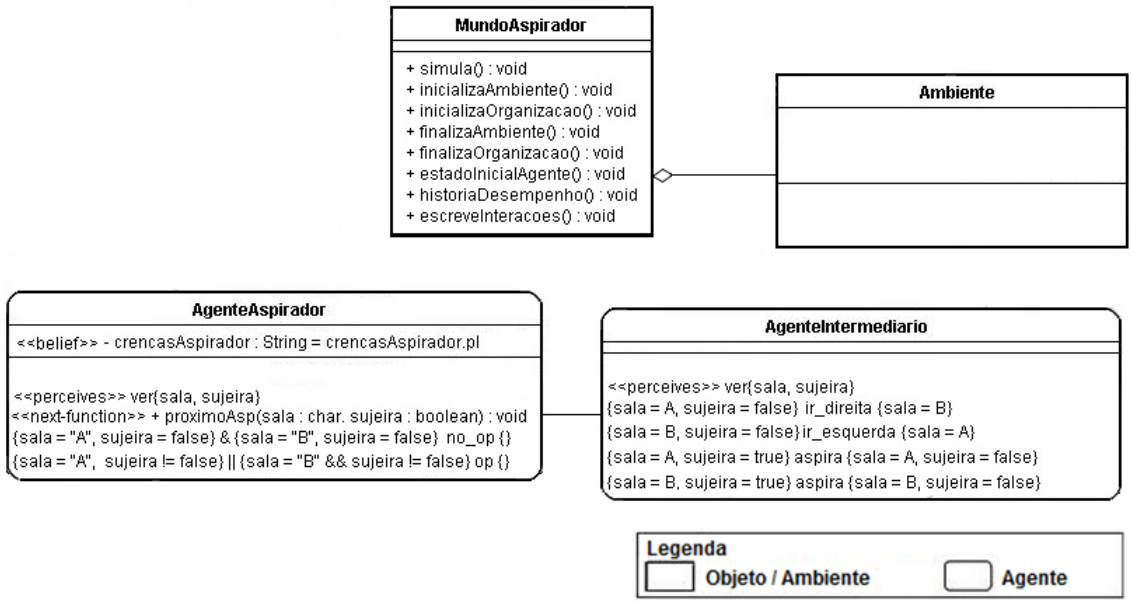

Figura 53 – Diagrama de classes para o sistema multi-agente aspirador de pó.

Na Figura 54 é apresentado o diagrama de papéis para o SMA do mundo do aspirador de pó. Na Figura 55 é apresentado o diagrama de organização, no qual podemos observar a representação do agente aspirador como um agente reativo baseado em conhecimento, e o agente intermediário como um agente reativo simples dentro da organização chamada de OrganizacaoAspirador.

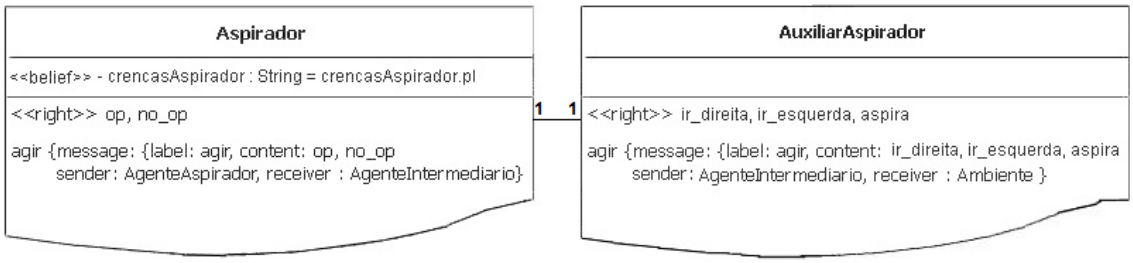

Figura 54 – Diagrama de Papéis de MAS-ML para o SMA aspirador de pó.

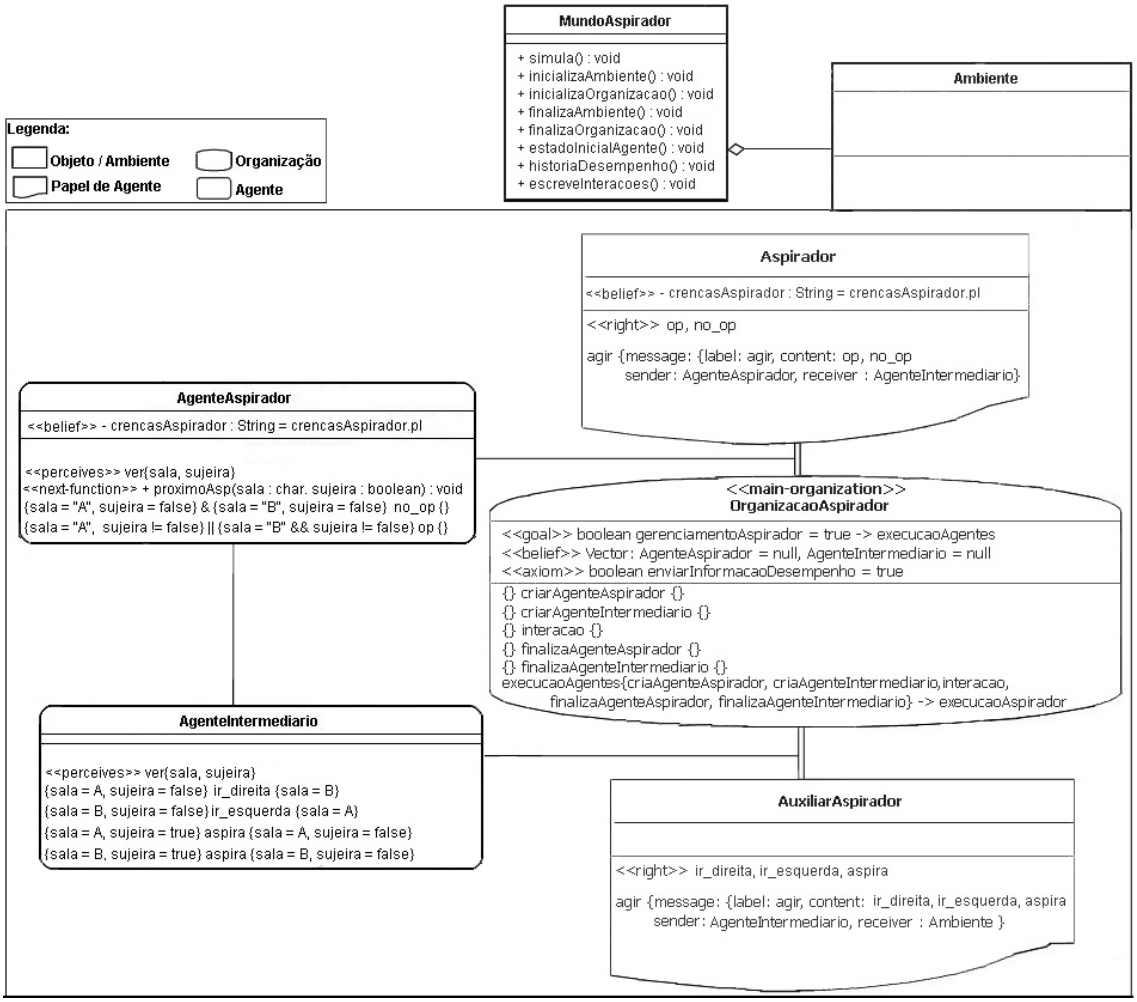

Figura 55 – Diagrama de Organização para o SMA aspirador de pó.

O diagrama de sequência para o mundo do aspirador de pó é apresentado na Figura 56. Podemos perceber que as ações tomadas pelo agente aspirador de pó (que é reativo baseado em conhecimento) são regidas pela função *próximo* e pelas regras *condição-ação*. Além disto, suas ações são representadas através de um conjunto de ações possíveis (no_op e op) e cada ação possui uma mensagem associada à ação tomada.

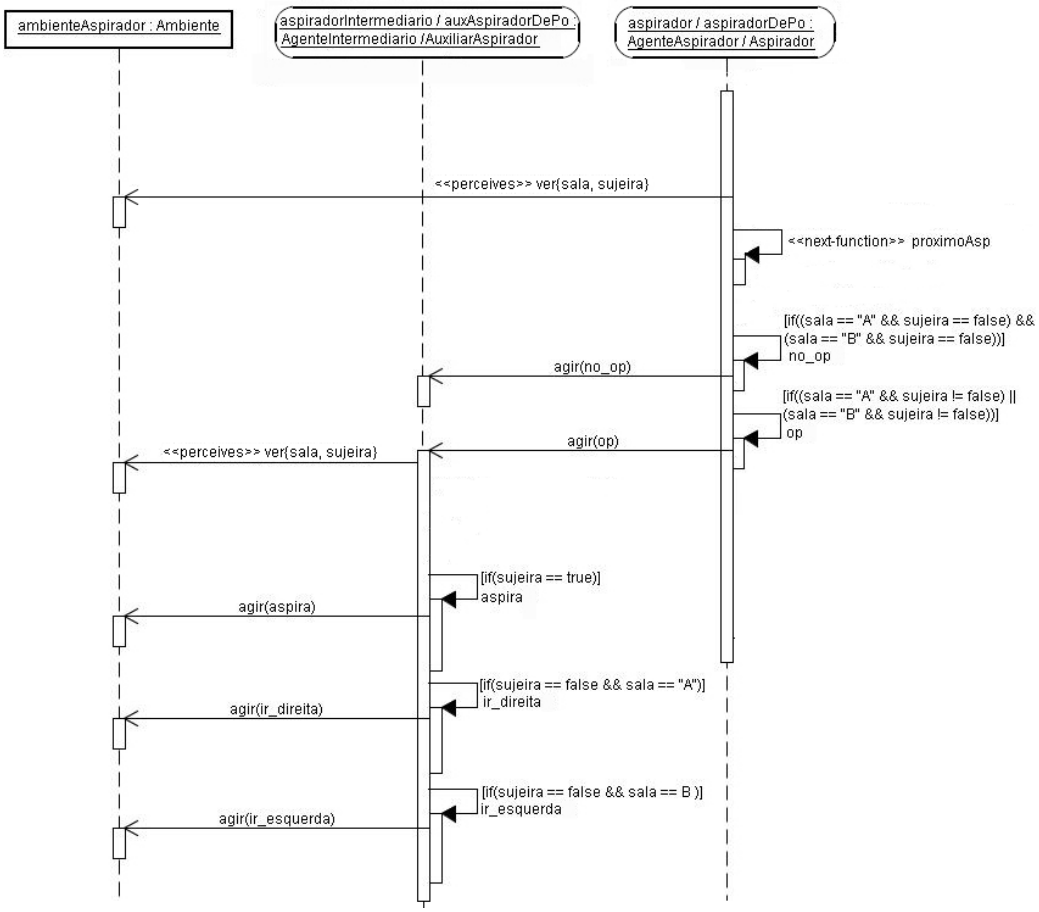

Figura 56 – Diagrama de Sequência para o SMA aspirador de pó.

O agente intermediário foi modelado como um agente reativo simples, portanto possui as regras condição-ação que são responsáveis por selecionar a ação que deverá ser executada no ambiente. O agente intermediário leva em consideração a percepção das ações do agente aspirador para agir ou não agir. As ações são representadas através de um conjunto de ações possíveis (ir_direita, ir_esquerda, aspira), onde cada ação possível possui uma mensagem associada.

As figuras 57 e 58 ilustram o diagrama de atividades para os agentes AgenteIntermediário e AgenteAspirador, respectivamente.

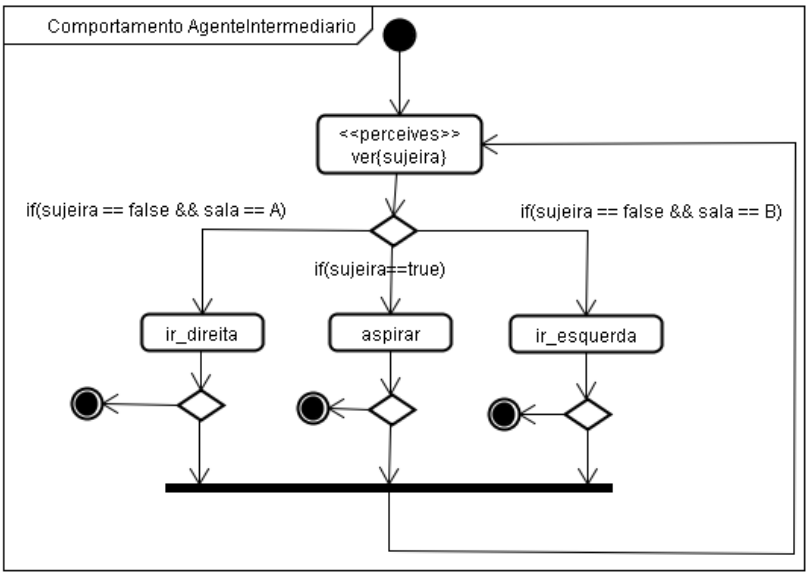

Figura 57 – Diagrama de Atividades para o AgenteIntermediário.

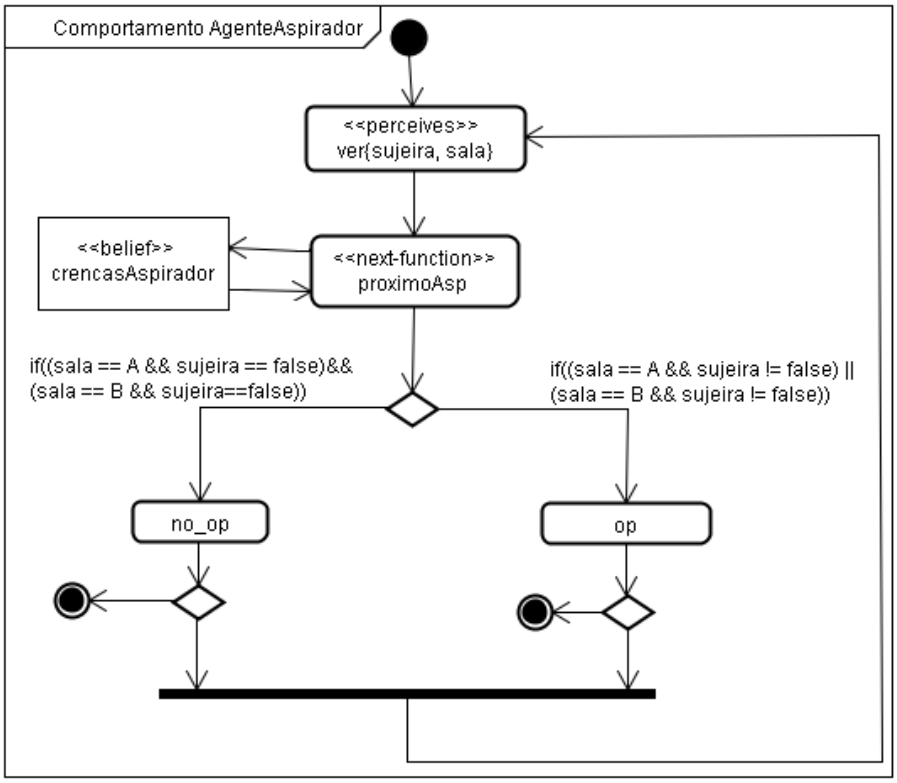

Figura 58 – Diagrama de Atividades para o AgenteAspirador.

6.3. Estudo de caso 3: TAC-SCM

TAC (*Trading Agent Competion*)[6] é um ambiente que possibilita a realização de leilões simultâneos, para testar técnicas, algoritmos e heurísticas a serem utilizadas em agentes de negociação. Há dois tipos de jogos na competição, TAC-Classic e TAC-

[6] Disponível em http://www.sics.se/tac.

SCM, este estudo de caso concentra-se na modelagem de um SMA para o TAC-SCM [Sadeh et al. 2003].

O TAC-SCM (*Supply Chain Management*, ou Gerenciamento de Cadeia de Fornecimento) tem como objetivo planejar e coordenar as atividades das organizações através da cadeia de fornecimento, desde a aquisição de matéria prima até a aquisição de bens de consumo finalizados. Cadeias de Suprimento são ambientes altamente dinâmicos, estocásticos e estratégicos [Arunachalam 2004].

O TAC-SCM foi projetado para capturar os desafios presentes em um ambiente integrado de aquisição de matéria-prima, produção de bens e oferta para clientes. O jogo descreve o cenário de uma cadeia de suprimentos para a montagem de computadores pessoais. Este cenário consiste de uma fábrica de computadores pessoais, fornecedores que provêem componentes para a montagem destes computadores e clientes que demandam computadores prontos [Sadeh et al. 2003]. O jogo envolve uma sequência de dias simulados (onde cada dia corresponde geralmente a 15 segundos), ou rodadas em que os agentes precisam realizar tarefas para gerenciarem a cadeia de suprimento. Os agentes têm uma conta no banco com saldo inicial igual a zero.

A cada dia, clientes lançam pedidos de orçamentos e selecionam os orçamentos submetidos pelos agentes com base na data de entrega e no preço de oferta. Os agentes são limitados pela capacidade de produção de suas linhas de montagem, e têm que obter componentes de um conjunto de oito fornecedores. Quatro tipos de componentes são requeridos para a construção de um computador pessoal: CPU, placa-mãe, memória principal e disco rígido. Cada tipo de componente está disponível em múltiplas versões. A demanda de clientes vem na forma de pedidos de orçamento para diferentes tipos de computadores pessoais, cada pedido requer possivelmente uma combinação diferente de componentes. O jogo começa quando um ou mais agentes se conectam a um servidor de jogo. O servidor simula os fornecedores e clientes, as transações bancarias, produção e serviço de estocagem de mercadoria para agentes individuais. O jogo ocorre sobre um número fixo de dias simulados e, no final, o agente com maior arrecadação em dinheiro no banco é declarado vencedor [Collins et al. 2006]. A Figura 59 ilustra o funcionamento do jogo TAC-SCM.

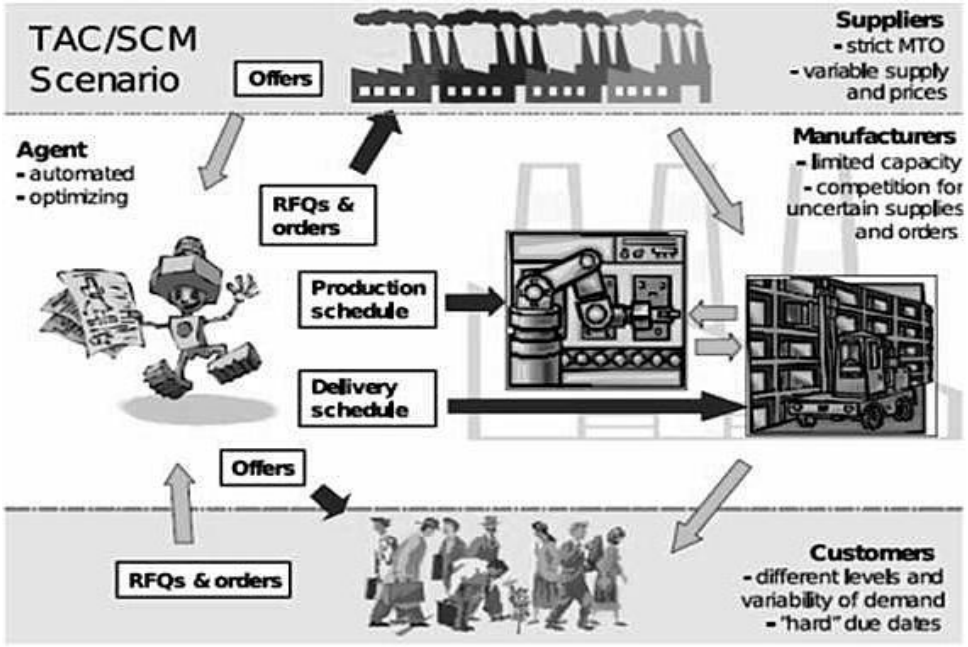

Figura 59 – Ilustração do cenário TAC-SCM [Collins et al. 2006].

6.3.1. Modelagem do SMA para TAC-SCM com MAS-ML 2.0

A modelagem do SMA para a problemática descrita é composta pelos seguintes agentes:

- **AgenteVendendor**: Modelado como um agente reativo simples, ele é capaz de oferecer computadores aos consumidores, recebe o pagamento e solicitar a montagem e entrega dos produtos.
- **AgenteComprador**: Modelado como um agente reativo baseado em conhecimento. Este agente decide quando fazer uma nova requisição de peças e realiza o pagamento das peças.
- **AgenteProdução**: É um agente baseado em objetivo guiado por planejamento. Este agente é responsável por gerenciar o estoque e montar os computadores de acordo com a demanda.
- **AgenteEntregador**: Agente baseado em objetivo guiado por plano. Este agente precisa seguir um conjunto de ações para entregar um produto ao cliente.
- **AgenteGerente**: Modelado como um agente baseado em utilidade, que é responsável por encontrar uma melhor maneira de alocação dos recursos da produção frente a demanda corrente, com o objetivo de maximizar o lucro e as vendas.

A arquitetura interna de cada agente foi definida levando em consideração a função especifica que cada um desempenha no SMA.

Os agentes Vendedor e Comprador são modelados como agentes reativos devido à necessidade de resposta rápida aos leilões. De acordo com Weiss (1999), agentes reativos respondem mais rapidamente às percepções do que os agentes pró-ativos. As figuras 60 e 61 ilustram os agentes reativos vendedor e comprador, respectivamente, nos diagramas estáticos de MAS-ML 2.0.

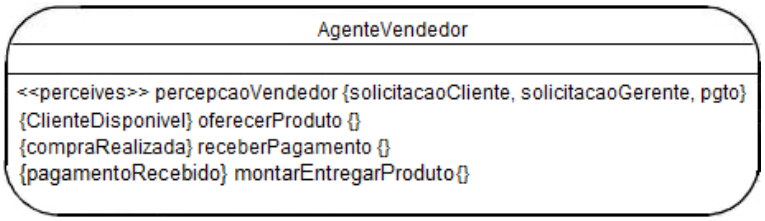

Figura 60 - AgenteVendedor proposto para o SMA do TAC-SCM

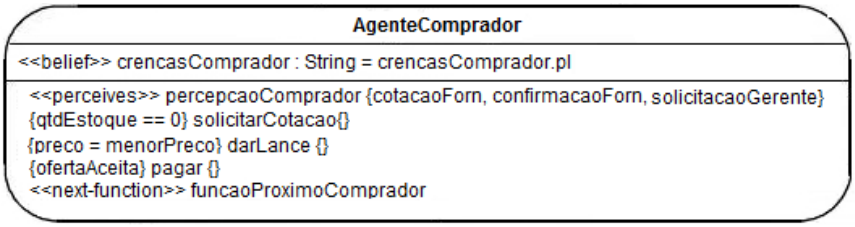

Figura 61 - AgenteComprador proposto para o SMA do TAC-SCM

O agente Produção necessita satisfazer a demanda atual através da produção de computadores. Para atingir este objetivo, ele não pode utilizar um plano pré-estabelecido, porque este cenário requer um conjunto de ações diferentes, dependendo da demanda atual. Assim, o agente Produção pode ser modelado de acordo com a arquitetura interna baseada em objetivo com planejamento. A Figura 62 ilustra a representação do AgenteProdução nos diagramas estáticos de MAS-ML 2.0.

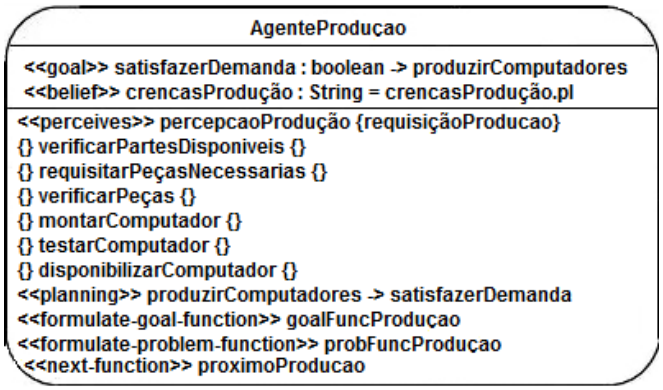

Figura 62 - Agente Produção proposto para o SMA do TAC-SCM

O agente Entregador é responsável por entregar os produtos aos clientes. Para atingir este objetivo, uma sequência de ações (plano) deve ser executada. A seguir a representação do agente Entregador nos diagramas estáticos de MAS-ML 2.0, modelado como um agente baseado em objetivo guiado por plano (Figura 63).

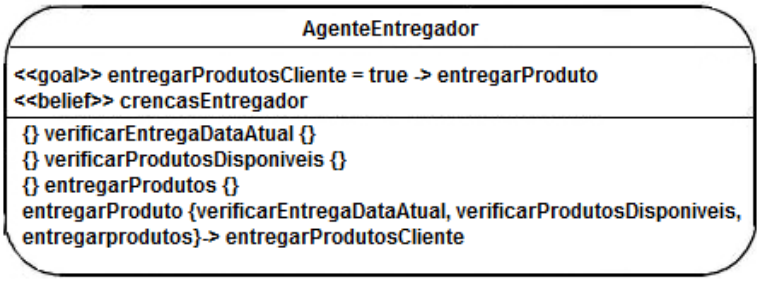

Figura 63 – AgenteEntregador proposto para o SMA do TAC-SCM

Finalmente, o agente Gerente é responsável por gerenciar todos os agentes do SMA proposto para o jogo. Este agente tenta maximizar o lucro e as vendas. Observe que seus objetivos podem estar em conflito: ao tentar maximizar o lucro, existe a possibilidade de o custo do produto sofrer aumento. Consequentemente, as vendas podem ser reduzidas. Analogamente, uma alternativa para tentar maximizar as vendas é reduzir o valor do produto e possivelmente o lucro.

Neste contexto, a arquitetura mais apropriada para o agente é a arquitetura baseada em utilidade. A Figura 64 mostra a modelagem para o agente Gerente nos diagramas estáticos de MAS-ML 2.0.

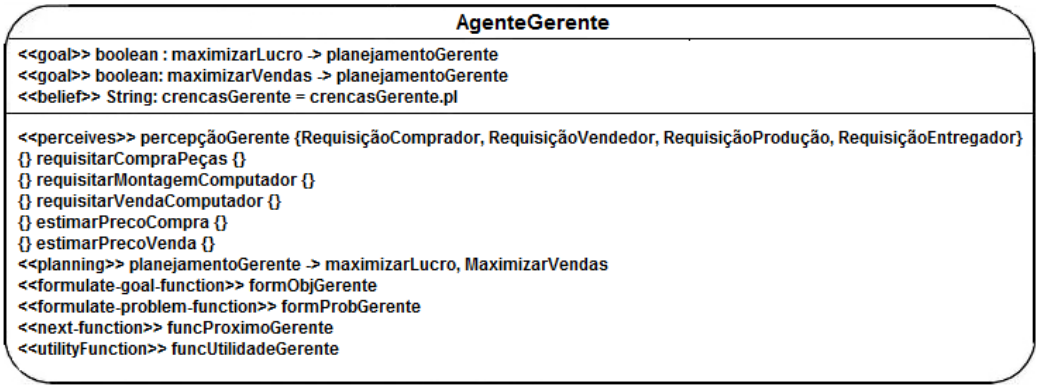

Figura 64 – AgenteGerente proposto para o SMA do TAC-SCM

Um agente, por sua vez, pode desempenhar um papel, com o intuito de orientar e restringir seu comportamento no contexto de uma organização. Deste modo, o AgenteVendedor desempenha o papel de Vendedor na organizaçãoTAC e o AgenteComprado desempenha o papel de Comprador na organizaçãoTAC.

Os papéis dos agentes reativos: AgenteVendedor e AgenteComprador são ilustrados nas figuras 65 e 66, respectivamente. Observe que estes papéis não possuem objetivos e, no caso do AgenteVendedor, também não há representação das crenças, conforme proposto na Seção 4.3.5.

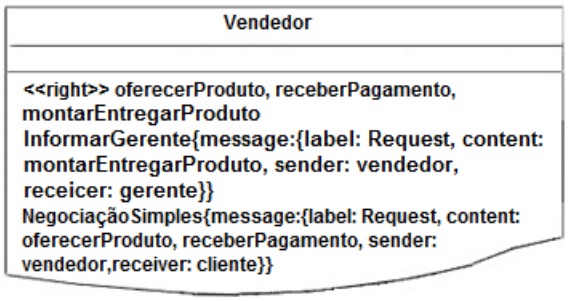

Figura 65 – Papel do AgenteVendedor proposto para o SMA do TAC-SCM.

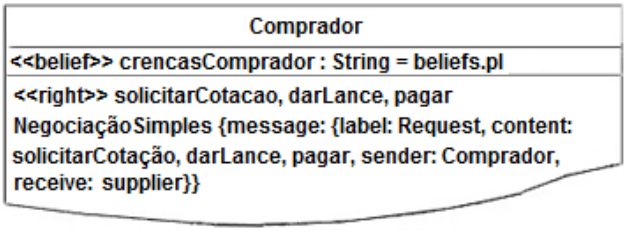

Figura 66 – Papel do AgenteComprador proposto para o SMA do TAC-SCM.

A representação de papeis para os agentes AgenteProdução, AgenteEntregador e AgenteGerente é ilustrada nas figuras 67, 68 e 69, respectivamente. Note que a extensão proposta não teve impacto na representação dos papéis para agentes pró-ativos, portanto a modelagem segue a definição original de MAS-ML.

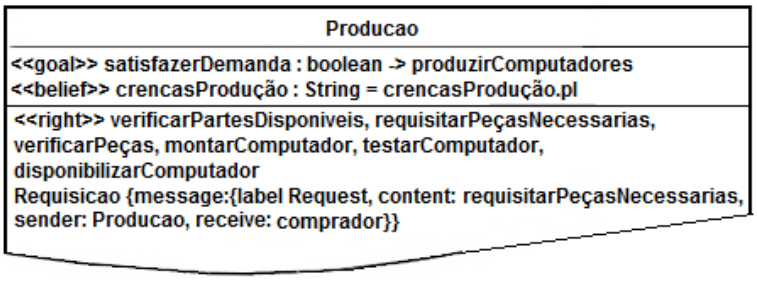

Figura 67 – Papel do AgenteProducao proposto para o SMA do TAC-SCM.

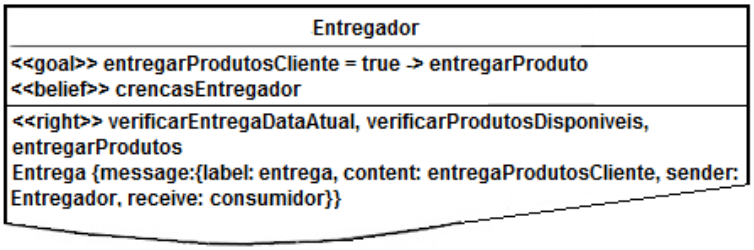

Figura 68 – Papel do AgenteEntregador proposto para o SMA do TAC-SCM.

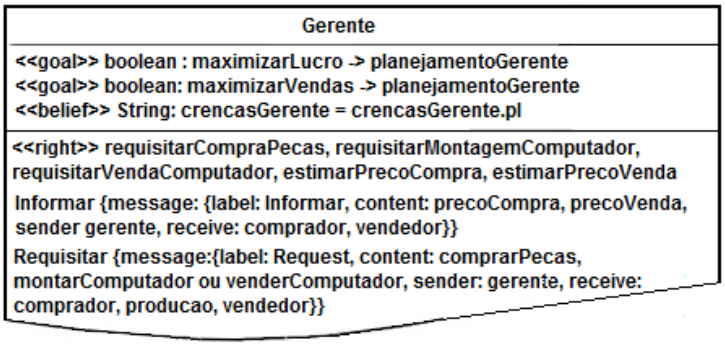

Figura 69 – Papel do AgenteGerente proposto para o SMA do TAC-SCM.

Na Figura 70 é ilustrado o diagrama de classes simplificado do SMA proposto para TAC-SCM. A estrutura interna dos agentes é omitida, tendo em vista que a representação detalhada dos mesmos já foi apresentada nesta seção. O ambiente também está representado de maneira simplificada.

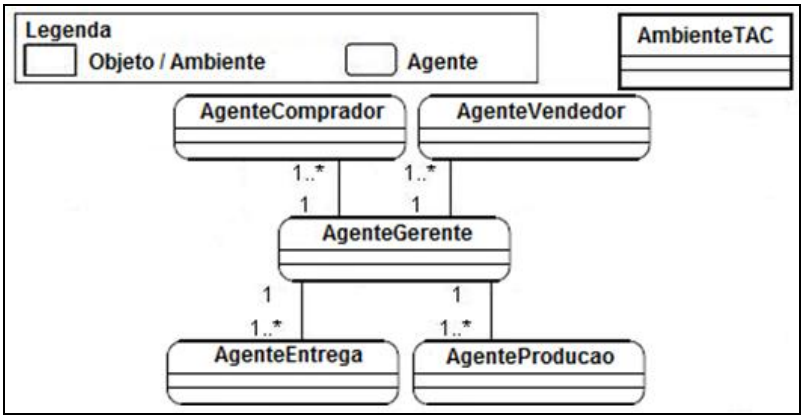

Figura 70 – Diagrama de classes proposto para o SMA do TAC-SCM.

Da mesma forma, nas figuras 71 e 72 são apresentados os diagramas de organização e de papéis simplificados, respectivamente.

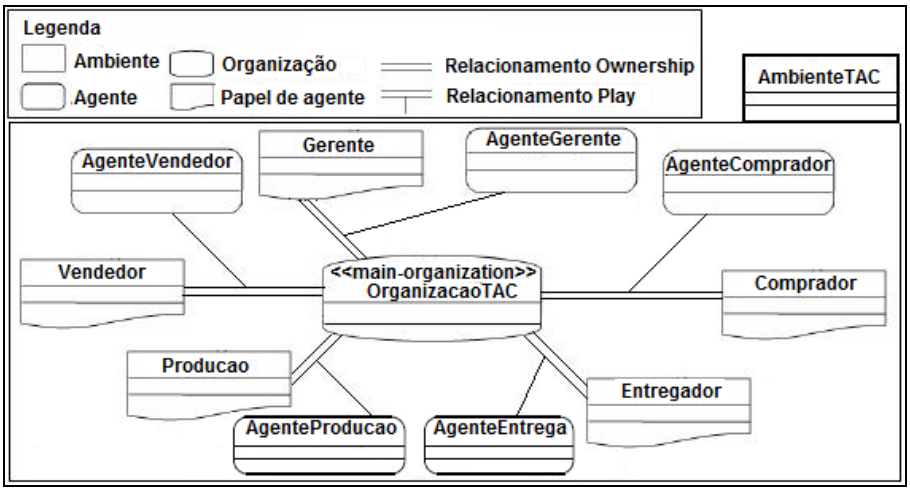

Figura 71 – Diagrama de organização proposto para o SMA do TAC-SCM.

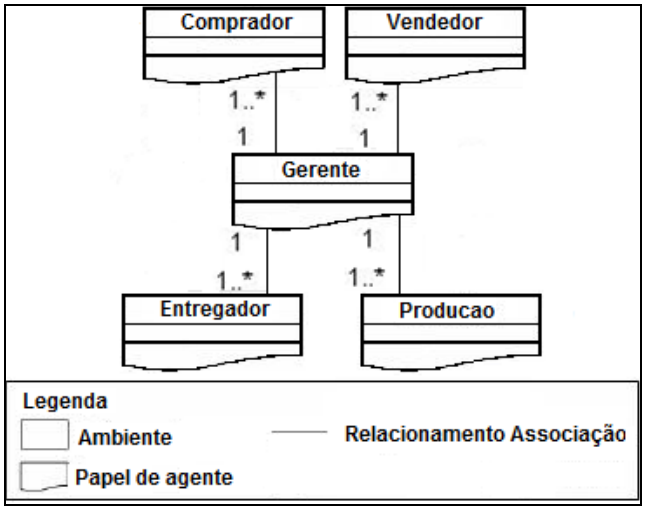

Figura 72 – Diagrama de papéis proposto para o SMA do TAC-SCM.

Na Figura 73, o diagrama de sequência do AgenteVendedor é apresentado. Este diagrama demonstra a execução do agente vendedor através de suas percepções e de um conjunto de ações possíveis associadas a uma regra condição-ação.

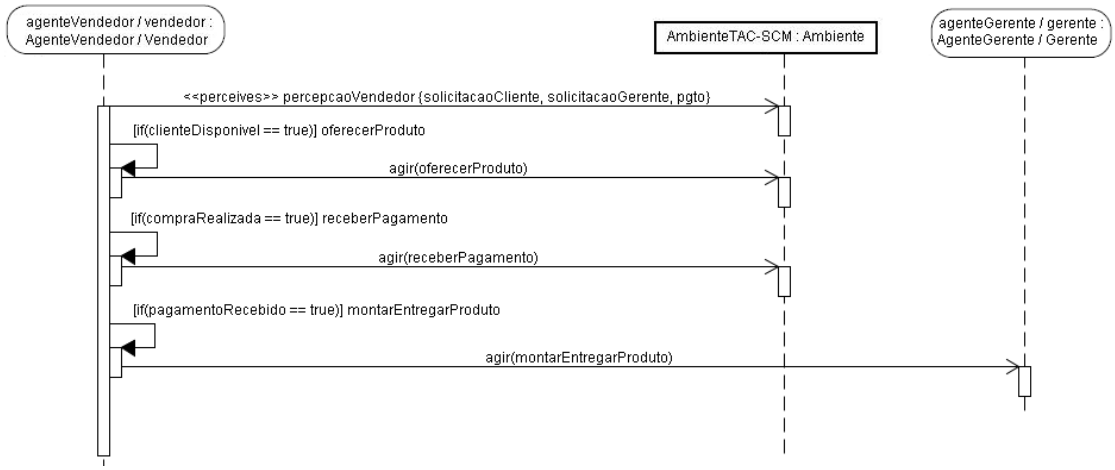

Figura 73 – Diagrama de sequência do AgenteVendedor.

Na Figura 74 o diagrama de sequência do AgenteComprador é representado através de suas percepções, da execução da função próximo e de um conjunto de ações possíveis associadas a uma regra condição-ação.

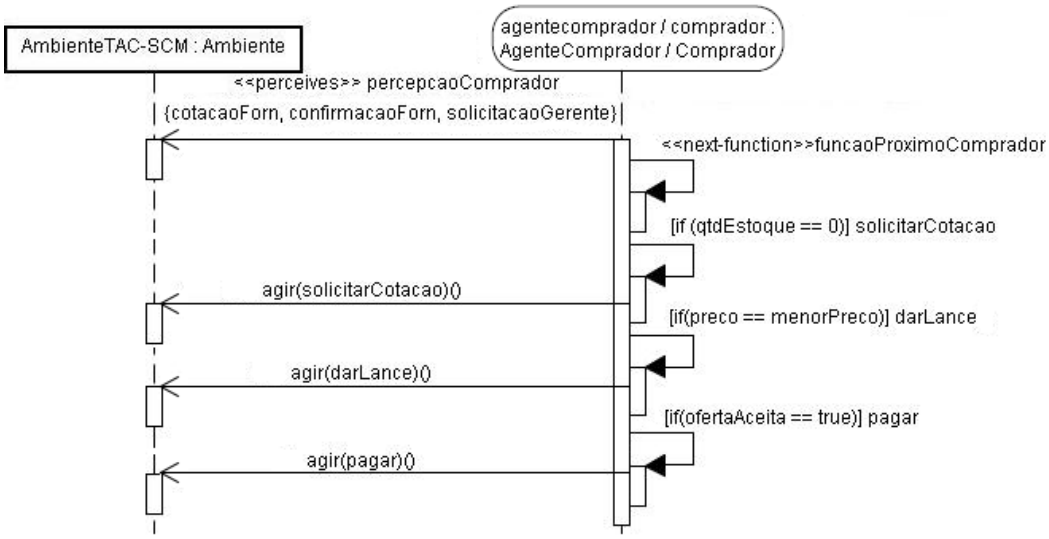

Figura 74 – Diagrama de sequência do AgenteComprador.

Na Figura 75 é ilustrado o diagrama de sequência para a execução do AgenteProducao. Note que as ações tomadas pelo AgenteProducao são regidas pela percepção, função próximo, função de formulação de objetivo, função de formulação de problema e planejamento. Além disto, suas ações são representadas através de um conjunto de ações possíveis.

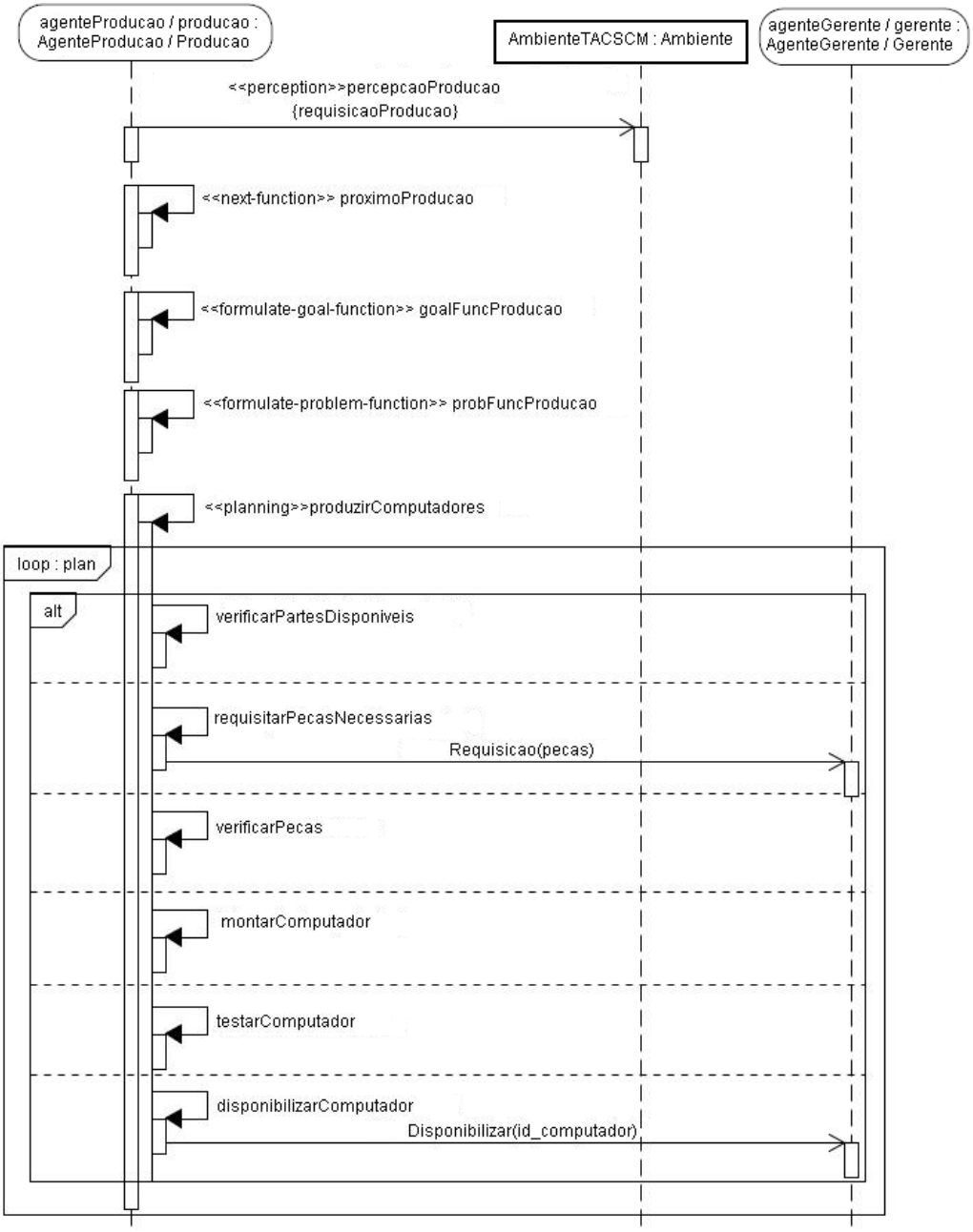

Figura 75 – Diagrama de sequência do AgenteProducao.

A Figura 76 descreve o diagrama de sequência para a execução do AgenteEntregador. Observe que as ações tomadas pelo AgenteEntregador são regidas por um plano, portanto trata-se de uma sequência de ações.

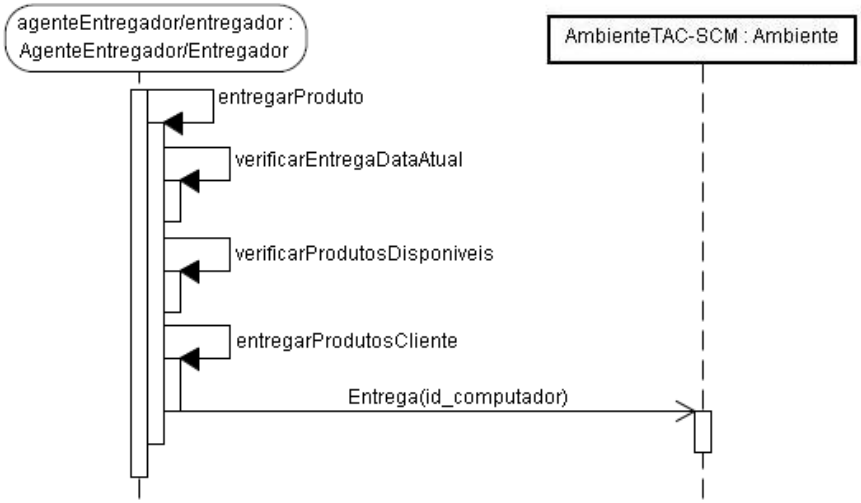

Figura 76 – Diagrama de sequência do AgenteEntregador.

Na Figura 77 apresentamos o diagrama de sequência para a execução do AgenteGerente. Neste caso, as ações tomadas pelo AgenteGerente são regidas pela percepção, função próximo, função de formulação de objetivo, função de formulação de problema, planejamento e função utilidade. Além disso, suas ações são representadas através de um conjunto de ações possíveis.

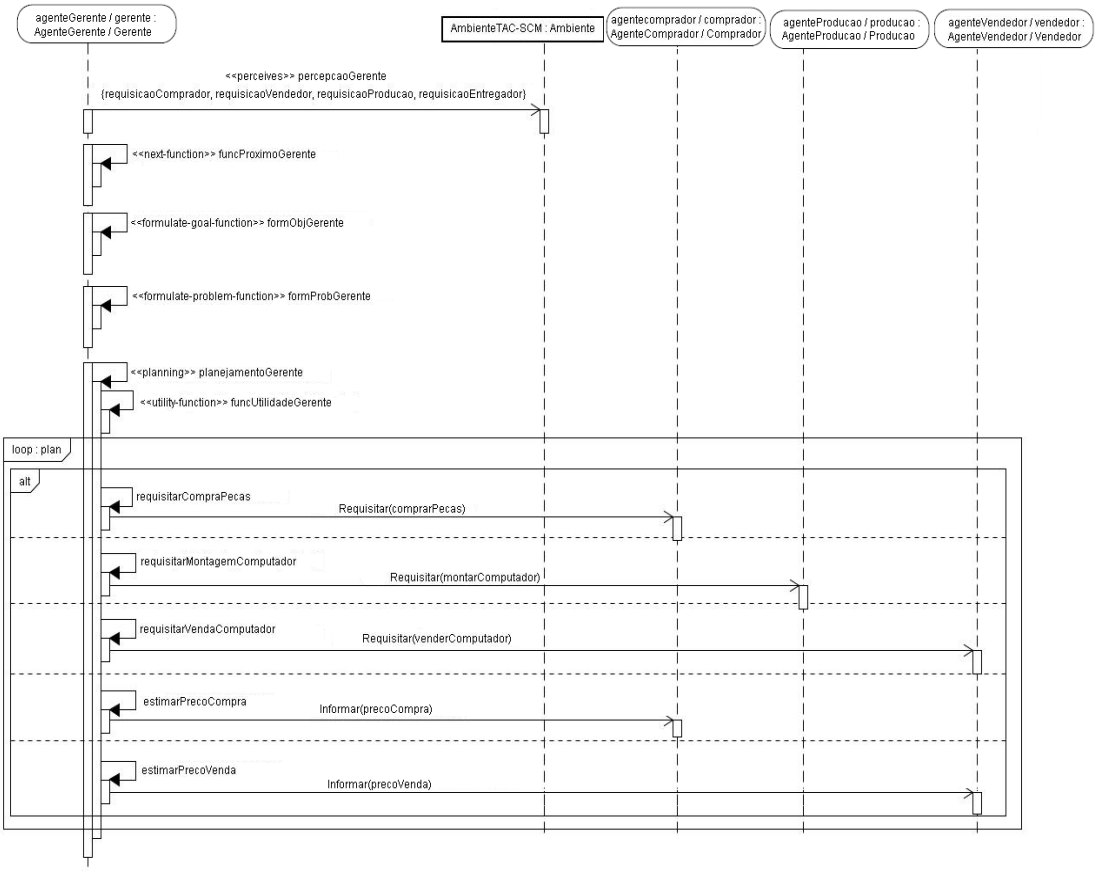

Figura 77 – Diagrama de sequência do AgenteGerente.

O diagrama de atividades descreve o comportamento de cada agente, os diagramas de atividades para os agentes: Vendedor, Comprador, Produção, Entregador e Gerente são apresentados nas figuras 78, 79, 80, 81 e 82, respectivamente.

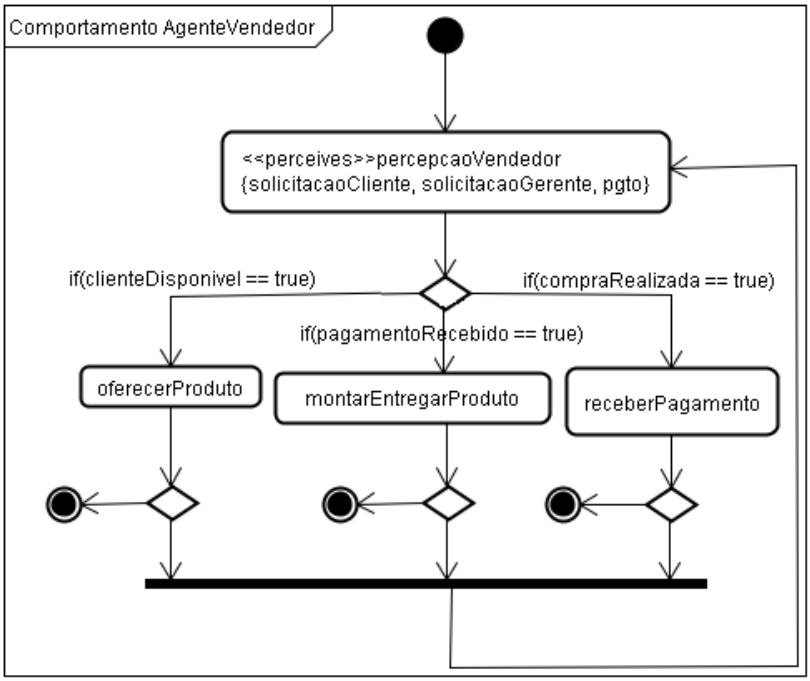

Figura 78 – Diagrama de atividades do AgenteVendedor.

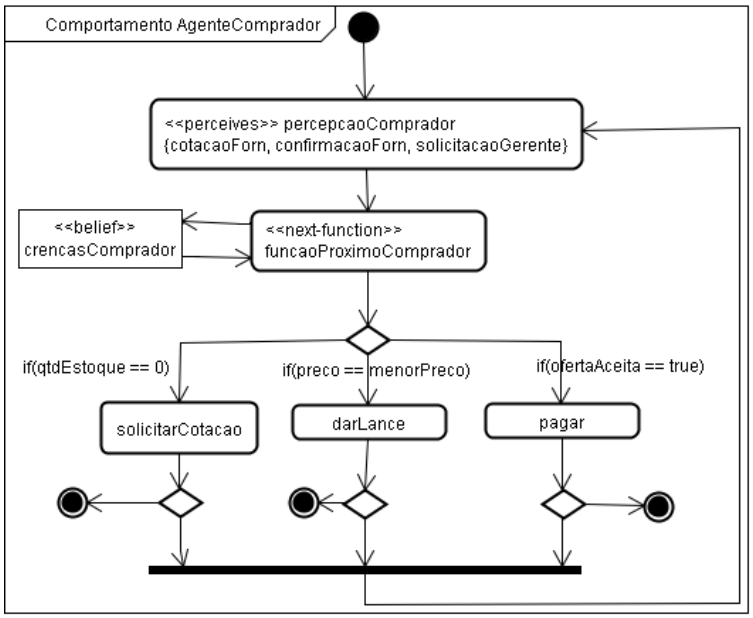

Figura 79 – Diagrama de atividades do AgenteComprador.

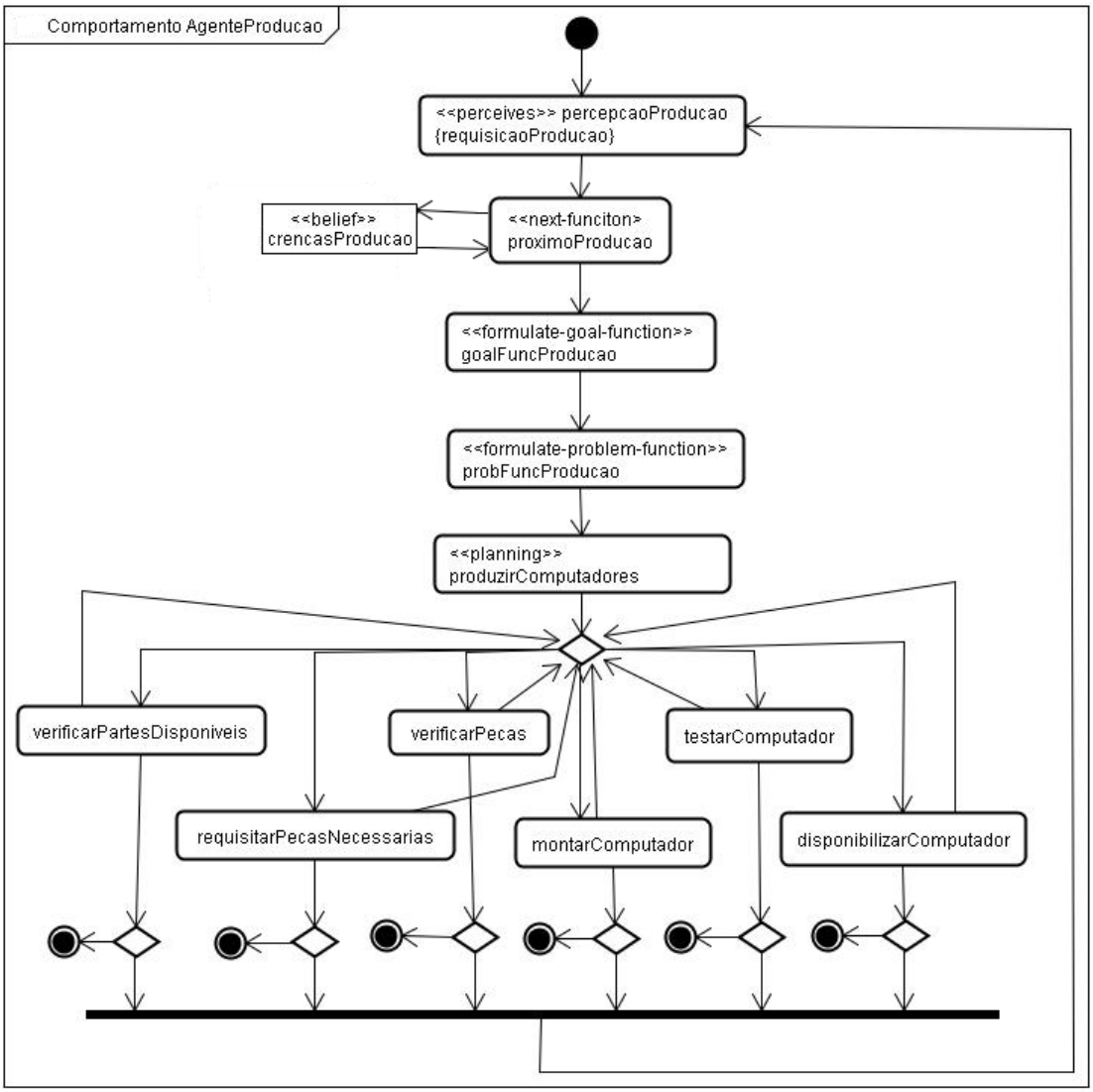

Figura 80 – Diagrama de atividades do AgenteProducao.

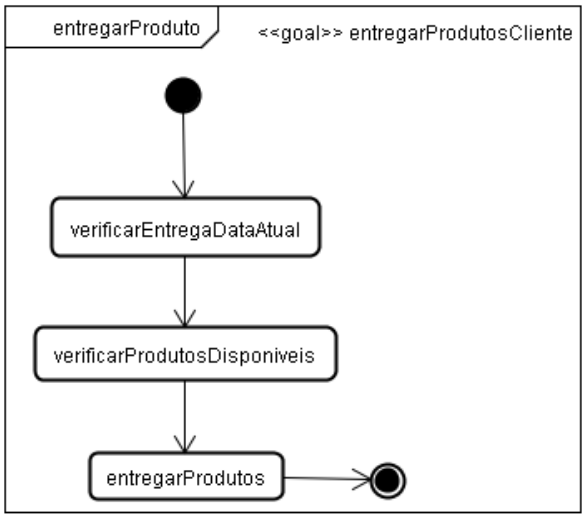

Figura 81 – Diagrama de atividades do AgenteEntregador.

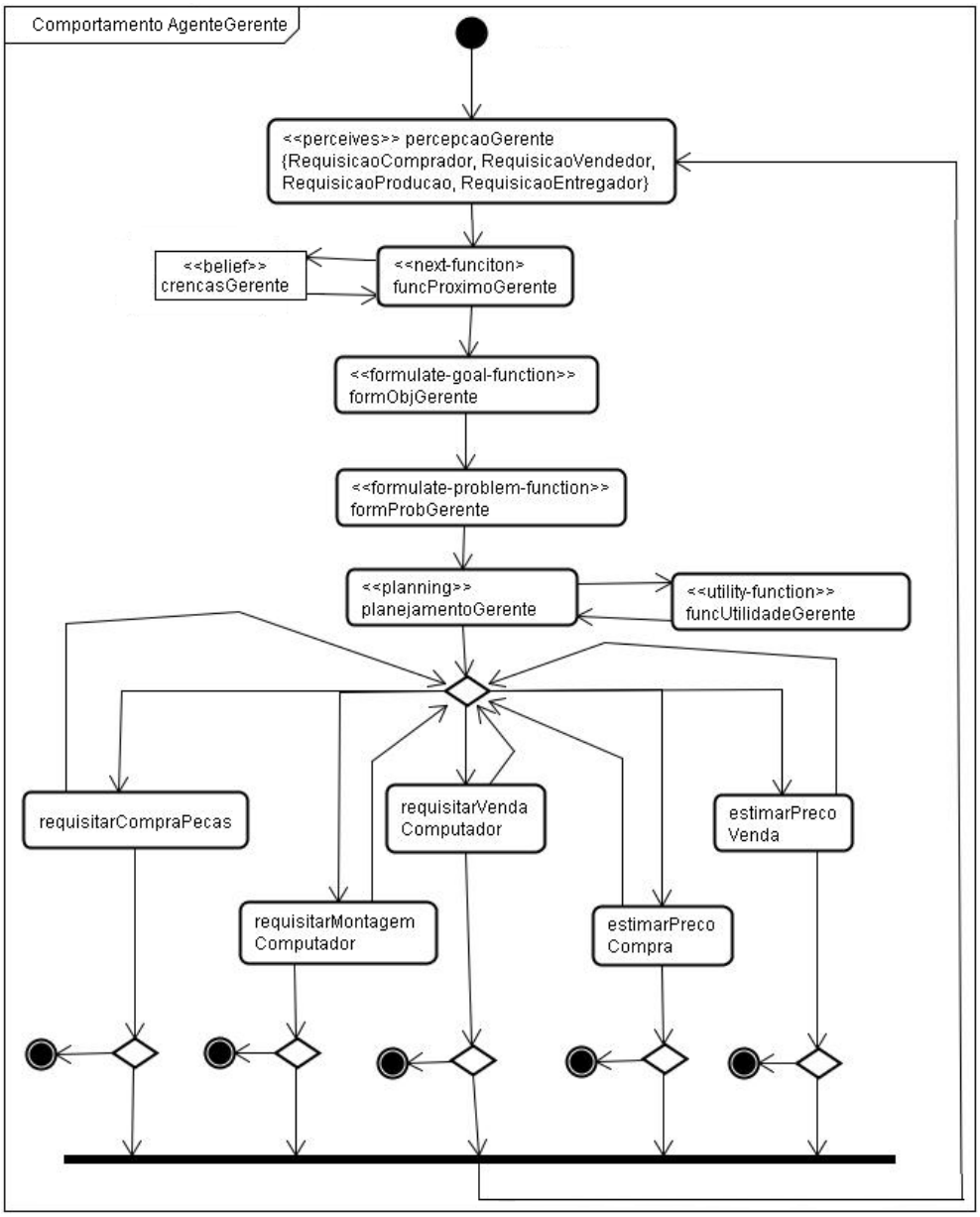

Figura 82 – Diagrama de atividades do AgenteGerente.

Através do estudo de caso desenvolvido para TAC-SCM, a necessidade de modelar agentes com diferentes arquiteturas em um único sistema fica evidenciada. Neste contexto, MAS-ML 2.0 foi aplicada na resolução de um problema real, demonstrando a versatilidade da extensão realizada para modelar SMAs com agentes de diferentes arquiteturas.

7. Considerações Finais

Nesta dissertação foram apresentadas as principais arquiteturas internas de agentes de software com o intuito de determinar as suas características, propriedades, atributos, componentes mentais e comportamento, de forma a especificar os elementos de projeto necessários para dar apoio à modelagem de SMAs envolvendo agentes com diversas arquiteturas.

Os estudos realizados apontaram a linguagem de modelagem MAS-ML como sendo a mais adequada a ser utilizada como ponto de partida para a modelagem das diversas arquiteturas internas, por ela contemplar as seguintes características: (i) modelar agentes baseados em objetivo com plano, (ii) prover uma extensão conservativa à linguagem UML, (iii) possuir um *framework* conceitual, (iv) oferecer suporte a objetos convencionais, (v) identificar papéis e (vi) modelar adequadamente ambientes e a interação entre agente e ambiente.

Com base na especificação das diversas arquiteturas, foi apresentada uma extensão do *framework* conceitual TAO e da linguagem de modelagem MAS-ML para possibilitar a modelagem das arquiteturas internas de agentes reativos, pró-ativos baseados em objetivo e pró-ativos baseados em utilidade, de forma completa e consistente.

MAS-ML foi originalmente projetada para dar suporte à modelagem de agentes pró-ativos orientados por objetivos baseados em plano pré-definido, no entanto, conceber agentes pró-ativos baseados em plano pré-definido em ambientes estocásticos e parcialmente observáveis ou dinâmicos pode ser uma tarefa bastante complexa e/ou produzir efeitos indesejados. A partir da necessidade de modelar agentes com arquitetura interna reativa, pró-ativa baseada em objetivo guiado por planejamento e pró-ativa baseada em utilidade, a linguagem MAS-ML apresentou algumas deficiências.

A evolução de MAS-ML para modelar as arquiteturas internas de agentes envolveu inicialmente a criação das metaclasses AgentPerceptionFunction e AgentPlanningStrategy e a criação de diversos estereótipos: <<next-function>>, <<formulate-goal-function>>, <<formulate-problem-function>> e <<utility-function>> associados a metaclasse AgentAction; <<perceives>> associado a AgentPerceptionFunction e <<planning>> associado a AgentPlanningStrategy. As

entidades de MAS-ML que sofreram alteração foram AgentClass e AgentRoleClass. Os diagramas de classes, organização e papéis foram alterados em consistência com a nova representação gráfica de AgentClass e de AgentRoleClass. O diagrama de papéis foi alterado no que diz respeito aos agentes reativos e, finalmente, os diagramas dinâmicos também foram alterados para contemplar a representação adequada dos novos elementos.

Devido à evolução em MAS-ML, em SMAs onde agentes com arquiteturas diferentes podem estar colaborando, como no estudo de caso do TAC-SCM, o projetista pode especificar nos diagramas de MAS-ML a arquitetura interna a ser utilizada pelo desenvolvedor para cada um dos agentes inteligentes do SMA a ser desenvolvido. Deste modo, o *gap* semântico entre a modelagem e a implementação pode ser reduzido.

Assim sendo, o benefício conseguido através da extensão é notório tanto para a área de Inteligência Artificial (IA) quanto para Engenharia de Software (ES). Do ponto de vista de IA, antes de codificar sistemas para simulação ou para outros fins, os desenvolvedores podem utilizar a linguagem MAS-ML estendida para gerar modelos mais próximos da concepção de agentes utilizada na área, facilitando consequentemente a implementação de tais sistemas.

Do ponto de vista de ES, o presente trabalho contribui especificamente para a modelagem de sistemas complexos, no contexto do paradigma orientado a agentes. A partir do suporte fornecido, o projetista pode gerar modelos mais adequados à arquitetura da solução, a serem utilizados como base para a codificação de SMAs.

Ainda no contexto de ES, propiciar o apoio automatizado às diferentes atividades inerentes ao desenvolvimento e construção de software é uma prática estimulada. A partir desta premissa, o presente trabalho propõe a extensão de uma ferramenta de modelagem existente de forma a implementar os novos conceitos propostos para a modelagem de SMAs utilizando a linguagem MAS-ML 2.0. A ferramenta de modelagem MAS-ML tool, que inicialmente possibilitava a modelagem do diagrama de classes de MAS-ML, foi evoluída para também dar apoio à geração do diagrama de organização, ambos gerados de forma consistente com a extensão proposta neste trabalho. Deste modo, os usuários da linguagem MAS-ML podem utilizar MAS-ML tool para automatizar o processo de modelagem e a validação dos modelos criados,

possibilitando a geração de modelos mais corretos e reduzindo a ocorrência de erros humanos.

7.1. Trabalhos Futuros

Existem alguns trabalhos futuros que poderão ser desenvolvidos com o intuito de dar continuidade ao trabalho apresentado nesta dissertação. Dentre eles podem ser citados:

(i) A possibilidade de extensão de MAS-ML 2.0 para outras arquiteturas internas, como por exemplo a arquitetura BDI.

(ii) A inclusão de aprendizado nas arquiteturas de agentes. A representação de agentes com aprendizado em MAS-ML 2.0 poderia ser realizada através de extensões semelhantes às propostas nesta dissertação.

(iii) A implementação do diagrama de papéis e dos diagramas dinâmicos na ferramenta MAS-ML tool. Para a implementação do diagrama de papéis, a representação do papel de agente no diagrama de organização criado pode ser aproveitada. Em relação aos relacionamentos, se torna necessária a definição do relacionamento de controle.

(iv) A geração de código a partir dos diagramas gerados pela ferramenta MAS-ML tool. Para esta tarefa, uma abordagem baseada em MDA (*Model Driven Architecture*) semelhante à utilizada por de Maria (2005) poderia ser utilizada.

(v) Criação de mecanismos em MAS-ML 2.0 para representar as técnicas de IA e estratégias de agentes que podem ser utilizadas na implementação de agentes. Na fase de modelagem é possível representar, através de notas textuais, a utilização de uma determinada estratégia para o desenvolvimento de uma atividade por parte do agente, e.g. rede neural ou lógica fuzzy. Porém, no momento da geração de código (item iv) pode ser necessário que esta informação seja expressa de forma mais especifica e detalhada. Além disto, existe a possibilidade de um agente com arquitetura interna baseada em objetivo com planejamento trace uma estratégia de execução na qual combine os componentes que compões sua arquitetura interna com um plano pré-definido, por exemplo. Neste caso, pode-se

utilizar um processo de extensão semelhante ao apresentado neste trabalho.

Referências Bibliográficas

Arunachalam, R (2004). The 2003 supply chain management trading agent competition. In: Third International Joint Conference on Autonomous Agents & Multi Agent Systems. July 2004. [S.l.: s.n.]. p. 113–120.

Bellifemine, F. L.; Caire, G.; Greenwood, D. (2007) Developing Multi-Agent Systems with JADE. [S.l.]: Wiley (Wiley Series in Agent Technology).

Bordini, R. H.; Wooldridge, M.; Hübner, J. F. (2007). Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology), John Wiley & Sons.

Castro, J.; Alencar, F.; Silva, C. (2006). Engenharia de Software Orientada a Agentes. In: Karin Breitman; Ricardo Anido. (Org.). Atualizações em Informática. Rio de Janeiro: PUC-Rio, p. 245-282.

Cervenka, R.; Trencansky, I.; Calisti, M.; Greenwood, D. (2005). AML: Agent Modeling Language. Toward Industry-Grade Agent-Based Modeling. In J. Odell, P. Giorgini, and J.P. Muller, editors, Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, pages 31–46. Springer-Verlag, Berlin.

Choren, R.; Lucena, C. (2004). Agent-Oriented Modeling Using ANote, 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2004), 3rd; The Institution of Electrical Engineers, IEE, Stevenage, UK, 2004, pp. 74-80, ISBN: 0-86341-431-1, May 24-25.

Collins, J.; Arunachalam, R.; Sadeh, N.; Eriksson, J.; Finne, N.; Janson, S. (2006). The Supply Chain Management Game for the 2007 Trading Agent Competition. [S.l.].

Czarnecki, K.; Eisenecker, U. (2000). Generative Programming - Methods, Tools, and Applications, Adison-Wesley, June 2000.

Dardenne, A.; Lamsweerde, A.; Fickas, S. (1993). Goal-directed Requirements Acquisition. Science of Computer Programming. v.20, p.3-50.

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

De Maria, B. A. (2005). Usando a Abordagem MDA no Desenvolvimento de Sistemas Multi-Agentes. Dissertação de Mestrado.PUC, Departamento de Informática, Rio de Janeiro.

De Maria, B. A.; Silva, V. T.; Lucena, C.J.P.; Choren, R. (2005). VisualAgent: A Software Development Environment for Multi-Agent Systems. Proceedings of the 19º Simpósio Brasileiro de Engenharia de Software (SBES 2005), Tool Track, Uberlândia, MG, Brazil, October 3-7, 2005.

D'Inverno, M.; Luck, M. (2001). Understanding Agent Systems. New York: Springer.

Dorigo, M.; Stutzle, T. (2004). Ant Colony Optimization. The MIT Press, Cambridge, Massachusetts.

Eclipse Platform (2009). www.eclipse.org, acessado em 15 de Julho de 2009.

EMF (2009), www.eclipse.org/modeling/emf/, Acessado em 15 de Julho de 2009.

Farias, K.; Nunes, I.; Silva, V. T. da; Lucena, C. J. P. de (2009). MAS-ML Tool: Um Ambiente de Modelagem de Sistemas Multi-Agentes. Fifth Workshop on Software Engineering for Agent-oriented Systems (SEAS 09) no 23º Simpósio Brasileiro de Engenharia de Software (SBES 2009), Fortaleza, CE, Brasil.

France, R.; Rumpe, B. (2007). Model-Driven Development of Complex Software: A Research Roadmap, in Future of Software Engineering (FOSE'07) co-located with ICSE'07, Minnesota, EUA, May 2007, pp. 37–54.

GMF (2009). www.eclipse.org/modeling/gmf/, Acessado em 15 de Julho de 2009.

Gonçalves, E. J. T.; Campos, G. L.; Cortés, M. I.; Silva, V. T. da (2009). Modelagem de Agentes Reativos utilizando MAS-ML. Fifth Workshop on Software Engineering for Agent-oriented Systems (SEAS 09) no 23º Simpósio Brasileiro de Engenharia de Software (SBES 2009), Fortaleza, CE, Brasil.

Gonçalves, E. J. T.; Cortés, M. I.; Campos, G. L.; Gomes, G. F.; Silva, V. T. (2010a). Towards the Modeling Reactive and Proactive Agents by Using MAS-ML. Aceito para publicação no track Agent-Oriented Methodologies, Infrastructures and Processes (AOMIP) no 25th ACM SYMPOSIUM ON APPLIED COMPUTING (SAC 2010), Sierre, Suíça.

Gonçalves, E. J. T.; Cortés, M. I.; Campos, G. L.; Gomes, G. F.; Silva, V. T. (2010b). Extending MAS-ML to Model Proactive and Reactive Agents. Aceito para publicação no Second International Conference on Agents and Artificial Intelligence (ICAART 2010), Valencia, Espanha

Henderson-Sellers, B.; Giorgini, P. (2005). Agent-Oriented Metodologies. Idea Group Publishing.

Hopcroft, J. E.; Ullman, J. D. (2002) Introdução à teoria dos autômatos, linguagens e computação. 2. ed. Elsevier.

Huget, M. Agent UML Class Diagrams Revisited. In: Bauer, B.; Fischer, K.; Muller, J.; Rumpe, B. (Eds.) Anais do Agent Technology and Software Engineering (AgeS), Alemanha, 2002.

Jennings, N. R. (1996). Coordination Techniques for DAI. In: Foundations of distributed artificial intelligence. Editores Greg O'hare; Nicholas Jennings. John Wiley and Sons.

Mellor, S. J.; Clark, A. N.; Futagami, T. (2003). Introduction: Model-Driven Development, IEEE Software, vol. 20, no. 5, pp.14-18, Sep/Oct, 2003, Guest Editors'.

Menezes, P. F. (2005). Linguagens Formais e Autômatos. 5. ed. Sagra Luzzatto. v. 3.

Odell, J.; Parunak, H. V. D., Bauer, B. (2000). Extending UML for Agents. Proc. of the Agent-Oriented. Information Systems Workshop (AOIS'00) at the 17th National conference on Artificial Intelligence (AIII'00) (3-17).

Padilha, T. P. P.; Jácome, T. F. (2002). O Uso de Técnicas de Modelagem de Agentes em Ambientes Educacionais. In: VI Congresso Iberoamericano de Informática Educativa.

Pokahr, A.; Braubach, L.; Lamersdorf, W. (2003). "Jadex: Implementing a BDI-Infrastructure for JADE Agents". EXP - In Search of Innovation (Special Issue on JADE), vol. 3, no. 3 , Telecom Italia Lab, Turin, Italy, S. 76-85.

Russell, S.; Norvig, P. (2004). Inteligência Artificial: uma abordagem moderna. 2 Ed. São Paulo: Prentice-Hall.

Sadeh, N.; Arunachalam, R.; Eriksson, J.; Finne, N.; Janson, S. (2003). TAC-03 A supply-chain trading competition. AI Magazine, v. 24, n. 1, p. 92–94.

Silva, P. S.; Mendes, M. J. (2003). Uma Abordagem para Incorporar Mecanismos de Inteligência Artificial a Agentes Móveis. XXI Simpósio Brasileiro de Redes de Computadores. Natal, Rio Grande do Norte. pp. 837-852.

Silva, V.; Garcia, A.; Brandao, A.; Chavez, C.; Lucena, C.; Alencar, P. (2003). Taming Agents and Objects in Software Engineering. In: Garcia, A.; Lucena, C.; Zamboneli, F.; Omicini, A; Castro, J. (Eds.), Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag, LNCS 2603, pp. 1-26, 2003, ISBN 978-3-540-08772-4.

Silva, V. T. da (2004). Uma linguagem de modelagem para sistemas multi-agentes baseada em um framework conceitual para agentes e objetos. Tese de doutorado. Rio de Janeiro: PUC, Departamento de Informática.

Silva, V. T. da; Choren, R.; Lucena, C. J. P. de (2004). Using the MAS-ML to Model a Multi-agent System. Em: Software Engineering for Multi-Agent Systems II, Berlin: Springer, p 349-351.

Silva, V.; Cortés, M.; Lucena, C. (2004). An Object-Oriented Framework for Implementing Agent Societies. MCC32/04. Technical Report, PUC-Rio. Rio de Janeiro, Brazil.

Silva, V. T.; Choren, R.; Lucena, C. J. P. de (2005). Using UML 2.0 Activity Diagram to Model Agent Plans and Actions. The International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2005), 4th. Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems, Netherlands, Holanda, pp. 594-600, v. 2, n.1, ACM, ISBN: 1-59593-094-9.

Silva, V. T.; Choren, R.; Lucena, C. J. P. de (2007). MAS-ML: A Multi-Agent System Modeling Language. Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA); In: Companion of the 18th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications; Anaheim, CA, USA, ACM Press, pp. 304-305.

Sommerville, I. (2007). Engenharia de Software, 8º edição. São Paulo: Pearson addison Wesley.

Wagner, G. (2003). The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior. Information Systems, v. 28, n.5, pp. 475–504.

Weiss, G. (1999). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Massachusetts.

Wooldridge, M.; Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. Knowledge Engineering Review, Vol. 10, No. 2. Cambridge: Cambridge University Press, 1995.

Wooldridge, M.; Jennings, N. (1994). Agent Theories, Architectures, and Languages: A Survey. In: European conference on Artificial Intelligence, 11, Amsterdã, Holanda, Anais do Workshop on Agent Theories, Architectures and Languages, p. 1-39.

Wooldridge, M.; Jennings, N. R.; Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, v.3, n.3, p.285-312, Springer, Netherlands.

Wooldridge, M. (2002). An Introduction to Multiagent Systems. John Wiley and Sons Ltd, February.

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

UML: Unified Modeling Language Specification, versão 2.2, OMG, (2009) Disponível em:<http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML>. Acessado em: 26 de agosto 2009.