Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo, Definição e Implementação de Ambiente de Ensino-Aprendizagem com Arquitetura de Agentes e Modelo Multidimensional de Aprendizagem

João Paulo de Freitas Matos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof. Célia Ghedini Ralha

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenadora: Prof. Maristela Terto Holanda

Banca examinadora composta por:

Prof. Célia Ghedini Ralha (Orientadora) — CIC/UnB
Prof. Germana Nóbrega de Menezes — CIC/UnB
Prof. Fernanda Lima — CIC/UnB

CIP — Catalogação Internacional na Publicação

Matos, João Paulo de Freitas.

Estudo, Definição e Implementação de Ambiente de Ensino-Aprendizagem com Arquitetura de Agentes e Modelo Multidimensional de Aprendizagem / João Paulo de Freitas Matos. Brasília : UnB, 2013. 94 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. Sistemas Multiagentes, 2. Informática na Educação, 3. Modelo Multidimensional

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo, Definição e Implementação de Ambiente de Ensino-Aprendizagem com Arquitetura de Agentes e Modelo Multidimensional de Aprendizagem

João Paulo de Freitas Matos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Célia Ghedini Ralha (Orientadora)
CIC/UnB

Prof. Germana Nóbrega de Menezes
CIC/UnB

Prof. Fernanda Lima
CIC/UnB

Prof. Maristela Terto Holanda
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 12 de Março de 2013

Dedicatória

Dedico este trabalho à meus pais, que me ensinaram desde o princípio o valor da educação.

Agradecimentos

Agradeço à minha orientadora Profa. Dra. Célia Ghedini Ralha que, mesmo em seu programa de Pós Doutorado e distante da Universidade de Brasília, não se recusou em me orientar neste trabalho. Ao contrário, me auxiliou, dedicou atenção e paciência que foram decisivos para a conclusão deste trabalho. Agradeço também à Profa. Dra. Germana Menezes da Nóbrega que me ofereceu a oportunidade de realização deste trabalho e me deu todo o suporte mesmo nos momentos de sua gestação. Por fim e não menos importante, agradeço todos os amigos e familiares, em especial *Elane Mayara Sousa* que foi paciente nas minhas ausências.

Resumo

Ambientes de ensino e aprendizagem oferecem ao aluno a possibilidade de determinação do seu modelo multidimensional, compreendido pelas dimensões cognitiva, metacognitiva e afetiva. O objetivo destes ambientes é a notificação do docente acerca do modelo dos seus alunos para a melhoria na sua didádica, fornecendo mais subsídios para promover ensino personalizado. Estes ambientes porém não possuem a arquitetura adequada para lidar com o problema, pois geralmente carecem de desempenho e possuem alta complexidade. Este trabalho propõe o estudo, definição e implementação de um ambiente de ensino e aprendizagem baseado em Sistema Multiagente para a inferência do modelo multidimensional em agentes específicos e grupos de trabalhos que personificam o estilo de aprendizagem do estudante.

Palavras-chave: Sistemas Multiagentes, Informática na Educação, Modelo Multidimensional

Abstract

Teaching and learning environment offer the opportunity for students of determining its multidimensional model, understood by cognitive, metacognitive and affective dimensions. The aim of these environments is notify the instructor about the model of their students to improve in their didatics, providing more subsidies to promote personalized learning. These enviromments, however, do not have an appropriate architecture for dealing with the problem, because usually lack from poor performance and have high complexity. This work proposes the study, the design and the implementation of an environment for teaching and learning based in Multiagent System for the inference of multidimensional model in specific agents and workgroups that personify the student learning style.

Keywords: Multiagent Systems, Informatics in Education, Multidimensional Model

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Cada pessoa possui uma forma preferencial de absorção do conhecimento. Seja por imagens, textos, teoria ou prática, durante uma situação de aprendizagem todos tendem a receber e processar melhor as informações que são recebidas de certa maneira, em detrimento a outras. Essa forma de recepção do conhecimento é nomeada estilo de aprendizagem.

O acesso à recursos tecnológicos, outrora caros e difíceis, tornaram-se presentes no cotidiano de muitas pessoas devido a facilidade de aquisição. Dessa forma, a viabilidade do aprendizado por meio do computador aumentou e começa a modificar o paradigma do professor detentor do conhecimento e o único responśavel pela sua transmissão.

Em um ambiente escolar a forma didática escolhida por um docente pode afetar o desempenho dos seus alunos, visto que a forma de transmissão do conhecimento escolhida pode prejudicar alguns estudantes. A situação ideal requer a existência de uma personalização do ensino para cada estudante, considerando as características inerentes à sua cognição. Inspirado por esse ideal, algumas ferramentas no âmbito da computação pretendem o ensino personalizado aos estudantes de acordo com o seu perfil de utilização.

Assim, conhecer os fatores relacionados ao processo de aprendizagem exige que as ferramentas computacionais aplicadas ao ensino consigam determinar eficientemente os estilos de aprendizagem. Porém, a complexidade das aplicações aumenta devido às abordagens que serão empregadas, bem como o aumento da exigência de processamento.

Algumas abordagens visam diminuir essa complexidade, permitindo a representação aproximada das características dos alunos em ambientes computacionais, de forma que seja possível representar diversos domínios de modelo para um único estudante. Por exemplo, o modelo do estudante elaborado na forma multidimensional a partir dos universos cognitivo, metacognitivo e afetivo.

Somado à isso, técnicas de computação distribuída são projetadas para rodar de forma descentralizada, com componentes e serviços rodando em diversos lugares distintos e comunicando-se uma com as outras através de mensagens. A arquitetura dessas aplicações é projetada objetivando o alto paralelismo, exibilidade, interoperabilidade, dentre outros aspectos, permitindo a coexistência de diversos modelos de estudantes em uma única aplicação.

Portanto, determinar o estilo de aprendizagem mostra-se uma estratégia fundamental para que a transmissão de informações e vivências entre alunos e professores torne-se mais

eficaz e perceptível, promovendo a criação de informações cada vez mais relevantes para o planejamento, acompanhamento e avaliação dos aprendizes.

1.1 Problema

Os ambientes educacionais de aprendizagem não possuem uma arquitetura apropriada para a inferência de modelos multidimensionais. A análise do modelo multidimensional do aluno identifica suas características particulares, compreendendo os modelos cognitivo, metacognitivo e afetivo, tornando-se informações fundamentais para o docente na busca pela didática mais recomendável para seus alunos. Porém a arquitetura cliente-servidor permite uma abordagem simples do problema, não usufruindo da possibilidade de representação do conhecimento individualizado em tempo real para a interação entre modelos de estudantes.

1.2 Objetivos

Tendo em vista o cenário atual apresentado, o presente trabalho tem como objetivo definir uma arquitetura distribuída na *web* para auxiliar o processo de ensino-aprendizagem por meio da abordagem de Sistema multiagente (SMA), criando insumos para auxiliar o docente na adoção da melhor estratégia didática de ensino.

Esta abordagem permitirá a construção e manutenção de um modelo multidimensional do estudante, a partir do qual os estilos de aprendizagem desse estudante poderão ser identificados. A abordagem de sistemas multiagentes permite a decomposição do problema na modelagem multidimensional em vários subproblemas menores, sendo capaz de diminuir a complexidade da resolucão. Além disso as características inerentes aos agentes possuem alto potêncial para análises mais detalhadas do modelo multidimensional, por exemplo a habilidade social e a proatividade podem permitir a interação com outros modelos de alunos visando comparações e validações do modelo.

Em corroboração ao objetivo do trabalho, a informação do estilo de aprendizagem ao docente pode permitir a correta orientação da sua didática em sala de aula, melhorando qualitativamente o ensino aos seus alunos.

Especificamente, os objetivos deste trabalho são:

- Projeto da arquitetura geral de um SMA, utilizando-se de uma metodologia apropriada;
- Definir e implementar a arquitetura geral da solução, incluindo os agentes assistentes de cognição, metacognição e afetivo;
- Propor uma interface do agente assistente de cognição com os atores externos do sistema: Docente e Estudante;

1.3 Metodologia

A metodologia de realização deste trabalho é constituída da revisão bibliográfica, modelagem da arquitetura, implementação e testes em laboratório da solução. A primeira

etapa consistiu do estudo dos conceitos de Informática na Educação (IE), Sistemas Multiagentes e levantamentos de uso de SMA em contextos pegagógicos. Este estudo foi importante para a orientação do desenvolvimento embasado na teoria e a comparação das ferramentas existentes com a solução proposta.

Em seguida, foi realizado o levantamento e estudo de metodologias aproriadas para modelagem de SMA existentes para a modelagem e construção da solução. Essas metodologias foram comparadas e uma delas escolhida para a modelagem do SMA.

Em sequência, houve o levantamento do *framework* necessário para implementação do SMA. Após levantamento e comparação das ferramentas disponíveis, o desenvolvimento da aplicação multiagente foi feito com o *framework JADE* o qual é completamente desenvolvido na linguagem *JAVA* e simplifica a implementação de SMA que cumprem as especificações FIPA.

Em seguida, foi necessário a escolha de uma ferramenta para desenvolvimento da interface *web* e a sua integração com o SMA. Baseado nos levantamentos e estudos, o *framework JBoss Seam* mostrou-se mais viável visto que facilita a construção de aplicações dinâmicas para a internet de forma simples e ágil.

Por fim, foram realizados testes em laboratório da solução proposta. Foram definidos cenários que simulavam o uso da aplicação por meio dos perfis de Aluno e Docente.

1.4 Estrutura do Trabalho

A divisão de capítulos foi feita da seguinte forma:

- Capítulo 2: Apresenta uma breve visão sobre as áreas de estudo envolvidas neste trabalho, tais como IE e SMA. Além disso, apresenta algumas ferramentas e tecnologias utilizadas, bem como os trabalhos correlatos e o seu comparativo.
- Capítulo 3: Contém a proposta de solução composta pela metodologia, modelagem da arquitetura e implementação.
- Capítulo 4: Exibe os testes realizados em laboratório por meio dos perfis dos usuários: Aluno e Docente.
- Capítulo 5: Apresenta algumas conclusões e abre perspectivas para trabalhos futuros.
- Capítulo 6: Apresenta o apêndice com todos os casos de uso escritos na modelagem do SMA.

Capítulo 2

Fundamentos básicos

Este capítulo apresenta os principais definições e conceitos necessários para o entendimento deste trabalho. Dividido em 4 seções, a primeira delas apresenta a teoria a respeito de IE. A Seção 2.2 aborda os conceitos de SMA necessários ao entendimento deste trabalho. A Seção 2.3 apresenta a teoria acerca da metodologia MASE, a decisão pela sua escolha e as fases de desenvolvimento. A Seção 2.4 reúne um estudo de ferramentas e tecnologias relacionadas a este trabalho. Por fim, a Seção 2.5 aborda alguns trabalhos correlatos.

2.1 Informática na Educação

A IE constitui-se um importante ramo de estudo. O uso do computador como meio de educação tornou-se essencial atualmente. Popularizando meios inacessíveis em um passado recente, a aquisição de conhecimento foi simplificada e reduziu-se a um computador com conexão com a internet para o acesso a diversos tipos de conhecimento.

O computador passa então a ser uma forma de ensino, ocasionando na descentralização da figura do professor neste processo e, segundo [18], promove o desenvolvimento cognitivo por meio de uma interação maior entre o aluno e o objeto de conhecimento.

Com o crescimento da internet, surgem ambientes específicos para aplicações interativas que auxiliam o estudante. Estes ambientes, chamados de Ambientes Virtuais de Aprendizagem (AVA), surgiram na década de 90 com os cursos à distância e mais tarde com ferramentas interativas e em tempo real [28]. Da necessidade em organizar as informações geradas pelos AVA, surgiram os Sistemas Gerenciadores de Conteúdo e Aprendizagem (LCMS - Learning and Content Managment System).

A interação entre as ferramentas tornou-se cada vez maior e tornou-se cada vez mais necessário o ensino personalizado ao estudante. A personalização está relacionada com a forma específica que cada estudante tem ao lidar com informações, a melhor forma de aprender e as suas habilidades. Entre outras palavras, os Sistemas começaram a interagir mais com os alunos a fim de determinar a forma como os alunos percebem e processam as informações.

2.1.1 Estilo de Aprendizagem

O estilo de aprendizagem é a forma mais eficiente em que o estudante absorve conhecimento ao receber e processar uma informação. Um modelo de estilo de aprendizagem visa classificar um estudante de acordo com uma escala que, organiza as formas as quais são possíveis receber e processar o conhecimento [21]. Estilos de aprendizagem não classificam em uma escala melhor ou pior do que outro estilo de aprendizagem, apenas categoriza o aluno de acordo com a forma de aprender.

O estilo de aprendizagem pode orientar melhor um docente a estimular um aluno durante o processo de desenvolvimento de suas habilidades, alterando o ambiente para o aluno de acordo com o estilo de forma a individualizar o processo didático do ensino. Existem diversos modelos de estilos de aprendizagem propostos na literatura, no entando será apresentado neste trabalho as teorias mais importantes dos modelos de estudantes.

Modelo de *Kolb*

A perspectiva da teoria experimental do aprendizado proposta em *Kolb* [28] enfatiza o fato da experiência no processo de aprendizagem. Kolb inicialmente baseia-se em um ciclo de aprendizado de quatro estágios: Experiência concreta, observação reflexiva, conceituação abstrata e experimentação ativa. Basicamente o ciclo funciona de forma à iniciar nas experiências concretas para prover fundamentos para o ciclo da observação reflexiva. A partir de observações e reflexões, novos conceitos abstratos são formados. Esses conceitos podem implicar em ações que podem ser colocadas em prática, criando assim novas experiências.

Kolb propôs quatro estilos de aprendizagem: Divergente, assimilador, convergente e acomodador. Estes estilos estão permeados nos ciclos de aprendizagem definidos. A Tabela 2.1 exibe os estilos de aprendizagem propostos por *Kolb* combinados com o ciclo de aprendizado e a separação em quatro categorias.

Modelo de *Felder*

O modelo de Felder [21] divide os seguintes estilos de aprendizagem:

- Sensorial/Intuitivo.
- Visual/Verbal.
- Activo/Reflexivo.
- Sequêncial/Global.

O estilo sensorial categoriza alunos que favorecem o aprendizado por meio dos seus sentidos. Com tendências à abordagem prática no processo da aprendizagem, costumam lidar melhor com fatos e observações. Dessa forma, algumas disciplinas que focam em conceitos abstratos, teorias, fórmulas (como física e química) tendem a ser mais complicadas para estes alunos.

Alunos do estilo intuitivo favorecem informações que florecem em seu raciocínio interno, como a memória, situações imaginárias e por meio de reflexões. Eles normalmente gostam variar o tipo de trabalho, não se importando com a sua complexidade.

Tabela 2.1: Estilos de Aprendizagem definidos por Kolb

Estilo de Aprendizagem	**Ciclos**	**Descrição**
Divergente	Experiência Concreta, Observação Reflexiva	Pessoas que utilizam a imaginação com mais facilidade para resolver problemas. São capazes de olhar um problema por diferentes perspectivas. Possuem maior interesse cultural e prazer ao adquirir informações.
Assimilador	Conceituação Abstrata, Observação Reflexiva	Possuem uma aproximação maior de raciocínio lógico, onde há ênfase em idéias e conceitos. São menos focados em pessoas e mais focados em conceitos abstratos. Valorizam mais as explicações detalhadas do que uma oportunidade prática.
Convergente	Conceituação Abstrata, Experimentação Ativa	Possuem inclinação maior á situações práticas, onde utilizarão o seu conhecimento para resolver um problema de forma experimental.
Acomodador	Experiência Concreta, Experimentação Ativa	Usam-se de outras pessoas para suas análises, confiando em experiências de terceiros para tirar suas próprias conclusões.

A categoria Visual agrupa alunos com mais eficiência em aprender com imagens, diferentemente dos alunos da Verbal categoria que são eficientes em materiais verbais, palavras escritas e discursos.

Estudantes do estilo Ativo tendem a aprender de forma a realizar tentativas, expressar suas idéias, realizar experimentações. Geralmente, conseguem trabalhar muito bem em grupos devido ao favorecimento da prática de exemplos. Em oposição, os estudantes reflexivos preferem um raciocínio mais internalizado e pensam bastante a respeito antes de tentar algo. Geralmente, preferem trabalhar sozinhos ou em pares.

O estilo Sequêncial reúne estudantes que entendem informações por meio de "fragmentos de conhecimento conexos", ou seja, partes de entendimento a respeito de um domínio que, juntos, comporiam uma imagem completa do conhecimento. Podem ser capazes de resolver problemas com um entendimento incompleto acerca de algum domínio. Todavia por não possuirem o entendimento completo, carecem de conseguir correlacionar as informações com outras áreas.

Por fim, o estilo Global possui alunos que detém domínio de grandes áreas de conhecimento, estes porém de forma desconexa. Em geral, ou detém todo o conhecimento a respeito de algo, ou não sabem nada. O processamento aparentemente é lento, mas depois do domínio total, conseguem fazer muito mais conexões com áreas correlatas do que alunos com o estilo de aprendizagem Sequêncial.

Modelo de *Honey* e *Mumford*

O modelo de Honey e Mumford faz uma alteração no ciclo de aprendizagem de Kolb e também define novos estilos de aprendizagem [30]:

- Ativo: Pessoas que garantem o seu aprendizado na prática.
- Pragmático: Pessoas que precisam identificar como aplicar no mundo real a teoria estudada. Possuem inclinação à experimentações e abordagens práticas.
- Reflexivo: Aprendem durante a observação e refletindo sobre determinada ação, sem necessariamente praticá-la. Visualizam a ação de diversas perspectivas para fomentar o seu conhecimento.
- Teórico: Primam o entendimento da teoria por trás das ações. O aprendizado de novas informações ocorre por modelos, conceitos e fatos.

Honey e Mumford desenvolveram duas versões de questionários de aprendizagem [26] para a determinação das preferências de aprendizagem do indivíduo.

2.1.2 Diagnóstico do Estilos de Aprendizagem

Existem diversas possibilidades de diagnosticar o estilo de aprendizagem. Por meio de questionários de estilo de aprendizagem é possível inferir explícitamente o estilo, exigindo que o indivíduo a ser analisado responda uma série de questões pertencentes à várias dimensões de aprendizagem.

O questionário utilizado neste trabalho [30] foi escolhido por se basear em vários outros instrumentos de investigação propostos por Butler, Felder e Silvermann, Honey e Mumford e Kolb (1984). O questionário analisa o aluno através de dezessete dimensões:

Tabela 2.2: Distribuição de Perguntas no Questionário de Estilo de Aprendizagem. Adaptado de [30]

Questões	**Dimensão do Estilo de Aprendizagem**
1,2,3	Estilo Sensorial
4,5,6	Estilo Intuitivo
7,8,9	Estilo Visual
10,11,12	Estilo Verbal
13,14,15	Estilo Sequêncial
16,17,18	Estilo Global
19,20,21	Estilo Divergente
22,23,24	Estilo Assimilador
25,26,27	Estilo Convergente
28,29,30	Estilo Acomodador
31,32,33	Estilo Ativo
34,35,36	Estilo Reflexivo
37,38,39	Estilo Teórico
40,41,42	Estilo Pragmático
43,44,45	Estilo Realista
46,47,48	Estilo Análitico
49,50,51	Estilo Pessoal

Acomodador, Analítico, Assimilador, Ativo, Convergente, Divergente, Global, Intuitivo, Pessoal, Pragmático, Realista, Reflexivo, Sensorial, Sequencial, Teórico, Verbal, Visual. Cada dimensão possui três perguntas, totalizando o número de cinquenta e uma questões objetivas.

As respostas possuem quatro alternativas: Discordo Totalmente, Discordo, Concordo e Concordo Totalmente, onde cada questão possui uma pontuação que corresponde respectivamente a 1, 2, 3 ou 4.

O estilo de aprendizagem é determinado por meio do somatório das respostas de cada dimensão de aprendizagem. A dimensão com maior pontuação é a predominante, apresentando-se então como o estilo de aprendizagem do aluno.

As questões das dimensões de aprendizagem foram distribuídas entre as perguntas no questionário, apresentadas na Tabela 2.2. Foram inseridas três questões correspondentes à cada dimensão integrante dos modelos propostos em [30].

Esta seção expôs os principais modelos de estudante presentes na literatura da IE, assim como o questionário de estilo de aprendizagem como uma ferramenta capaz de determinação do estilo de aprendizagem do estudante deste trabalho. É necessário agora o entendimento de técnicas computacionais para a implementação deste questionário. A seção seguinte apresentará a teoria a respeito de SMA, que será a base para o desenvolvimento deste trabalho, definido os aspectos da infraestrutura do ambiente de ensino-aprendizagem proposto.

2.2 Sistemas Multiagente

Esta seção visa expor os fundamentos acerca de SMA. Primeiramente serão apresentados as definições iniciais a respeito da Inteligência Artificial (IA) e a teoria relacionada a agentes. Em seguida o trabalho disserta a respeito das arquiteturas relacionadas aos agentes. Só então são apresentados os conceitos de SMA: Interação dos agentes, os aspectos da comunicação neste ambiente e por fim a teoria sobre ontologias.

2.2.1 Conceitos

Em [33] é possível identificar que a definição de IA foi separada em quatro grupos. A distinção foi feita com base na classificação das definições conforme o seu entendimento de raciocínio, comportamento, performance humana e racionalidade. A racionalidade é entendida como um conceito ideal de inteligência inerente à um sistema. Os quatro grupos que definem IA são:

- Sistemas que pensam como humanos: Define IA em termos de raciocínio baseado no modelo cognitivo humano. A evidência de programas que podem operar como humanos é feita por meio da comparação da entrada, saída e o tempo de execução do programa com padrões humanos.
- Sistemas que pensam racionalmente: Esta definição está relacionada à lógica e procurava desenvolver programas que poderiam resolver qualquer problema descrito em uma linguagem formal com notações lógicas.
- Sistemas que agem como humanos: Os sistemas desta definição são classificados por meio do teste de *Turing*. O teste basicamente consiste de um entrevistador humano que, após algumas perguntas escritas ao programa, deve identificar se as respostas são ou não de um humano.
- Sistemas que agem racionalmente: Baseada na noção de agente racional, um agente é uma entidade que age e possui controle autônomo, percebe o ambiente, adapta a mudanças e é capaz de comprir outras metas.

O crescimento dos estudos relacionados a IA permitiu a ramificação em diversas áreas de atuação, possibilitando a resolução de diversos desafios relacionados à aplicações modernas. Uma das áreas de atuação é o auxílio na execução de aplicações que resolvem problemas de alta complexidade.

A complexidade de várias aplicações exige muito de *hardwares* mais modestos e o seu tempo de execução é inviável, necessitando-se um maior investimento e consequentemente encarecendo o seu valor. Dessa forma outras abordagems fazem-se necessárias, como a distribução da aplicação em vários computadores que dividem a sua execução. Este é o campo de estudo da IA Moderna segundo a abordagem de Russel e Norvig [33] que, corroborando com a última definição de IA, propõe: Sistemas que são compostos por vários agentes coletivos atuando de forma a distribuir o seu trabalho outros. Cada agente pode possuir uma capacidade diferente, sendo possível realizar a tarefa de modo paralelo.

Os agentes são entidades (reais ou virtuais) que funcionam de forma autônoma em um ambiente [33], ou seja, não necessitam de intervenção humana para realizar processamento.

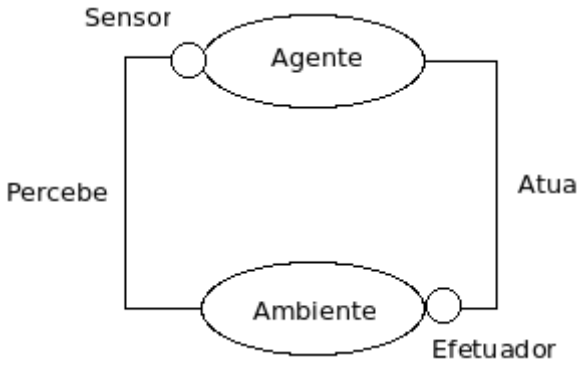

Figura 2.1: Representação do funcionamento básico do agente em um ambiente.

Esse ambiente de funcionamento do agente geralmente contém vários outros agentes e é possível a comunicação entre eles através do ambiente por meio de troca de mensagens.

A Figura 2.1 representa o funcionamento básico de agentes, existindo em um ambiente e atuando de forma a percebê-lo por meio de sensores, analisando os dados da interação inicial e agindo novamente sobre o ambiente de forma a modificá-lo pelos seus efetuadores.

Alguns agentes seguem o princípio básico de racionalidade: Sempre objetivam suas ações pela escolha da melhor ação possível segundo seus conhecimentos. Logo é possível inferir que a ação de um agente nem sempre alcança o máximo desempenho, parâmetro definido para medir o grau de sucesso da ação de um agente com base nos seus objetivos. São estes os chamados Agentes Racionais.

Como dito anteriormente, agentes estão presentes em um ambiente e não possuem controle total sobre o mesmo, sendo possível no máximo a influência com a sua atuação ou a criação de outros agentes. Podemos separar ambientes em classes: Software, Físico e Relidade virtual (simulação de ambientes reais em *software*). Em [39] são definidos os ambientes com propriedades inerentes à seu funcionamento:

- Observável: Neste tipo de ambiente, os sensores dos agentes conseguem ter percepção completa do ambiente. Por exemplo, um sensor de movimento consegue ter visão total em um ambiente aberto.
- Determinística: O próximo estado do ambiente é sempre conhecido dados o estado atual do ambiente e as ações dos agentes. O oposto do ambiente determinístico é o estocástico, quando não há certeza do seu estado. Por exemplo, agentes dependentes de eventos climáticos.
- Episódico: A experiência do agente é dividida em episódios, onde cada episódio é a percepção do agente e a sua ação.
- Sequêncial: A ação tomada pelo agente pode afetar o estado do ambiente e ocasionar na mudança de estado
- Estático: O ambiente não é alterado enquanto um agente escolhe uma ação.
- Discreto: Existe um número definido de ações e percepções do agente para o ambiente em cada turno.

Tabela 2.3: Listagem de SMA com propriedades de medida de performance, ambiente, atuadores e sensores

Tipo de agente	**Medida de performance**	**Ambiente**	**Atuadores**	**Sensores**
Sensores de estacionamento	Avarias no veículo	Carro e garagens	Freio do carro, controle de velocidade	Sensor de proximidade
Jogos com oponente computador	Quantidade de vitórias	Software	Realizar jogada	Percepção do tabuleiro
Agentes hospitalares	Saúde do paciente	Paciente, ambiente médico	Diagnósticos	Entrada de sintomas do paciente

- Contínuo: As percepções e ações de um agente modificam-se em um espectro contínuo de valores. Por exemplo, temperatura de um sensor muda de forma contínua.

A Tabela 2.3 mostra alguns exemplos de agentes, apresentando as suas características em termos de possibilidade de medida de performance, possíveis ambientes, atuadores e sensores. A primeira linha da Tabela 2.3 apresenta um exemplo de agente atuando em um veículo como um sensor de estacionamento. Responsável por auxiliar o motorista no ato de estacionar o carro, o seu ambiente é da classe físico (considerando o carro e o ambiente onde está o carro). Seu sensor de proximidade é a percepção do ambiente e, caso detecte que está próximo de um obstáculo, pode atuar nos freios dos carros diminuindo a velocidade e evitando colisões. Avarias no carro podem indicar um mal funcionamento do sensor.

A segunda linha da Tabela 2.3 apresenta um agente atuando em um jogo avulso. Esse ambiente é dito dinâmico, pois a cada jogada de um oponente (real ou não), o agente deve analisar a jogada feita, calcular sua próxima jogada e executá-la. O objetivo principal do agente é a vitória. O ambiente que o agente atua é um software e o seu atuador é um algum mecanismo que permite que ele realize a jogada. O sensor é o mecanismo no qual o agente irá perceber a jogada realizada pelo oponente.

Por fim, a última linha da Tabela 2.3 expõe um exemplo de um agente médico atuando em um ambiente estático: Um paciente. Esse ambiente é classificado como estático por não ser alterado pelo agente nesse exemplo, porém é possível classificá-lo diferente em outras situações. O objetivo principal é monitorar a saúde do paciente, logo a medida de performance será a aproximação ou não do diagnóstico médico. Seu atuador não será diretamente no ambiente (corpo humano), será na forma de relatórios médicos e seus sensores podem variar de acordo com a doença a ser monitorada.

Segundo [39] é possível separar as principais noções dos agentes em dois grupos. O primeiro, chamado de noção fraca, compreende as características de reatividade, proatividade e habilidade social. O conceito de reatividade está ligado com a percepção do ambiente e a reação no mesmo. Proatividade é a característica do agente tomar iniciativa e agir sem a necessidade de nenhum estímulo. Habilidade social é a capacidade de interação com outros agentes. Já a noção forte de agente envolve os seguintes aspectos:

- Mobilidade: Possibilidade de movimentação pelo ambiente, por exemplo, transferência em uma rede.
- Veracidade: O agente não transmite informações falsas.
- Benevolência: Capacidade de ajudar os outros agentes.
- Racionalidade: A ação do agente não impedirá a realização dos seus objetivos.
- Cooperação: Cooperação com o usuário.

2.2.2 Arquiteturas de agentes

A arquitetura de agentes varia de acordo com a complexidade da sua autonomia, ou seja, com a capacidade de reagir aos estímulos do ambiente. Conforme verificado em [33], os tipos de arquitetura são: orientadas à tabela, reflexiva simples, reflexiva baseado em modelo, baseada em objetivo, baseada em utilidade.

A primeira arquitetura a ser explorada é o agente orientado à tabelas. Todas as ações dos agentes são conhecidas e estão gravadas em uma tabela. Assim, quando o agente receber o estímulo ele já terá a ação a ser tomada previamente gravada em sua memória. Logo para construir esse tipo de agente, fica claro que além de saber todas percepções possíveis, será necessário definir ações apropriadas para todas. Isso ocasiona a criação de tabelas muito complexas e o tamanho pode facilmente passar a ordem de milhões de registros dependendo do número de entradas.

A arquitetura reflexiva simples é um dos tipos mais elementares de agente. Nele, o agente seleciona a ação com base unicamente na percepção atual, desconsiderando assim uma grande tabela de decisões. A decisão é tomada com base em regras de condição-ação: Se uma condição ocorrer, uma ação será tomada. Por exemplo, supondo um agente médico que determina o diagnóstico de uma doença no paciente caso exista alguma anomalia no organismo (Por exemplo, um paciente com febre). Uma condição-ação poderia ser:

if anomalia-organismo then diagnóstico-médico

Esse tipo de agente é bastante simples, o que é uma vantagem comparado à arquitetura de tabela. Porém, essa abordagem requer um ambiente totalmente observável, visto que esse tipo de agente possui uma inteligência bastante limitada. No exemplo do agente médico existem diversas maneiras de se detectar uma anomalia no organismo do paciente, seria necessário conhecer todas as formas para usarmos uma abordagem reativa simples.

A arquitetura baseada em modelos funciona de maneira similar a anterior. Nessa abordagem, é levado em conta a parte do ambiente que não é visível neste momento. E para saber o "momento atual" de um agente, é necessário guardar a informação de estado consigo. Para atualizar o estado do agente, é necessário conhecer como o mundo desenvolve-se independente do agente (no caso do exemplo, como o organismo funciona) e é necessário saber as ações dos agentes no ambiente. Esses conhecimentos acerca do ambiente são definidos como modelo de mundo. Do ponto de vista do agente, essa abordagem é considerada como baseada em modelo.

Na arquitetura baseada em objetivo, as ações do agente são executadas apenas se o aproximam de alcançar um objetivo. Para isso, será necessário algo além do estado atual do ambiente: Será necessário informações do objetivo a ser atingido. Assim o agente pode combinar as informações de estado e objetivo para determinar se deve ou não agir

sobre o ambiente. Este modelo permite uma maior flexibilização das ações em determinados ambientes, visto que suas decisões são representadas de forma explícita e podem ser modificadas, com a consequência de aumento da complexidade desta abordagem. É interessante notar que esse tipo de arquitetura não trata ações com objetivos conflitantes.

Por fim, a arquitetura baseada em utilidade não utiliza apenas objetivos para realizar a próxima decisão, mas dá ao agente a capacidade de fazer comparações sobre o estado do ambiente e as ações a serem tomadas: Quais delas são mais baratas, confiáveis, resilientes e rápidas do que as outras. A capacidade de avaliação do agente é chamada de função de utilidade, que mapeia uma sequência de estados em um número real que determina o grau de utilidade. Esse mecanismo possibilita a decisão racional de escolha entre vários objetivos conflitantes. Por exemplo, escolher entre um objetivo mais barato ao invés de escolher entre o mais rápido.

2.2.3 Interação entre Agentes

Os agentes interagem entre si em um ambiente SMA, que são sistemas onde estes agentes são capazes de se comunicar, possuindo uma linguagem de alto nível para isso. Um agente possui um objetivo que, normalmente, é distinto dos objetivos de outros agentes. Além disso, ele pode ou não cooperar com outros para a realização de uma tarefa.

Em [36], podemos encontrar as seguintes características principais em ambientes multiagente:

- Fornecem protocolos específicos para comunicação e interação. Cada ambiente tem as suas particularidades: Alguns são em uma única máquina, outros são compartilhados com o mundo real e outros são distribuídos. Cabe a cada ambiente definir um protocolo onde todos agentes devem obedecer para comunicar-se.
- São tipicamente abertos.
- Contém agentes que são autônomos e individualistas.

É trivial imaginar que um SMA é designado para a solução de problemas de forma distribuída, onde o problema é fragmentado entre os agentes que, juntos, trabalham concorrentemente para a resolução deste problema, podendo ou não agir de forma cooperativa.

É importate notar a distinção de conceitos como cooperação na literatura de SMA. Em [39], existem duas principais diferenças do conceito entre as duas abordagens.

A primeira diferença é a designação distinta dos objetivos dos agentes. Em um ambiente com vários agentes, eles devem trabalhar estratégicamente para alcançar seus objetos. A possibilidade de agentes não cooperarem é perfeitamente plausível. A segunda diferença está na ação de forma autônoma, ou seja, tomar suas próprias decisões em tempo de execução sem interferências humanas ou de outros agentes. Logo, um ecossistema de agentes deve ser capaz de coordenar dinâmicamente suas ações, cooperando com outros agentes para atingir os objetivos. Em aplicações distribuídas, esses comportamentos já são desenhados durante o planejamento do software.

A forma de agentes resolverem problemas foi baseada na técnica distribuição cooperativa de resolução de problemas - *cooperative distributed problem solving* (CDPS). Em [20], a técnica CDPS consiste de uma rede de baixo acoplamento provida de sofisticados nós

resolvedores de problemas que precisam cooperar entre si, pois nenhum deles possui recursos, informações e *expertise* suficientes para resolver algum problema sozinho. Cada nó possui uma *expertise* diferente que pode resolver parte do problema.

Inicialmente essa técnica assumiu que os problemas fossem de ordem benevolente. Isso significa que, implicitamente, os agentes compartilham o mesmo objetivo de resolver o problema proposto. Isso implica que todos eles ajudarão sempre que possível, mesmo tendo prejuízos na execução da ação, pois o objetivo geral de todos será a resolução do problema maior. Esse cenário é plausível desde que uma organização ou entidade tenha o controle de (ou possa modelar) todos os agentes.

Em um cenário mais realista (e de maior enfoque dos estudos de SMA), agentes podem pertencer à sociedades com interesses próprios, diferentes de outras sociedades. Logo, é possível ocorrer conflito de interesses neste ambiente, situação que força os agentes a cooperarem com os outros para alcançarem seus objetivos.

O processo CDPS pode ser divido em três etapas:

- decomposição do problema - o problema é quebrado em instâncias menores e dividido entre os agentes. Esses subproblemas podem ser quebrados em diversas subpartes, permitindo uma maior decomposição e consequentemente um maior número de agentes trabalhando na resolução.
- solução do subproblema - os subproblemas são resolvidos pelos agentes. Isso pode significar troca de informações entre os agentes, por exemplo, dos que resolveram as suas instâncias compartilhando informações com outros agentes.
- integração da solução - as soluções dos subproblemas são integradas em uma resolução completa do problema original.

Com uma solução compartilhada de resolução de problemas, a arquitetura SMA mostra-se bastante robusta neste quesito. É necessário porém saber dos detalhes da comunicação entre os agentes, a forma de envio das mensagens, as suas linguagens, bem como outras particularidades.

2.2.4 Comunicação

A comunicação é um dos aspectos mais importantes no desenvolvimento de SMA. Problemas de sincronização entre as partes que se comunicam devem ser devidamente estudados. A situação mais simples possível da comunicação, onde o agente A envia uma mensagem ao agente B que está prontamente disponível para receber a mensagem nem sempre é a o cenário mais recorrente. Para tanto, é necessário entender os pormenores da comunicação desta abordagem.

Em uma aplicação normal (*desktop* ou *web*), a comunicação entre objetos pode ser mais simplificada. Por exemplo, supondo uma aplicação em que existe dois objetos, a e b e que o objeto a tenha um método público chamado *m1*. O objeto b pode ser comunicar com o objeto a por meio do método *m1*, provavelmente da seguinte forma *a.m1(args)*, onde *args* são os argumentos enviados ao objeto a e a sintaxe pode ser diferente da apresentada, dependendo da linguagem de programação. É importante notar que o controle da execução do método *m1* não está no objeto a, mas sim no objeto b, que decide o momento o qual o método será invocado.

Esse cenário de comunicação é diferente em um ambiente SMA. Supondo dois agentes a e b, onde o agente a tem a capacidade de executar a ação α. O agente b não poderá invocar diretamente o método que corresponde à ação α, visto que os agentes são autônomos e independentes: Cada um tem somente tem total controle sobre suas ações e seus estados. O agente precisará enviar a solicitação da execução da ação α por meio de mensagem. Isso porém não garante que o agente a executará esta ação, pois pode não ser do seu interesse. Os agentes podem também influênciar o comportamento de outros agentes, alterando seu estado interno para a execução de ações e cooperando para o cumprimento do objetivo de outros agentes.

A comunicação dos agentes é baseada na teoria dos atos de fala (*Speech act theory*) e trata a comunicação como uma ação. A teoria dos atos de fala, publicada em 1962 por John Austin [13], propõe que certas expressões de linguagem natural, ou atos de fala, possuem a característica implícita de suscintar uma ação em um interlocutor, causando assim uma mudança de estado da mesma forma que uma ação física. Essas expressões descrevem as ações por meio de desejos, habilidades e crenças.

Em [38] é definido que a teoria dos atos de fala apresenta duas características:

- A distinção entre o significado proferido por uma expressão e a forma como essa expressão é utilizada.
- Expressões de todos os tipos podem ser consideradas como atos de fala, pois têm capacidade de mudar o mundo de alguma forma.

Posteriormente o trabalho de John Searle [35], relacionado ao de Austin, separa uma ação de um ato entre orador (*speaker*) e ouvinte(*hearer*) identifincando propriedades e condições que um discurso deve possuir para realizar ações sucedidas. Além disso, ele classifica alguns atos de discursos em 5 classes:

- Representativas - Ato de um orador representar uma verdade para o ouvinte. Pode ser entendido como uma ação de informar (*inform*).
- Diretivas - Tentativa do orador de fazer algum ouvinte realizar alguma ação por meio do seu ato. Pode ser entendido como uma ação de requisição (*request*).
- Comissivas - O orador toma alguma atitude em relação à uma ação em andamento.
- Expressivas - Expressa algum estado psicológico.
- Declarações - Causa algum efeito relacionado a determinado assunto.

A comunicação então baseia-se nos atos de fala para prever a interação com seres humanos e é definida por meio de semânticas definidas pela teoria da IA [39]. O formalismo para a comunicação foi escolhido de forma que foi possível representar os atos de discursos em uma lógica multimodal, que contém os operadores de desejos, habilidades e crenças dos atos de discurso.

Da mesma forma que a teoria dos atos de fala influênciou na arquitetura da comunicação, ela influênciou também as várias linguagens de comunicação dos agentes. Linguagens foram desenvolvidas para, não apenas representar ações de agentes, mas também para representar o conhecimento entre sistemas autônomos. No início dos anos 90, duas linguagens foram desenvolvidas pelo consórcio *Knowledge Sharing Effort*, encabeçados pela agência norte americana *Defense Advanced Research Projects Agency*(DARPA) [7].

Tabela 2.4: Listagem de atributos de uma mensagem em KQML

Parâmetro	**Significado**
sender	Remetente da mensagem.
receiver	Destinatário da mensagem.
reply-with	Identifica se o remetente espera uma respota. Em caso positivo, o campo *in-reply-to* deve ser preenchido com a referência para a resposta.
in-reply-to	Campo conténdo a referência para a resposta solicitada.
language	Linguagem em que o campo *content* está escrito.
ontology	Indica a forma que deve ser interpretada o conteúdo do campo *content*.
content	Conteúdo da mensagem.

- *Knowledge Query and Manipulation Language* (KQML) - Protocolo designado para a comunicação de agentes, em uma arquitetura que esses agentes sejam projetados para resolver problemas da arquitetura cliente-servidor [31]. Não existe o foco com o conteúdo da mensagem.
- *Knowledge Interchange Format* (KIF) - Criada para facilitar a troca de conhecimento entre agentes, suas declarações são providas de significados que podem ser compreensíveis a qualquer agente que conheça a estrutura da linguagem. Não possui foco na transmissão da mensagem [23].

KQML

A linguagem KQML define um protocolo para comunicação de agentes, onde cada mensagem tem um enunciado performativo (*performative*), que varia com o seu objetivo, e em seguida os parâmetros da mensagem. O KQML define vários enunciados performativos que distinguem-se pelo objetivo da mensagem, sendo dividas em três categorias: Discursivas, intervenção e facilitação e *networking*. Por exemplo, uma mensagem com o tipo performativo *ask-one* indica que o agente remetente A deseja saber uma resposta do agente B sobre o conteúdo da mensagem. Em [31], é possível verificar que a última versão da linguagem define mais de trinta enunciados performativos.

Os maioria dos parâmetros são opcionais, sendo os mais importantes: *content* e *receiver*. A Tabela 2.4 lista os principais parâmetros em uma mensagem nesta linguagem, bem como seu significado.

O formato da mensagem é completamente compreensível aos humanos. Na mensagem apresentada na Listagem 2.1 é possível identificar na primeira linha o enunciado performativo *ask-one*, onde será uma mensagem de consulta. Nas linhas abaixo, visualizamos todos os parâmetros da mensagem antecedidos por (:). O primeiro parâmetro da mensagem é *receiver* como controle-estoque, ou seja, esse será o destinatário da mensagem. O segundo parâmetro *language* tem o valor PROLOG indicando que a sintaxe do conteúdo está escrita em PROLOG. O próximo parâmetro, *ontology* informa a ontologia que expressa o conteúdo. Por fim, o parâmetro *content* indica o conteúdo da mensagem, no caso, uma consulta escrita em PROLOG perguntando o preço de um computador.

Listagem 2.1: Exemplo de mensagem em KQML

```
(ask-one
        :receiver controle-estoque
        :language PROLOG
        :ontology PRODUTOS
        :content  (PRECO COMPUTADOR ?price)
)
```

A Listagem 2.2 apresenta um exemplo de diálogo escrito em KQML onde é possível identificar a interação entre dois agentes. A primeira mensagem do diálogo possui o enunciado performativo *evaluate*, significando que o emissor A deseja avaliar o conteúdo com B. Nos parâmetros é possível notar que a linguagem da mensagem é PROLOG, utiliza a ontologia PRODUTOS e o conteúdo é uma consulta em prolog. O parâmetro *reply-with* cria uma referência para a consulta do conteúdo.

Na segunda mensagem, o seu enunciado performativo é *reply*, significando uma mensagem do tipo resposta. O parâmetro *in-reply-to q1* especifica essa mensagem como resposta à q1, ou seja, à consulta da mensagem anterior. Dessa forma a linguagem consegue distinguir respostas de um mesmo remetente. Os outros parâmetros são o emissor B, destinatário A, linguagem PROLOG e o conteúdo da mensagem, o valor da consulta q1.

Listagem 2.2: Exemplo de diálogo em KQML

```
(evaluate
        :sender A
        :receiver B
        :language PROLOG
        :ontology PRODUTOS
        :reply-with q1
        :content  (PRECO COMPUTADOR ?price)
)
(reply
        :sender B
        :receiver A
        :language PROLOG
        :ontology PRODUTOS
        :in-reply-to q1
        :content  (=2000.00)
)
```

De acordo com [39], a adoção desta linguagem pela comunidade de SMA foi significante, mas sofreu diversas críticas e ocasionando o desenvolvimento de novas linguagens:

- A fluídez e a não restrição do KQML fez com que diversas implementações extendidas surgissem, impossibilitando a interoperabilidade entre sistemas;
- Mecanismos de transporte do KQML nunca foram bem definidos, causando problemas de diversas implementações destes mecanismos e prejudicando novamente a interoperabilidade;
- A semântica do KQML nunca foi formalmente definida, ocasionando em má interpretações dos enunciados performativos;

- A linguagem não possui enunciados performativos adequados para algumas semânticas. Por exemplo, a inexistência do enunciado *comissives*.

KIF

A linguagem foi desenvolvida para expressar conhecimento a cerca de um determinado domínio, sendo possível assim a troca de conhecimentos entre agentes. De acordo com [23] a linguagem possui as seguintes características:

- Semântica declarativa, sendo possível entender o seu significado sem a necessidade de um interpretador para manipulação das expressões;
- Logicamente compreensível;
- Capacidade de reproduzir meta-conhecimento, ou seja, conhecimento a respeito da representação do conhecimento. Com isso, é possível exibir novas representações de conhecimento sem a necessidade de modificar a linguagem.

A linguagem é baseada na lógica de primeira ordem onde são definidos operadores como existe($\exists$) e para todo($\forall$), possibilitando aos agentes a expressão de diversas propriedades e domínios. Além disso, ela define um vocabulário básico para a expressão de tipos básicos (números, caracteres, strings) e algumas funções padrões para lidar com esses tipos de dados (menor que, maior que, soma, dentre outros).

O exemplo de código da Listagem 2.3 ilustra um exemplo de expressão na linguagem KIF, validando que a temperatura de m1 é 83 graus Célsius.

Listagem 2.3: Exemplo de expressão de conteúdo com a linguagem KIF. Fonte: [39]

```
(= (temperature ml ) ( scalar 83 Celsius))
```

2.2.5 Linguagem de Comunicação de Agentes FIPA

Após as críticas à linguagem KQML, o consórcio *Foundation for Intelligent Physical Agents* (FIPA) começou a trabalhar em 1995 no desenvolvimento da padronização de SMA. O núcleo dessa padronização foi o desenvolvimento de uma linguagem de comunicação de agentes (ACL) comum à todas as plataformas.

Baseado na linguagem KQML, a estrutura das mensagens é a mesma e os seus atributos são semelhantes. A maior diferença entre as duas linguagens são os enunciados performativos. Foram definidos 20 tipos de mensagens performativas, muito semelhante ao KQML, porém com definições formais das interpretações dessas mensagens e não definindo nenhuma linguagem para o conteúdo da mensagem.

Devido ao fato da linguagem KQML não possuir enunciados performativos adequados, a ACL da FIPA recebeu total atenção na definição formal da semântica da linguagem. A Tabela 2.5 contém um breve resumo dos enunciados performativos disponíveis em [4]. É importante ressaltar que a especificação disponível em [4] possui toda a descrição formal de cada enunciado performativo aqui descrito.

As performativas *request* e *inform* são consideradas principais pela especificação da FIPA, pois orientam toda a linguagem de comunicação. Além disso, é possível derivar as outras performativas por meio destas.

Tabela 2.5: Listagem de Enunciados Performativos

Tipo	**Descrição**
Accept Proposal	Declaração de aceite de proposta feita por um agente.
Agree	Performativa feita por um agente indicando que aceitou o *request* feito por outro agente.
Cancel	Cancela uma mensagem de *request*, informando que não irá mais participar daquela conversação.
Call for Proposal	Performativo que indica o início de uma negociação entre agentes.
Confirm	A confirmação permite ao emissor da mensagem confirmar a veracidade do conteúdo da mensagem.
Disconfirm	Similar à confirmação, porém informando ao emissor da não certeza do conteúdo da mensagem.
Failure	Permite ao agente indicar que a tentativa de executar uma ação falhou.
Inform	Informa à um destinatário que o conteúdo da mensagem é verdadeiro, implicando que o remetente da mensagem também acredita no seu conteúdo. É uma das mais importantes performativas feitas pela FIPA.
Inform If	Envia uma mensagem a qual o seu conteúdo pode ser verdadeiro ou falso.
Inform Ref	Similar ao *Inform If*, com a diferença que ao invés de questionar se é verdadeiro ou falso, ele solicita o valor de uma expressão para o remetente.
Not Understood	Informa à um agente que não entendeu por que determinada ação deve ser realizada. Usada quando o estado interno do agente não é compatível com a mensagem.
Propagate	Consiste em propagar o conteúdo da mensagem para um grupo de agentes.
Propose	Permite um agente realizar uma proposta em uma negociação para outro agente.
Proxy	Permite ao destinatário da mensagem agir como um *proxy* para os agentes que estão descritos no conteúdo da mensagem.
Query If	Permite à um agente consultar um destinatário sobre a validade do conteúdo.
Query Ref	Similar ao *query-if*, porém o agente remetente irá consultar o valor de uma expressão
Refuse	Indica que um determinado agente não irá executar uma ação que foi determinada por outro agente.
Reject Proposal	Permite um agente rejeitar uma proposta feita por outro agente em uma negociação.
Request	Permite à um agente requisitar que outro agente execute determinada ação.
Request When	Permite à um agente requisitar que outro agente execute determinada ação quando a proposição (no conteúdo da mensagem) for verdadeira.
Request Whenever	Similar ao *request-when*, porém o agente nunca irá executar a ação quando a proposição (no conteúdo da mensagem) for verdadeira.
Subscribe	O conteúdo será uma proposição e o remetente da mensagem será notificado sempre que essa proposição for verdadeira.

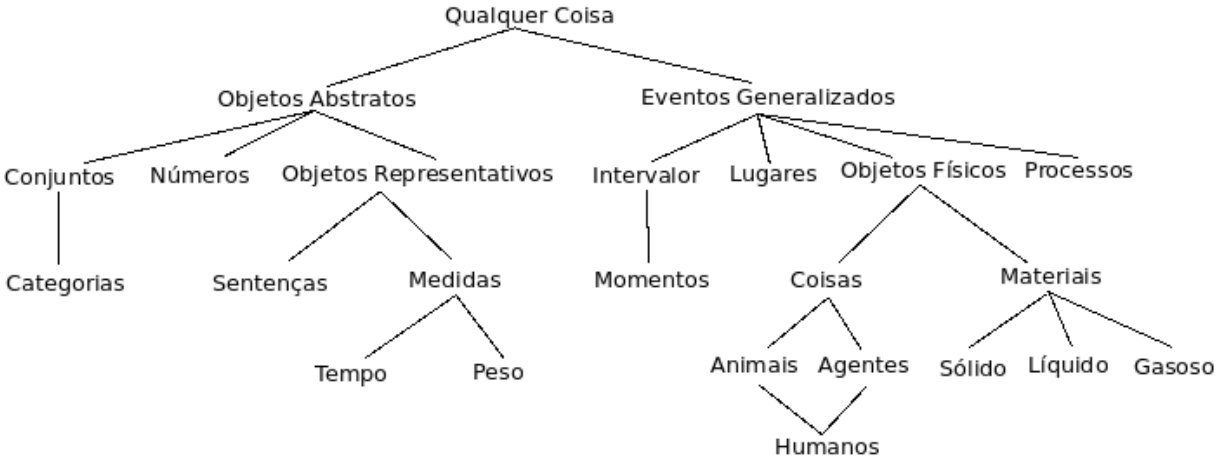

Figura 2.2: Ontologias superiores do mundo, cada uma indicando um conceito ou especialização do seu superior.

2.2.6 Ontologias

Ontologia é um um ramo da Filosofia que dedica-se a estudar e representar a natureza do ser, existência ou realidade. De acordo com [27] é um modelo que consiste na representação de um vocabulário com definições precisas dos termos, axiomas formais que restringem uma interpretação e expressões bem formadas de uso dos termos. Segundo [33], a representação de conceitos e objetos pode ser entendida como Engenharia Ontológica:

> *...concentrating on general concepts-such as Actions, Time, Physical Objects, and Beliefs - that occur in many different domains. Representing these abstract concepts is sometimes called ontological engineering.*

A possibilidade de representação de conhecimento por meio de ontologias é vasta. A sua forma de especificação pode ser entendida como uma hierarquia, onde os conhecimentos são organizados na forma de árvore, possibilitando a inserção de novos nós de conhecimento a qualquer momento na estrutura.

A Figura 2.2 representa a organização geral de conceitos, chamado de ontologias superiores. As ontologias gerais estão representadas no topo da árvore, e as suas especialidades vão crescendo no sentido das folhas.

Em um ambiente SMA além de especificar uma linguagem para comunicação, dois agentes podem comunicar-se com relação à um determinado domínio de aplicação: Podem negociar valores de carteiras em uma organização financeira, podem trocar mensagens sobre os dados analisados de performance de veículos, enfim, dois agentes podem comunicar-se usando a mesma ontologia.

Existem muitas linguagens que foram desenvolvidas para expressar ontologias. A mais comum delas é a *eXtensible Markup Language* (XML), uma linguagem de marcação que organiza os dados de forma hierarquizada.

A Listagem 2.4 descreve uma ontologia de domínio de países escrita na linguagem XML. Nela observa-se a ontologia superior, País. Cada país é composto por um conjunto de estados, conténdo as propriedades nome e sigla. Cada estado possui um conjunto de cidades. Cada cidade possui os atributos nome e população. Esses atributos poderiam ser escritos de forma diferente no exemplo da Listagem 2.4, porém sem alteração de semântica.

Listagem 2.4: Representação uma ontologia simples de cidades.

```
<Pais nome="Brasil">
        <Estado nome="Goias" sigla="GO">
                <Cidade>
                        <Nome>Goiania</Nome>
                        <Populacao>1.300.000</Populacao>
                </Cidade>
                <Cidade>
                        <Nome>Anapolis</Nome>
                        <Populacao>342.347</Populacao>
                </Cidade>
                <Cidade>
                        <Nome>Aparecida de Goiania</Nome>
                        <Populacao>474.219</Populacao>
                </Cidade>
        </Estado>
        <Estado nome="Sao Paulo" sigla="SP" >
                <Cidade>
                        ...
                </Cidade>
                ...
        </Estado>
        ...
<Pais>
```

2.3 Metodologias de Sistemas Multiagente

Atualmente muitas abordagens de projeto de desenvolvimento de software conseguem com sucesso definir uma metodologia para construção, implantação e manutenção do software. A definição dessas engenharias de software possuem diversos casos de sucesso, em sua maioria nas áreas de Análise de Projetos Orientados à Objetos.

Pela característica autônoma dos agentes, não é possível compará-los à simples objetos que serão invocados por outros objetos. Não existe interação direta, apenas coordenação de ações via conversação para cada agente atingir suas próprias metas. Portanto, o advento da abordagem SMA trouxe à tona a necessidade de outras metodologias de desenvolvimento diferente daquelas, devido à diferença de abordagem do software.

Foram definidas algumas metodologias para modelagem e desenvolvimento de SMA. A metodologia *Gaia* [40] consiste em uma abordagem não iterativa que define análise e *design* de sistemas baseados em agentes. Esta metodologia é considerada genérica e compreensível pois pode ser aplicada a um nível muito amplo de SMA, lidando tanto com escopos mais gerais (sociedade), quanto com escopos mais específicos (agentes). Ela define as regras e tarefas em termos de uma organização, que é sinônimo de sociedade de agentes ou um SMA.

A metodologia *PASSI* (*Process for Agent Societies Specification and Implementation*) é designada para o acompanhamento do desenvolvimento do sistema desde o levanta-

mento dos requisitos até a codificação [17]. Seus modelos de projetos são baseados nas especificações da FIPA, garantindo assim a compatibilidade com o SMA a ser desenvolvido. O *PASSI* define uma metodologia cinco modelos de trabalho: Requisito do Sistema, sociedade de Agentes, implementação dos agentes, modelo de código e modelo de implantação.

A metodologia *Multiagent Systems Engineering* (MASE) é definida por meio dos requisitos e metas iniciais do SMA. Estes são levantados de forma iterativa e a partir de então são modeladas em diversas etapas as tarefas para cumprir os objetivos. O MASE utiliza-se de alguns modelos gráficos para a descrição dos agentes, seus objetivos, suas interações e a sua arquitetura. Em [34], a metodologia do MASE é baseada nas mais tradicionais metodologias de desenvolvimento de software, dividida em duas fases principais: Análise e Design.

Neste trabalho, foi escolhido a metodologia de desenvolvimento MASE. Em contraste com as outras abordagens, a metodologia *GAIA* não permite a adição de novos requisitos no momento de *design*. Supondo uma situação onde estuda-se detalhadamente o *design* do sistema, ocasionalmente podem surgir alguns requisitos ou tarefas adicionais derivados desta análise. Por não permitir a adição destes requisitos, o *GAIA* mostra-se uma abordagem mais rígida e pode ser desvantajosa se comparado com outras e, portanto, foi descartada.

A necessidade do acompanhamento até o código do desenvolvimento dos agentes feito pela metodologia *PASSI* poderia exigir um conhecimento mais aprofundado do framework de desenvolvimento do SMA, necessitando assim da escolha e estudo prévio deste *framework*. Este trabalho optou então pelo não acoplamento entre o projeto de modelagem e o código e, portanto, determinou o *MASE* para o seu desenvolvimento.

Justificada a escolha da metodologia, a primeira fase do MASE, chamada de análise, consiste no levantamento e entendimento dos requisitos com o objetivo de levantar um conjunto de regras, as quais são associadas à tarefas que devem ser realizadas para o sistema atingir seus objetivos. Ao fim dessa fase, são gerados alguns artefatos que embasarão a próxima etapa da metodologia. A fase de análise pode ser dividida nos seguintes passos:

- Capturar Metas;
- Desenvolver Casos de Uso;
- Refinar Regras.

A fase de design consiste na modelagem do SMA de acordo com as regras levantadas na fase anterior. O objetivo é a definição das conversações que existirão entre os agentes, bem como a arquitetura geral do sistema. Essa fase pode ser divida em quatro passos:

- Criar as Classes dos Agentes;
- Construir Conversações;
- Montagem dos Agentes;
- Design do Sistema.

2.3.1 Análise

De acordo com [34], a fase de análise objetiva o desenvolvimento de um conjunto de regras, que descrevem uma funcionalidade para uma entidade em termos de tarefas necessárias para o sistema atingir seus objetivos.

Captura de Metas

O primeiro passo dessa fase, Captura de Metas, consiste na transformação da especificação inicial do sistema em um conjunto estruturado de metas. Entende-se que as *metas* que o sistema leva em consideração são relacionadas ao sistema (e não ao usuário), visto que são mais estáveis e as metas do sistema devem satisfazer, de forma geral, os objetivos do usuário [34].

Dividida em dois subpassos, o primeiro deles é a identificação de todas as metas. A partir da documentação inicial do sistema, com os requisitos funcionais, são extraídos os objetivos de cada cenários. É importante ressaltar que as metas devem descrever de forma geral e suscinta o comportamento do sistema, descrevendo o que o comportamento deve fazer, que é distinto de "como deve ser feito".

As metas são divididas em quatro tipos e podem ser classificadas em mais de um: Sumário, particionada, combinada e não funcional. A meta sumário é derivada de outras metas semelhantes para que se generalize em uma meta pai. Meta particionada é a meta que, quando todas suas submetas são atingidas, a meta passa a ser atingida. Metas combinadas são metas que são agrupadas devido à semelhaça de seus objetivos. As metas não funcionais estão relacionadas ao cumprimento objetivos que são requisitos não funcionais do sistema.

O segundo subpasso consiste na estruturação de metas em forma hierarquizada. É necessário separar quais metas são mais abstratas e de que forma elas podem ser agrupadas em forma de hierarquia. Com isso, eliminam-se algumas metas que são repetidas e identificam-se quais metas são atingidas por meio de outras submetas.

Desenvolver Casos de Uso

Neste passo o objetivo é entender o comportamento e o fluxo de execução do sistema por meio dos casos de uso, além de haver um entendimento maior sobre como o sistema comunicar-se-á. Para tanto, este passo visa o levantamento e criação dos casos de uso do sistema, bem como o diagrama de sequência para detalhar a ordem dos eventos de cada cenário.

Os casos de uso geralmente são levantados a partir dos requisitos iniciais do sistema. Nele, são identificados os participantes (atores) e a sua interação com o sistema, esclarecendo a comunicação de alguns módulos do sistema.

O diagrama de sequência define os eventos que cada interação pode criar, mostrando a ordem de execução destes eventos e a sua comunicação. Para cada caso de uso é criado no mínimo um diagrama de sequência. O objetivo principal é o levantamento dos eventos e das regras.

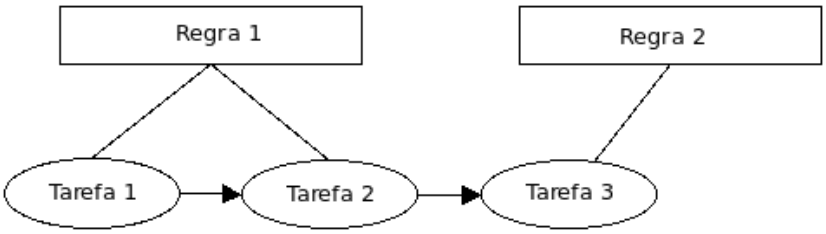

Figura 2.3: Representação utilizada no MASE Role Model.

Refinar Regras

O último passo da fase de análise, o refinamento de regas consiste na associação metas e seus diagramas de sequências às regras e suas respectivas tarefas. A associação de tarefas à regras é a melhor forma de modelagem de SMA [34].

Em geral, a associação de metas às regras é de um para um. Durante esta etapa, algumas considerações relativas ao desenho do sistema devem ser levadas em consideração. Caso exista alguma alteração (adição de uma nova meta, alteração de caso de uso, dentre outras), a metodologia permite que o analista retorne aos passos anteriores e remodele a solução. Algumas metas podem ser combinadas em apenas uma regra, simplificando o desenho do sistema.

A interface com sistemas externos geralmente é tratada como uma regra diferente, visto que sua complexidade pode variar. O MASE não modela explicitamente um ator humano, pois o considera como um ator externo ao sistema.

Após a associação das regras, elas são estruturadas em uma modelagem chamada *MASE Role Model*, que possui informações de interações entre as tarefas. A Figura 2.3 representa a notação do *MASE Role Model* utilizada por [34], onde os são utilizados retângulos para expressar as regras, elipses para identificar as tarefas e as setas entre as tarefas representam as suas comunicações. Essas comunicações podem ser por meio de mensagens, caso as regras estejam separadas em agentes diferentes.

Caso as regras compartilhem tarefas será necessário remodelar a composição da regra, visto que o MASE não permite a duplicação de tarefas.

De forma geral, cada regra possui uma série de tarefas que podem (ou não) executar paralelamente. Cada tarefa possui um comportamento que pode depender de outras regras para o cumprimento de seu objetivo. Preocupando-se com as conversações entre tarefas, o MASE define neste passo a criação do *Concurrent Task Diagram*. Ilustrado na Figura 2.4, é representado por meio de autômatos de estados finitos devido a facilidade de construção e entendimento. A transição consiste de uma mudança de estado do agente, podendo envolver um processamento ou uma comunicação externa.

A mudança de estado do automato é equivalente à mudança de estado do agente. A sintaxe da transição pode ser expressa por meio da notação da Listagem 2.5 que define o gatilho a ser disparado, condições para a execução e as mensagens transmitidas.

Listagem 2.5: Sintaxe da mudança de estado.

```
trigger [guard] ^ transmission(s)
```

O *token* trigger representa um evento que inicia a mudança de estado, geralmente vindo de outra tarefa da mesma regra. O *token [guard]* é a verificação da validade desta

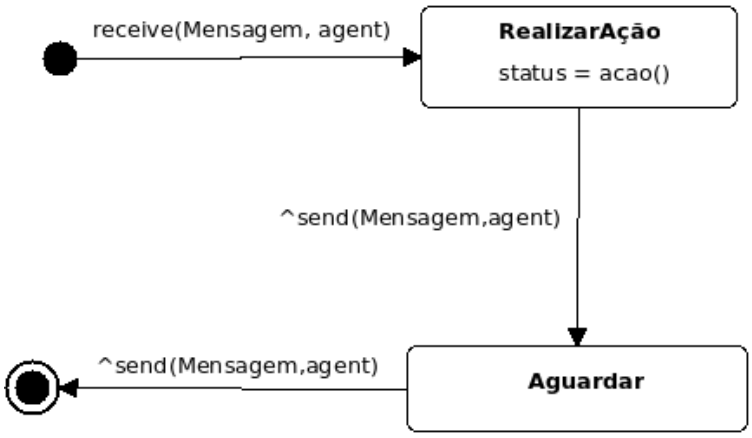

Figura 2.4: Representação utilizada no *Concurrent Task Diagram*.

condição e a mudança de estado só é efetivada após a validade desta condição. Por fim ocorrem as transmissões *transmission*, que podem conter o parâmetro *s*.

Para comunicações externas, dois eventos especiais foram definidos: O evento *send* indicando o envio de mensagem e o evento *receive*, indicando o recebimento de mensagem.

A mensagem sempre possui um cabeçalho performativo (definida pela FIPA, representa o objetivo da mensagem) com a seguinte sintaxe:*performative(p1...pn)*, onde *p1...pn* indicam os parâmetros da mensagem.

Após a definição dos *Concurrent Tasks Diagrams*, o analista pode combinar tarefas, a fim de diminuir a complexidade do sistema. Com isso, o sistema já possui a complexidade das regras determinadas, bem como as tarefas necessárias para atingir seus objetivos.

2.3.2 Design

A fase de design é divida em quatro passos: Criar Classes dos Agentes, Construir Conversações, Montagem dos Agentes e Design do Sistema. O principal objetivo desta fase é projetar os agentes e suas interações de acordo com os insumos construídos na fase anterior.

Criando as Classes dos Agentes

Neste passo, os agentes são criados com base nas regras definidas na fase anterior. Para cada agente criado deve existir pelo menos uma regra associada, caso contrário o levantamento de regras mostra-se incompleto. Dessa forma, o MASE garante que todas os objetivos do sistema são atingidos, já que as regras do sistema estão relacionados com as metas que foram levantadas na etapa anterior.

Ao fim deste passo é necessária a criação de um novo diagrama, o *Agent Class Diagram* representado na Figura 2.5. Nele, as classes dos agentes são associadas com as regras levantadas e as comunicações entre as classes.

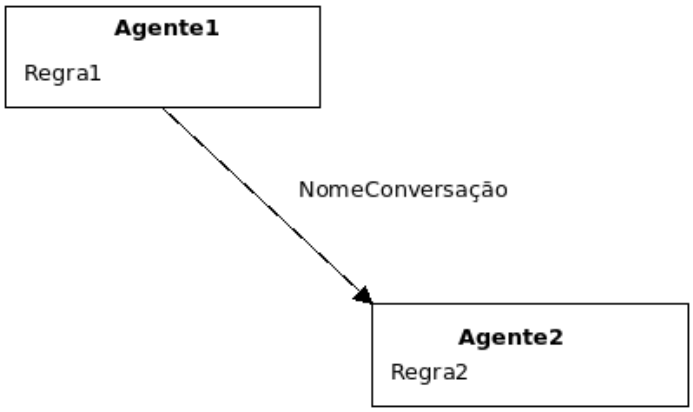

Figura 2.5: Representação utilizada no *Agent Class Diagram*.

Construir Conversações

A próxima etapa da segunda fase diz respeito às conversações que existirão no SMA. Cada detalhe da conversação entre dois agentes deverá ser planejada para que haja a correta comunicação entre os agentes e a quantidade de erros seja minimizada.

Quando o agente recebe uma mensagem, ele automaticamente compara com as suas conversações ativas [34]. Caso exista alguma conversação semelhante, o agente muda o seu estado e realiza as ações necessárias para atingir esse estado. Caso contrário, é assumido que o agente emissor deseja iniciar uma nova conversação e o receptor compara com as suas possibidades de tipos de conversação disponíveis para participar.

Neste passo, é criado o Diagrama de Comunicação. Nele, as conversações são montadas por meio do mesmo autômato de estados finitos, utilizado na fase análise, passo *refinar regras*. As transições de estado são as conversações que são feitas pelo agente.

A Listagem 2.6 apresenta a sintaxe utilizada na conversação de agentes. A sintaxe indica que a expressão é definida por meio das mensagens recebidas, condições, ações e mensagens transmitidas. O *token rec-mess* indica que a mensagem com os parâmetros *args1* foi recebida caso a condição *cond* seja verdadeira. Então o método *action* é chamado e a mensagem *trans-mess* com os argumentos *args2*. Todos os elementos da mensagem são opcionais.

Listagem 2.6: Sintaxe da conversação entre dois agentes.

```
rec-mess(args1) [cond] / action ^ trans-mess(args2)
```

O diagrama de comunicação apresentado na Figura 2.6 é definido para os dois lados da conversação: Iniciador e Receptor. considerando o lado iniciador da conversação.

Montagem dos Agentes

Neste passo, o estado interno dos agentes é criado. É necessário nesta etapa definir a arquitetura dos agentes e a sua composição. Para tanto, é necessário criar o Diagrama de Arquitetura de Agentes, onde são representados os componentes arquiteturais dos agentes.

Os componentes internos dos agentes podem ser atributos dos agentes, ou métodos auxiliares. Um agente pode ter vários componentes internos e estes componentes obviamente se comunicarão.

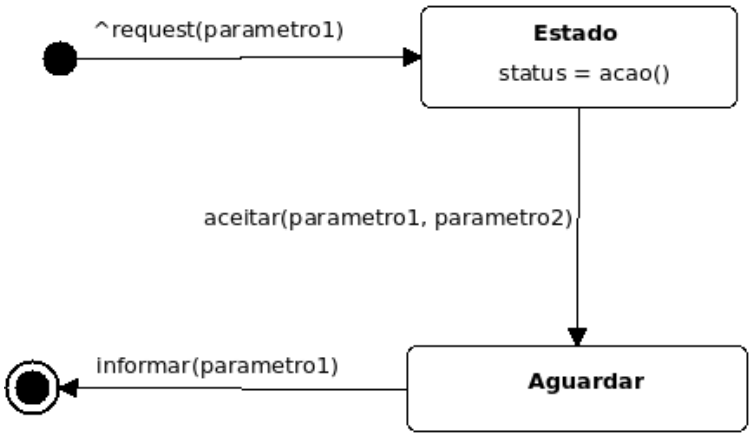

Figura 2.6: Exemplo de conversação utilizada no Diagrama de Comunicação do lado do iniciador da conversação.

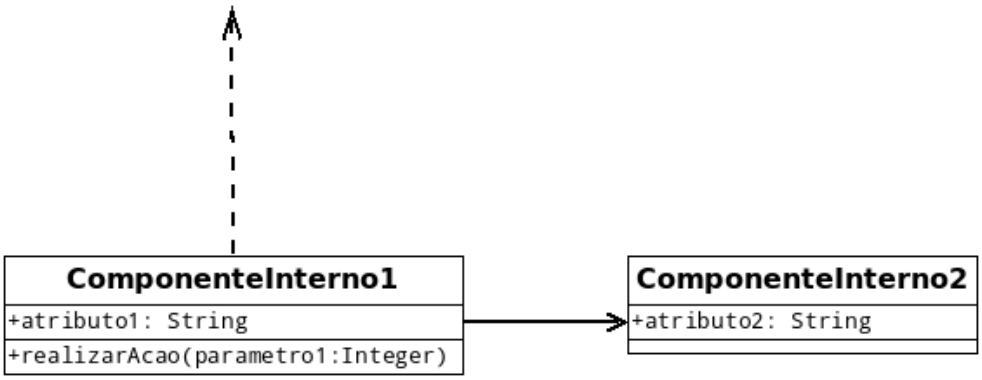

Figura 2.7: Notação utilizada na arquitetura de agentes.

No diagrama de arquitetura de agentes apresentado na Figura 2.7, a linha tracejada significa interação com sistemas externos, enquanto a outra significa interação com outros agentes do sistema.

As setas tracejadas indicam dependência com módulos externos ao agente, enquanto as outras representam dependência entre componentes.

Design do Sistema

A fase final da metodologia MASE consiste na criação de um diagrama de *deploy* dos agentes. Este diagrama consiste na representação do número de agentes que serão criados por cada máquina da aplicação, bem como suas localizações no sistema.

A Figura 2.8 apresenta o diagrama de *deploy*, expressando os ambientes criados e seus agentes internos. O retângulo com linhas pontilhadas representam os ambientes que podem existir na aplicação. Os outros retângulos representam os agentes e as linhas que os ligam representam as suas interações.

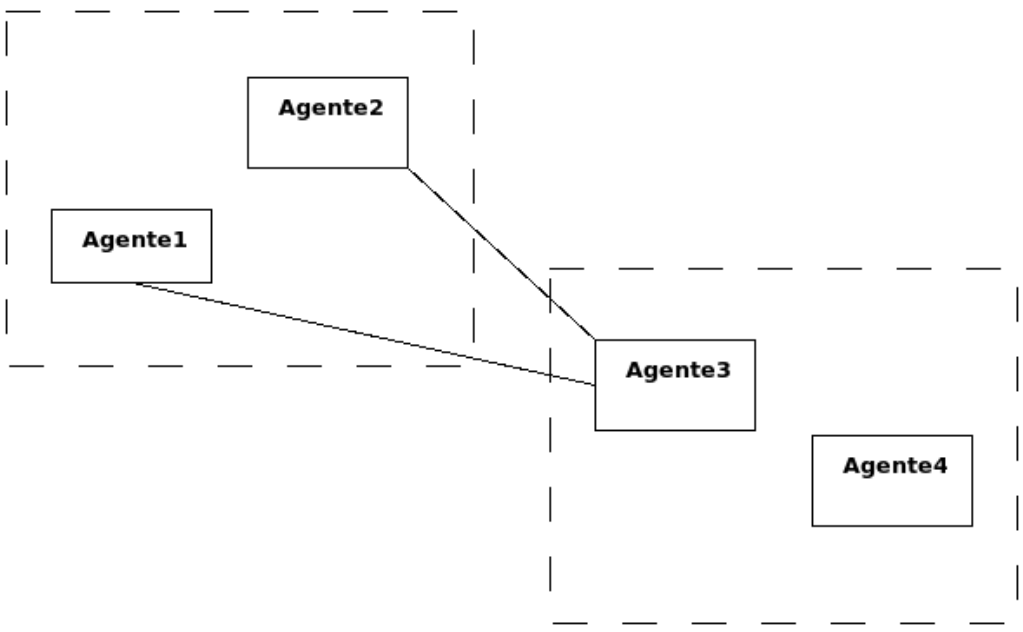

Figura 2.8: Notação utilizada no diagrama de deploy.

Ferramenta *AgentTool*

O *AgentTool* é uma ferramenta que dá suporte à metodologia MASE para a modelagem de um SMA. A versão 1.8.1 dá suporte à todas fases e passos do MASE [3], provendo suporte para a criação de diagramas, detalhamento de conversações e casos de uso simplificados.

A Figura 2.9 apresenta a tela principal onde é possível visualizar as principais abas disponíveis. Ela mostra a tela para desenvolvimento do diagrama de metas hierarquizado. O avanço ou regresso do desenvolvimento é percebido durante o preenchimento das abas, que é feito de forma sequêncial da esquerda para a direita, acompanhando todos os passos da metodologia.

Conforme o avanço da modelagem do SMA, algumas abas serão habilitadas de acordo com a seleção dos objetos nos diagramas. Por exemplo, cada comunicação de tarefas no diagrama de regras deve possuir um detalhamento. Dessa forma, a aba *Task Panel* torna-se automaticamente visível ao usuário quando há esse tipo de interação.

2.4 Estudos de Ferramentas e Tecnologias Relacionadas

Nesta seção estão reunidos as tecnologias que auxiliares ao desenvolvimento do SMA. A primeira seção explica alguns diagramas que são definidos na *Unified Modeling Language* (UML), necessários para entender os passos do MASE.

Em seguida é explicado os conceitos relacionados ao *middleware* JADE, bem como a sua arquitetura, o funcionamento dos agentes e os agentes que possuem interface gráfica.

Por fim, esta seção disserta a respeito da ferramenta JBoss Seam, responsável por interligar os principais *frameworks* JAVA e facilitar o desenvolvimento de aplicações *web* dinâmicas.

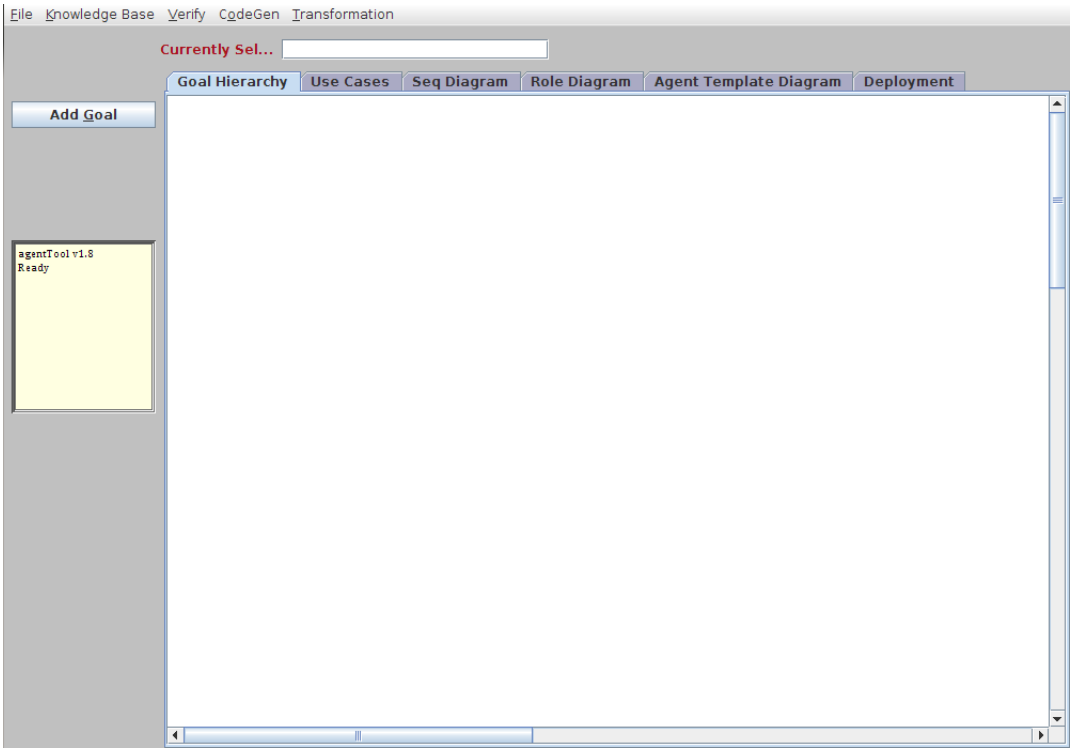

Figura 2.9: Interface da ferramenta *AgentTool*.

2.4.1 UML

A Linguagem Unificada de Modelagem, UML é uma linguagem visual que foi desenvolvida para a representação do software por meio de imagens objetivando o entendimento dos artefatos de forma rápida e clara, resultando em uma semântica para o projeto em questão. Segundo [22] o UML faz parte de uma família de notações gráficas que ajudam na descrição e concepção de sistemas de software, principalmente em sistemas concebidos utilizando o paradigma da orientação à objetos (OO).

O UML é um padrão não proprietário, controlado pelo consórcio *Object Management Group* [10]. Seu nascimento é datado em 1997 [22], surgindo a partir da união de diversas linguagens e ferramentas de surgiram na década de 80 e 90. A linguagem ajuda o entendimento de como o software foi projetado, como ocorre a comunicação entre seus objetos, como suas classes são organizadas, quais são os atores que são envolvidos na utilização do software, dentre outras possibilidades de representação.

Segundo [22] é possível separar o uso do UML de três formas distintas de modelagem conforme o objetivo de uso: Rascunho, planta de software e como linguagem de programação.

A utilização como rascunho é facilita a comunicação entre as pessoas envolvidas no projeto, sejam desenvolvedores discutindo funcionalidades do software ou gestores explicando funcionalidades em alto nível. O objetivo neste uso é a comunicação de alguns aspectos do sistema de forma rápida, sem a necessidade de formalizar artefatos para o projeto.

A utilização do UML como planta de software são documentos detalhados que são criados para documentação do software, sendo divida em duas sub-categorias: Engenharia

reversa e engenharia normal. Na engenharia reversa, os diagramas são gerados a partir de uma ferramenta que faz a leitura do código fonte e gera os diagramas desejados, que são utilizados para auxiliar o leitor no entendimento do sistema. Na engenharia normal, a idéia é modelar o sistema detalhadamente antes de qualquer desenvolvimento, prevendo quais serão os módulos do sistema, bem como a sua a comunicação.

No uso como linguagem de programação, o UML é utilizado para geração de código executável por ferramentas avançadas de modelagem. Esse modo requer a modelagem de estado e comportamento do sistema, para fins de detalhar todo o comportamento e lógica do sistema em código.

Diagramas UML

O UML descreve 13 tipos de diagramas, conforme a Figura 2.10, os quais podem ser categorizados como estruturais e comportamentais. Apesar da grande quantidade diagramas envolvidos no UML, de acordo com [29], a maioria dos processos de desenvolvimento de software utilizam-se de poucos modelos devido à maior significância de alguns. A metodologia de desenvolvimento de SMA utiliza-se apenas dos seguintes diagramas:

- Diagrama de Caso de uso;
- Diagrama de Sequência.

Diagrama de Caso de Uso Casos de uso são relatos textuais que são utilizados para descobrir e descrever os requisitos do sistema. Consiste da descrição de como um ator utiliza uma funcionalidade do sistema para atingir algum objetivo relacionado. Em [29], os casos de uso devem ser prioritariamente desenvolvidos de forma textual e o seu respectivo diagrama deve ser desenvolvido de forma secundária, somente para ilustrar o relato textual.

Um dos objetivos do caso de uso é a facilidade do levantamento dos requisitos, tanto para os analistas de um sistema, quanto para os clientes envolvidos. A definição de uma modelagem em comum facilita o entendimento das partes interessadas, tornando o caso de uso uma boa maneira de simplificar o entendimento do comportamento do sistema [29], bem como envolver todos as partes interessadas(*stakeholders*) na sua construção. Em [18], o caso de uso é um contrato de como será o comportamento do sistema. Este contrato será feito por meio dos atores que existirão, da sua interação com o sistema, bem como os cenários existentes.

Duas definições fazem-se necessárias para o entendimento do caso de uso. A primeira delas é o "Ator" do caso de uso. Ele é um objeto com um comportamento definido no sistema. É possível definir o ator como uma pessoa, organização ou mesmo o próprio sistema (quando utiliza serviços do próprio sistema), desde que tenham sempre um papel relacionado. Existem três tipos de atores relacionados ao sistema:

- Ator Principal: Seus objetivos são satisfeitos por meio da utilização do sistema.
- Ator de Suporte: Fornece algum serviço para o sistema.
- Ator de Bastidor: Expressa algum interesse pelo comportamento do caso de uso.

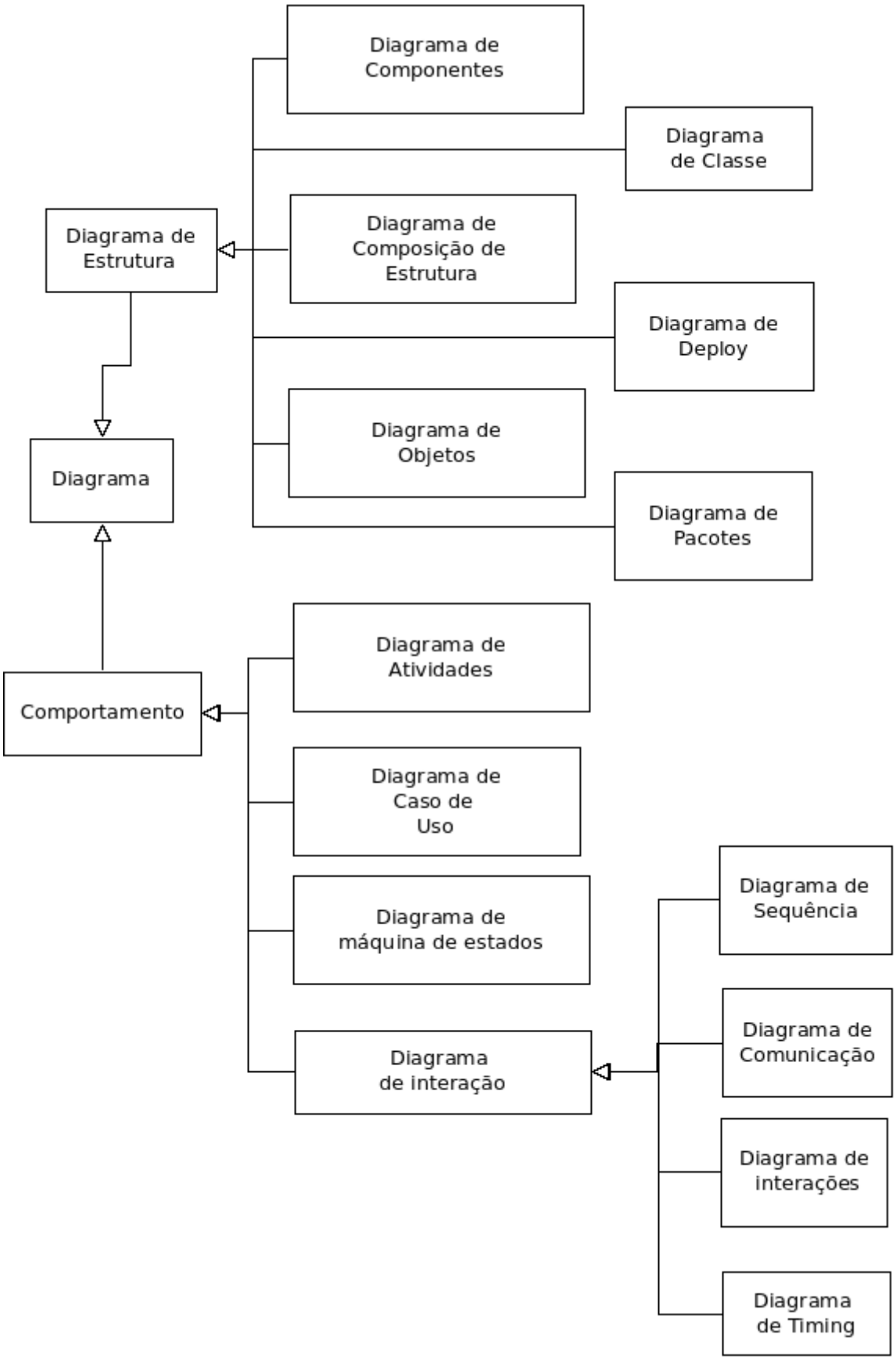

Figura 2.10: Categorização dos Diagramas UML 2.0 [22].

Tabela 2.6: Estruturação Detalhada de Caso de Uso

Seção do Caso de Uso	**Significado**
Nome do Caso de Uso	Nome do caso de uso, iniciando-se com um verbo.
Escopo	Escopo descrito pelo caso de uso.
Nível	Podem ser níveis de objetivo de usuário (quando descrevem os cenários para atingir o objetivo do usuário) ou nível de subfunção (subpassos para dar suporte a um objetivo de usuário).
Ator Principal	O ator que procura os serviços para atingir seus objetivos.
Interessados e Interesses	Significado.
Pré-Condições	Condições que antecedem o caso de uso e são necessárias para atingir os objetivos.
Garantia de Sucesso	Objetos que podem ser analisados após a execução do caso de uso a fim de validar a correta execução do sistema.
Cenário de Sucesso Principal	Chamado também de fluxo básico, este cenário descreve o fluxo principal do sistema que satisfaz os interesses dos *stakeholders*.
Extensões	Chamado também de fluxos alternativos, são fluxos auxiliares ou cenários de erros que são relacionados ao cenário de sucesso principal.
Requisitos Especiais	Registram requisitos não funcionais do sistema e que estão relacionados com o caso de uso.
Lista de Variantes Tecnologias de Dados	Listagem de dificuldades técnicas, desafios técnicos que valem a pena registrar no caso de uso.
Frequência de ocorrência	Frequência de ocorrência deste caso de uso.

A segunda definição envolvida é a de cenário. Um cenário é uma sequência de interações entre os atores e o sistema. Os cenários são separados por ações de interesses de atores. Logo o caso de uso pode ser considerado como um conjunto de cenários de interações de atores com o sistema.

Dessa forma, o caso de uso deve deixar claro os requisitos funcionais do sistema, bem como o seu comportamento. Existem três formas de se escrever um caso de uso, diferindo em seu nível de formalidade e formatos: Resumido, informal e completo. Este trabalho usará o nível de caso de uso completo, devido ao nível de detalhe de sua estruturação. Os casos de uso são estruturados de diversas formas. De acordo com [29], o padrão de estruturação mais adotado foi desenvolvido por Alistar Cockburn [18], descrito na Tabela 2.6.

A diagramação do caso de uso por UML é uma forma de representação do sistema, mostrando fronteiras do sistema, comunicação e comportamento entre os atores. A Figura 2.11 representa a sugestão de diagramação visual do caso de uso, propondo forma de representação de atores, casos de uso e atores auxiliares.

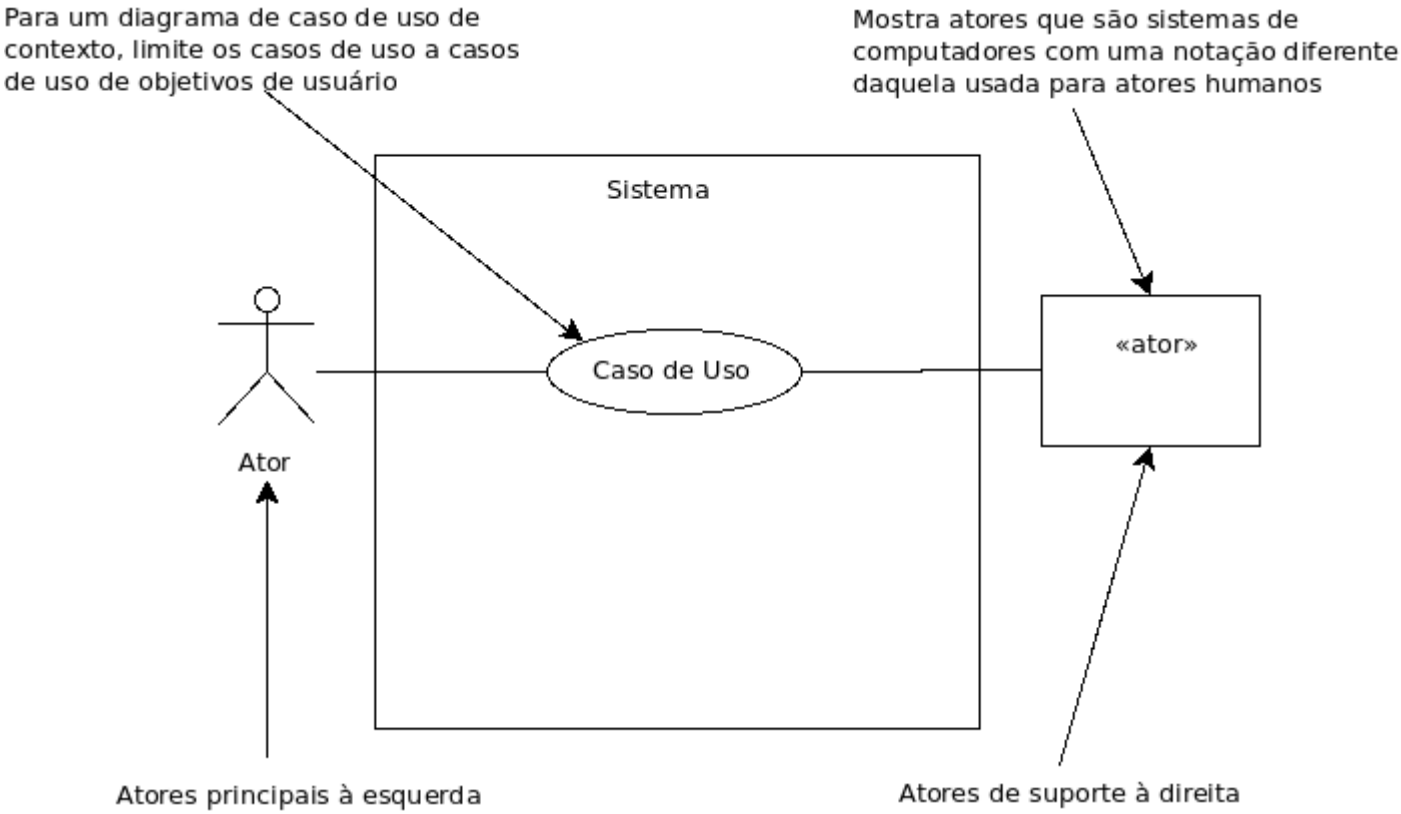

Figura 2.11: Sugestões de notação de caso de uso proposto por [29]

Diagrama de Sequência O Diagrama de Sequência é um documento criado para ilustrar os eventos descritos em um caso de uso de forma sequencial e temporal, mostrando a interação de atores externos ao sistema e os eventos que eles geram durante essa interação. O UML possui uma notação específica para este diagrama, onde todos os elementos são representados.

Neste diagrama, são representados para cada cenário do caso de uso os eventos que os atores geram, bem como a ordem da sua interação. No diagrama os atores são representados na parte superior (com a mesma notação do diagrama de caso de uso). Abaixo dos atores é apresentada a linha de tempo, crescendo de cima para baixo.

Durante a interação do ator com o sistema, eventos de sistema são gerados e iniciam toda a execução do cenário do caso de uso, ou operação do sistema. A execução dos eventos ocorre até o último evento cronológico na linha do tempo. Os eventos gerados pela interação entre os atores ocorrem na linha do tempo de forma cronológica e ordenada com a mesma ordem dos eventos do cenário do caso de uso.

A nomenclatura dos eventos deve sempre iniciar com um verbo, podendo ser seguida de um substantivo. Além disso, deve-se sempre expressar a nomenclatura em níveis genéricos verbais, nunca detalhando a funcionalidade do sistema.

A Figura 2.12 ilustra a notação de diagramas de sequência, onde é possível identificar a interação entre um ator e uma entidade do sistema. É fácil notar que os eventos estão ocorrendo de forma ordenada de cima para baixo.

O primeiro evento é iniciado pelo ator, onde o método (listarProdutos()) da Entidade é invocado. Esse método gera a resposta (produtos) para o ator. A interação seguinte

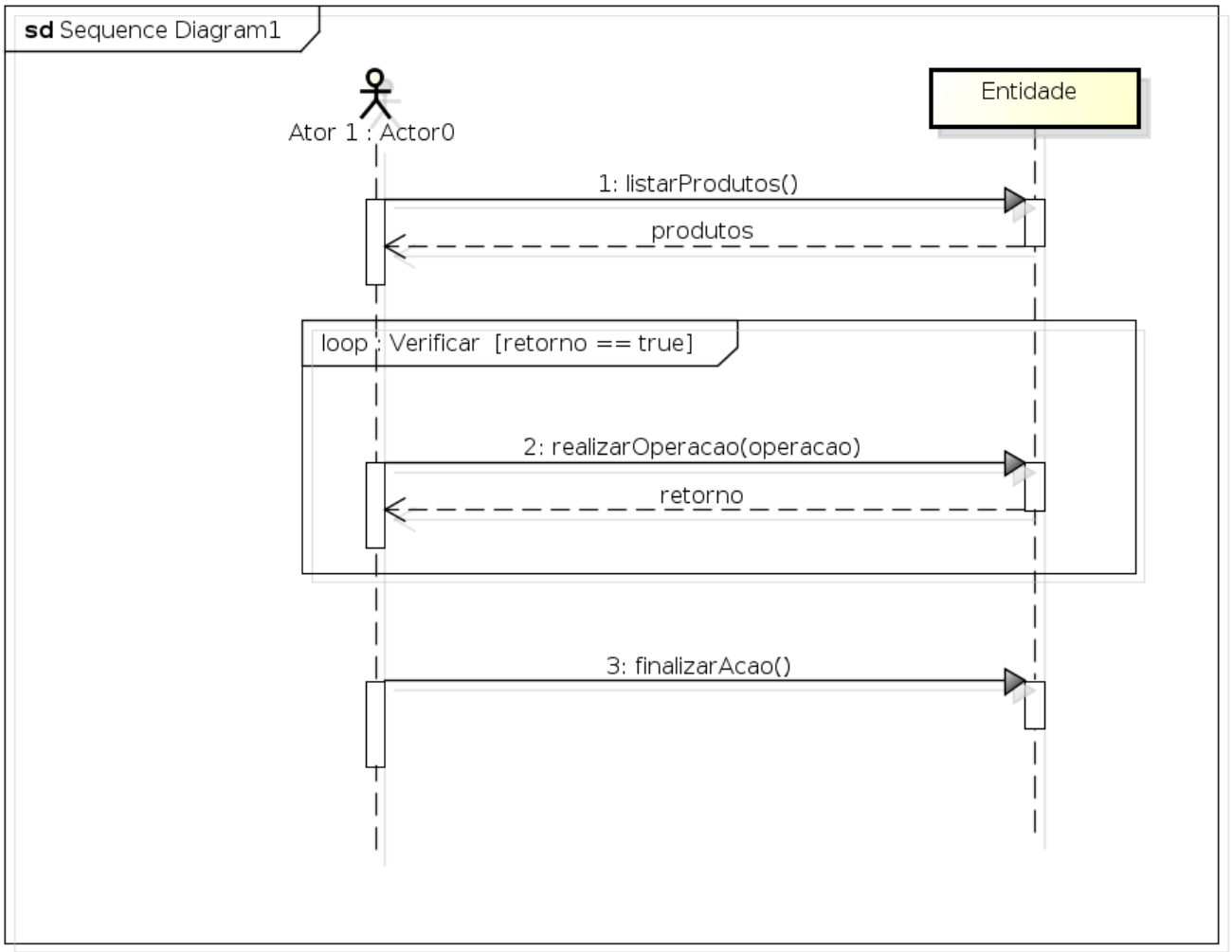

Figura 2.12: Notação de diagrama de sequência, exibindo a comunicação de um ator e uma entidade (sistema)

acontece dentro de um retângulo, do tipo *loop* e de nome "Verificar". Esse retângulo permite que o diagrama de sequências represente um loop, onde todos os eventos serão repetidos enquanto a condição de guarda for verdadeira, no caso do exemplo, enquanto retorno for igual a *true*. O UML permite diversos operadores além do loop, como por exemplo a negação, a assertiva, o *break*, dentre outros.

Dentro do retângulo, o ator chama o método *(*realizarOperacao), enviado o parâmetro *operação*. A entidade retorna um valor, que será testado na guarda para a continuidade da conversação. Por fim, o ator chama o método *finalizarAcao* da entidade. Por não haver outra interação em seguida, o cenário é considerado como encerrado.

A importância do desenho de um diagrama de sequência está no fato de detalhar os eventos do sistema que são gerados pela interação de atores externos, pois assim é possível ter uma análise comportamental do sistema com base nesses eventos. Neste nível de análise, é possível estudar o comportamento do sistema no nível de projetar o que ele faz, porém sem necessariamente explicar como o faz [29].

O diagrama de sequência geralmente está relacionado com o caso de uso, primeiramente pelo fato de descrever um cenário de caso de uso, mas também pelo fato de o caso de uso conter todos os detalhes do cenário. O diagrama de sequência apenas deixará claro a interação entre os atores e os eventos derivados dessa interação.

2.4.2 Ferramentas de SMA

Os *frameworks* projetados para o desenvolvimento de SMA visam simplificar o desenvolvimento de aplicações ao implementar uma infra-estrutura básica que provê recursos ao programador. O seu objetivo é garantir os recursos que são essenciais em um SMA, possibilitando o foco do programador no desenvolvimento da aplicação. Diversas ferramentas foram criados para a criação de SMA, cada um com características distintas.

O *AgentMom* [8] é um *framework* de comunicação que foi criado na linguagem JAVA para o desenvolvimento de agentes, conversações e a passagem de mensagens simples nestas conversações. Possui a capacidade de criação de grupos específicos para envio de mensagens (*multicast*) e encriptação do conteúdo da mensagem.

O *MaDKit* [9] é uma ferramenta para desenvolvimento SMA também desenvolvido em JAVA, onde é possível definir uma plataforma pelo modelo organizacional, agrupando agentes e determinando regras para este grupo.

O *Java Agent Development Framework* (JADE) é um *middleware* criado para facilitar o desenvolvimento de SMA. Desenvolvido inicialmente pela Telecom Italia e desde 2000 é um projeto open source. Seu desenvolvimento de baixo acoplamento permite que seja possível integrar várias bibliotecas auxiliares (*addons*) para facilitar o desenvolvimento de aplicações.

Segundo [16], ele foi desenvolvido seguindo todas as especificações da FIPA, o que garante uma intercomunicação com outras plataformas. O JADE foi desenvolvido na linguagem JAVA, possibilitando o uso de diversas bibliotecas e frameworks desenvolvidos na linguagem. O middleware provê várias funcionalidades básicas que abstraem e simplificam o desenvolvimento de aplicações, com o objetivo do desenvolvedor estar mais preocupado com o comportamento do agente do que com a infra estrutura da plataforma.

Este trabalho optou pela escolha da ferramenta JADE, pois no contraste com as outras ferramentas, foi possível destacar alguns aspectos. O *framework MaDKit* não permite

o desenvolvimento de agentes com comportamento e interação complexos. Tanto o *framework AgentMon* quanto o *MaDKit* não possuem a conformidade das suas mensagens com linguagem ACL, especificado pela FIPA. Dessa forma, as duas ferramentas foram descartadas.

Dada a escolha da ferramenta, é necesário entender as particularidades do JADE. O sistema de mensagem no JADE funciona de forma assíncrona. Um agente não precisa estar necessariamente disponível para receber as mensagens, visto que elas são enfileiradas e processadas em ordem. Além disso, não é necessário que um agente tenha uma referência para outro agente a fim de comunicar-se.

A arquitetura de um SMA desenvolvido em JADE funciona de forma semelhante à rede P2P (*Peer-to-Peer*), onde cada agente possui um nome único na plataforma - Agent ID (AID) - e é livre para entrar e sair a qualquer momento durante a execução. Uma plafatorma JADE possui normalmente os seguintes elementos:

- Agent Management System (AMS) é o agente responsável por supervisionar toda a plataforma e criar um elo entre os agentes. Esse tipo de serviço é chamado de *white pages* e indexa todos os agentes da plataforma.
- Directory Facilitator (DF) é o agente responsável por registrar todos os serviços e prover a funcionalidade de busca para os agentes. Este tipo de serviço é chamado de *yellow pages*.

Os agentes executam em threads separadas garantindo o não compartilhamento de recursos para evitar condições de corrida. A plataforma é responsável também por manter o ciclo de vida dos agentes. Durante a criação dos agentes, eles são automaticamente registrados no serviço de *white pages*

A mobilidade de agentes entre máquinas também é feita de forma transparente pelo JADE. Em [16], a mobilidade do agente pode transportar o estado do agente (sob certas condições) entre processos e máquinas.

Uma das características mais importantes do *middleware* é o suporte nativo à ontologias e linguagens de comunicação FIPA. As ontologias podem ser modeladas (expressando ações, conceitos e predicados) de forma simples e clara, além de ser possível restringir as ontologias à determinados agentes que devem a conhecer.

Arquitetura

A Figura 2.13 representa o ambiente de execução JADE, ilustrando os principais elementos da arquitetura distribuídos em três máquinas (dois servidores e um *desktop*). Cada máquina possui um container e vários agentes.

O container principal(*main container*), presente na máquina do centro, possui algumas diferenças entre os outros, chamados de *Container-1* e *Container-2*, abringando agentes primordiais na plataforma que atendem às especificações da FIPA (número 23 e 61 [15]). O container principal possui o componente *container table* (CT), que possui as referências para os objetos e os endereços de comunicação dos nós que compõe a plataforma.

Além disso, o container principal possui uma tabela global para descrição de agentes (GADT) que registra todos os agentes da plataforma e o seu endereço. Os outros containeres possuem uma cópia em cache da tabela para, caso a tabela principal seja corrompida, possa ser substituída.

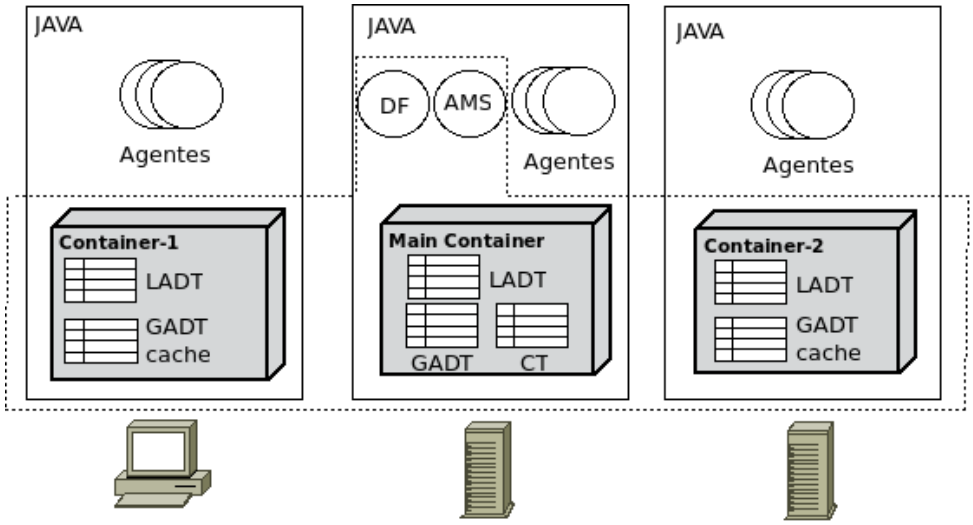

Figura 2.13: Representação da arquitetura principal do JADE.Adaptado de [16].

O primeiro deles é o *Directory Facilitator* (DF), agente responsável por registrar todos os serviços disponíveis na plataforma. O segundo agente é o *Agent Management System* (AMS) é o agente responsável por supervisionar toda a plataforma, registrando os agentes que estão rodando, bem como o seu estado.

Implementação dos Agentes

O JADE define a classe abstrata *Agent*, que é a base para todos os agentes definidos. O desenvolvedor tem apenas o trabalho de extender esta classe e implementar o comportamento no método *setup()*. O fato de extender a classe abstrata implica em herdar várias características já definidas pelo JADE (registro, configuração, etc.) e métodos que podem ser chamados para a implementação do comportamento do agente.

Cada agente possui um identificador único (Agent ID - AID) que identifica o agente em toda plataforma. Por padrão, o formato do AID possui primeiramente o nome do agente seguido do caracter '@', por fim o endereço da plataforma onde o agente está. Este AID é atribuído durante o registro do agente no AMS. Neste registro, é possível também registrar os serviços do agente no DF.

A implementação do método *setup()* é invocado durante o registro do agente e deve estabelecer um comportamento para ele. Este comportamento diz respeito à ação que será realizada pelo agente durante a ocorrência de um evento. O registro/cancelamento dos comportamentos é feito pelos métodos exibidos na Listagem 2.7.

Listagem 2.7: Exemplo de registro de comportamento nos agentes.

```
void addBehaviour( Comportamento )
void removeBehaviour( Comportamento )
void addSubBehaviour( Comportamento )
void removeSubBehaviour( Comportamento )
```

Os comportamentos são separados em primitivos e compostos. A diferença entre ambos é a possibilidade dos comportamentos compostos poderem agregar vários outros

comportamentos simples ou compostos, sendo eles: *ParallelBehaviour, SequentialBehaviour*. De maneira distinta, os comportamentos primitivos tem relação direta com o tempo, acontecendo durante um período de espera ou após o envio de uma mensagem. São eles: *SimpleBehaviour, CyclicBehaviour, TickerBehaviour, OneShotBehaviour, WakerBehaviour* e *ReceiverBehaviour*.

Ciclo de Vida dos Agentes

O JADE implementa o ciclo de vida especificado pela FIPA. Os possíveis estados da plataforma são:

- *INITIATED* - Após a criação do objeto, antes do registro do objeto no AMS, o agente assume o estado de iniciado. Este estado significa que o agente ainda não está disponível para a execução de ações na plataforma.
- *ACTIVE* - Neste estado o agente é registrado no AMS, possuindo assim o AID e o endereço. Ele está pronto para a execução do trabalho na plataforma.
- *SUSPENDED* - O agente está com as atividades suspensas e está em modo ocioso.
- *WAITING* - O agente está bloqueado esperando algum evento acontecer para executar alguma ação. Tipicamente, este estado é usado para fazer o agente esperar por alguma mensagem.
- *DELETED* - O agente está destruído e sua thread de execução é terminada. O seu registro será removido do AMS e a sua referência removida da JVM.
- *TRANSIT* - O agente está movendo-se de uma plataforma para uma nova localização. Mesmo em transito, é possível enviar mensagens para este agente, visto que serão empilhadas na sua fila de mensagens e posteriormente processadas quando ele assumir o estado ACTIVE.

Para cada uma das transições de estados existem métodos que são invocados em um momento anterior. Eles são úteis para a execução de ações que antecedem a mudança de estado. Por exemplo: Durante a mudança do estado ACTIVE para DELETE, o método *doDelete()* é encarregado de implementar ações que antecedam o fim do agente, como que o agente desfaça o registro dos seus serviços no DF.

Interface Gráfica

O JADE permite o desenvolvimento de agentes com suporte à interface gráfica. Dessa forma, é possível desenvolver uma interação simples do agente com o ser humano.

Por padrão, o JADE utiliza diversos agentes que utilizam-se de interfaces gráficas para a comunicação dos humanos que, dentre outras funcionalidades, permitem o envio de dados, controle do agente e testes da plataforma. Ferramentas como gerência dos *containers*, visualização do DF, criação de mensagens a partir de agentes falsos (*sniffers*), dentre outros são disponibilizados nativamente para o desenvolvedor. Na Figura 2.14 é apresentada a interface básica do agente RMA, ilustrando o ambiente de execução. É possível identificar apenas o *container* principal da aplicação. Nele, existem os três agentes principais registrados descritos anteriormente: AMS, DF e o próprio RMA.

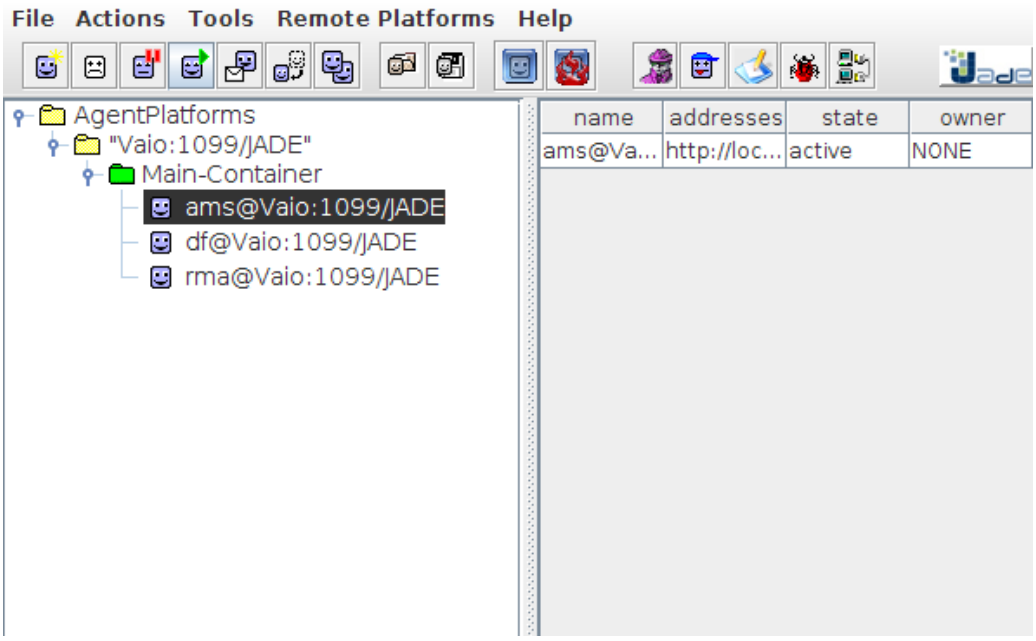

Figura 2.14: Apresentação da Interface do agente RMA.

2.4.3 JBoss Seam

No mundo corporativo do JAVA, muitos frameworks são responsáveis por partes específicas de uma aplicação. Seguindo as especificações propostas para a plataforma (JSR), as aplicações implementam integrações com o banco de dados (Hibernate, JPA), integração com a camada de visualização (Struts 1 [1] e 2 [2], JSF [5]), contextos Java e injeção de dependência [6]. Porém a integração destes frameworks nem sempre é trivial, demandando muito tempo dos desenvolvedores para a correta configuração.

Neste aspecto surge o *JBoss Seam*. Ele é um *framework* que reune as principais tecnologias de desenvolvimento *web* na linguagem Java. Ele integra as tecnologias *Asynchronous JavaScript and XML* (AJAX), *JavaServer Faces* (JSF), *Java Persistence* (JPA), *Enterprise Java Beans* (EJB 3.0) e *Business Process Management* (BPM) em uma única ferramenta que objetiva o desenvolvimento ágil de aplicações e o foco do programador na lógica de negócio [11].

A pilha de aplicações do Seam, representada na Figura 2.15, contém todos as tecnologias que são utilizados pelo *framework*. Caso o desenvolvedor deseje utilizar outras tecnologias, o Seam provê configurações para que outras ferramentas possam ser integradas facilmente à aplicação.

Em [12], uma das principais ferramentas do Seam é o *seam-generator*. Com ele, é possível gerar uma estrutura básica de projeto, com arquivos de construção, bibliotecas compatíveis e as configurações necessárias para o início do desenvolvimento e o *deploy* em um servidor de aplicação.

Além disso, uma das grandes vantagens do *seam-generator* é a geração automática de código, criando a partir de uma tabela no banco de dados todas as operações necessárias para a visualização, inserção, exclusão e atualização de dados (CRUD), diminuindo assim o tempo de desenvolvimento de aplicações.

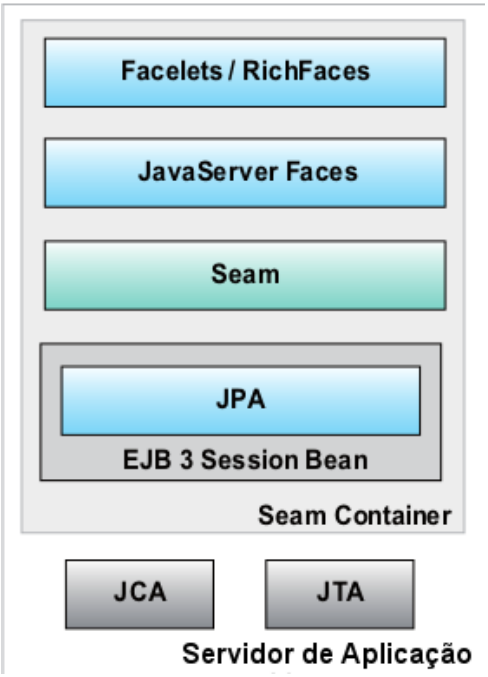

Figura 2.15: Representação da pilha de aplicações do Seam [12].

Portanto este trabalho optou por utilizar o *JBoss Seam*, pois engloba outras importantes ferramentas em uma única solução, facilitando assim o desenvolvimento da aplicação *web*. A facilidade pode ser constatada a partir dos aspectos já mencionados, como integração nativa entre as ferramentas da sua pilha de aplicações e geração automática de código.

2.5 Trabalhos Correlatos

É possível fazer comparações deste trabalho com diversas áreas de conhecimentos correlatos. Os trabalhos presentes nesta seção foram selecionados a partir de um levantamento que buscou soluções para melhorias na educação com o apoio de tecnologia. Estes trabalhos estão relacionados com a área de atuação deste trabalho, seja em IA, Estilos de Aprendizagem, SMA ou AVA.

Na área de auxílio de alunos, alguns trabalhos assemelham-se com o propósito deste trabalho. A implementação de um agente artificial integrado à um tutor inteligente [37] que identifica dificuldades de aprendizagem por meio de técnicas de IA. Ele prima pela agradável interação com humanos com o intuito de facilitar a compreensão da ferramenta. Sua abordagem porém não é voltada para a abordagem multidimensional de inferência do aluno, bem como a solução não prevê a inferência de dados vindos de outros AVA.

O trabalho [14], na área de identificação de conhecimentos, visa a determinação de estilos de aprendizagem dos alunos por meio da plataforma chamada "Ferramenta de Identificação de Perfis de Aprendizes – FIPA", uma aplicação multicamadas que faz a identificação do perfil por meio de questionários de estilos de aprendizagem. Apesar da inferência do estilo de aprendizagem, esta plataforma utiliza uma abordagem distinta deste trabalho, sendo uma arquitetura cliente-servidor.

Esta arquitetura não possui alguns benefícios da abordagem SMA, conforme [25]: A possibilidade de representação de entidades individuais como, no domínio deste trabalho, a modelagem do conhecimento de cada aluno por meio de agentes. Outra característica é a representação das suas interações por meio de equações, regras ou operações lógicas que podem flexibilizar a modelagem. Por fim, modelos baseados em agentes são recomendáveis para simulações devido as suposições realisticas inerentes a seu domínio.

O trabalho [24] utiliza-se da abodagem SMA para a adaptação do ensino na internet. A adaptação ocorre conforme as características individuais de cada aluno e pode modificar características como a estratégia de ensino, seleção de materiais que são mais adequados ao aluno de acordo com o seu perfil. Este ambiente assemelha-se bastante com o presente trabalho, porém o seu foco não é na modelagem multidimensional do aluno.

A modelagem multidimensional tem por objetivo a separação clara das dimensões do aluno para restringir o domínio de atuação de cada agente. Dessa forma, cada a gente expressará uma quantidade menor de propriedades de cada dimensão, sendo possível obter uma informação mais precisa e expressiva de cada dimensão, conforme pode ser visto nas simulações em [25].

Por fim, o AVA EDULIVRE [32] possibilita a inferência implícita de modelos e perfis individuais de estudantes da educação infantil. Com o perfil de cada aluno e de todos os alunos em geral, o educando poderá desenvolver nortear seu trabalho da melhor forma possível. A abordagem deste trabalho não utiliza-se de agentes, diferindo assim do foco deste trabalho. Além disso, o EDULIVRE apresenta a estratégia de ser o AVA e não permite integração com outros ambientes.

Conforme exposto alguns trabalhos na área de IE utilizam de diversas tecnologias ou abordagens para a construção de ambientes de ensino-aprendizagem. Todavia, o capítulo seguinte apresentará uma proposta que tende melhorar o modelagem do perfil de aprendizagem por meio da definição de uma arquitetura específica baseada em SMA. Essa arquitetura apresenta aspectos que tangem o modelo multidimensional do aluno propondo uma divisão de agentes para inferência do perfil e aspectos da infraestrutura do ambiente de ensino-aprendizagem.

Capítulo 3

Proposta de Solução

O presente capítulo detalha a modelagem da solução e a arquitetura proposta. A primeira Seção 3.1 apresenta os diagramas que foram construídos a partir as duas fases da metodologia MASE. A segunda Seção 3.2 mostra a arquitetura da solução em um escopo mais geral, considerando além do SMA a plataforma *web* e a sua comunicação.

A arquitetura básica do sistema já havia sido previamente decidida de acordo com [19], bem como o nome da solução: *Frank*. Basicamente, a aplicação funciona da seguinte forma: O aluno deverá autenticar-se em uma plataforma *web* para a determinação do seu estilo de aprendizagem. Em seguida, o professor autentica-se na plataforma e verifica o estilo de aprendizagem dos seus aluno. A verificação do estilo de aprendizagem e manutenção do modelo multidimensional é feita por agentes. Logo, a solução proposta foi desenvolvida em duas aplicações: *Web* e SMA.

A aplicação SMA é baseada em grupos de trabalho por aluno. Após a autenticação do aluno na plataforma *web*, o SMA realiza a criação dos agentes do grupo de trabalho designado para ele. Esses agentes são responsáveis pela manutenção do modelo multidimensional do aluno na plataforma e são atualizados conforme ocorre o envio de dados dos ambientes externos para os agentes do grupo de trabalho.

Após a escolha da metodologia MASE, justificada no capítulo 2, foram levantados todos os requisitos e regras a partir de [19], onde foi compreendido inicialmente o problema. Em seguida, uma série de artefatos foram gerados em consonância com a metodologia até o último passo, a geração dos agentes. Ao fim da metodologia, a solução foi composta por sete diferentes tipos de agentes:

1. *GatewayAgent* - Agente responsável pela interface com a plataforma web que irá interagir com os alunos e docentes.

2. *WebServiceAgent* - Agente responsável pela interface com o Ambiente Virtual de Aprendizagem.

3. *ManagerAgent* - Agente responsável pelo controle da plataforma

4. *WorkgroupAgent* - Responsável pelos agentes cognitivo, afetivo e metacognitivo.

5. *CognitiveAgent* - Responsável pelo modelo cognitivo do aluno.

6. *MetacognitiveAgent* - Responsável pelo modelo metacognitivo.

7. *AfectiveAgent* - Responsável pelo modelo afetivo.

O primeiro agente, *GatewayAgent*, é responsável pela interação direta com a plataforma web. Essa plataforma irá interagir com os alunos (por meio de questionários de estilo de aprendizagem) e docentes (notificando o estilo de aprendizagem dos seus alunos) e encaminhará os dados para o agente em questão, que repassará ao ambiente.

O agente interface *WebServiceAgent*, é responsável pela comunicação com os web services do SMA Frank, que é a forma escolhida para futuras interações com AVA. Por fim, o agente *ManagerAgent* é responsável pela criação dos agentes.

A partir dos agentes e regras gerados, foi implementado o SMA na linguagem JAVA utilizando o *framework* JADE. Em seguida, foi desenvolvido a plataforma *web* utilizando *JBoss Seam*, responsável pela interface do grupo de trabalho de agentes com o aluno. Após a conclusão da plataforma, foi necessário integrar as duas aplicações por meio do módulo *JadeGateway* disponibilizada nativamente pelo JADE.

Por fim, a solução foi testada por meio de cenários que simulavam o uso por meio dos atores Aluno e Docente. A demonstração foi feita a partir de um aluno inferindo o seu estilo de aprendizagem e em seguida do docente visualizando o estilo de aprendizagem dos alunos de sua turma.

As seções seguintes detalham a metodologia, justificam a arquitetura da solução por meio da modelagem definida na metodologia MASE.

3.1 A Modelagem

A modelagem foi desenvolvida utilizando-se a ferramenta *agentTool*. A ferramenta possui meios para diagramar todas as fases e passos do MASE, auxiliando o analista em todos os diagramas necessários, além de gerar código automático para alguns frameworks de SMA.

A metodologia é dividida em duas fases: Análise e Design. Na primeira Seção 3.1.1 é desenvolvida a fase de análise responsável pelo levantamento de requisitos e entendimento das regras e tarefas. A segunda fase, na Seção 3.1.2, é responsável pela criação dos agentes, conversações e componentes internos.

3.1.1 Análise

A metodologia inicia-se com o passo de captura das metas. Para tanto, foi necessário primeiramente um levantamento inicial dos requisitos do SMA. Os requisitos foram levantados e compreendidos por meio da sua documentação inicial [19], onde é possível listar:

1. O sistema deve manter um modelo do estudante, onde será determinado o seu estilo de aprendizagem e será notíficado ao docente.

2. O sistema deve assistir (auxiliar) o aluno por meio de um grupo de trabalho.

3. O sistema deve fazer interface com Ambiente Virtual de Aprendizagem, a fim de estabelecer comportamentos do estudante.

4. O sistema deve criar uma modelagem cognitiva do aluno, onde são mantidas informações sobre o desempenho, de acordo sua interação em Ambiente Virtual de Aprendizagem, e informações a respeito do seu estilo de aprendizagem.

5. O sistema deve criar uma modelagem metacognitiva do aluno, onde são armazenadas informações com o intuito de melhorar processos de aprendizagem de domínios específicos.

6. O sistema deve criar uma modelagem afetiva do estudante, especificamente a respeito da modelagem da personalidade e emoções do estudante.

7. O sistema deve fazer interface com Ambiente Virtual de Aprendizagem.

8. O sistema deve refutar ou confirmar o estilo de aprendizagem do aluno a partir do desempenho relacionado à interação com o sistema e/ou com Ambiente Virtual de Aprendizagem.

9. O sistema SMA deve atualizar o modelo do estudante com base em inferências a partir dos registros de trabalho do estudante.

10. O sistema deve construir o modelo do estudante a partir de uma modelagem explícita, ou seja, a partir do feedback explícito do estudante (questionário).

11. O sistema deve construir o modelo do estudante a partir de uma modelagem implícita, ou seja, a partir do desempenho obtido nas ferramentas de aprendizado.

A partir destes requisitos, foi possível estabelecer metas que o sistema deveria atingir para satisfazê-los:

1. Manter um modelo do estudante.

2. Auxiliar o aluno por meio de um grupo de trabalho (Workgroup).

3. Notificar ao docente.

4. Interface com ambientes de virtuais de aprendizagem.

5. Interface com o Aluno.

6. Criar modelagem cognitiva.

7. Criar modelagem metacognitiva.

8. Criar modelagem modelagem afetiva.

9. Criar modelagem da personalidade.

10. Criar modelagem das emoções do estudante.

11. Confirmar estilo de aprendizagem do aluno.

12. Refutar estilo de aprendizagem do aluno.

13. Construir modelo de desempenho do aluno.
14. Construir modelagem explícita.
15. Construir modelagem implícita.
16. Construir modelo de estilo de aprendizagem do aluno.

A partir do levantamento, foi possível observar que a meta 1 abrange o escopo geral de toda a aplicação, sendo possível estabelecer como meta do sistema. A meta 2 indica que o SMA deverá criar um grupo de trabalho para cada aluno que estiver usando o sistema. O grupo de trabalho deve ser composto pelos agentes cognitivo, afetivo e metacognitivo, onde cada um possui suas respectivas metas. No levantamento, foram previstas três interfaces: Interface *web* com o Aluno, notificação do docente (via *web*) e interface com ambientes de virtuais de aprendizagem.

A hierarquia de metas do SMA Frank, apresentada na Figura 3.1, foi desenvolvida para graduar as metas que podem ser atingidas com o cumprimento de outras. Os retângulos em cinza representam metas particionadas.

Foram levantados 5 principais casos de uso, alguns com fluxos alternativos que representam fluxos alternativos no caso de uso, mas não necessariamente implicam em uma execução no SMA, apenas na parte *web*, como por exemplo: Erro de Login. Para fins de detalhamento, este trabalho utiliza-se da notação completa de desenvolvimento de casos de uso.

O primeiro caso de uso, descrito na Apêndice A.1, diz respeito à modelagem cognitiva do aluno. Existem dois cenários possíveis: Modelagem implícita (principal cenário de sucesso) e modelagem explícita (cenário alternativo). O SMA deverá processar o questionário de estilos de aprendizagem, respondido pelo aluno, para inferir explicitamente o seu modelo cognitivo e deverá analisar as respostas enviadas por ele para inferir explicitamente o seu modelo cognitivo.

O segundo caso de uso, descrito no Apêndice A.2, descreve o cenário de notificação do docente. Nele, o docente é autenticado no sistema e o sistema exibe uma lista de alunos disponíveis nas mais diversas turmas. O docente seleciona um aluno e então o sistema exibe os dados relativos ao modelo do aluno.

O terceiro caso de uso, descrito respectivamente no Apêndice A.3, dizem respeito à inferência do modelo afetivo. De forma análoga, o caso de suo descrito em A.4 ilustra a inferência do modelo metacognitivo do aluno.

O Apêndice A.5 descreve o último caso de uso que foi levantado para promover a interação entre os diferentes AVA e o SMA Frank. Devido a possibilidade dos AVA serem desenvolvidos em qualquer linguagem, é necessário utilizar-se de uma forma de comunicação comum entre aplicações.

Logo o SMA Frank irá utilizar de *WebServices* para a comunicação externa, garantindo que diversas aplicações poderão interagir com o SMA. Para novos AVAs, tudo o que precisará ser feito é a implementação da assinatura do serviço no *WebService*. Dessa forma a solução garante uma intervenção mínima no código do AVA, exigindo menos tempo na codificação da comunicação e garantindo o foco na inferência à ser feita pelo SMA.

Após o desenvolvimento dos casos de uso, foi necessário refinar os diagramas de sequência. Todos os diagramas foram desenvolvidos na ferramenta *agentTool*, visto que ele

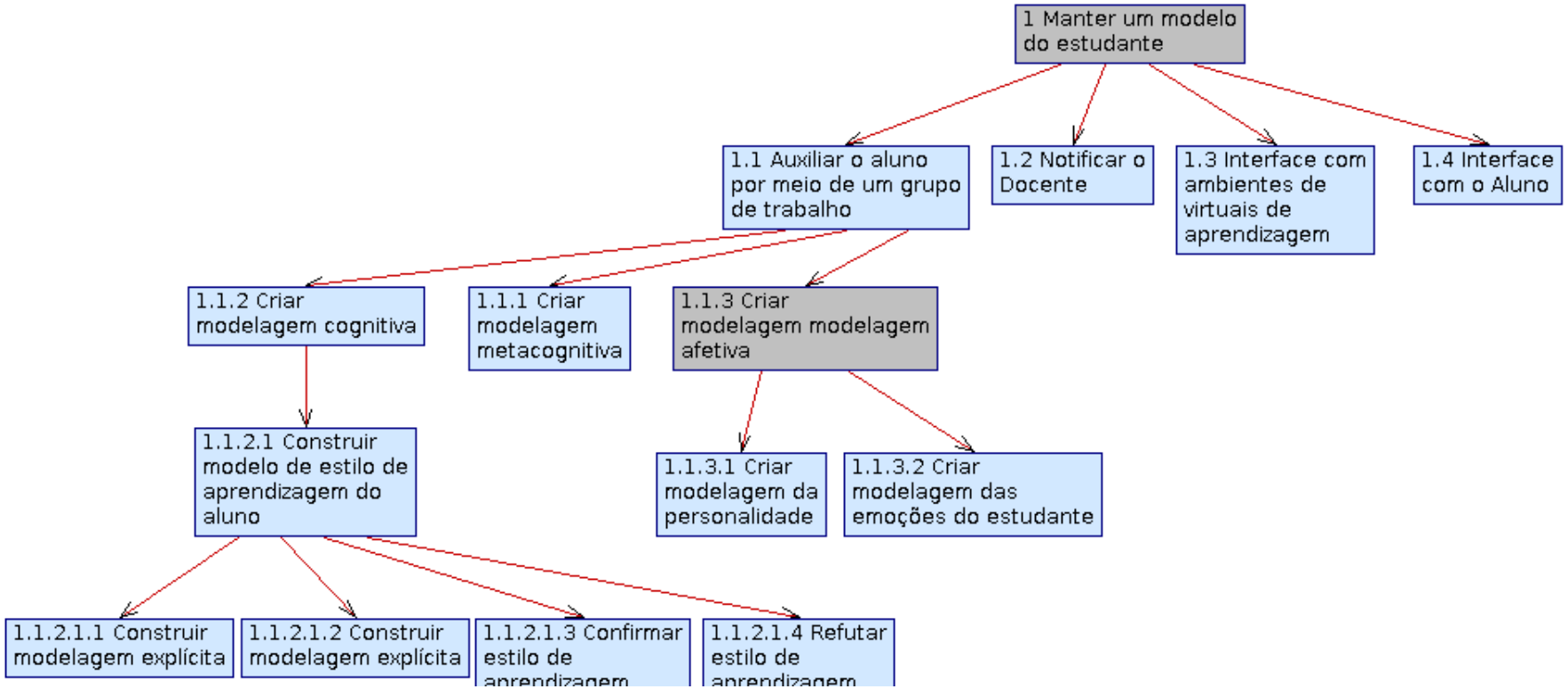

Tabela 3.1: Hierarquia de Metas do SMA Frank.

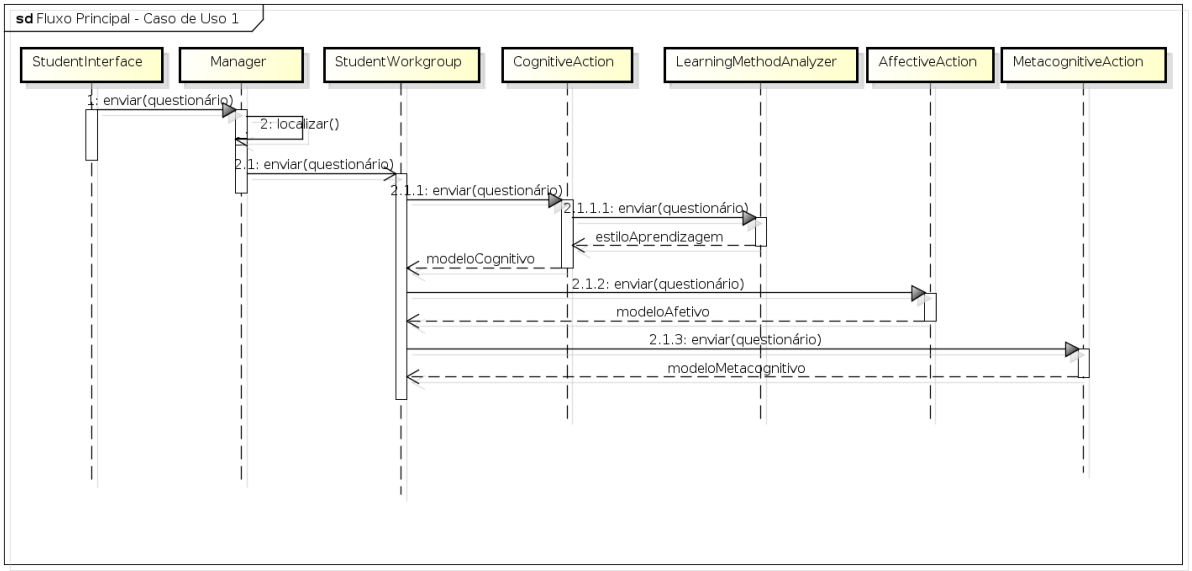

Figura 3.1: Diagrama de sequência do fluxo principal, caso de uso 1.

acompanha todas as fases do MASE. Para o primeiro caso de uso, foram desenvolvidos dois diagramas de sequência distintos: Um para o fluxo principal e outro para o fluxo alternativo.

No diagrama de sequência do caso de uso 1, apresentado na Figura 3.1, existem 6 regras. O fluxo do Sistema inicia-se com a regra *StudentInterface*, onde ele envia os dados de questionário para o *Manager*. Este então gera um evento de localização do aluno. Em seguida, gera o evento enviar para a regra *StudentWorkGroup*, com o parâmetro *questionário*. A regra *StudentWorkGroup* gera o evento de enviar para as regras *CognitiveAction*, *AffectiveAction* e *MetacognitiveAction*. Eles retornam respectivamente os modelos *Cognitivo*, *Afetivo* e *Metacognitivo*. A regra *cognitiveAction* ainda gera mais um evento de envio para a regra *LearningMethodAnalyzer*, que retorna o estilo de aprendizagem.

A Figura 3.2 apresentação o fluxo de exceção do primeiro caso de uso, onde ilustra a regra *WebServiceInterface* que recebe os dados do AVA e envia para a regra Manager por meio do evento *enviar*. Após esse evento, a regra *StudentWorkgroup* recebe o evento e reenvia para *CognitiveAction*. Em seguida ele envia para as regras *LearningMethodAnalyzer* e *PerformanceAnalyzer* que vão inferir o estilo de aprendizagem e a performance. Por fim, com estes dados, o modelo cognitivo é retornado para a regra *StudentWorkgroup*.

A Figura 3.3 ilustra a inferência afetiva descrita no fluxo principal do caso de uso 3. O processo de comunicação das regras *WebServiceInterface* e *StudentWorkgroup* funciona de forma semelhante ao diagrama anterior. A regra *AffectiveAction* gera um evento de inferência de modelagem afetiva.

O diagrama de sequência do fluxo principal caso de uso 4, apresentado na Figura 3.4, funciona de forma similar ao caso de uso anterior, com a diferença de que a regra *MetacognitiveAction* realiza a inferência da modelagem metacognitiva.

Por fim a Figura 3.5 apresenta o diagrama de sequência do fluxo principal do caso de uso5. Nele é mostrado a comunicação da regra *WebServiceInterface* com a regra *Manager*. A primeira realiza a validação de dados e em seguida o envio de dados. Após receber os dados, a regra *Manager* localiza o aluno e continua o fluxo de execução.

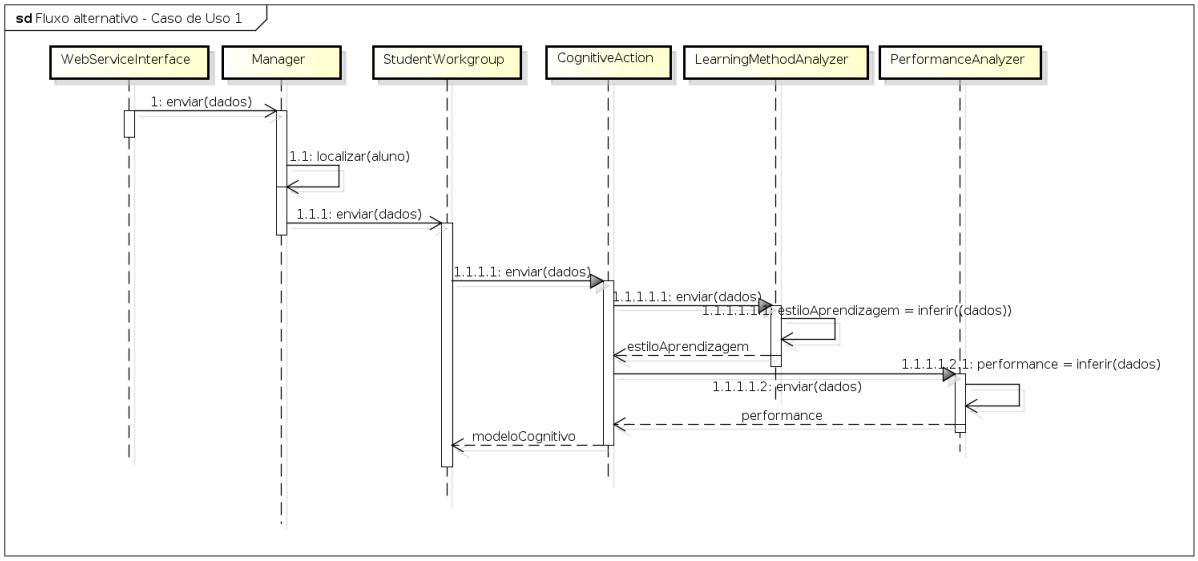

Figura 3.2: Diagrama de sequência do fluxo de exceção, caso de uso 1.

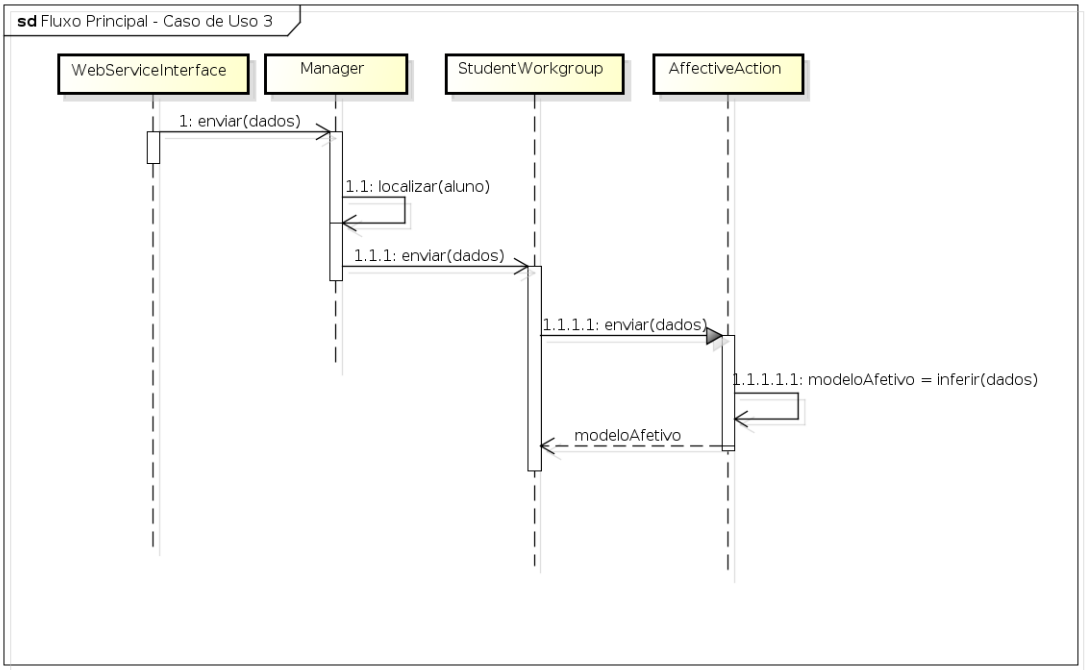

Figura 3.3: Diagrama de sequência do fluxo principal, caso de uso 3.