



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estudo, Definição e Implementação de Ambiente Baseado em Modelo Multidimensional e Abordagem de Sistema Multiagente

João Paulo de Freitas Matos

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

Orientador
Prof. Célia Ghedini Ralha

Brasília
2013

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

Coordenador: Prof. Alexandre Zaghetto

Banca examinadora composta por:

Prof. Célia Ghedini Ralha (Orientador) — 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 Baseado em Modelo Multidimensional e Abordagem de Sistema Multiagente / João Paulo de Freitas Matos. Brasília : UnB, 2013.

151 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



Estudo, Definição e Implementação de Ambiente Baseado em Modelo Multidimensional e Abordagem de Sistema Multiagente

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

Prof. Célia Ghedini Ralha (Orientador)
CIC/UnB

Prof. Germana Nóbrega de Menezes Prof. Fernanda Lima
CIC/UnB CIC/UnB

Prof. Alexandre Zaghetto
Coordenador do Bacharelado em Ciência da Computação

Brasília, 12 de Março de 2013

Dedicatória

Dedico a....

Agradecimentos

Agradeço a....

Resumo

A ciência...

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

Abstract

The science...

Keywords: Multiagent Systems, Informatics in Education, Multidimensional Model

Sumário

1	Introdução	1
1.1	Problema	2
1.2	Objetivos	2
1.3	Metodologia	3
1.4	Estrutura do Trabalho	3
2	Fundamentos básicos	4
2.1	Informática na Educação	4
2.1.1	Modelo de Honey e Munford	5
2.2	Sistemas Multiagentes	5
2.2.1	Inteligência artificial	5
2.2.2	Agente	6
2.2.3	Arquitetura de agentes	8
2.2.4	Sistemas Multiagentes	9
2.2.5	Comunicação	11
2.2.6	Linguagem de Comunicação de Agentes FIPA	15
2.2.7	Ontologias	15
2.3	Multiagent Systems Engineering	18
2.3.1	Análise	18
2.3.2	Design	21
2.4	Ferramentas Utilizadas	23
2.4.1	Unified Modeling Language	24
2.4.2	Java Agent Development Framework	30
2.4.3	JBoss Seam	33
2.5	Trabalhos Correlatos	34
3	Proposta de Solução	36
3.1	Metodologia	37
3.2	A Modelagem	37
3.2.1	Análise	37
3.2.2	Design	49
3.3	Arquitetura	52
3.3.1	Frank Web	52
3.3.2	SMA Frank	53
3.3.3	Integração Entre as Aplicações	53

4	Experimentações	55
4.1	Metodologia de Testes	55
4.2	Demonstração da Interface com Aluno	55
4.3	Demonstração da Interface com Docente	59
4.4	Resultados	59
5	Conclusões e Trabalhos Futuros	62
	Referências	63

Lista de Figuras

2.1	Esquematização do funcionamento básico de um agente em um ambiente. .	6
2.2	Ontologias superiores do mundo, cada uma indicando um conceito ou especialização do seu superior.	17
2.3	Representação utilizada no MASE Role Model.	20
2.4	Representação utilizada no <i>Concurrent Task Diagram</i>	20
2.5	Representação utilizada no <i>Agent Class Diagram</i>	22
2.6	Exemplo de conversação utilizada no Diagrama de Comunicação do lado do iniciador da conversação.	22
2.7	Notação utilizada na arquitetura de agentes.	23
2.8	Notação utilizada no diagrama de deploy.	24
2.9	Categorização dos Diagramas UML 2.0. Adaptado de [20].	26
2.10	Sugestões de notação de caso de uso proposto por [22]	28
2.11	Notação de diagrama de sequência, exibindo a comunicação de um ator e uma entidade (sistema)	29
2.12	Representação da arquitetura principal do JADE. Adaptado de [13].	31
2.13	Apresentação da Interface do agente RMA	34
2.14	Representação da pilha de aplicações do Seam. [9]	35
3.1	Diagrama de sequência do fluxo principal, caso de uso 1.	41
3.2	Diagrama de sequência do fluxo de exceção, caso de uso 1.	42
3.3	Diagrama de sequência do fluxo principal, caso de uso 3.	42
3.4	Diagrama de sequência do fluxo principal, caso de uso 4.	43
3.5	Diagrama de sequência do fluxo principal, caso de uso 5.	44
3.6	Diagrama <i>MASE Role Model</i> gerado para o SMA Frank.	45
3.7	Detalhamento da tarefa "Validar Dados" que pertence à regra <i>WebServiceInterface</i>	45
3.8	Detalhamento da tarefa "Enviar Questionário" que pertence à regra <i>StudentInterface</i>	46
3.9	Detalhamento da tarefa "Autenticar Aluno" que pertence à regra <i>StudentInterface</i>	46
3.10	Detalhamento da tarefa "Determinar WG do Aluno" que pertence à regra <i>Manager</i>	46
3.11	Detalhamento da tarefa "Criar Workgroup" que pertence à regra <i>Manager</i>	47
3.12	Detalhamento da tarefa "Processar Dados" que pertence à regra <i>StudentWorkgroup</i>	47
3.13	Detalhamento da tarefa "Atualizar Modelos" que pertence à regra <i>StudentWorkgroup</i>	48

3.14	Detalhamento da tarefa "Inferir Modelo Afetivo"que pertence à regra <i>AffectiveAction</i>	48
3.15	Detalhamento da tarefa "Inferir Modelo Metacognitivo"que pertence à regra <i>MetacognitiveAction</i>	48
3.16	Detalhamento da tarefa "Analisar Estilo de Aprendizagem"que pertence à regra <i>LearningMethodAnalyzer</i>	49
3.17	Diagrama de Classes do SMA Frank.	49
3.18	Detalhamento da Conversação 1 no lado do Iniciador.	50
3.19	Detalhamento da Conversação 1 no lado do Respondedor.	50
3.20	Detalhamento da Conversação 4 no lado iniciador.	51
3.21	Detalhamento da Conversação 4 no lado iniciador.	51
3.22	Detalhamento da Conversação 7 no lado iniciador.	51
3.23	Detalhamento da Conversação 7 no lado recebedor.	52
4.1	Tela Inicial do Sistema.	56
4.2	Tela de Autenticação do Sistema.	56
4.3	Tela de Erro de Autenticação do Sistema.	57
4.4	Tela de convite ao preenchimento do questionário de estilos de aprendizagem.	58
4.5	Tela de preenchimento do questionário.	59
4.6	Tela de visualização do estilo de aprendizagem inferido.	60
4.7	Tela inicial do usuário com o perfil de docente.	60
4.8	Tela de visualização do estilo de aprendizagem por turma.	61

Lista de Tabelas

2.1	Listagem de sistemas multiagentes com propriedades de medida de performance, ambiente, atuadores e sensores	7
2.2	Listagem de atributos de uma mensagem em KQML	13
2.3	Listagem de Enunciados Performativos	16
2.4	Estruturação Detalhada de Caso de Uso	27
3.1	Hierarquia de Metas do SMA Frank.	40
3.2	Estruturação das Tarefas por Regra	44

Capítulo 1

Introdução

Com o crescente desenvolvimento da computação que penetra cada vez mais em diversas áreas de conhecimento, a demanda por processamento tende a crescer rapidamente, tornando o desenvolvimento de aplicações cada vez mais complexo, exigindo cada vez mais desempenho e consequentemente poder de processamento. Para a execução dessas aplicações em tempo hábil, são necessários investimentos cada vez mais altos em *hardwares* melhores, existindo porém um fator limitante (custo ou tecnologia).

Além disso, atualmente a grande quantidade de informação disponível exige análises cada vez mais precisas e detalhadas da informação processada tendo em vista a ajuda de tomada de decisões. Dessa forma, aplicações que antes eram centralizadas em uma única máquina transformaram-se em aplicações distribuídas em várias máquinas que são concorrentes e assíncronas.

A partir dessa motivação, aplicações 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. Cada módulo pode ter um objetivo específico, como capturar e processar eventos no ambiente computacional, processar informações, persistir eventos no banco de dados, enfim, uma vasta gama de operações que são assíncronas e independentes. A arquitetura dessas aplicações é projetada objetivando o alto paralelismo, flexibilidade, interoperabilidade, dentre outros aspectos.

Diversos *frameworks* tentam lidar com o problema da computação distribuída, porém alguns usam arquiteturas que são baseadas em processamento ordenado: Os dados são processados ordenadamente, não usufruindo de todo o processamento que poderia ocorrer se fosse realmente paralelo. Outros *frameworks* são embasados em tecnologias que não são recomendáveis em um ambiente distribuído: O uso de recurso um compartilhado e bloqueante, que pode prejudicar o processamento em larga escala.

A computação distribuída toma formas ainda mais interessantes quando aplicada a contextos sensíveis a sociedade em geral. Áreas de atuação como Bolsa de Valores, Análise de Redes Sociais, Análise de Mídia Social, Informática na Educação, dentre outras. Em especial a esta última área, a possibilidade de auxílio no aprendizado do aluno eleva a importância deste setor na computação.

Na perspectiva da Informática na Educação (IE), a abordagem chamada Sistemas Tutores Inteligentes permite a representação de conhecimento de forma muito interessante. É possível a construção um modelo onde se representa o estudante (o objeto a quem se deve ensinar), o domínio do conteúdo (o conteúdo a ser ensinado) e o modelo pedagógico

(a forma a ser ensinada). Esta abordagem permite o uso de vários conceitos da Inteligência Artificial para determinação de formas como o estudante é mais eficiente em aprender, ou seja, o seu estilo de aprendizagem.

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.

Assim, conhecer os fatores relacionados ao processo de aprendizagem exige que as ferramentas computacionais aplicadas ao ensino consigam determinar eficientemente os estilos de aprendizagem. Dessa forma, aplicações tendem a ser mais complexas vistas ao alto grau de processamento que estas técnicas podem exigir.

1.1 Problema

Os ambientes educacionais de aprendizagem não possuem uma arquitetura apropriada para a inferência de modelos multidimensionais, pois a abordagem baseada em cliente-servidor não é apropriada para tal finalidade.

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.

Esta abordagem permitirá a construção e manutenção de um modelo multidimensional do estudante, a partir do qual os estilos de aprendizagens desse estudante poderão ser identificados e informados ao docente. A abordagem de sistemas multiagentes permite a decomposição do problema na modelagem multidimensional em vários subproblemas menores, diminuindo a complexidade da resolução. Além disso, a habilidade social dos agentes pode permitir a interação com outros modelos de alunos visando comparações e validações do modelo.

Especificamente, os objetivos específicos deste trabalho são:

- Objetivo específico 1: Obter uma modelagem da arquitetura geral do SMA Frank utilizando-se da metodologia *Multiagent System Engineering* (MASE), proposta como uma solução de Engenharia de Software para o desenvolvimento de SMA;
- Objetivo específico 2: Obter uma implementação da arquitetura geral do SMA Frank;
- Objetivo específico 3: Obter uma implementação da arquitetura dos agentes assistentes de cognição, metacognição e afetivo;
- Objetivo específico 4: propor uma interface do agente assistente de cognição com o estudante;
- Objetivo específico 5: propor uma interface do agente assistente de cognição com o docente;

1.3 Metodologia

A metodologia utilizada para a realização deste trabalho é composta das seguintes atividades:

- Aprofundado estudo dos conceitos de Informática na Educação, Sistemas Multiagentes e *Multiagent Systems Engineering* (MASE).
- Estudo e pesquisa do *middleware* JADE e a sua integração com aplicações externas.
- Baseado nos estudos feitos, a modelagem da Solução utilizando a metodologia MASE.
- Desenvolvimento da aplicação multiagente com base na pesquisa a respeito do *JADE*.
- Desenvolvimento da aplicação web e a sua integração com o sistema multiagente.
- Testes em laboratório da solução desenvolvida e suas conclusões.

O aprofundamento dos estudos em Informática na Educação e Sistemas Multiagentes é importante para orientar o desenvolvimento deste trabalho com base na teoria e garantir as melhores práticas.

Norteados pela teoria acerca do *Multiagent Systems Engineering*, a modelagem da solução será composta por uma série de diagramas que irão justificar as escolhas da arquitetura proposta.

Usando tecnologias existentes e consolidadas, a arquitetura proposta irá usar o framework *JADE*, que é completamente desenvolvido na linguagem *JAVA* e simplifica a implementação de Sistemas Multiagentes (SMA) que cumprem as especificações FIPA. A arquitetura proposta também englobará uma interface web que utiliza a plataforma *open source JBoss Seam*, desenvolvida para auxiliar a construção de aplicações dinâmicas para a internet de forma simples e ágil.

Os testes deste trabalho serão realizados por meio de simulações com alunos e docentes. Considerando o cenário do aluno, ele deve autenticar-se e utilizar o sistema para a verificação do seu estilo de aprendizagem, bem como visualizar a criação dos agentes do seu grupo de trabalho. O cenário do docente deve possibilitar a visualização do estilo de aprendizagem dos seus alunos.

1.4 Estrutura do Trabalho

Este trabalho está dividido em capítulos visando facilitar a leitura e organizar os conceitos que perfazem o desenvolvimento deste trabalho:

- Capítulo 2 contém todos os fundamentos teóricos necessários para o desenvolvimento desse trabalho.
- Capítulo 3 contém a proposta de solução composta pela metodologia, modelagem da arquitetura e implementação.
- Capítulo 4 contém os testes realizados.
- Por fim, o capítulo 5 relata a conclusão e trabalhos futuros.

Capítulo 2

Fundamentos básicos

Este capítulo apresenta os principais conceitos e definições necessários para o entendimento deste trabalho. A seção 2.1 apresenta alguns conceitos básicos em *Unified Modeling Language* (UML), que são necessários para o entendimento da modelagem deste trabalho. A seção 2.2 disserta sobre conceitos a respeito da informática na educação. A seção 2.3 aborda a teoria sobre Sistemas Multiagentes necessária para este trabalho. A seção 2.4 contém a metodologia *Multiagent System Engineering*, desenvolvida para a criação de Sistemas Multiagentes. A seção 2.5 e 2.6 abordam o funcionamento dos frameworks JADE e Jboss Seam, respectivamente. Por fim, a seção 2.7 detalha alguns trabalhos correlatos.

2.1 Informática na Educação

O uso do computador como meio de educação tornou-se essencial atualmente. Popularizando meios que, antigamente, eram muito caros ou raros, a aquisição de conhecimento tornou-se muito simples bastando um computador com acesso a internet para o acesso à 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 [17], promove o desenvolvimento cognitivo por meio de uma interação maior entre o aluno e o objeto de conhecimento.

Dessa forma a Informática na Educação (IE) constitui-se um importante ramo de estudo.

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 a distância e mais tarde com ferramentas interativas e em tempo real [23].

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 à cada estudante. A personalização está relacionada com a forma específica que cada estudante tem ao lidar com informações, a melhor forma de aprender, 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.

Em [19], um modelo de estilo de aprendizado classifica um estudante de acordo com um número, ou posição, em uma escala que, classifica as formas as quais são possíveis receber e processar as informações. Estilos de aprendizagem não são classificatórios, portanto não existe um mau ou bom estilo de aprendizagem. Apenas um diferente do outro.

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.

O modelo mais famoso pela literatura da IE, o modelo de Felder trata dos os seguintes estilos de aprendizagem:

- Sensing/Intuiting
- Visual/Verbal
- Active/Reflective
- Sequential/Globa

2.1.1 Modelo de Honey e Munford

O modelo de Butler possui cinco dimensões de estilos de aprendizagem:

- Divergente
- Pragmático
- Realista
- Pessoal
- Analítico

2.2 Sistemas Multiagentes

Este capítulo visa introduzir o conceito de sistemas multiagentes (SMA). Para tanto é necessário mostrar conceitos que são base para o entendimento de SMA, iniciando pela apresentação alguns conceitos a cerca de Inteligencia Artificial (IA). Em seguida o trabalho disserta sobre a teoria relacionada à Agente, bem como suas arquiteturas. Só então são apresentados os conceitos de Sistemas Multiagentes (SMA), comunicação em um ambiente SMA e por fim a teoria sobre ontologias.

2.2.1 Inteligência artificial

A definição de IA pode variar em duas dimensões principais [24]. Usando a definição de sistemas computacionais que agem racionalmente temos:

Computational Intelligence is the study of the design of intelligent agents.

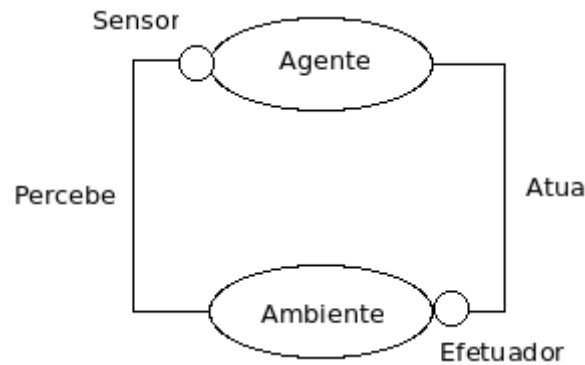


Figura 2.1: Esquemática do funcionamento básico de um agente em um ambiente.

Nessa definição, é importante ressaltar que o agente é uma entidade que atua racionalmente, esperando-se que essa racionalidade e outras características o diferencie de simples programas.

Com o crescimento dos estudos relacionados a este campo a inteligência artificial ganhou várias áreas de atuação, permitindo assim a resolução de diversos desafios relacionados às aplicações modernas. Uma das áreas de atuação é o auxílio na execução de aplicações que resolvem problemas de alta complexidade.

Atualmente várias aplicações podem exigir demais de *hardwares* mais modestos tornando inviável o tempo de execução daquela aplicação, sendo necessário um investimento maior e conseqüentemente encarecendo o seu valor. Dessa forma outras abordagens fazem-se necessárias, como a distribuição da aplicação em vários computadores que dividem a sua execução. Este é o campo de estudo da Inteligência Artificial Distribuída: Sistemas que são compostos por vários agentes coletivos, ou seja, distribuem o trabalho uns com os outros. Cada agente pode possuir uma capacidade diferente, sendo possível realizar a tarefa de modo paralelo.

2.2.2 Agente

Os agentes são entidades (reais ou virtuais) que funcionam de forma autônoma em um ambiente [27], ou seja, não necessitam de intervenção humana para realizar processamento. 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.

Na representação do funcionamento de agentes 2.1, os agentes funcionam de forma a perceber o ambiente em que estão por meio de sensores, fazem análises com base na interação inicial e por fim podem agir sobre o ambiente de forma a modificá-lo por meio de efetadores.

Alguns agentes seguem o princípio de racionalidade básico: 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, sendo desempenho o 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. O agente não tem controle total do ambiente, ele pode no máximo influenciá-lo com a sua atuação ou criar outros agentes. Podemos separar ambientes em classes: Software, Físico e Relidade

Tabela 2.1: Listagem de sistemas multiagentes com propriedades de medida de performance, ambiente, atuadores e sensores

Tipo de agente	Medida de performance	Ambiente	Atuadores	Sensores
Sensores de estacionamento	Avárias 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

virtual (simulação de ambientes reais em software). Em [33] temos, em geral, 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 dado o estado atual do ambiente e as ações dos agentes. O oposto do ambiente determinístico é o estocástico, quando não temos certeza do estado do ambiente. 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.
- Sequencial: 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.
- 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.

Em 2.1 é mostrado alguns exemplos de agentes, apresentando as suas características já discutidas nesse trabalho.

A primeira linha da tabela 2.1 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. Avárias no carro podem indicar um mal funcionamento do sensor.

A segunda linha da tabela é apresentado o exemplo de 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 pelo seu oponente, irá calcular sua próxima jogada e a realizará. 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 2.1 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 ser 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.

Os agentes possuem características inerentes à abordagem [33]. Os agentes possuem os aspectos de *reatividade*, *proatividade* e *habilidade social*. O conceito de reatividade está ligado com o agente perceber o ambiente e reagir. Proatividade é a característica do agente tomar a iniciativa e agir sem a necessidade de nenhum estímulo. Habilidade social é a capacidade de interação com outros agentes.

Os agentes possuem também os seguintes aspectos: Mobilidade, veracidade, benevolência, racionalidade e cooperação. As definições são:

- Mobilidade: O Agente deve poder mover-se no ambiente, por exemplo, em uma rede.
- Veracidade: Agente não comunica informações falsas.
- Benevolência: Agente ajudará os outros.
- Racionalidade: O agente não irá agir de forma a impedir a realização de seus objetivos.
- Cooperação: O agente coopera com o usuário.

2.2.3 Arquitetura 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 [27], 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 dessa arquitetura 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 levará a tabelas muito complexas e o tamanho pode facilmente passar a ordem de milhões dependendo do número de entradas.

A arquitetura reflexiva simples é um dos tipos mais simples 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 de regras condição-ação: Se uma condição ocorrer, uma ação será tomada. Por exemplo, vamos supor um agente médico

que determina o diagnóstico de uma doença no paciente caso exista alguma anomalia no organismo (Por exemplo, 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. O agente que usa esse tipo de abordagem é chamado de agente baseado em modelo.

Na arquitetura baseada em objetivo, as ações do agente são tomadas 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 do estado e o objetivo para determinar se deve ou não agir sobre o ambiente. Essa arquitetura porém é obviamente mais complexa e de certa forma ineficiente. Porém ela permite uma maior flexibilização das ações em determinados ambientes, visto que suas decisões são representadas de forma explícita e podem ser modificadas. É interessante notar que esse tipo de arquitetura não trata ações com objetivos conflitantes.

E 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.

2.2.4 Sistemas Multiagentes

Sistemas multiagentes são sistemas compostos por vários agentes capazes de se comunicar, possuindo uma linguagem de alto nível para isso. Um agente possui um objeto que, normalmente, é distinto dos objetivos de outros agentes e pode ou não cooperar com outros agentes para a realização de uma tarefa.

Em [30], 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 sistema multiagente é designado para a solução de problemas de forma distribuída, onde o problema é distribuído entre os agentes que, juntos, trabalham concorrentemente para a resolução deste problema, podendo ou não trabalhar de forma cooperativa.

É importante notar a distinção de conceitos como *cooperação* na literatura de sistemas multiagentes. Em [33], existem duas principais diferenças do conceito entre as duas abordagens.

A primeira delas é que agentes são designados de forma diferente, com objetivos diferentes. Em um ambiente com vários agentes, eles devem trabalhar estrategicamente para alcançar seus objetos. A possibilidade de agentes não cooperarem é perfeitamente plausível.

A segunda diferença está no fato do agente agir 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 dinamicamente 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 [18], a técnica CDPS consiste de uma rede de baixo acoplamento provida de sofisticados nós resolvidores 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 os agentes 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 modele) 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 o conflito de interesses neste ambiente, situação que força os agentes a cooperarem com os outros para alcançarem seus objetivos.

O processo CDPS pode ser dividido em três etapas:

- decomposição do problema.
- solução do subproblema.
- integração da solução.

Com uma solução compartilhada de resolução de problemas, a arquitetura de sistemas multiagentes 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.

2.2.5 Comunicação

A comunicação é um dos aspectos mais importantes no desenvolvimento de SMAs. 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 nesta 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 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 influenciar 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 (do inglês *Speech act theory*) e trata a comunicação como uma ação. A teoria dos atos de fala, publicada em 1962 [10] por John Austin, onde ele percebe que certas expressões de linguagem natural, ou atos de fala, possuem a característica de realizar ação em um interlocutor, causando assim uma mudança de estado da mesma forma que uma ação física. Logo, as expressões descrevem as ações por meio de desejos, habilidades e crenças.

Em [32], a teoria dos atos de fala possuem duas características:

- A distinção entre um o significado expressado por uma expressão e a forma como essa expressão é utilizada.
- Expressões de todos os tipos podem ser considerados como atos, pois mudam o mundo de alguma forma.

Posteriormente o trabalho de John Searle [29], relacionado ao de Austin, separa uma ação de um ato entre orador(*speaker*) e ouvinte(*hearer*) identifica propriedades e condições que um discurso deve conter para realizar ações sucedidas. Além disso, ele classifica alguns atos de discursos em 5 classes:

- Representativas - Representa o 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 Inteligência Artificial [33]. 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 influenciou na arquitetura da comunicação, ela influenciou também nas 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) [6].

- *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 [25]. 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 [21].

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 divididas em três categorias: Discursivas, intervenção/mecânica 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 [25], 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 2.2 lista os principais parâmetros em uma mensagem nesta linguagem, bem como seu significado.

O formato da mensagem é completamente compreensível aos humanos. Na mensagem 2.1 podemos verificar na primeira linha o enunciado performativo *ask-one*, onde será uma mensagem de consulta. Nas linhas abaixo, visualizamos todos os parâmetros da mensagem antecidos 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 espessa o conteúdo.

Tabela 2.2: Listagem de atributos de uma mensagem em KQML

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

Por fim, o parâmetro *content* que indica o conteúdo da mensagem, no caso, uma consulta escrita em PROLOG perguntando pelo preço de um computador.

Listing 2.1: Exemplo de mensagem em KQML

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

Um exemplo de diálogo escrito em KQML pode ser visto na sequência de mensagens [2.2](#)

Listing 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)
)
```

A primeira mensagem do diálogo possui o enunciado performativo é *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 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.

Em [33], a adoção desta linguagem pela comunidade de SMA foi significativa, mas sofreu diversas críticas:

- A fluidez e a não restrição do KQML fez com que diversas implementações estendidas surtissem, 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*.

Dessa forma, novos desenvolvimentos de linguagens fizeram-se necessários.

KIF

A linguagem foi desenvolvida para expressar conhecimento a cerca de um determinado domínio, sendo possível assim a troca de conhecimentos entre agentes. Em [21], a linguagem possui as seguintes características:

- Tem uma semântica declarativa, sendo possível entender o seu significado sem a necessidade de um interpretador para manipulação das expressões.
- É logicamente compreensível.
- É provida com a capacidade de reproduzir meta-conhecimento, ou seja, conhecimento a respeito da representação do conhecimento. Com isso, é possível reproduzir 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, domínios, dentre outros. 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 trecho de código 2.3 ilustra um exemplo de expressão na linguagem KIF, validando que a temperatura de m1 é 83 graus Célsius.

Listing 2.3: Exemplo de expressão de conteúdo com a linguagem KIF. Fonte: [33]
(= (temperature m1) (scalar 83 Celsius))

2.2.6 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 SMAs. O núcleo dessa padronização foi o desenvolvimento de uma linguagem de comunicação de agentes (ACL) padronizada para 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 definindo formalmente as interpretações dessas mensagens e não definindo nenhuma linguagem para o conteúdo da mensagem.

Devido ao fato da linguagem KQML não ter performativos adequados, a ACL da FIPA recebeu total preocupação na definição formal da semântica da linguagem. A tabela 2.3 contém um breve resumo dos enunciados performativos disponíveis em [3]. É importante notar que a especificação [3] possui toda a descrição formal de cada enunciado performativo aqui descrito.

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

2.2.7 Ontologias

Ontologia é um ramo da Filosofia que dedica-se a estudar e representar a natureza do ser, existência ou realidade. É uma formalização dos conceitos e relacionamentos que podem existir em um determinado universo. Para a Inteligência Artificial, é tudo aquilo que pode ser representado. Em o autor [27], 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 bastante 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 conhecimento a qualquer momento.

Em 2.2, é possível observar 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 de Sistemas Multiagentes, 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, dentre outros exemplos. Em outras palavras, 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.

Tabela 2.3: Listagem de Enunciados Performativos

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

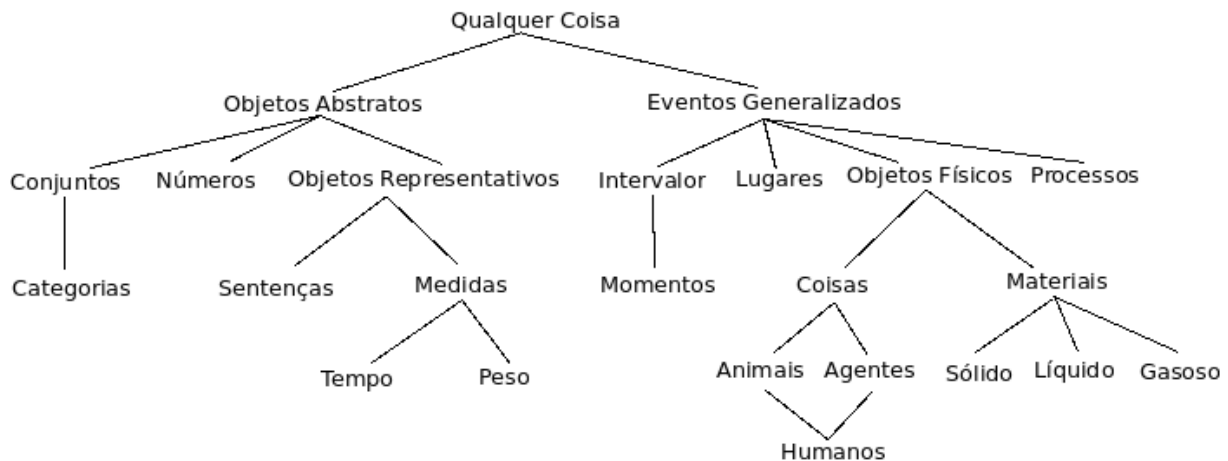


Figura 2.2: Ontologias superiores do mundo, cada uma indicando um conceito ou especialização do seu superior.

O exemplo 2.4 está descrito escrito na linguagem XML que representa o domínio de países. Nele podemos ver a ontologia superior, País. Cada país é composto por um conjunto de estados, contendo 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 trecho 2.4, porém sem alteração de semântica.

Listing 2.4: Exemplo de código XML representando uma ontologia simples de cidades.

```

<País nome="Brasil">
  <Estado nome="Goiás" 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>
  ...
</País>

```

2.3 Multiagent Systems Engineering

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. O advento de sistemas multiagentes trouxe à tona a necessidade de outras metodologias de desenvolvimento diferente daquelas, visto a diferença de abordagem no software.

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.

O desenvolvimento de uma metodologia para projetar sistemas multiagentes surgiu, chamada *Multiagent Systems Engineering* (MASE), onde os requisitos e metas do SMA são levantados e a partir de então as tarefas e agentes são projetadas para lhes atender. O MASE utiliza-se de alguns modelos gráficos para a descrição dos agentes, seus objetivos, suas interações e a sua arquitetura.

Em [28], a metodologia do MASE é baseada nas mais tradicionais metodologias de desenvolvimento de software, dividida em duas fases principais: Análise e Design.

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

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

2.3.1 Análise

Em [28], a fase de análise objetiva o desenvolvimento de um conjunto de regras, que descrevem uma funcionalidade para uma entidade, as quais as atividades descrevem o que deve ser feito para o sistema atingir o 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 [28].

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

As metas podem ser divididas em 4 tipos, podendo 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 semelhança de seus objetivos. As metas não funcionais são relativas à cumprir 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 irá se comunicar. 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. Esses diagramas são criados para cada caso de uso, podendo haver mais de um para cada caso de uso. O objetivo principal é o levantamento dos eventos e das regras.

Refinar Regras

O último passo da fase de análise, o refinamento de regras consiste na associação metas e seus diagramas de sequências às regras e suas respectivas tarefas. A associação de tarefas às regras é a melhor forma de modelagem de Sistemas Multiagentes [28].

Em geral, a associação de metas às regras é de um para um, não sendo uma regra necessariamente. Durante esta etapa, algumas considerações relativas ao desenho do sistema devem ser levadas em consideração. Caso exista alguma alteração (adição de uma

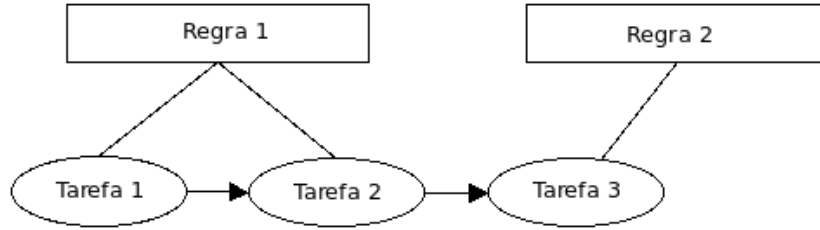


Figura 2.3: Representação utilizada no MASE Role Model.

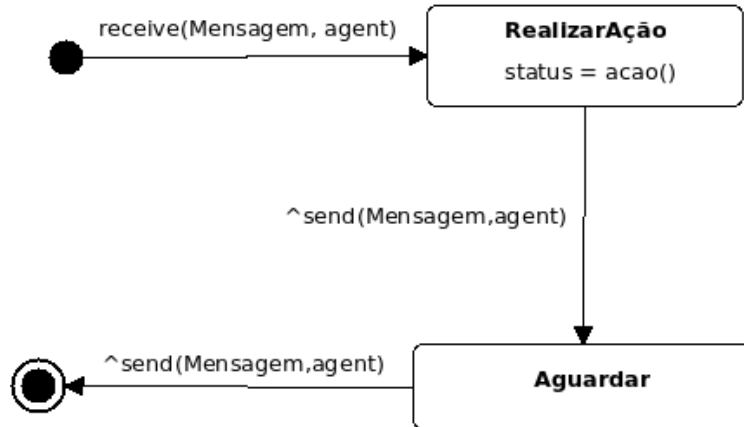


Figura 2.4: Representação utilizada no *Concurrent Task Diagram*.

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 estruturas em uma modelagem chamada *MASE Role Model*, contendo informações de interações entre as tarefas. A representação a notação do *MASE Role Model* 2.3 utilizada por [28] utiliza-se de 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 determina neste passo a criação do *Concurrent Task Diagram*.

O diagrama é representado por meio de autômatos de estados finitos 2.4, 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 de pode ser expressa por meio de uma notação 2.5 que define o gatilho

a ser disparado, condições para a execução e as mensagens transmitidas.

Listing 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 do código *guard*, ocorrendo a mudança de estado somente quando a condição for verdadeira. Por fim ocorrem as transmissões, 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 dividida 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 necessário a criação de um novo diagrama, o *Agent Class Diagram* 2.5. Nele, as classes dos agentes são associadas com as regras levantadas e as comunicações entre as classes.

Construir Conversações

A próxima etapa da segunda fase diz respeito às conversações que existirão no SMA. Aqui, cada detalhe da conversação entre dois agentes deverá ser planejado 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 [28]. Caso exista alguma conversação semelhante, o agente muda o seu estado e realiza as ações relativas para atingir esse estado. Caso contrário, é assumido que o agente emissor deseja iniciar uma nova conversação e o receptor compara com as suas possibilidades de tipos de conversação disponíveis para participar.

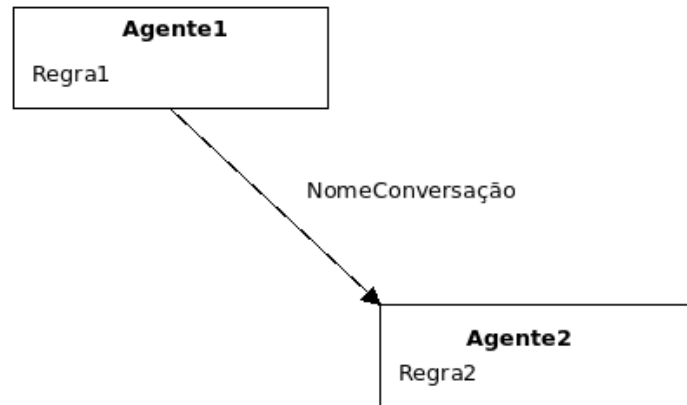


Figura 2.5: Representação utilizada no *Agent Class Diagram*.

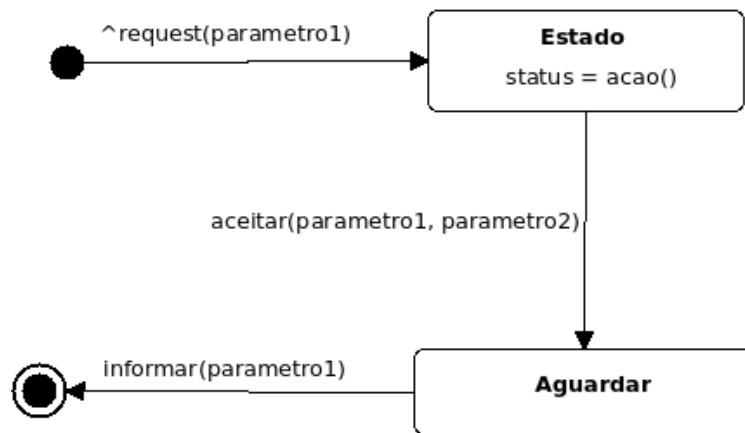


Figura 2.6: Exemplo de conversa utilizada no Diagrama de Comunicação do lado do iniciador da conversa.

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*. Os estados do autômato são as ações que devem ser realizadas pelo agente. As transições de estado são as conversações que são feitas pelo agente.

A sintaxe usada na conversa de agentes 2.6 é uma expressão que 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.

Listing 2.6: Sintaxe da conversa entre dois agentes.

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

O Diagrama de Comunicação 2.6 é definido para os dois lados da conversa: Iniciador e Receptor. considerando o lado iniciador da conversa.

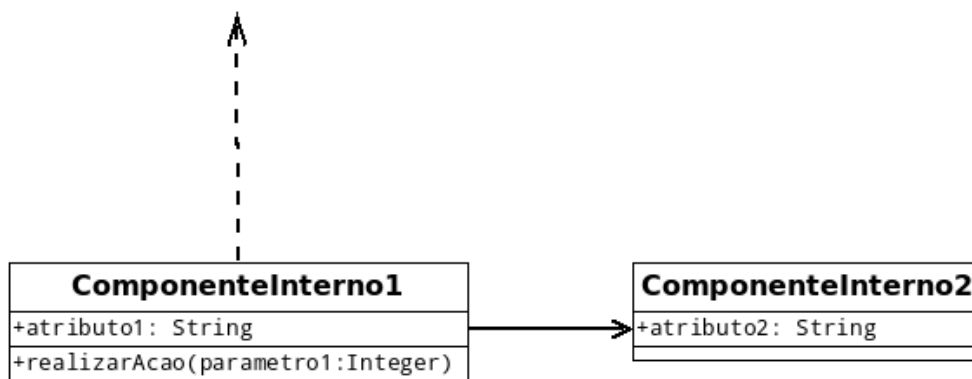


Figura 2.7: Notação utilizada na arquitetura de agentes.

Montagem dos Agentes

Neste passo, o estado interno dos agentes é criado. É necessário nesta etapa definir a arquitetura dos agentes e os componentes que irão compor esta arquitetura. Para isso, 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. No mínimo, cada agente criado no Diagrama de Agentes deve ser definir uma arquitetura interna.

No Diagrama de Arquitetura de Agentes 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 atributos externos ao agente, enquanto as outras setas 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 representação do diagrama de *deploy* 2.8 expressa os ambientes criados, bem como os agentes internos. O retângulo com linhas pontilhadas representam os ambientes que podem existir na aplicação. Os outros retângulos caixas representam os agentes e as linhas que os ligam representam as suas interações.

2.4 Ferramentas Utilizadas

Nesta seção estão reunidos as tecnologias que auxiliaram ao desenvolvimento do SMA. A primeira seção explica alguns diagramas que são definidos na *Unified Modeling Language*, necessários para entender os passos da *Multiagent System Engineering*.

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.

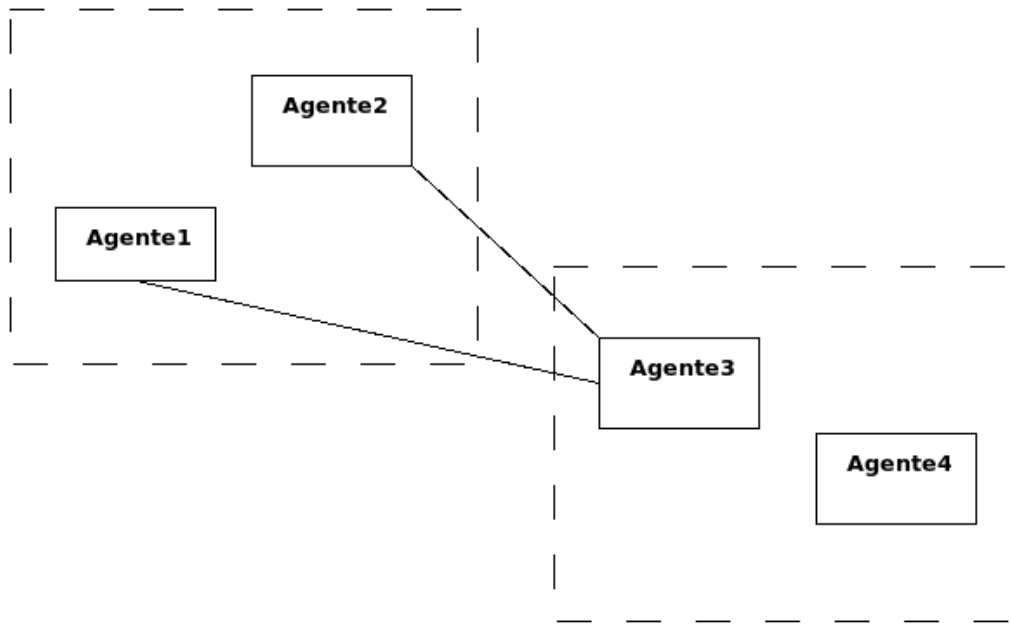


Figura 2.8: Notação utilizada no diagrama de deploy.

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

2.4.1 Unified Modeling Language

A Linguagem Unificada de Modelagem, *Unified Modeling Language* (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 e resultando em uma semântica para o projeto em questão. Em [20] 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* [7]. Seu nascimento é datado em 1997 [20], 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.

É possível separar o uso do UML de três formas distintas [20], diferindo entre as modelagens utilizadas e o objetivo de uso. As três formas são: Rascunho, planta de software e como linguagem de programação.

A utilização como rascunho é utilizada para facilitar 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 dividida 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 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 2.0 descreve 13 tipos de diagramas [2.9](#) que podem ser categorizados como estruturais e comportamentais

Apesar da grande quantidade de diagramas envolvidos no UML, nem todos os processos de desenvolvimentos de software utilizam todos eles. Em [\[22\]](#), a grande extensão de diagramas UML não é usada pela maioria das metodologias. A metodologia de desenvolvimento de sistema multiagente 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 [\[22\]](#), 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 entre as partes faz do caso de uso uma boa maneira de simplificar o entendimento do comportamento do sistema [\[22\]](#), bem como envolver todos as partes interessadas do sistema(*stakeholders*) na construção do mesmo. Em [\[15\]](#), 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.

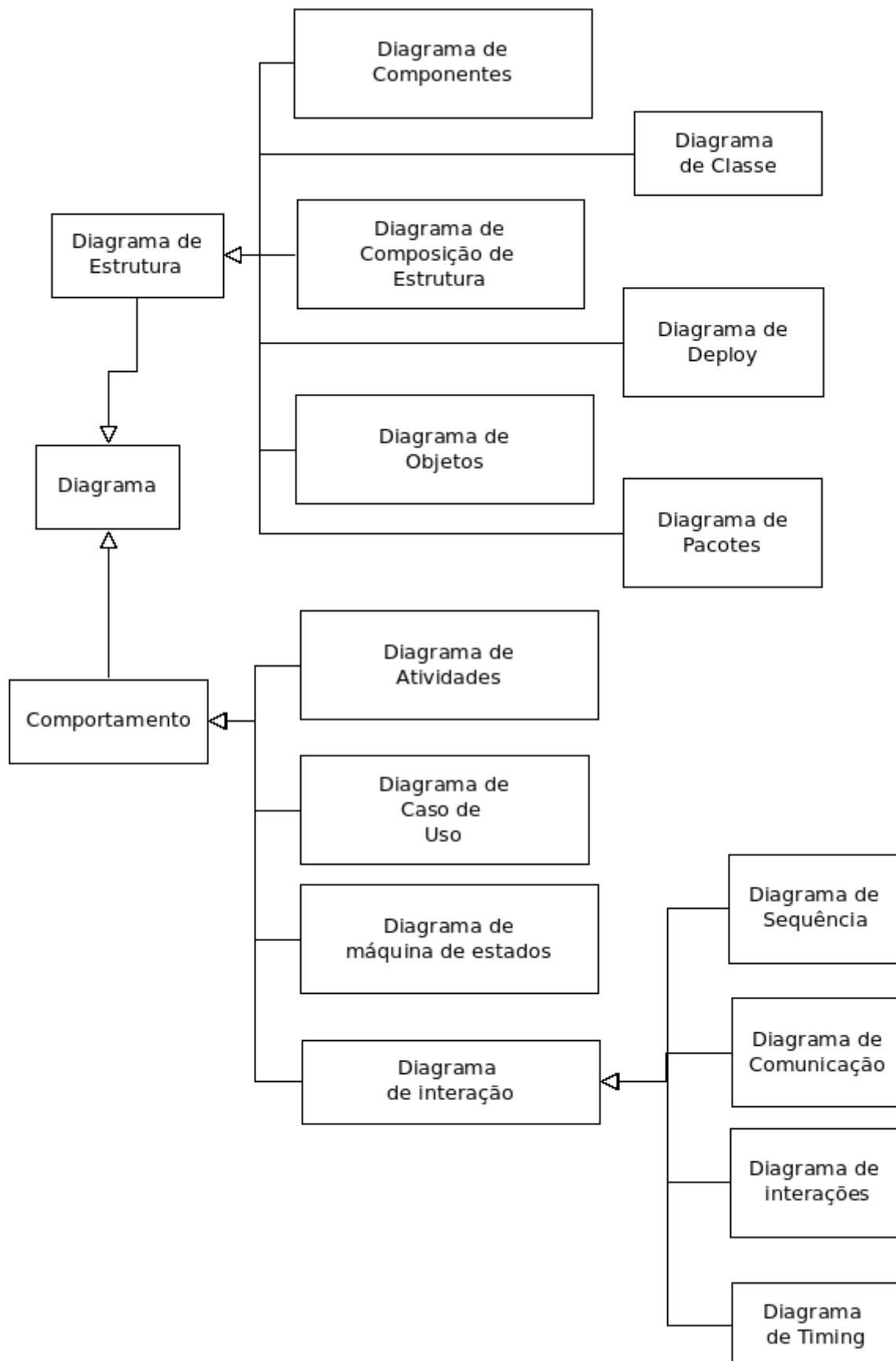


Figura 2.9: Categorização dos Diagramas UML 2.0. Adaptado de [20].

Tabela 2.4: 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 interessados.
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 Tecnológicas 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

- Ator de Bastidor: Expressa algum interesse pelo comportamento do 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 fato de ser estruturado e mostrar mais detalhes. Os casos de uso são estruturados de diversas formas. Em a literatura, sendo a mais famosa [22] a estrutura utilizada por Alistar Cockburn [15] e detalhada neste trabalho. 2.4

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. Em 2.10 é mostrado a sugestão de diagramação do caso de uso, propondo forma de representação de atores, casos de uso e atores auxiliares.

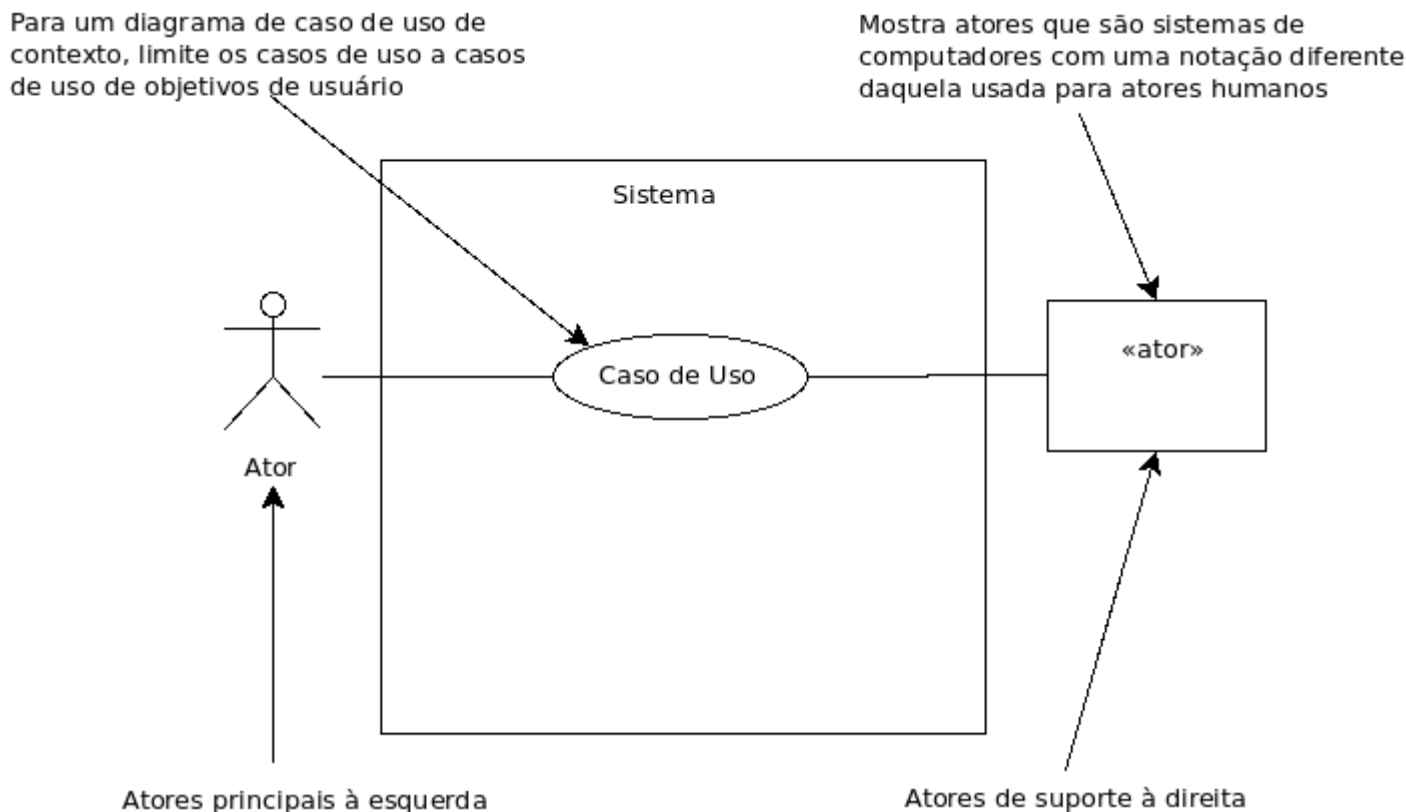


Figura 2.10: Sugestões de notação de caso de uso proposto por [22]

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. A 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.

Na ilustração da notação de diagramas de sequência 2.11 é possível identificar a interação entre um ator e uma entidade do sistema. É fácil identificar 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

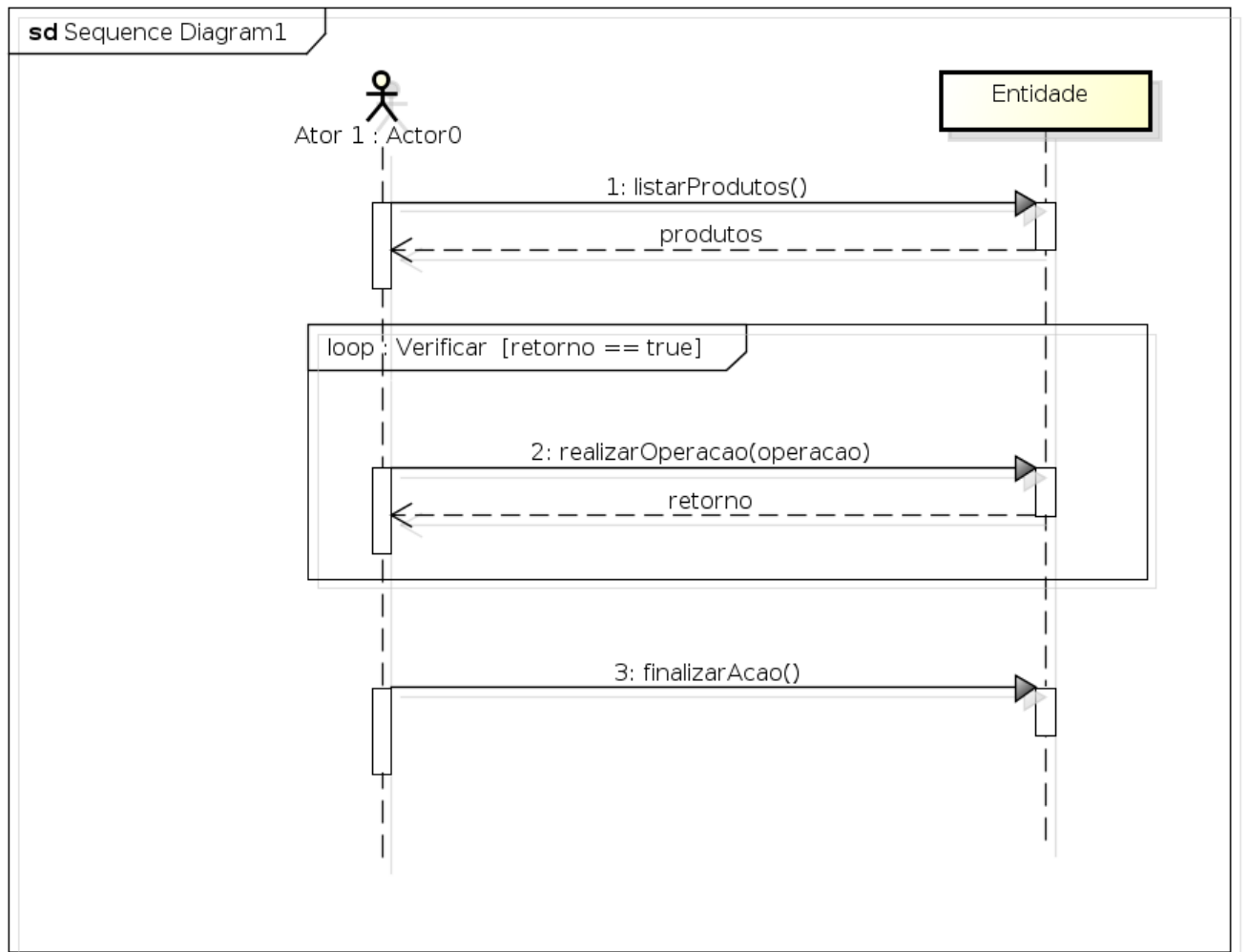


Figura 2.11: Notação de diagrama de sequência, exibindo a comunicação de um ator e uma entidade (sistema)

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 *operacao*. 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 e projetar o comportamento do sistema no nível de projetar o que ele faz, porém sem necessariamente explicar como o faz [22].

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. Ele apenas deixará claro a interação entre os atores e os eventos derivados dessa interação.

2.4.2 Java Agent Development Framework

JADE é um *middleware* desenvolvido em 1988 pela Telecom Italia, posteriormente (2000) tornando-se em um projeto open source. Seu desenvolvimento de baixo acoplamento permite que seja possível integrar várias bibliotecas auxiliares (*addons*) para facilitar o desenvolvimento de aplicações.

Em [13], 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.

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 plataforma 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.

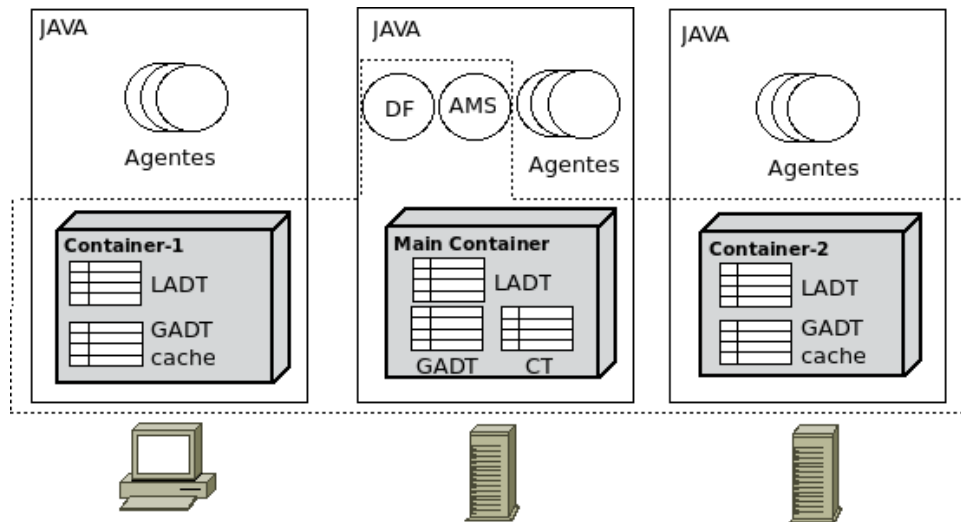


Figura 2.12: Representação da arquitetura principal do JADE. Adaptado de [13].

- 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 [13], 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 representação 2.12 do ambiente de execução JADE ilustra 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*, abrigando agentes primordiais na plataforma que atendem às especificações da FIPA (número 23 e 61 [12]). 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.

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 estender esta classe e implementar o comportamento no método *setup()*. O fato de estender 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 caractere '@', 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.

Conforme dito anteriormente, o método *setup()* deverá ser implementado e, no mínimo, deverá ser estabelecido um comportamento para o agente. 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 no trecho 2.7.

Listing 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 representação básica do exemplo do agente RMA 2.13, é apresentada sua interface gráfica com o ambiente de execução em tempo real. É possível identificar apenas o *container* principal da aplicação. Nele, existem os três agentes principais registrados descritos anteriormente: AMS, DF e o próprio 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 [4]), contextos Java e injeção de dependência [5]. 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 reúne 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 [8].

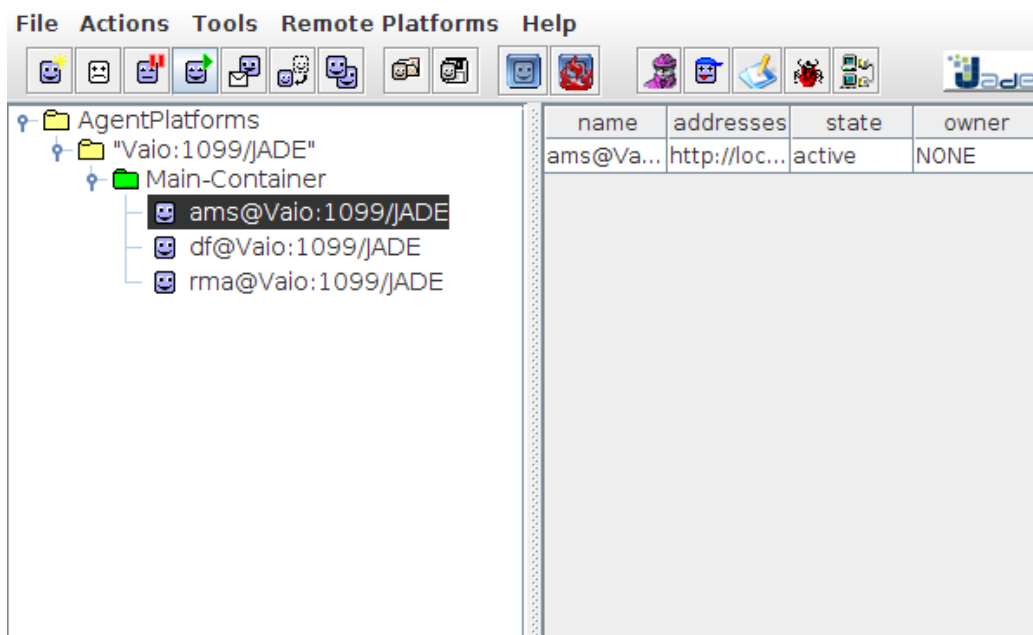


Figura 2.13: Apresentação da Interface do agente RMA

A pilha de aplicações do Seam 2.14 são todas as tecnologias que são utilizadas pelo JBoss Seam. Caso o desenvolvedor deseje utilizar outras tecnologias, o Seam provê configurações para que outras ferramentas possam ser integradas facilmente à aplicação.

Em [9], 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.

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 Inteligência Artificial, Estilos de Aprendizagem, Sistemas Multiagentes ou Ambientes Virtuais de Aprendizagem.

Na área de auxílio de alunos, alguns trabalhos assemelham-se com o propósito deste trabalho. A implementação de um agente artificial integrado a um tutor inteligente [31] que identifica dificuldades de aprendizagem por meio de técnicas de Inteligência Artificial. 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

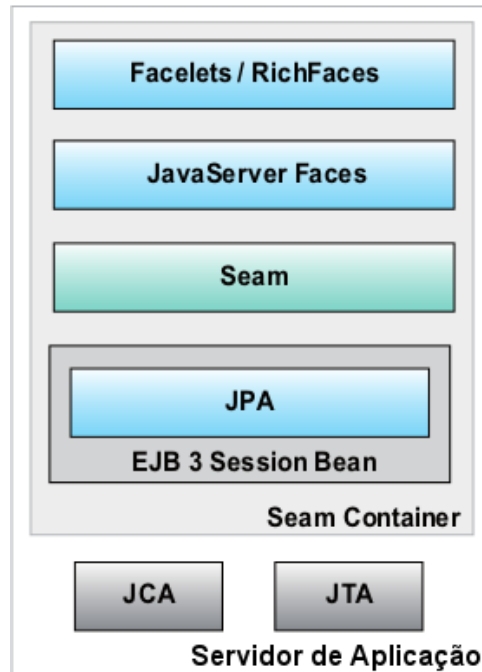


Figura 2.14: Representação da pilha de aplicações do Seam. [9]

de inferência do aluno, bem como a solução não prevê a inferência de dados vindos de outros Ambientes Virtuais de Aprendizagem.

Existem trabalhos na área de identificação de conhecimentos [11], que visam a determinação de estilos de aprendizagem dos alunos por meio da plataforma chamada "Ferramenta de Identificação de Perfis de Aprendizizes – FIPA", uma aplicação multicamadas que faz a identificação do perfil é feita 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.

O trabalho [14] utiliza-se de Sistemas Multiagentes 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.

Por fim, o Ambiente Virtual de Aprendizagem EDULIVRE [26] constitui-se de um Ambiente Virtual de Aprendizagem que 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 é com uso de agentes, diferindo assim do foco deste trabalho. Além disso, o EDULIVRE apresenta a estratégia de ser o Ambiente Virtual de Aprendizagem e não permite integração com outros ambientes.

Capítulo 3

Proposta de Solução

O presente capítulo apresenta as características da aplicação Frank, bem como detalha a modelagem da solução e a sua arquitetura. Além disso, o capítulo explica a metodologia de desenvolvimento utilizada do trabalho.

Basicamente, a solução é 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. *AffectiveAgent* - Responsável pelo modelo afetivo.

A solução é baseada em grupos de trabalho por aluno. Após a autenticação do aluno em uma plataforma web, o sistema multiagente realiza a criação dos agentes de grupo de trabalho para o aluno: *WorkgroupAgent*, *CognitiveAgent*, *MetacognitiveAgent*, *AffectiveAgent*. Esses agentes são responsáveis pela manutenção do modelo multidimensional do aluno na plataforma e são atualizados conforme os dados forem enviados pelos agentes interfaces com o ambiente externo.

O primeiro agente interface, *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 os Ambientes Virtuais de Aprendizagem. Por fim, o agente *ManagerAgent* é responsável pela criação dos agentes.

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

3.1 Metodologia

Para a realização deste trabalho, foi planejada uma metodologia de desenvolvimento na qual objetivou-se a construção da arquitetura do SMA por meio de uma metodologia de desenvolvimento de SMAs. Inicialmente, foi necessário um levantamento de bibliografias relacionadas à Informática na Educação para o entendimento do problema do trabalho, além de levantamentos de uso de SMA em contextos pedagógicos.

Em seguida foi aplicada, foi realizado estudos aprofundados na metodologia MASE para a implementação deste trabalho. Após o levantamento inicial, seu uso é dividido em duas fases, conforme explicado na seção 2.3, sendo a primeira delas essencial para o levantamento dos requisitos necessários para cumprir os objetivos. A segunda fase foi importante para a distribuição das regras entre os agentes que existem no sistema. As primeiras subseções deverão detalhar a modelagem.

A arquitetura básica do sistema já havia sido previamente decidida [16], sendo necessário detalhar seus requisitos por meio da metodologia MASE. Além disso, os Ambientes Virtuais de Aprendizagem deveriam ser capazes de comunicar-se com a solução proposta, ou seja, a comunicação deve ser feita de forma independente da linguagem de programação. Por fim, foi definido [16] o nome da aplicação solução a ser desenvolvida: *Frank*.

Posteriormente o trabalho seguiu-se com a implementação do SMA na linguagem JAVA, utilizando-se das ferramentas JADE e JBoss Seam. As subseções seguintes deverão detalhar a implementação do Sistema Multiagente e da camada web, bem como a sua integração.

Por fim, a solução foi testada por meio de cenários que simulavam o uso por meio dos atores Aluno e Docente. A demonstração está detalhada no capítulo 4.

3.2 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. A primeira fase 3.2.1, responsável pelo levantamento de requisitos e entendimento das regras e tarefas, e a segunda fase 3.2.2 responsável pela criação dos agentes, conversações e componentes internos.

3.2.1 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 [16], onde é possível listar:

1. O sistema deve manter um modelo do estudante, onde será determinado o seu estilo de aprendizagem e será notificado 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 3.1 foi desenvolvida para encontrar quais metas poderiam ser estabelecidas 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 diz respeito à modelagem cognitiva do aluno. Existem dois cenários possíveis: Modelagem implícita (principal cenário de sucesso) e modelagem explícita (cenário alternativo). Basicamente 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 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 trecho possui a descrição do caso de uso.

Os terceiro e quarto casos de uso dizem respeito à inferência do modelo afetivo e metacognitivo do aluno, respectivamente.

Por fim, o último caso de uso foi levantado para prever a interação do AVA com o SMA Frank. Devido a possibilidade dos Ambientes Virtuais de Aprendizagem serem desenvolvidos em qualquer linguagem, é necessário utilizar-se de uma forma de comunicação comum entre aplicações.

Logo o SMA Frank irá utilizar-se 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

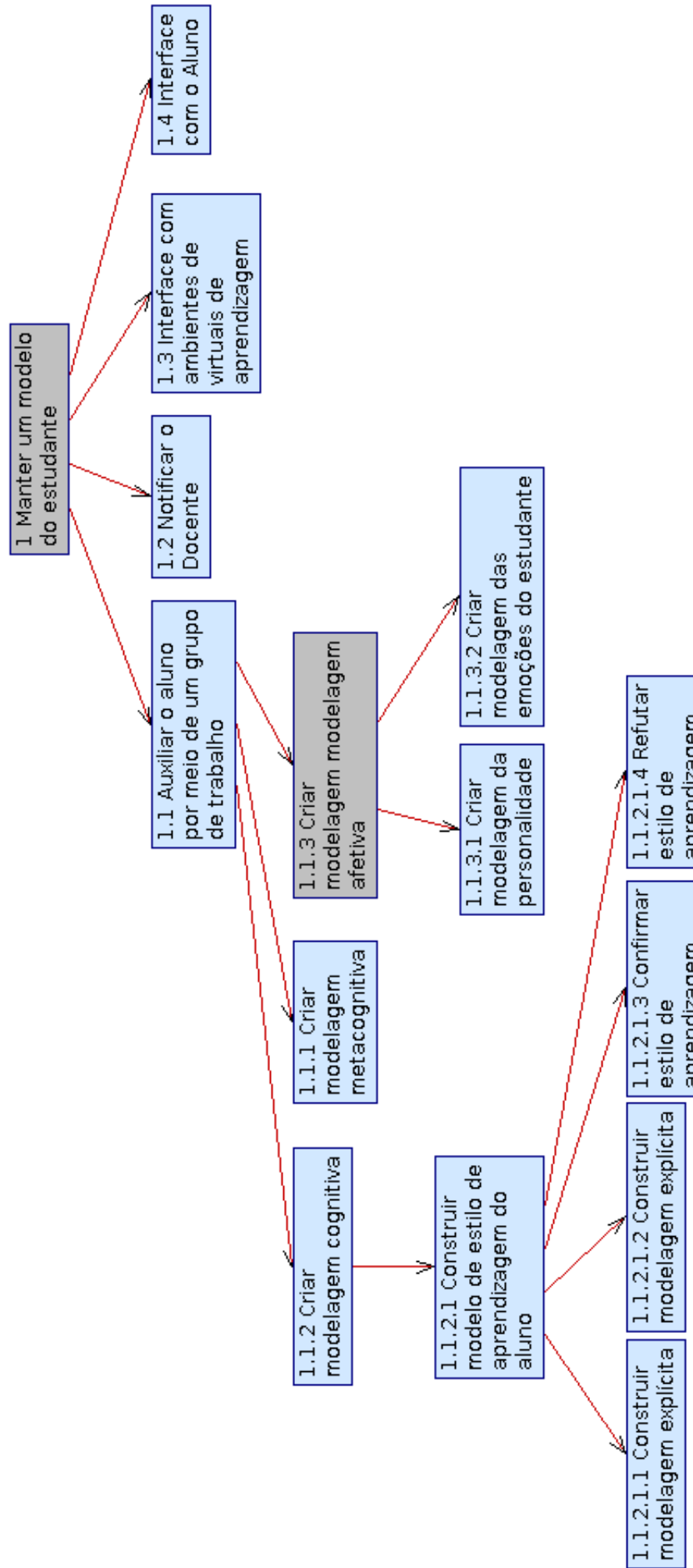


Tabela 3.1: Hierarquia de Metas do SMA Frank.

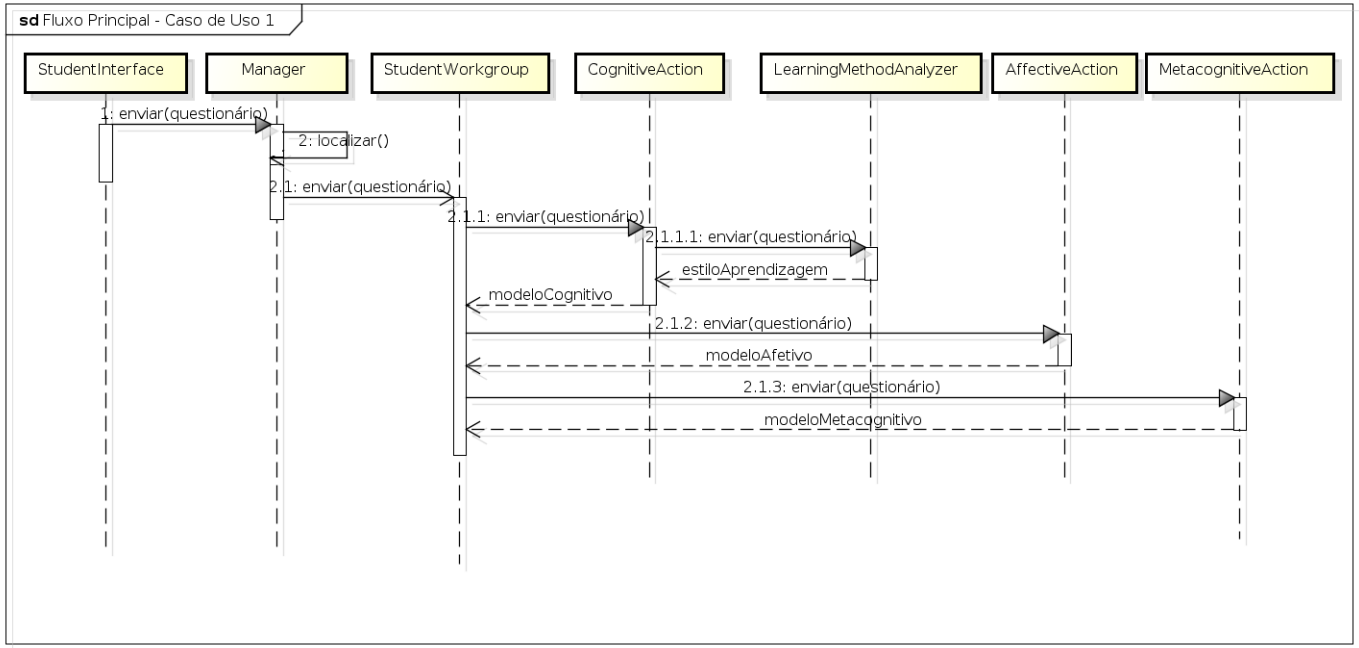


Figura 3.1: Diagrama de sequência do fluxo principal, caso de uso 1.

tempo na codificação da comunicação e garantindo o foco na inferência à ser feita pelo SMA. Segue a descrição do caso de uso.

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 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.

A imagem 3.1 refere-se ao fluxo principal do caso de uso 1. Neste diagrama de sequência, 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 imagem 3.2 representa o fluxo de exceção do primeiro caso de uso. A regra *WebServiceInterface* 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 imagem 3.3 representa o fluxo principal do caso de uso 3, inferência afetiva. 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.

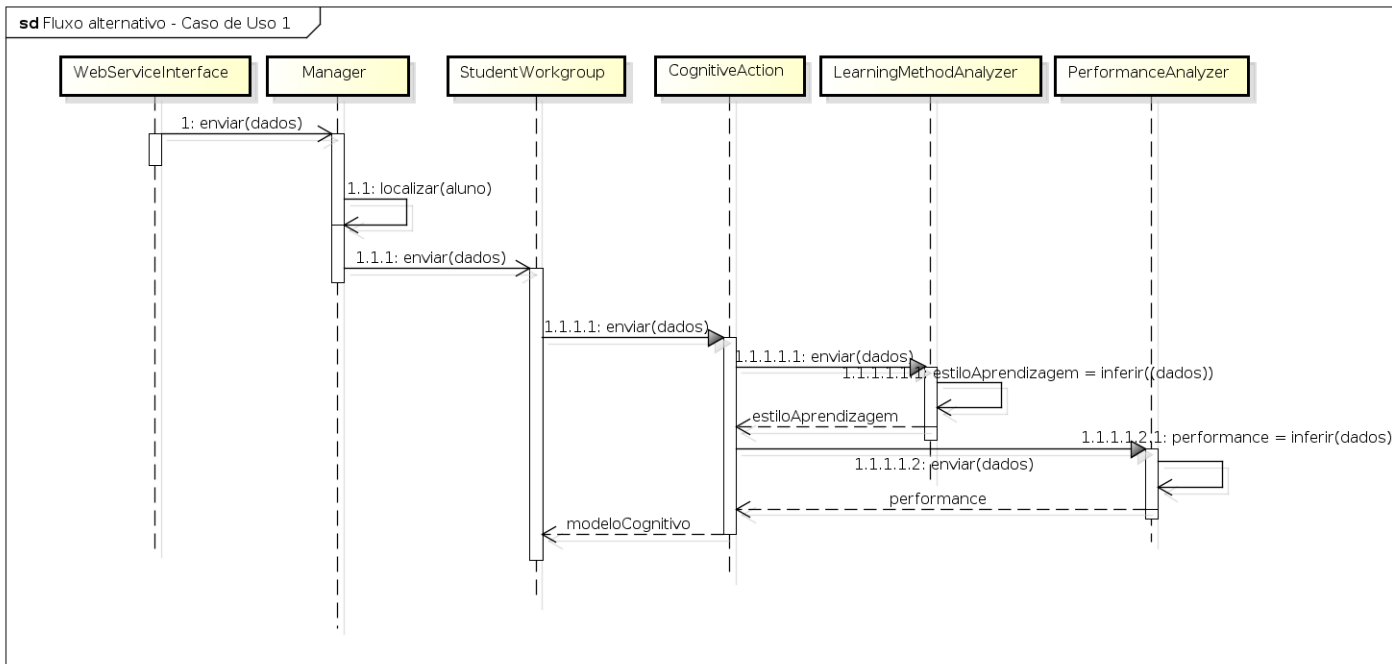


Figura 3.2: Diagrama de sequência do fluxo de exceção, caso de uso 1.

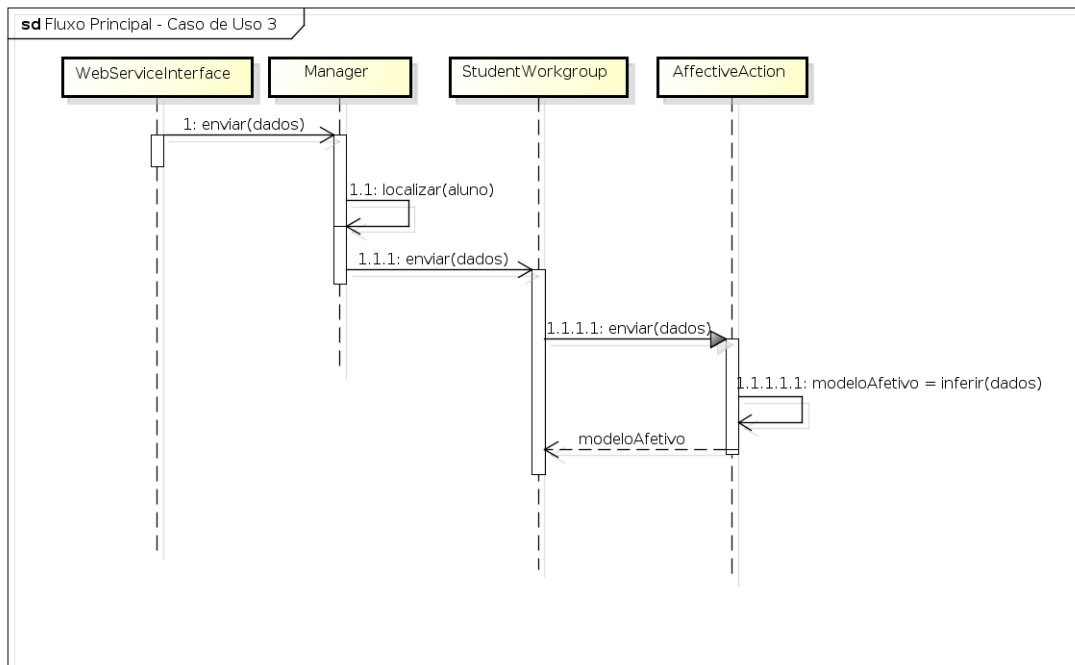


Figura 3.3: Diagrama de sequência do fluxo principal, caso de uso 3.

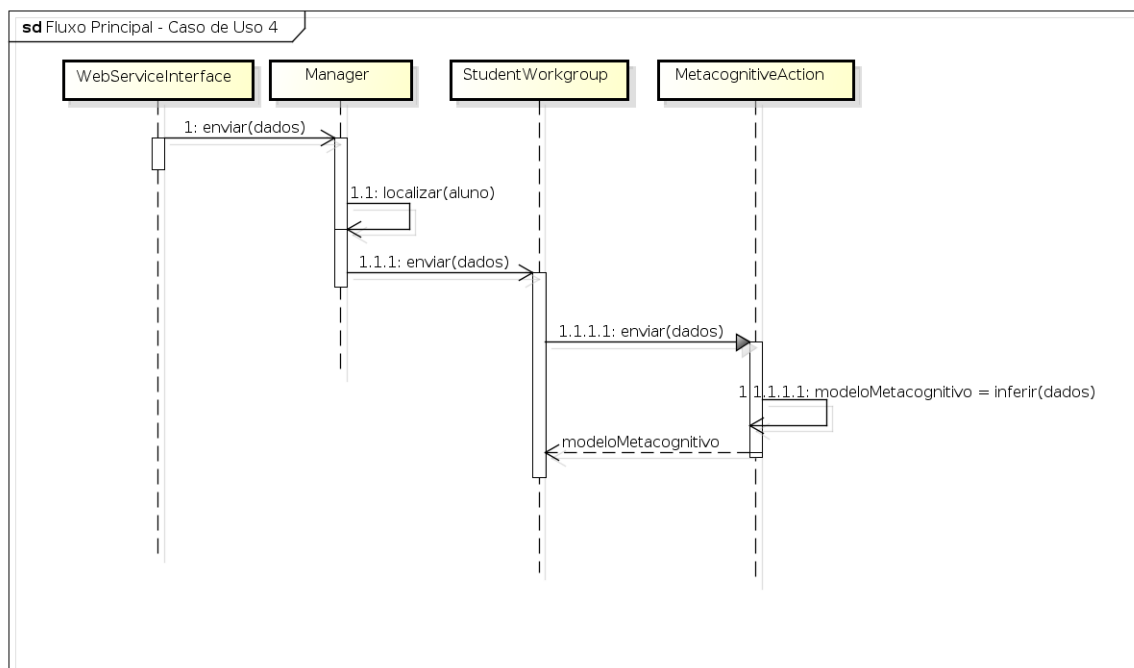


Figura 3.4: Diagrama de sequência do fluxo principal, caso de uso 4.

O diagrama de sequência 4 3.4 funciona similar ao caso de uso anterior, com a diferença de que a regra *MetacognitiveAction* realiza a inferência da modelagem metacognitiva.

Por fim o último diagrama de sequência 3.5 representa 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.

Por fim, após o levantamento de todas as regras foi necessário criar tarefas que satisfaçam o cumprimento das regras. Em outras palavras, é necessária a criação do *MASE Role Model*. A tabela 3.2 representa as metas criadas para o SMA, bem como as suas respectivas tarefas. É importante ressaltar que, devido aos objetivos deste trabalho, houve um refinamento muito maior do agente cognitivo. Os agentes afetivos e metacognitivos foram apenas projetados na arquitetura.

A imagem 3.6 apresenta o *MASE Role Model* e a sua estruturação das regras e tarefas. De forma geral, a regra *Manager* é a responsável pela gerência de todo o SMA. Ela pode receber os dados da regra *StudentInterface* (Interface Web da Aplicação) ou *WebServiceInterface* (Ambientes Virtuais de Aprendizagem). Além disso, a regra *StudentInterface* possui uma tarefa para autenticação do aluno, onde ela encaminha uma mensagem para a regra *Manager* que cria o workgroup do aluno.

A regra *StudentWorkgroup* será a regra responsável pela gerência do grupo de trabalho do aluno. Ela recebe os dados da regra *Manager* e reencaminha para as regras *Cognitive*, *Affective* e *Metacognitive*. As linhas tracejadas de cor azul representam comunicações internas entre as regras.

A ferramenta *agentTool* automaticamente indicou a criação dos *Concurrent Tasks Diagrams* para cada tarefa. O fluxo de execução pode iniciar-se nas regras *StudentInterface* ou *WebServiceInterface*. Elas representam respectivamente as interações com o Aluno e

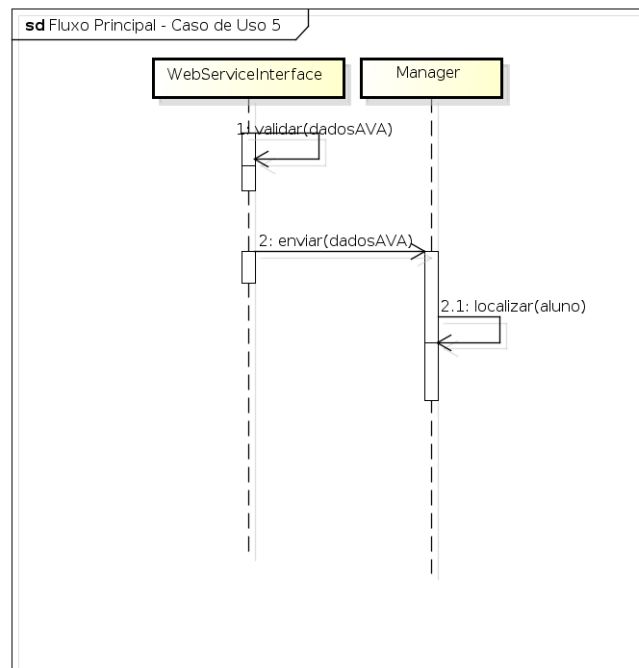


Figura 3.5: Diagrama de sequência do fluxo principal, caso de uso 5.

Tabela 3.2: Estruturação das Tarefas por Regra

Regra	Tarefas
StudentInterface	Validar Dados, Autenticar Aluno
WebServiceInterface	Validar Dados
Manager	Determinar Workgroup do Aluno, Criar Workgroup do Aluno
StudenWorkgroup	Processar Dados, Atualizar Modelo
CognitiveAction	Separar Dados de Aprendizagem, Atualizar Modelo Cognitivo, Atualizar Performance
MetacognitiveAction	Inferir Modelo Cognitivo
AffectiveAction	Inferir Modelo Afetivo
LearningMethodAnalyzer	Analisar Estilo de Aprendizagem
PerformanceAnalyzer	Analisar Performance

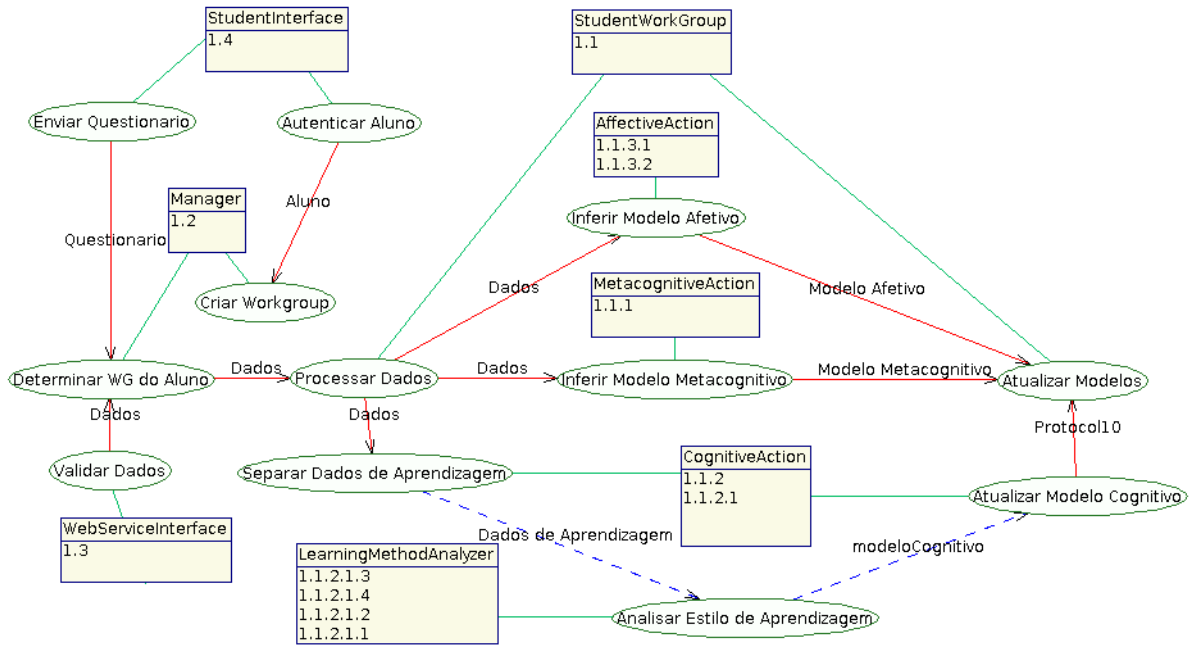


Figura 3.6: Diagrama *MASE Role Model* gerado para o SMA Frank.

com o Ambiente Virtual de Aprendizagem.

Na regra *WebServiceInterface*, o diagrama da tarefa "Validar Dados" 3.7 inicia com o recebimento de uma mensagem de um agente "a" e os dados "DadosWebService". A tarefa passa ao estado "ValidarDados", onde ele recupera as variáveis "idUserio" e "status" (verificação se o usuário é válido no ambiente). Em seguida a variável "status" é testada: Caso seja inválida o agente "a" recebe um código de erro. Caso contrário, a tarefa muda para o estado "enviarDados". Basicamente, o estado procura onde está a tarefa "manager" e encaminha os dados "DadosWebService".

O diagrama da tarefa "Enviar Questionário" 3.8, da regra *StudentInterface* inicia-se com uma mensagem da plataforma web contendo os dados do questionário. Esses dados são convertidos para uma linguagem comum no SMA. Em seguida, esse questionário é enviado para o manager.

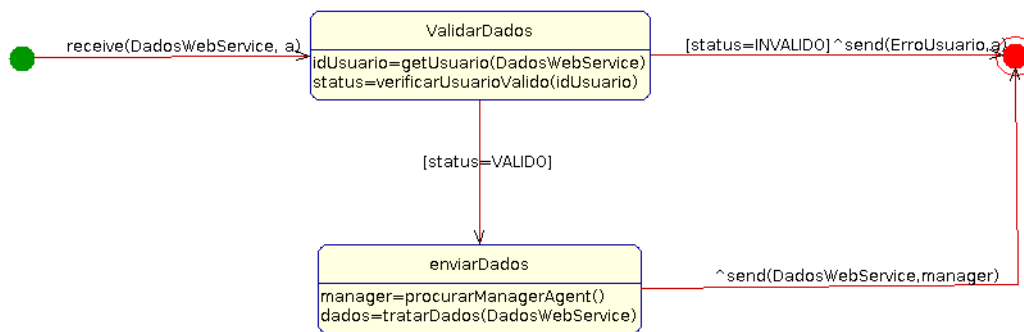


Figura 3.7: Detalhamento da tarefa "Validar Dados" que pertence à regra *WebServiceInterface*.

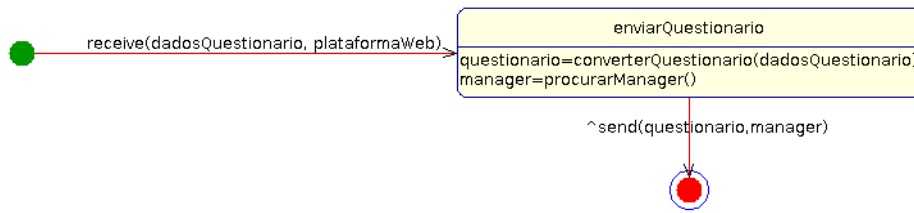


Figura 3.8: Detalhamento da tarefa "Enviar Questionário" que pertence à regra *StudentInterface*.

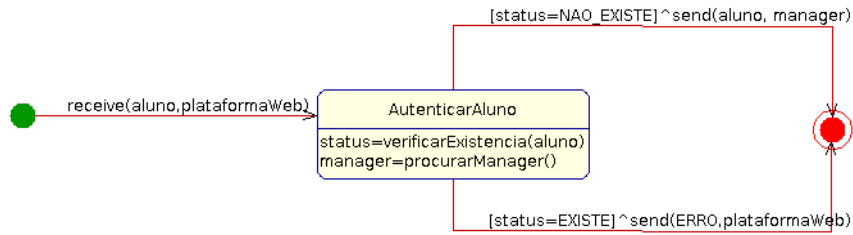


Figura 3.9: Detalhamento da tarefa "Autenticar Aluno" que pertence à regra *StudentInterface*.

Ainda na regra *StudentInterface*, o diagrama da tarefa "Autenticar Aluno" 3.9 inicia com uma mensagem da plataforma web contendo o aluno a ser criado. A tarefa verifica a existência do aluno e em caso negativo, envia a mensagem de criação à tarefa *Manager*. Caso contrário, envia uma mensagem de erro à plataforma Web.

Agora na regra *Manager*, o diagrama da tarefa "Determinar WG do Aluno" 3.10 (Determinar Workgroup do Aluno) inicia-se com entrada de uma mensagem recebida de um agente *agent* e o estado "DeterminarWorkgroup". Basicamente, ele apenas procura o grupo de trabalho do aluno. Por fim, pelo teste de validade do grupo de trabalho (*wg = VALIDO*), os dados são enviados para o respectivo grupo de trabalho caso ele seja válido. Caso contrário, é enviado uma mensagem de erro ao agente que iniciou a conversação.

O diagrama da tarefa "Criar Workgroup" 3.11 da regra *Manager* inicia-se com entrada de uma mensagem vinda da tarefa *studentInterface* e a mudança para o estado "criarWorkgroup". O estado cria os agentes cognitivo, metacognitivo, afetivo e o workgroup propriamente dito. Por fim, caso o workgroup seja válido, ou seja, não tenha ocorrido nenhum erro durante a criação, é enviado um código de sucesso à tarefa *studentInterface*.

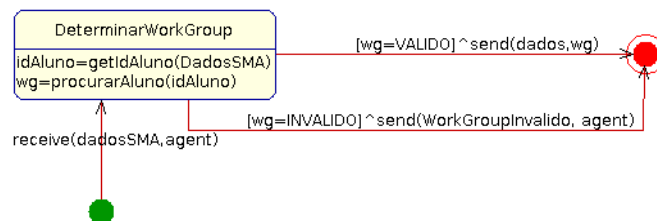


Figura 3.10: Detalhamento da tarefa "Determinar WG do Aluno" que pertence à regra *Manager*.

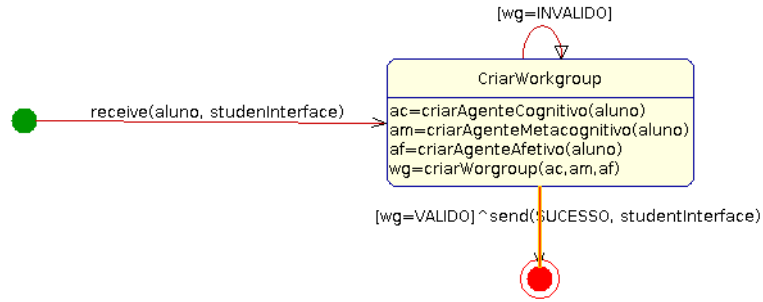


Figura 3.11: Detalhamento da tarefa "Criar Workgroup" que pertence à regra *Manager*.

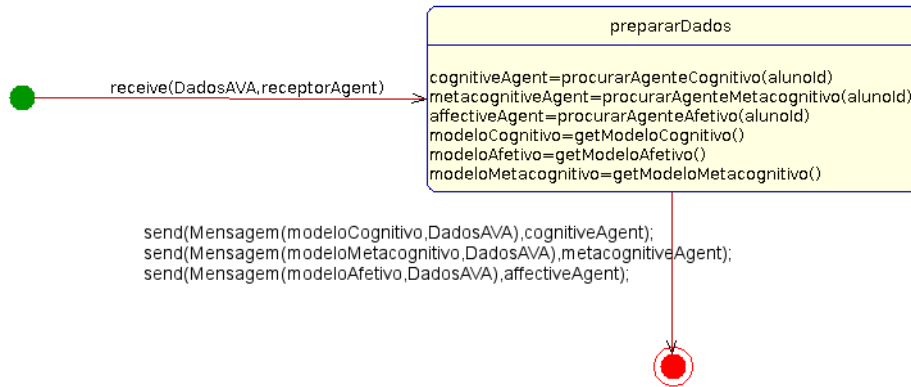


Figura 3.12: Detalhamento da tarefa "Processar Dados" que pertence à regra *StudentWorkgroup*.

Caso contrário, o workgroup deve ser criado novamente.

Agora na regra *StudentWorkgroup*, o diagrama da tarefa "Processar Dados" 3.11 mostra que os modelos do aluno são separados e enviados junto com os dados do SMA para os respectivos agentes.

O diagrama da tarefa "Atualizar Modelos" 3.13 pode parecer mais complexo, porém sua execução é basicamente simples. Ele representa o recebimento das inferências cognitiva, metacognitiva e afetiva. A transição inicia-se com o recebimento de um desses modelos. A tarefa vai para o estado "Aguardar Modelo Completo", onde verifica se os três modelos já foram inferidos. Se a verificação for válida, a tarefa é terminada e o objetivo de inferência dos modelos é atingido. Caso contrário, a tarefa aguarda as outras inferências e muda seu estado quando elas chegarem, repetindo assim o fluxo inicial.

As regras *AffectiveAction* e *MetacognitiveAction* possuem tarefas semelhantes. Os seus diagramas 3.14 e 3.15 basicamente mostram que elas recebem a mensagem do *workgroup* e realizam a sua inferência. Devido ao objetivo deste trabalho não ser estudar especificamente essas inferências, mas, criar a arquitetura do SMA Frank, foi previsto apenas os agentes e a etapa de inferência dos modelos.

A regra *CognitiveAction* possui um detalhamento maior, visto que será necessário a inferência explícita do estilo de aprendizagem do aluno. Para tanto, as regras possuem uma comunicação interna (representada pela linha tracejada azul) significando que provavelmente estarão no mesmo agente. De forma simples as tarefas da regra *CognitiveAction*

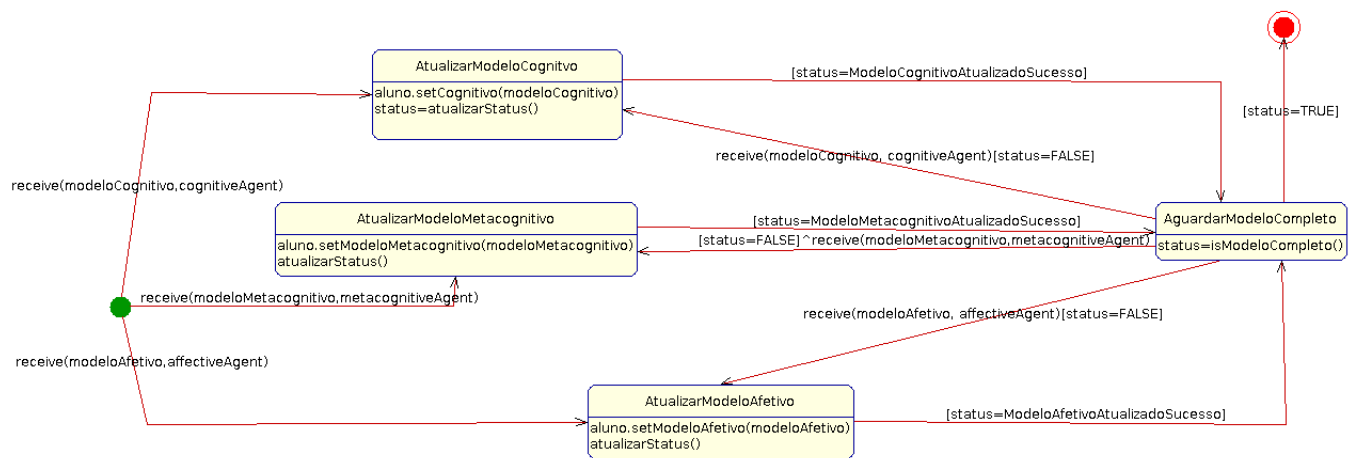


Figura 3.13: Detalhamento da tarefa "Atualizar Modelos" que pertence à regra *StudentWorkgroup*.

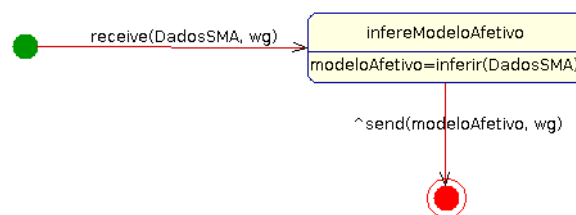


Figura 3.14: Detalhamento da tarefa "Inferir Modelo Afetivo" que pertence à regra *AffectiveAction*.

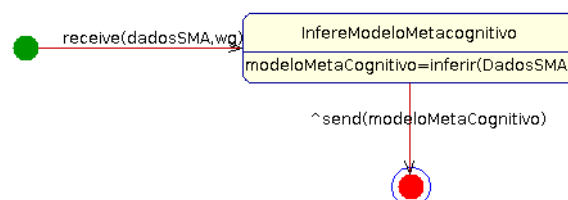


Figura 3.15: Detalhamento da tarefa "Inferir Modelo Metacognitivo" que pertence à regra *MetacognitiveAction*.

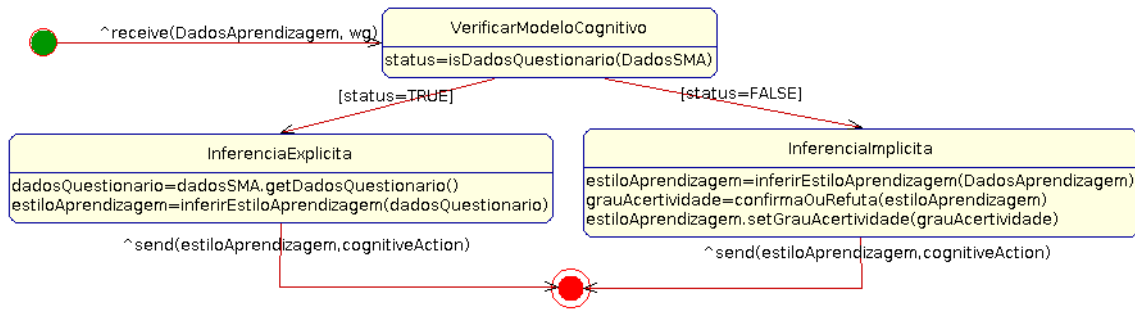


Figura 3.16: Detalhamento da tarefa "Analisar Estilo de Aprendizagem" que pertence à regra *LearningMethodAnalyzer*.

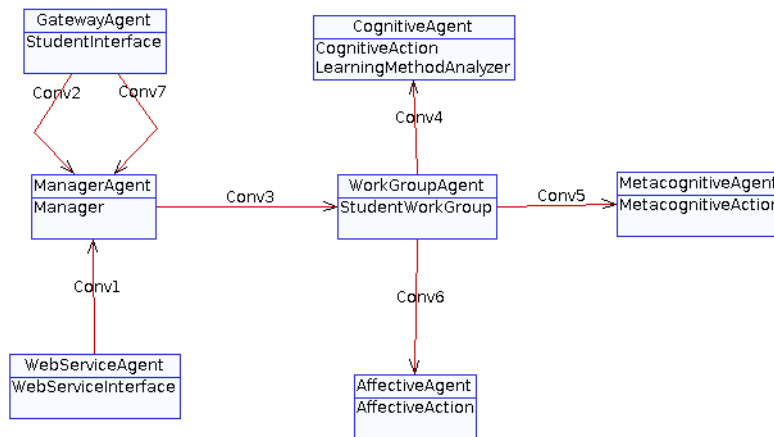


Figura 3.17: Diagrama de Classes do SMA Frank.

"Separar Dados de Aprendizagem" e "Atualizar Modelo" fazem o semelhante que já foi mostrado em outras tarefas. Portanto, o diagrama destas tarefas foi suprimido.

A regra *LearningMethodAnalyzer* possui a tarefa, onde seu diagrama 3.16 mostra o processo de análise do estilo de aprendizagem. Basicamente, o diagrama mostra que a inferência deve ser do tipo explícita caso os dados sejam o questionário. Caso contrário, a inferência deve ser implícita e pode variar de acordo com o ambiente virtual de aprendizagem.

3.2.2 Design

Após a conclusão da primeira etapa do MASE, é necessário definir as classes dos agentes, bem como as suas conversações. A arquitetura 3.17 do SMA Frank é composta pelos seguintes agentes:

- GatewayAgent - Possui a regra *StudentInterface*, responsável pela interface com a plataforma web, ou seja, com o estudante e o docente.
- WebServiceAgent - Possui a regra *WebServiceInterface*, é responsável pela interface com os ambientes virtuais de aprendizagem.

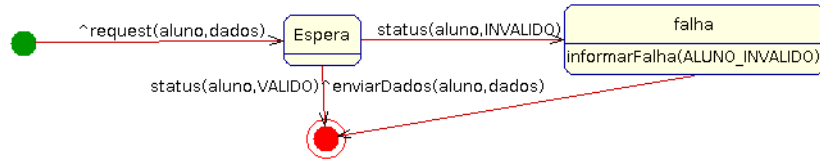


Figura 3.18: Detalhamento da Conversação 1 no lado do Iniciador.

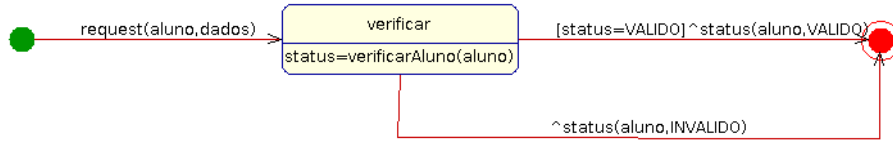


Figura 3.19: Detalhamento da Conversação 1 no lado do Respondedor.

- Manager - Possui a regra *Manager*, é responsável pela gerência da plataforma: Criação dos agentes, encaminhamento de mensagens.
- WorkgroupAgent - Possui a regra *StudentWorkgroup*, responsável pela gerência do grupo de trabalho de um determinado aluno.
- CognitiveAgent - Possui as regras *CognitiveAction*, *LearningStyleAction*. Responsável pela inferência do modelo cognitivo do aluno.
- AffectiveAgent - Possui a regra *AffectiveAction*. Responsável pela inferência do modelo afetivo do aluno.
- MetacognitiveAgent - Possui a regra *MetacognitiveAction*. Responsável pela inferência do modelo metacognitivo do aluno.

As conversações são definidas da seguinte forma:

- Conv1 - Conversação do *WebServiceAgent* com o *ManagerAgent*
- Conv2 - Conversação do *GatewayAgent* com o *ManagerAgent*
- Conv3 - Conversação do *ManagerAgent* com o *WorkgroupAgent*
- Conv4, Conv5 e Conv6 - Conversação do *WorkGroupAgent* com os agentes *CognitiveAgent*, *MetacognitiveAgent* e *AffectiveAgent*, respectivamente.

A primeira conversação (conv1) ocorre de forma simples. No lado do iniciador da conversação 3.18, ele solicita ao respondedor a verificação da existência do aluno na plataforma e entra no estado de espera. Caso a resposta seja *usuarioValido*, o iniciador irá enviar a mensagem *enviarDados* e encerrar a execução normalmente. Caso contrário, entrará no estado de falha, onde informará erro de execução. No lado respondedor da conversação 3.19, ela se inicia com o request do iniciador e o estado verificar, onde é verificado a existência do aluno. Por fim, é enviado uma resposta sobre a existência do aluno para o iniciador da conversação.

A segunda conversação (conv2) possui a mesma dinâmica da conversação 1, portanto seu diagrama será suprimido.

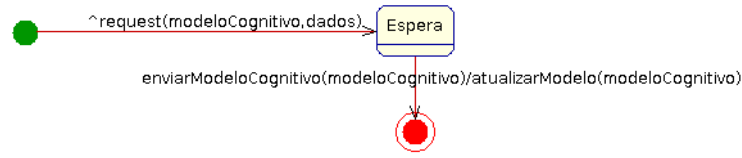


Figura 3.20: Detalhamento da Conversação 4 no lado iniciador.

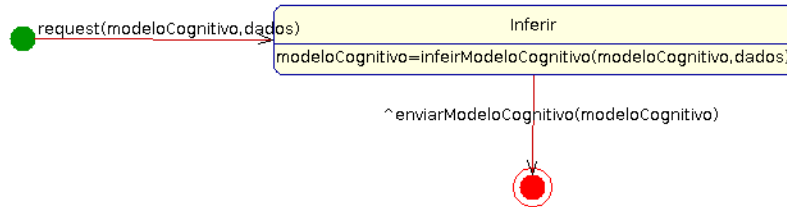


Figura 3.21: Detalhamento da Conversação 4 no lado iniciador.

A conversação 3 (conv3) é bastante simples. Não há nenhum estado de transição durante a conversação, pois é apenas um encaminhamento de dados para o grupo de trabalho do aluno.

As conversações 4, 5 e 6 (conv4, conv5 e conv6) possuem dinâmica bastante semelhante. Do lado do iniciador, basicamente ele envia um request com o modelo a ser inferido, os dados a serem analisados e entra em estado de espera. Em seguida, quando receber a resposta, ele atualiza o modelo cognitivo. A imagem 3.20 detalha a conversação 4, que pode ser generalizado para a 5 e 6.

O lado do receptor recebe o request e infere o modelo do aluno. Por fim, apenas reenvia novamente ao iniciador da conversação. A imagem 3.21 detalha a conversação 4, que pode ser generalizado para a 5 e 6.

A última conversação, conv7 3.22, é representa a conversação para criação do grupo de trabalho do aluno. No lado do iniciador, ele requisita a verificação da existência do aluno no ambiente. Caso não exista, ele envia a resposta de criação do grupo de trabalho para o receptor da conversação.

Do lado do receptor 3.23, ele recebe o request e faz a verificação da existência do aluno. Caso já exista, a conversação é encerrada. Caso contrário, ele entra no estado de espera e em seguida faz a criação do workgroup.

A arquitetura de deploy do trabalho deve ser dinâmica, visto que os agentes de trabalho devem ser instanciados dinamicamente. Funcionando em ambiente descentralizado, o SMA Frank precisa balancear a carga de uso entre os ambientes disponíveis. Dessa forma, o diagrama de deploy foi desnecessário.

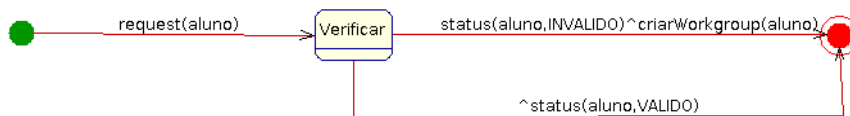


Figura 3.22: Detalhamento da Conversação 7 no lado iniciador.

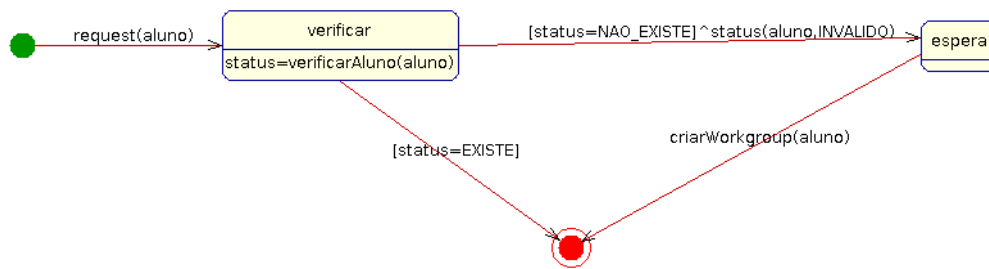


Figura 3.23: Detalhamento da Conversação 7 no lado receptor.

3.3 Arquitetura

A arquitetura do sistema foi separada em duas aplicações: A parte Web chamada de *Frank Web* e a parte Sistema Multiagente chamada de *SMA Frank*. Esta seção pretende relatar a arquitetura interna de cada parte, bem como as dificuldades encontradas na integração entre ambas.

A solução foi separada devido aos seguintes aspectos:

- Maior Distribuição - É possível replicar a parte web em vários nós de uma rede, possibilitando assim um maior ganho de performance da aplicação.
- Menor Complexidade - Os objetivos das aplicações estão separados, modularizando assim as responsabilidades de cada parte.
- Menor Dificuldade - A parte Web é responsável pela interação com o usuário e com o banco de dados. Estes pontos seriam muito mais difíceis de implementar no framework JADE.

3.3.1 Frank Web

A parte Web utiliza-se do framework JBoss Seam, conforme dito anteriormente, um framework que acelera o tempo de desenvolvimento de aplicações dinâmicas para a Web. Para facilitar a manutenção do código e a reescrita de alguma parte dele, foi adotado o padrão de projetos Model View Controller (MVC). Esse padrão divide a arquitetura do sistema em três partes:

- Apresentação - Responsável pela apresentação dos dados para o usuário em uma página web.
- Controladora - Determina o componente a ser executado.
- Modelo - Representação das entidades, auxiliam na interação com o banco dados.

As ferramentas disponibilizadas pelo framework geraram todo o código para visualização, inserção, atualização e exclusão de dados. Além disso, geraram todas as páginas para que o usuário possa interagir com o sistema.

Após a geração do código inicial, foi necessário uma implementar algumas funcionalidades específicas da aplicação Frank-Web. A primeira delas foi implementar a ação de

autenticação conjunta com o SMA. Quando o aluno fizer login no sistema, a aplicação web envia uma mensagem ao sistema multiagente para a criação do grupo de trabalho.

Em seguida, foi necessário a implementação a tela de respostas do questionário feita pelo usuário.

Por fim, a implementação da tela de notificação ao docente do estilo de aprendizagem do aluno.

3.3.2 SMA Frank

O SMA Frank foi desenvolvida com o *middleware* JADE, devido à sua grande aceitação na comunidade de Sistemas Multiagentes e o extenso suporte da comunidade. A aplicação está dividida entre os agentes criados e os seus comportamentos (justificados na seção anterior).

A comunicação entre os agentes foi implementado utilizando-se de ontologias. O pacote de ontologia do JADE permite a criação de abstrações muito robustas, descartando neste primeiro momento a utilização de bibliotecas de terceiros.

No JADE, os agentes devem adicionar comportamentos, que são disparados quando o agente recebe uma mensagem. Os comportamentos são separados conforme o objetivo do agente.

Foram implementadas duas ontologias. A primeira, chamada de *FrankManagementOntology*, tem a função de gerenciamento da plataforma. Logo as ações de criação e destruição de agentes estarão presentes nela.

A ontologia *ModelInferOntology* possui a função de modelar todos os conceitos relacionados à inferência do modelo do aluno. Portanto, conceitos como estilo de aprendizagem, modelo cognitivo, afetivo e metacognitivo devem estar nesta ontologia. É importante ressaltar que os modelos não estão completos, sendo necessário que em trabalhos futuros sejam modeladas as ontologias.

3.3.3 Integração Entre as Aplicações

A integração entre o *Frank Web* e o *SMA Frank* pareceu bastante complexa em uma análise inicial. Por não compartilhar a mesma Máquina Virtual Java (JVM), a dificuldade de integração pareceu alta.

O Jade porém possui uma forma nativa em que aplicações externas podem se conectar com o ambiente em execução. A classe *DynamicJadeGateway* registra um agente na plataforma, que atuará como uma ponte entre a aplicação e o ambiente de execução.

Esse agente funcionará de forma distinta dos outros agentes da plataforma. O agente não receberá mensagens da aplicação Web, ao invés disso receberá objetos JAVA que representarão comandos. Para os diferentes comandos, o agente pode lançar diferentes mensagens na plataforma SMA. A aplicação *Frank SMA* implementa 6 comandos:

- AnswerCommand
- CreateAgentCommand
- DestroyAgentCommand
- DimensionCommand

- ProcessQuestionnaireCommand
- RequestCognitiveModelCommand

Capítulo 4

Experimentações

Esta seção apresenta o protótipo desenvolvido para a parte web da solução. Serão mostradas as telas de interação com o aluno e professor. Para tanto, foi definida uma metodologia 4.1 das experimentações que explicará como foi dividido os cenários de testes. A seção 4.2 mostra o fluxo de execução desde o primeiro acesso do aluno à visualização do seu estilo de aprendizagem. A seção 4.3 mostra o fluxo de execução do docente que deseja verificar o estilo de aprendizagem de seus alunos.

Por fim, o capítulo de resultados 4.4 destaca todos os resultados esperados do cenários do aluno e dos objetivos do projeto.

4.1 Metodologia de Testes

A metodologia de testes foi dividida em dois cenários. O primeiro deles é o cenário onde o aluno faz o primeiro acesso ao sistema e deseja conhecer o seu estilo de aprendizagem. É seguido então um fluxo de execução para que este aluno possa realizar o questionário de estilos de aprendizagem. No fim do fluxo, espera-se que o aluno saiba o seu estilo de aprendizagem.

O segundo cenário aborda o acesso do docente ao sistema, desejando saber o estilo de aprendizagem dos alunos de sua turma. É demonstrado um fluxo de execução para que ele autentique-se no sistema e acesse os dados de sua turma. Ao final deste fluxo, é esperado que o docente saiba o estilo de aprendizagem de todos os alunos que realizaram o questionário. As duas próximas seções mostram os fluxos explicados.

4.2 Demonstração da Interface com Aluno

O fluxo inicia-se quando o aluno acessa a interface principal do Sistema 4.1, mostrando uma tela de boas vindas. O usuário seleciona vai para a tela de autenticação e o Sistema exige o login e a senha de acesso 4.2. Caso o usuário informe o login inválido o Sistema apresenta a tela de erro 4.3, solicitando novamente os dados para autenticação.

Após a autenticação, o usuário com perfil de aluno realiza o primeiro acesso ao Sistema. Ele detecta que o usuário não fez o questionário de estilos de aprendizagem e apresenta a tela de convite ao preenchimento do mesmo 4.4.

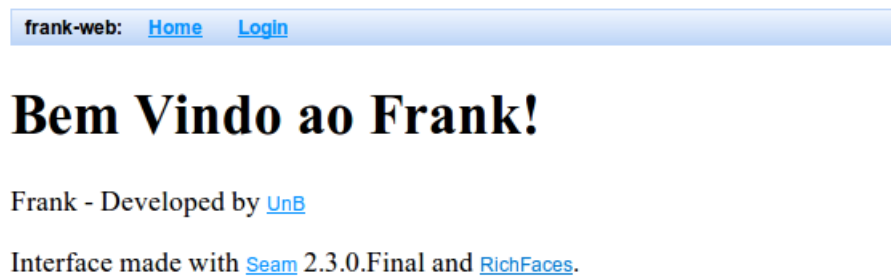


Figura 4.1: Tela Inicial do Sistema.

The screenshot shows a web browser window displaying a login form. At the top, there is a blue header bar with the text "frank-web:" followed by two links, "Home" and "Login", both in blue. Below the header, there is a blue bar with the word "Login" in white. The main content area has a light blue background. It starts with the text "Por favor, você deve autenticar-se" in black. Below this, there are two input fields: "Login" with the text "professor" and "Senha" with masked characters "*****". Below the password field, there is a checkbox labeled "Lembrar de mim?". At the bottom of the form, there is a blue button with the word "Login" in white.

Frank - Developed by [UnB](#)
Interface made with [Seam](#) 2.3.0.Final and [RichFaces](#).

Figura 4.2: Tela de Autenticação do Sistema.

frank-web: [Home](#) [Login](#)

- Falha no Login

Login

Por favor, você deve autenticar-se

Login

Senha

Lembrar de mim? ☐

[Login](#)

Frank - Developed by [UnB](#)

Interface made with [Seam](#) 2.3.0.Final and [RichFaces](#).

Figura 4.3: Tela de Erro de Autenticação do Sistema.

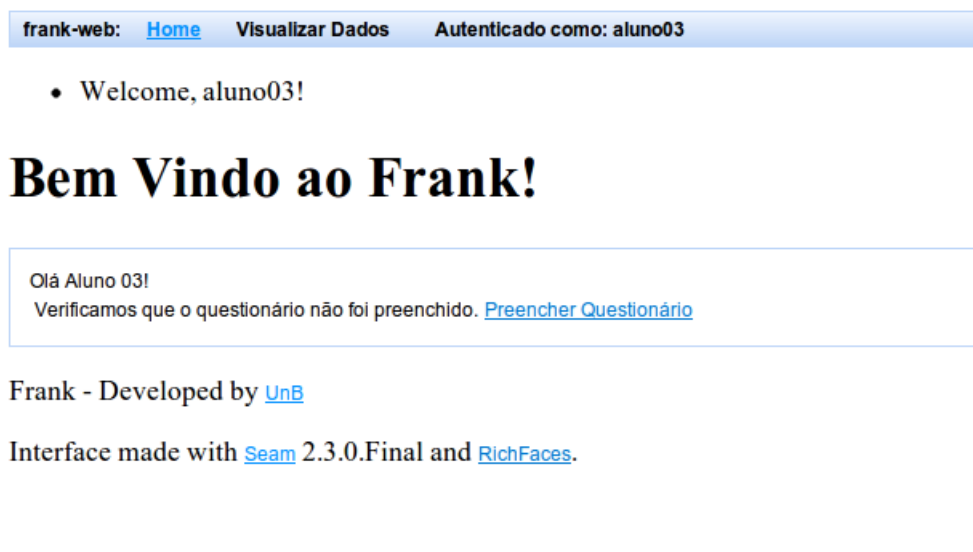


Figura 4.4: Tela de convite ao preenchimento do questionário de estilos de aprendizagem.

Titulo	Resposta
Prefiro discutir questões concretas e não perder tempo com idéias abstratas.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Tendo a ser perfeccionista.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Nas reuniões, apoio as idéias práticas e realistas, independentemente dos métodos o mais importante é que as coisas funcionem.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Entusiasmo-me ter que fazer algo novo e diferente.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Preocupo-me em interpretar, cuidadosamente a informação disponível antes de tirar uma conclusão.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Incomoda-me que as pessoas não tomem as coisas a sério.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
É mais importante para mim que o professor apresente a matéria em etapas sequenciais claras.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Quando escrevo um texto, eu prefiro trabalhar (pensar, escrever) diferentes partes do texto e ordená-los depois.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente
Eu sou capaz de formular respostas originais e criativas, com frequência.	<input type="radio"/> Discordo Totalmente <input type="radio"/> Discordo <input type="radio"/> Concordo <input type="radio"/> Concordo Totalmente

Frank - Developed by [UnB](#)

Interface made with [Seam](#) 2.3.0.Final and [RichFaces](#).

Figura 4.5: Tela de preenchimento do questionário.

O usuário então entra na tela de preenchimento do questionário 4.5, preenche todas as respostas de acordo com suas características e em seguida seleciona o botão "Confirmar". O Sistema envia os dados para o SMA e o seu estilo de aprendizado é inferido.

Por fim, o Sistema exibe o resultado final do processamento do SMA na tela de boas vindas 4.6. Dessa forma, o usuário verá sempre o seu estilo de aprendizagem atualizado ao acessar o Sistema.

4.3 Demonstração da Interface com Docente

O fluxo do docente é semelhante ao fluxo do aluno até a autenticação passo de autenticação 4.2. O Sistema distingue os usuários neste momento e, no caso do Docente, renderiza uma tela diferente devido ao seu perfil. Ele renderiza a tela de visualização de turmas às quais pertencem ao docente em questão.

O Docente seleciona a turma desejada e o Sistema apresenta a tela de visualização de estilos de aprendizagem dos alunos pertencentes àquela turma. É importante notar que nem todos os alunos possuem um estilo de aprendizagem inferido visto que, estes devem fazer acesso ao Sistema para a inferência do estilo ou, no futuro, podem possuir desvios no seu estilo de aprendizagem.

4.4 Resultados

Após a execução das experimentações, foi possível observar vários aspectos do sistema. O primeiro deles é a assistência do aluno por meio do grupo de trabalho específico para ele, com os agentes cognitivo, metacognitivo e afetivo. Com esse grupo de trabalho será possível a inferência do seu modelo multidimensional por meio de dados de Ambientes Virtuais de Aprendizagem.

frank-web: [Home](#) Visualizar Dados Autenticado como: aluno03

Bem Vindo ao Frank!

Olá Aluno 03!

Estilo de Aprendizagem: Assimilador

Descrição: Aprendem por observação reflexiva e conceitualização abstrata...

Frank - Developed by [UnB](#)

Interface made with [Seam](#) 2.3.0.Final and [RichFaces](#).

Figura 4.6: Tela de visualização do estilo de aprendizagem inferido.

frank-web: [Home](#) Visualizar Dados Autenticado como: professor

- Welcome, professor!

Bem Vindo ao Frank!

Olá Professor!


Você possui 1 turma neste momento

Data Inicio	Data Fim	Quantidade de Alunos	Ação
2/10/2013	3/12/2013	6	View

Frank - Developed by [UnB](#)

Interface made with [Seam](#) 2.3.0.Final and [RichFaces](#).

Figura 4.7: Tela inicial do usuário com o perfil de docente.

frank-web: Home Visualizar Dados Autenticado como: professor		
Turma Details		
Data de Inicio 2/10/13 Data de Fim 3/12/13		
Edit Done		
 Docente		
Nome	Estilo de Aprendizagem	Action
João Paulo	Acomodador	View
Aluno 04		View
Aluno 05		View
Aluno 06		View
Aluno 03	Assimilador	View

Frank - Developed by [UnB](#)

Interface made with [Seam 2.3.0.Final](#) and [RichFaces](#).

Figura 4.8: Tela de visualização do estilo de aprendizagem por turma.

Além disso, por meio de preenchimento de questionário e consequentemente a modelagem explícita, houve a determinação do seu estilo de aprendizagem. O agente cognitivo possui inteligência para interpretar as respostas do questionário e determinar o seu estilo de aprendizagem.

Por fim, através do cenário do docente, foi possível visualizar a notificação do estilo de aprendizagem dos seus alunos feito pela plataforma.

Capítulo 5

Conclusões e Trabalhos Futuros

Este capítulo analisa o trabalho realizado discutindo o alcance dos objetivos por meio da arquitetura e protótipo desenvolvidos. Em seguida, são indicados uma série de possíveis trabalhos futuros em cima da plataforma Frank.

Este trabalho visou o auxílio de docentes na observação de seus alunos, bem como melhoria na sua didática de ensino ao observar a melhor forma que seus alunos podem absorver os conhecimentos adquiridos em sala de aula. Em face disso, a arquitetura proposta objetivou traçar os modelos cognitivo, afetivo e metacognitivo do aluno, criando insumos para o professor desenvolver melhor suas didáticas de ensino.

Ao fim deste trabalho, foi possível definir e implementar a arquitetura geral do SMA Frank:

- Dois agentes de interfaces com o ambiente externo;
- Um agente para controle do ambiente do SMA;
- Um grupo de trabalho para cada aluno, composto por 4 agentes: O grupo de trabalho e os agentes cognitivo, metacognitivo e afetivo.

Além disso, a implementação do protótipo da plataforma web é capaz de interagir tanto com os alunos, quanto com os seus professores.

Para tanto, com os objetivos alcançados, conclui-se que este trabalho possui uma potencial importância para a sociedade como um todo, pois auxilia o complexo e longo processo de aprendizagem de alunos e possui capacidade para a melhoria significativa do trabalho do docente.

A arquitetura do SMA Frank está proposta, porém é necessária a continuação de diversos aspectos na arquitetura. Primeiramente, como trabalho futuro faz-se necessário obter a modelagem metacognitiva e afetiva dos alunos, além das suas respectivas formas de inferências implícitas.

Como trabalho futuro, é necessário uma integração com um ambiente de aprendizagem dedicado e realizar testes reais com alunos e docentes para a validação das inferências realizadas.

Por fim, é desejável que a plataforma web possua um controle maior do ambiente SMA. Ou seja, uma administração web que controle todo o SMA, sendo possível a criação e remoção remota de agentes, o balanceamento dos agentes entre as diversas máquinas disponibilizadas para aplicação, a visualização do estado atual de cada grupo de trabalho de cada aluno.

Referências

- [1] About apache struts 1. <http://struts.apache.org/development/1.x/>. Acessado em 18/02/2013. 33
- [2] About apache struts 2. <http://struts.apache.org/development/2.x/>. Acessado em 18/02/2013. 33
- [3] Fipa communicative act library specification. <http://www.fipa.org/specs/fipa00037/index.html>. Acessado em 01/02/2013. 15
- [4] Javaserer faces technology. <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>. Acessado em 18/02/2013. 33
- [5] Jsr 299: Contexts and dependency injection for the java ee platform. <http://jcp.org/en/jsr/detail?id=299>. Acessado em 18/02/2013. 33
- [6] The knowledge sharing effort. <http://www-ksl.stanford.edu/knowledge-sharing/papers/kse-overview.html>. Acessado em 01/02/2013. 12
- [7] Object management group. <http://www.omg.org/>. Acessado em 05/01/2013. 24
- [8] The seam framework - next generation enterprise java development. <http://www.seamframework.org/>. Acessado em 10/02/2013. 33
- [9] D. Allen. *Seam in action*. Manning, 2009. vii, 34, 35
- [10] John L. Austin. *How to do things with words*. Harvard U.P., Cambridge, Mass., 1962. 11
- [11] Gilberto Bravos Bativa and Itana Stiubiener. Ferramenta de identificação de perfis de aprendizes-fipa. 35
- [12] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. Jade programmer's guide. *university of Parma*, pages 200–2003, 2002. 31
- [13] Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007. vii, 30, 31

- [14] Alessandra Rodrigues Débora Nice Ferrari José Emiliano Alex Francisco Oliveira Cláudio Geyer, Adriana Soares Pereira. Semeai - sistema multiagente de ensino e aprendizagem na internet. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 1, pages 293–299, 2001. 35
- [15] A. Cockburn. *Writing effective use cases*. Agile software development series. Addison-Wesley, 2001. 25, 27
- [16] Germana Menezes da Nóbrega. Frank: Re-utilização de abordagens e ferramentas da ie em um sistema multi-agente para identificação de estilos de aprendizagem. Outubro 2011. 37
- [17] Gelson Luiz Daldegan de Pádua. A epistemologia genética de jean piaget. *Revista FACEVV/ 1^o Semestre de*, (2):22–35, 2009. 4
- [18] E.H. Durfee, V.R. Lesser, and D.D. Corkill. Trends in cooperative distributed problem solving. *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):63–83, 1989. 10
- [19] Richard M Felder and Linda K Silverman. Learning and teaching styles in engineering education. *Engineering education*, 78(7):674–681, 1988. 5
- [20] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. vii, 24, 26
- [21] Michael Genesereth, Richard E. Fikes, Ronald Brachman, Thomas Gruber, Patrick Hayes, Reed Letsinger, Vladimir Lifschitz, Robert Macgregor, John McCarthy, Peter Norvig, and Ramesh Patil. Knowledge interchange format version 3.0 reference manual, 1992. 12, 14
- [22] C. LARMAN. *Utilizando UML e Padrões*. Bookman, 3 edition, 2008. vii, 25, 27, 28, 30
- [23] Eduardo PENTERICH. Ambientes virtuais de aprendizagem. *Sala de Aula e Tecnologias*. São Paulo: Editora da Universidade Metodista de São Paulo, 2005. 4
- [24] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998. 5
- [25] A. Preece. Knowledge query and manipulation language: A review. *University of Aberdeen, Aberdeen, Tech. Rep*, 1997. 12
- [26] Humberto Rabelo, Wolgrand Cardoso Braga Jr, Leônidas Leão Borges, Ed Porto Bezerra, Edna Gusmão de Góes Brennand, Tatiana Aires Tavares, and Guido Lemos de Souza Filho. Identificação do perfil individual intelectual do educando no ambiente virtual de aprendizagem edulivre. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 1, 2010. 35
- [27] Stuart Russel and Peter Novig. *Artificial Intelligence - A Modern Approach*. Number 1. Pearson Education, Upper Saddle River, New Jersey 07458, 1995. 6, 8, 15

- [28] A.D. SCOTT, F. Mark, and H.S. CLINT. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03):231–258, 2001. 18, 19, 20, 21
- [29] J.R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cam: [Verschiedene Aufl.]. Cambridge University Press, 1969. 11
- [30] Carlos Vinícius Sarmiento Silva. Agentes de mineração e sua aplicação no domínio de auditoria governamental. Acessado em 18/11/2011. Disponível em <http://monografias.cic.unb.br/dspace/handle/123456789/318>, Abril 2011. 9
- [31] Volmer Campos Soares and Vandro Roberto Vilardi Rissoli. Agente inteligente no apoio ao ensino-aprendizagem. In *Anais do Simpósio Brasileiro de Informática na Educação*, volume 1, 2011. 34
- [32] J. Verschueren and J.O. Östman. *Key Notions for Pragmatics*. Handbook of Pragmatics Highlights. John Benjamins Publishing Company, 2009. 11
- [33] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Number 1. John Wiley and Sons Ltd, Krst Sussex PO10 1JJD, England, 2004. 7, 8, 10, 12, 14