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

GILLEANES THORWALD ARAUJO GUEDES

Um Metamodelo UML para a Modelagem de Requisitos em Projetos de Sistemas MultiAgentes

Tese apresentada como condição parcial
para obtenção do título de doutor
em ciência da computação

Profa. Dra Rosa Maria Vicari
Orientadora

Porto Alegre, março de 2012

CIP - Catalogação na Publicação

Guedes, Gilleanes Thorwald Araujo
Um Metamodelo UML para a Modelagem de Requisitos em Projetos de Sistemas MultiAgentes / Gilleanes Thorwald Araujo Guedes. -- 2012.
229 f.

Orientadora: Rosa Maria Vicari.

Tese (Doutorado) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2012.

1. AOSE. 2. Requirements Modeling. 3. Metamodels. 4. AgentRole_Actor. 5. Internal Use Case. I. Vicari, Rosa Maria, orient. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Dr. Carlos Alexandre Netto
Vice-Reitor: Prf. Dr. Rui Vicente Oppermann
Pró-Reitor de Pós-graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Dr. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Dr. Álvaro Freitas Moreira
Bibliotecária-chefe: Beatriz Regina Bastos Haro

Agradecimentos

Gostaria de, neste espaço, agradecer as seguintes pessoas que permitiram a conclusão deste trabalho.

À professora doutora Rosa Maria Vicari pela orientação e apoio.

Ao professor doutor Marcelo Soares Pimenta pelas sugestões e conselhos.

Ao Thiago Thompsen Primo, Elder Rizzon dos Santos, Michelle Denise Leonhardt Camargo e Gustavo Kieling, meus colegas durante o doutorado.

Às professoras doutoras Carla T. L. L. Silva Schueneman e Viviane Torres da Silva pelo auxílio e material fornecido.

Ao meu pai, William Guedes pelo auxílio e apoio.

À Ederson Silveira pelo Apoio ao aplicar o meu trabalho.

A Soraia Silva Prietch pelo grande apoio na minha liberação para cursar o doutorado.

A Carlos Emilio Padilla Severo e Leonivan Dias Moraes pelo apoio.

ÍNDICE

Índice 4
Lista de Abreviaturas e Siglas 7
Lista de Figuras 8
Lista de Tabelas 13
ABSTRACT 14
RESUMO 15
1 Introdução 16
1.1 Justificativa 16
1.2 Objetivos 18
1.3 Contribuições Esperadas 19
1.4 Estratégia de Pesquisa 20
1.4.1 Fases da Pesquisa 21
1.5 Cronograma 23
2 Revisão de Literatura 25
2.1 UML, Metamodelos e Perfis 25
2.2 Agent Unified Modeling Language (AUML) 26
2.3 AORML – Agent-Object-Relationship Modeling Language – Linguagem de Modelagem Agente-Objeto-Relacionamento 29
2.3.1 Objetos e Agentes 29
2.3.2 Compromissos e Reivindicações 30
2.3.3 Modelos AOR externos 30
2.3.4 Ações 32
2.3.5 Diagramas de Quadro de Interação 33
2.3.6 Diagramas de Seqüência de Interação 34
2.3.7 Modelos AOR Internos 35
2.3.8 Agentes Externos e Internos 35
2.3.9 Quadros de Reação 36
2.3.10 Modelagem dos Direitos e Deveres dos Agentes Internos 37
2.3.11 Regras de Reação 37
2.3.12 Considerações Finais 38
2.4 Agent Modeling Language (AML) 40
2.4.1 Semi-Entidades da AML 41
2.4.2 Entidades Fundamentais da AML 41
2.4.3 Aspectos Sociais 42
2.4.4 Unidades de Organização 43
2.4.5 Relacionamentos Sociais 44
2.4.6 Papéis e Propriedades de Papéis 44
2.4.7 Modelagem de Interações 47
2.4.8 Extensões Específicas de Atos de Fala para Interações UML 49

2.4.9 Observações (Percepções) e Interações de Efeito 49
2.4.10 Modelagem de Capacidades e Comportamento 50
2.4.11 Modelagem de Comportamento Social 50
2.4.12 Modelagem de Aspectos Mentais 52
2.4.13 Considerações Finais 54
2.5 MAS-ML – Multi-Agent System Modeling Language – Linguagem de Modelagem de Sistema Multi-Agente 55
2.5.1 Definição de Entidades de Sistemas Multi-Agentes 55
2.5.2 Definição de Características Estruturais de Sistemas Multi-Agentes 60
2.5.3 Aspectos Estáticos da MAS-ML 61
2.5.4 Aspectos Dinâmicos da MAS-ML 61
2.5.5 Definição de Características Comportamentais de Sistemas Multi-Agentes 62
2.5.6 Definição de Relacionamentos de Sistemas Multi-Agentes 64
2.5.7 Diagramas Estruturais da MAS-ML 65
2.5.8 Diagramas Dinâmicos da MAS-ML 70
2.5.9 Considerações Finais 75
2.6 Metamodelo de Agência proposto por Silva (2007) 75
2.6.1 Perfil de Agência 78
2.6.2 Diagrama Arquitetural (Architectural Diagram) 79
2.6.3 Diagrama de Comunicação (Communication Diagram) 79
2.6.4 Diagrama Ambiental (Environmental Diagram) 80
2.6.5 Diagrama Intencional (Intentional Diagram) 80
2.6.6 Diagrama Racional (Rational Diagram) 81
2.6.7 Diagrama de Plano (Plan Diagram) 81
2.6.8 Considerações Finais 82
2.7 Metodologias AOSE – Agent Oriented Software Engineering – Engenharia de Software Orientada a Agente 82
2.7.1 Gaia 82
2.7.2 MaSE – Multiagent Systems Engineering – Engenharia de Sistemas Multiagentes 83
2.7.3 MESSAGE/UML - Methodology for Engineering Systems of Software Agents – Metodologia para Engenharia de Sistemas de Agentes de Software 90
2.7.4 PASSI (Process for Agent Societies Specification and Implementation – Processo para Especificação e Implementação de Sociedade de Agentes) 101
2.7.5 Prometheus 113
2.7.6 Tropos 126
2.7.7 Resumo das Características das Linguagens e Metodologias Estudadas Comparadas ao Metamodelo 130
2.8 Princípios de Projeto Propostos por Vicari (2007) 131
3 Um Metamodelo UML voltado para a Modelagem de Requisitos em Projetos de Sistemas Multi-Agentes 138
3.1 Noções Básicas de Agência 138
3.2 Engenharia de Requisitos 139
3.3 Metaclasses Utilizadas na Modelagem de Casos de Uso em UML 142
3.3.1 Pacote Common Behaviors, Subpacote BasicBehaviors e Metaclasse BehavioredClassifier 143
3.3.2 Pacote Classes, Kernel, Metaclasse NameSpace, Metaclasse NamedElement, Metaclasse DirectedRelationship, Metaclasse Constraint e Metaclasse RedefinableElement 143

3.3.3 Metaclasse Classifier 143
3.4 O Metamodelo Proposto 144
3.5 Estudo de Caso I – O sistema MCOE (Multi-agent CO-operative Environment – Ambiente Co-operativo Multi-agente) 152
3.6 Estudo de Caso II – O ambiente AME-A - Ambiente Multi-Agente de Ensino-Aprendizagem 159
3.7 Adequação do Metamodelo aos Princípios de Projeto Propostos por Vicari (2007) 165
3.8 Estensibilidade das Metaclasses Definidas no Metamodelo 169
3.9 Estudo de Caso III – Sistema de Apoio Pedagógico ao Aluno – Sistema APA 169
3.9.1 Arquitetura de Camadas Proposta por Vicari (2007) 173
3.10 Mapeamento de Conceitos do Metamodelo Proposto para a Linguagem MAS-ML 175
3.11 Mapeamento de Conceitos do Metamodelo Proposto para a Linguagem AML 181
3.12 Documentação Textual dos Casos de Uso Internos 184
3.13 Como Aplicar o Metamodelo 191
3.14 Validação do Metamodelo Proposto 192
3.14.1 Comparação dos Diagramas Produzidos por Meio do Metamodelo com Diagramas Produzidos por Meio da UML 192
3.14.2 Validação por Justificativa 196
3.15 Ferramenta CASE Utilizada 200
4 Discussão e Considerações Finais 201
5 Trabalhos Futuros 205
referências 206
Anexo I 212
Sistema Multiagente para Criação/Edição/Alteração de OA 212
1 apresentação do Sistema 213
1.1 Sistema Multiagente de Autoria de OA 215
1.2 Agentes do Sistema 218
1.3 Casos de Uso do Sistema de Autoria 220
2 Aplicação METAMODELO no Sistema Multiagente para Autoria de AO 221
3 Comentários e opnião sobre aplicação do metamodelo sUgerido 227
REFERÊNCIAS BIBLIOGRAFICAS 228

LISTA DE ABREVIATURAS E SIGLAS

ACL	Agent Communication Language – Linguagem de Comunicação de Agente
AML	Agent Modeling Language – Linguagem de Modelagem de Agente
AOR	Agent-Object-Relationship – Relacionamento-Objeto-Agente
AOSE	Agent Oriented Software Engineering – Engenharia de Software Orientada a Agente
AORML	Agent-Object-Relationship Modeling Language – Linguagem de Modelagem Agente-Objeto-Relacionamento
AR	Análise de Requisitos
AUML	Agent Unified Modeling Language – Linguagem de Modelagem Unificada de Agente
BDI	Belief, Desire, Intention – Crença, Desejo, Intenção
DCU	Diagrama de Casos de Uso
DP	Design Principles – Princípios de Projeto
FIPA	Foundation for Intelligent Physical Agents – Fundação para Agentes Físicos Inteligentes
ITS	Intelligent Tutoring System – Sistema Tutor Inteligente
MAS	MultiAgent System – Sistema Multi-Agente
MAS-ML	Multi-Agent System Modeling Language – Linguagem de Modelagem de Sistema Multi-Agente
OMG	Object Management Group – Grupo de Gerenciamento de Objeto
PS	Problem Solving – Solução de Problema
SMA	Sistema Multi-Agente
SMI	Social Mediated Interactions – Interações Sociais Mediadas
STI	Sistema Tutor Inteligente
UAM	Users and Agent Modeling – Modelagem de Usuários e Agentes
UML	Unified Modeling Language – Linguagem de Modelagem Unificada

LISTA DE FIGURAS

Figura 2.2.1: Metamodelo de Superestrutura para o Diagrama de Classes da AUML .. 27
Figura 2.2.2: Comparação entre o Diagrama de Seqüência da AUML Original (esquerda) e as Novas Melhorias deste Diagrama Incorporadas a partir da UML em sua Versão 2.0 (direita) 28
Figura 2.3.1: Modelos AOR externo e interno 29
Figura 2.3.2: Elementos Centrais de Modelagem Externa AOR – Modelo Original. 31
Figura 2.3.3: Elementos Centrais de Modelagem Externa AOR – Modelo Atual 31
Figura 2.3.4: Um Diagrama de Agente AOR para o Domínio de Bibliotecas de Universidade. 32
Figura 2.3.5: Um Evento de Ação Comunicativa, um Evento de Ação Não Comunicativa e um Evento de Não-Ação. 33
Figura 2.3.6: O Quadro de Interação Entre a Biblioteca Central e as Bibliotecas de Departamento. 33
Figura 2.3.7: Um processo de Interação Não-social Envolvendo um Agente e seu Ambiente Inanimado. 34
Figura 2.3.8: Um Processo de Interação Social Envolvendo Três Agentes – Formato Original 34
Figura 2.3.9: Tipos de Meta-Entidade da Modelagem AOR Interna. 35
Figura 2.3.10: Tipo de Agente Externo, Agente Interno e Tipo de Agente Interno – Formato Original. 36
Figura 2.3.11: Diagrama de Quadro de Reação 36
Figura 2.3.12: Diagrama de Padrão de Reação. 37
Figura 2.3.13: Direitos e Deveres. 37
Figura 2.3.14: Um Padrão de Interação. 38
Figura 2.3.15: Diagrama de Casos de Uso Representando as Interações entre Agente por meio de Casos de Uso e Agentes por meio de um Ator e pela Fronteira do Sistema 39
Figura 2.4.1: Níveis de Definição AML. 40
Figura 2.4.2: Exemplo de Entidades, seus Relacionamentos e Fornecimento e Uso de Serviços. 42
Figura 2.4.3 – Exemplo de Modelo de Estrutura de Organização em Nível de Classe. . 43
Figura 2.4.4: Exemplo de Modelagem de Estrutura Social. 45
Figura 2.4.5: Exemplo de Instanciação e Execução de Papel de Entidade. 46
Figura 2.4.6: Tipos de papéis de entidade e associações play. 46
Figura 2.4.7: Exemplo de instanciação e interpretação de papel de entidade. 47
Figura 2.4.8: Exemplo de uma Interação Comunicativa. 48
Figura 2.4.9: Exemplo de uma Interação Social com Mudança de Papel de Entidade... 49
Figura 2.4.10: Exemplo de Fragmentos de Comportamento, Observações, Interações de Efeito e Capacidades. 50

Figura 2.4.11: Exemplo de Interação Social com Mudanças de Papel de Entidade..... 51
Figura 2.4.12: Exemplo de Atividade Compreendendo as Ações Sociais Induzindo Mudanças de Papéis de Entidade. 52
Figura 2.4.13: Exemplo de um Modelo Mental..... 53
Figura 2.4.14: Exemplo de Crenças e Objetivos Individuais e Gerais. 54
Figura 2.5.1: Metamodelo MAS-ML: Novas Metaclasses (parte I). 56
Figura 2.5.2: Metamodelo MAS-ML: Novas Metaclasses (parte II)..... 56
Figura 2.5.3: AgentClass 57
Figura 2.5.4: AgentRoleClass..... 59
Figura 2.5.5: Metamodelo MAS-ML: Novas Características Comportamentais. 60
Figura 2.5.6: Metamodelo MAS-ML: Novos Relacionamentos. 64
Figura 2.5.7: Exemplo de Diagrama de Classe. 66
Figura 2.5.8: Exemplo de Diagrama de Organização (parte I)..... 67
Figura 2.5.9: Exemplo de Diagrama de Organização (parte II). 68
Figura 2.5.10: Exemplo de Diagrama de Papel 69
Figura 2.5.11: Modelagem de Planos e Ações - Diagrama de Sequência..... 71
Figura 2.5.12: Modelagem de um Protocolo - Diagrama de Sequência. 72
Figura 2.5.13: Agente comprometendo-se com um papel (Diagrama de Sequência). ... 73
Figura 2.5.14: Diagrama de Atividade. 74
Figura 2.5.15: Movendo de Organização e Ambiente. 75
Figura 2.6.1: Metamodelo de Agência Refletindo Conceitos Intencionais..... 77
Figura 2.6.2: Metamodelo de Agência Refletindo Conceitos de Interação. 78
Figura 2.6.3: Estereótipos de Agência. 79
Figura 2.6.4: Diagrama Arquitetural. 79
Figura 2.6.5: Diagrama de Comunicação. 80
Figura 2.6.6: Diagrama Ambiental. 80
Figura 2.6.7: Diagrama Intencional. 81
Figura 2.6.8: Diagrama Racional. 81
Figura 2.7.1 – Etapas da Metodologia MaSE..... 83
Figura 2.7.2 — Diagrama de Hierarquia de Objetivos 85
Figura 2.7.3 — Diagrama de Seqüência 86
Figura 2.7.4 – Modelo de Papéis 86
Figura 2.7.5 – Modelo de Papéis MaSE..... 87
Figura 2.7..8 — Diagrama de Tarefas Concorrentes 87
Figura 2.7.9 – Diagrama de Classes de Agentes 88
Figura 2.7.10 — Diagrama de Classes de Comunicação 89
Figura 2.7.11 – Conceitos MESSAGE centralizados no Agente 92
Figura 2.7.12 – Nível 0 – Visão de Organização – Relacionamentos Estruturais..... 95
Figura 2.7.13 – Nível 0 – Visão de Organização – Relacionamentos de Conhecimento 96
Figura 2.7.14 – Nível 0 – Diagrama de Implicação de Objetivo/Tarefa..... 97
Figura 2.7.15 – Nível 0 – Diagrama de Fluxo de Trabalho (Workflow). 97
Figura 2.7.16 – Nível 1 – Diagrama de Organização (Relacionamentos de Conhecimento). 98
Figura 2.7.17 – Nível 1 – Diagrama de Estrutura de Delegação 99
Figura 2.7.18 – Diagrama de Nível de Interação..... 100
Figura 2.7.19 – Diagrama de Informação de Domínio 101
Figura 2.7.20 – Modelos e Fases da Metodologia PASSI..... 102
Figura 2.7.21 – Diagrama de Descrição de Domínio para o Estudo de Caso da Livraria 103

Figura 2.7.22 – Parte do Diagrama de Identificação de Agente Obtida a partir da Fase de Descrição de Domínio 104
Figura 2.7.23 – Diagrama de Identificação de Papéis para o cenário no qual o PurchaseMonitor Anuncia a Necessidade de Compra de Livros 105
Figura 2.7.24 – As Tarefas do Agente PurchaseManager 106
Figura 2.7.25 – Diagrama de Ontologia de Domínio 107
Figura 2.7.26 – Diagrama de Ontologia de Comunicação 108
Figura 2.7.27 – Diagrama de Descrição de Papéis para o Cenário de Compra de Livros 109
Figura 2.7.28 – Parte do diagram de Definição de Estrutura Multi-Agente para o Estudo de Caso de Livraria 110
Figura 2.7.29 – Diagrama de Definição de Estrutura de Agente-Único para o Agente PurchaseManager 111
Figura 2.7.30 – Diagrama de Descrição de Comportamento Multi-Agente 112
Figura 2.7.32 – Fases da Metodologia Prometheus. 114
Figura 2.7.33 – Diagrama de Cenários 116
Figura 2.7.34 – Fases da Metodologia Prometheus incluindo os objetivos e processo do sistema 116
Figura 2.7.35 – Diagrama de Visão Geral de Objetivos 117
Figura 2.7.36 – Diagrama de Papéis do Sistema 117
Figura 2.7.37 – Diagrama de Visão Geral de Análise 118
Figura 2.7.38 – Diagrama de Familiaridade de Agente (Agent Acquaintance Diagram). 119
Figura 2.7.39 – Diagrama de Acoplamento de Dados. 119
Figura 2.7.40 – Diagrama de Ligação de Papel de Agente. 120
Figura 2.7.41 – Diagrama de Visão Geral do Sistema. 120
Figura 2.7.42 – Diagrama de Visão Geral do Sistema. 121
Figura 2.7.43 – Diagrama de Interação (à esquerda) e Diagrama de Protocolo (à direita) 121
Figura 2.7.44 - Diagrama de Visão Geral de Agente 123
Figura 2.7.45 – Diagrama de Visão Geral de Agente. 123
Figura 2.7.46 – Diagrama de Capacidade Contendo Planos (Reserve book plan). 124
Figura 2.7.1: Sub-domínios de Aplicação e Tipos de Agente. 132
Figura 2.7.2: Divisão de Conhecimento de Agentes. 134
Figura 3.1: Conceitos Usados na Modelagem de Casos de Uso. 142
Figura 3.2: Metamodelo UML para a Modelagem de Requisitos Funcionais Específicos para o Projeto de Sistemas Multi-Agentes. 145
Figura 3.3: Metaclasse AgentRole_Actor 146
Figura 3.4: Metaclasses Reactive_AgentRole e Cognitive_AgentRole Derivadas a Partir da Metaclasse AgentRole_Actor 147
Figura 3.5: Metaclasse InternalUseCase 148
Figura 3.6: Metaclasses Perception, Action, Goal e Plan 149
Figura 3.7: Metaclasses Plan Extend e Plan Extension Point 151
Figura 3.8: Metaclasse IncludeActionPerception 152
Figura 3.9: Atores e Casos de Uso Normais Representando o Estudante, os Personagens do Jogo e as Funcionalidades Disponíveis. 153
Figura 3.10: Papel de Agente Reativo e Ações modelados por Meio de um AgentRole_Actor e Casos de Uso Internos Estereotipados. 154
Figura 3.11: Papel de Agente Cognitivo, Percepções, Objetivo e Plano modelados por Meio de um Ator AgentRole e Casos de Uso Internos Estereotipados 155

Figura 3.12: Detalhamento do Objetivo "Aid in the Maintenance of Energy Level" (Auxiliar na Manutenção do Nível de Energia) 156
Figura 3.13: Student Knowledge Diagram – Diagrama de Conhecimento do Aluno .. 157
Figura 3.14: Detalhamento do Plano "Help the Student" por meio de um Diagrama de Sequência 158
Figura 3.15: Tutor Knowledge Diagram – Diagrama de Conhecimento do Tutor..... 158
Figura 3.16: Diagrama de Casos de Uso Completo para a Modelagem dos Requisitos Funcionais do sistema MCOE 159
Figura 3.17: Atores Normais Modelando Papéis Externos, Casos de Uso Normais Modelando Funcionalidades e AgentRole_Actors Estereotipados Modelando Papéis de Agente Reativos e Cognitivos 161
Figura 3.18: Casos de Uso Internos com Estereótipos Goal, Plan, Perception e Action 161
Figura 3.19: Requisitos Relativos ao Papel de Agente Methodology and Teaching Plan 163
Figura 3.20: Requisitos relativos ao Papel de Agente Learning Orientation 163
Figura 3.21: Requisitos relativos ao Papel de Agente Learning Analysis 164
Figura 3.22: Requisitos Relativos ao Papel de Agente Knowledge Application Orienting 164
Figura 3.23: Diagrama de Casos de Uso Completo para a Modelagem dos Requisitos do Sistema AME-A 165
Figura 3.24: O Metamodelo UML Estendido para Suportar os Tipos de Agente Propostos por Vicari (2007). 166
Figura 3.25: Modelagem dos Requisitos do Projeto AME-A de Acordo com os Novos Estereótipos Criados. 167
Figura 3.26: Arquitetura de Camadas Proposta por Vicari (2007). 168
Figura 3.27: Requisitos Modelados dentro da Arquitetura em Camadas Proposta por Vicari (2007) para o Papel de Agente Student Modeling. 168
Figura 3.28: Diagrama de Casos de Uso para a Modelagem de Requisitos do Sistema APA 172
Figura 3.29: Requisitos Modelados dentro da Arquitetura em Camadas Proposta por Vicari (2007) para o Papel de Agente Pedagogical Support 174
Figura 3.30: Mapeamento do AgentRole_Actor Student Modeling e os Casos de Uso Internos a ele Associados em uma AgentRoleClass 175
Figura 3.31: Diagrama Parcial de Organização em MAS-ML Mapeado a Partir do Diagrama de Casos de Uso do Projeto AME-A. 176
Figura 3.32: Diagrama de Papéis para o Ambiente AME-A. 179
Figura 3.33: Diagrama de Seqüência Referente ao Objetivo de Modelar o Perfil do Aluno pelo Papel de Agent Student Modeling 180
Figura 3.34: Diagrama de Seqüência Referente ao Plano "Apply questionary" 181
Figura 3.35: Mapeamento de AgentRole_Actors em Tipos de Papel de Entidade AML 182
Figura 3.36: Observações e Interações de Efeito do Tipo de Papel de Entidade Methodology and Teaching Plan 183
Figura 3.37: Modelo Mental. 183
Figura 3.38: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao sistema MCOE. 192

Figura 3.39: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao projeto AME-A. 193
Figura 3.40: Alternativa para Modelagem AME-A por meio da UML Padrão Utilizando Associações de Extensão. 194
Figura 3.41: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao projeto do sistema APA. 195
Figura 3.42: Alternativa para Modelagem do sistema APA por meio da UML Padrão Utilizando Associações de Extensão. 196

LISTA DE TABELAS

Tabela 2.5.2: Estereótipos de Papel 72
Tabela 2.7.1 – Esquema de Papel 99
Tabela 2.7.1: Resumo das Linguagens e Metodologias Comparadas ao Metamodelo. 130
Tabela 3.1: Documentação do Caso de Uso Interno Student Modeling 184
Tabela 3.2: Documentação do Caso de Uso Interno Apply Questionary 185
Tabela 3.3: Documentação do Caso de Uso Interno Student Remodeling 186
Tabela 3.4: Documentação do Caso de Uso Interno Choose Methodology and Teaching Plan..... 186
Tabela 3.5: Documentação do Caso de Uso Interno Change Methodology and Teaching Plan..... 187
Tabela 3.6: Documentação do Caso de Uso Interno Present Material for Learning 188
Tabela 3.7: Documentação do Caso de Uso Interno Select Material for Learning 188
Tabela 3.8: Documentação do Caso de Uso Interno Verify Student Knowledge 189
Tabela 3.9: Documentação do Caso de Uso Interno Boosting Student 190
Tabela 3.10: Documentação do Caso de Uso Interno Evaluation Student 190
Tabela 3.11: Documentação do Caso de Uso Interno Apply Evaluation..... 191

ABSTRACT

This PhD thesis is inserted within the context of the AOSE (Agent-Oriented Software Engineering) area, a recently-emerged field dealing with the software engineering of multi-agent systems which mixes concepts of Artificial Intelligence and Software Engineering together.

This new area emerged from new challenges faced by the software engineers when designing multi-agent systems, since this kind of system presents characteristics that set them apart from other types of software, precisely for including software agents, autonomous and proactive entities that execute functions in the system, owning their own goals and able to perceive and act upon the surrounding environment without the intervention of external users.

This work describes a UML metamodel developed for the modeling of the specific functional requirements for multi-agent systems projects. Its development was based on the perception that among the studied UML-derived languages for the multi-agent systems project, none of them had developed mechanisms for requirements modeling on this kind of software, leading us to create a UML metamodel for this purpose.

Along this thesis we shall describe the UML-derived languages we studied to be applied in the multi-agent systems project, the developed metamodel, its adaptation to the Vicari (2007) design principles, three case studies on which the metamodel was applied, plus a mapping proposal for the concepts defined in the metamodel into MAS-ML and AML languages concepts, as well as a validation proposal for the metamodel and the diagrams created by means of it.

Keywords: AOSE, Requirements Modeling, Metamodels; Stereotypes, Agents, Agent Roles, AgentRole_Actor, Internal Use Case, Layers Architecture of Vicari (2007), MAS-ML, AML.

RESUMO

A presente tese de doutorado está inserida dentro do contexto da área de AOSE – Agent-Oriented Software Engineering, uma área surgida recentemente voltada para a engenharia de software de sistemas multi-agentes que mescla conceitos tanto da Inteligência Artificial como da Engenharia de Software.

Esta nova área surgiu devido aos novos desafios enfrentados pelos engenheiros de software ao projetar sistemas multi-agentes, uma vez que este tipo de sistema apresenta características que os diferenciam de outros tipos de software, precisamente o fato de possuírem agentes de software, entidades autônomas e pró-ativas que executam funções no sistema, possuidoras de objetivos próprios e capazes de perceber e agir sobre o ambiente que os cerca sem a intervenção de usuários externos.

Este trabalho descreve um metamodelo UML desenvolvido para a modelagem de requisitos funcionais específicos para projetos de sistemas multi-agentes. O seu desenvolvimento baseou-se na constatação de que, apesar de já existirem linguagens derivadas da UML para o projeto de sistemas multi-agentes, nenhuma das linguagens estudadas desenvolveu mecanismos para a modelagem dos requisitos deste tipo de software, o que levou-nos a criar um metamodelo UML para este propósito.

No decorrer desta tese serão descritas as linguagens estudadas derivadas da UML para ser aplicadas no projeto de sistemas multi-agentes, o metamodelo desenvolvido, sua adaptação aos princípios de projeto de Vicari (2007), três estudos de caso onde o metamodelo foi aplicado, além de uma proposta de mapeamento dos conceitos definidos no metamodelo para os conceitos das linguagens MAS-ML e AML, bem como uma proposta para a validação do metamodelo e dos diagramas criados por meio dele.

Palavras-Chave: AOSE, Requirements Modeling, Metamodels; Stereotypes, Agents, Agent Roles, AgentRole_Actor, Internal Use Case, Layers Architecture of Vicari (2007), MAS-ML, AML.

1 INTRODUÇÃO

Ao longo dos anos, pesquisadores na área de engenharia de software têm procurado definir métodos de desenvolvimento de software e linguagens de modelagem com o objetivo de estabelecer etapas bem definidas para a construção de software de maneira a tornar o desenvolvimento de sistemas mais robusto, rápido, organizado, confiável, mais fácil de manter e reutilizar e com maior qualidade.

Já na área de Inteligência Artificial, tem-se disseminado o uso de agentes inteligentes como auxiliares em softwares aplicados a diversos domínios (JAYATILLEKE, 2006), (RODRIGUEZ, 2011). Esta prática tem demonstrado ser uma boa alternativa para o desenvolvimento de sistemas complexos, acarretando um aumento no desenvolvimento de sistemas multi-agentes.

Porém, o desenvolvimento deste tipo de sistema apresentou novos desafios à área de Engenharia de Software, o que levou ao surgimento de uma nova sub-área que mescla conceitos tanto da Engenharia de Software como da Inteligência Artificial, conhecida como AOSE – Agent Oriented Software Engineering ou Engenharia de Software Orientada a Agentes – cujo objetivo é propor métodos e linguagens/notações para projetar e modelar softwares apoiados por agentes.

Dentro desse contexto, algumas tentativas foram feitas para adaptar e estender a UML (Unified Modeling Language – Linguagem de Modelagem Unificada) gerando novas linguagens para o projeto de sistemas multi-agentes. Contudo, nenhuma das linguagens estudadas preocupou-se em estender e aplicar alguns dos recursos da UML, tais como o diagrama de casos de uso, que é principalmente empregado para o levantamento e análise de requisitos, uma fase essencial para a concretização de um bom projeto de software.

Considerando a importância da fase de análise de requisitos para um bom projeto de software e considerando a falta de mecanismos para a modelagem de requisitos funcionais por parte das linguagens estudadas, optou-se por desenvolver um metamodelo UML para modelar requisitos funcionais específicos para projetos de sistemas multi-agentes. Para atingir este objetivo, foi adaptado e estendido o diagrama de casos de uso para este fim, uma vez que este diagrama é utilizado principalmente para a identificação dos requisitos funcionais que o sistema necessitará, identificando os atores que interagirão com o software, quais as funcionalidades necessárias ao sistema e quais atores poderão utilizar estas funcionalidades.

1.1 Justificativa

Autores da área de Engenharia de Software, como Sommerville (2003) e Presman (1995), afirmam que a identificação inadequada dos requisitos é uma das principais causas do fracasso de projetos de software. Sendo assim, consideramos de suma importância a identificação clara e correta das necessidades de um software antes que

ele venha a ser projetado e implementado, evitando que o software venha a ser desenvolvido sem atender as reais necessidades que deveria enfocar.

Dentro desse contexto, a UML é uma linguagem padrão de modelagem adotada internacionalmente pela área de engenharia de software, sendo amplamente aplicada durante as fases de análise de requisitos e projeto de sistema.

Algumas tentativas já foram feitas para adaptar a UML para o projeto de sistemas multi-agentes, criando-se novas linguagens de modelagem, no entanto, nenhuma das linguagens estudadas preocupou-se em criar mecanismos para modelar os requisitos funcionais específicos necessários aos sistemas multi-agentes, identificando suas funcionalidades e os atores que as poderiam utilizar, uma função costumeiramente desempenhada pelo diagrama de casos de uso da UML quando aplicado na análise/projeto de sistemas orientados a objeto.

O diagrama de casos de uso padrão da UML, no entanto, não suporta a representação de conceitos como agentes de software ou os papéis que estes interpretam em um sistema multi-agente e é igualmente incapaz de modelar os objetivos, planos, percepções e ações associados a estes papéis.

Dessa maneira, levando em consideração que a análise de requisitos é extremamente importante para o sucesso de um projeto de software, que a UML é uma linguagem de modelagem padrão amplamente aceita e compreendida pela área de engenharia de software e que nenhuma das linguagens estudadas que estenderam a UML para o projeto de sistemas multi-agentes preocupou-se em criar novas metaclasses para serem utilizadas pelo diagrama de casos de uso de forma a modelar requisitos funcionais necessários aos sistemas multi-agentes, nós nos propusemos a desenvolver um metamodelo UML que preencha esta lacuna.

Nesse metamodelo nós criamos novas metaclasses de forma que o diagrama de casos de uso possa ser utilizado para identificar os requisitos funcionais internos de um sistema multi-agente, mais especificamente, quais papéis de agentes deverão ser suportados pelo software, quais serão os objetivos associados a estes papéis, que condições deverão ser satisfeitas para que estes objetivos se tornem intenções, que planos deverão existir para que os objetivos sejam atingidos, quais as percepções necessárias a cada agente que assumir um determinado papel e quais ações serão permitidas a um papel específico.

A escolha da UML como a linguagem base a partir da qual estendemos nosso metamodelo justifica-se por esta ser uma linguagem amplamente aceita e compreendida nos meios acadêmicos e na indústria de engenharia de software, o que facilitaria a compreensão das inovações fornecidas pelo metamodelo. Além disso, a UML é uma linguagem genérica projetada com o objetivo de ser aplicada a todos os domínios e quando isto não for possível para ser facilmente estendida e adaptada a eles.

Nós optamos por estender diretamente a UML e não alguma das linguagens já derivadas a partir desta com o objetivo de que nossa extensão pudesse ser utilizada por quaisquer das linguagens que estenderam a UML para o projeto de sistemas multi-agentes e que não produziram mecanismos para a modelagem de requisitos funcionais específicos para este tipo de software.

Quanto à justificativa para desenvolver um metamodelo exclusivamente voltado para a modelagem de requisitos funcionais para sistemas multi-agentes, acreditamos que os requisitos deste tipo de software tenham características particulares, baseando-nos em Rodriguez (2011) que afirma que a modelagem de requisitos em sistemas multi-

agentes requer abstrações, técnicas e notações que tenham sido adaptadas particularmente para este tipo de domínio.

Além disso, Xiao (2003) afirma que abordagens orientadas a agentes estão se tornando populares para a modelagem de requisitos e Yu (1997) e Xiao (2003), afirmam que conceitos de agente tem sido introduzidos na engenharia de requisitos primariamente como construções de modelagem para caracterizar elementos ativos no ambiente.

Yu (1997) acrescenta que abordagens orientadas a agentes para engenharia de requisitos serão de particular interesse para ambientes nos quais exista um alto grau de computação distribuída aberta e nos quais ocorram mudanças freqüentes. O autor afirma também que, para que um framework de engenharia de requisitos seja verdadeiramente orientado a agente, a identidade e a existência de um agente como uma construção ao nível de engenharia de requisitos precisa ser determinada dentro deste nível, baseado sobre critérios de engenharia de requisitos. Yu (1997) declara ainda que definir como agentes participam de cenários, em que grau, combinação de ação e reflexão, são questões importantes para investigar em um framework de engenharia de requisitos orientado a agentes.

Tiryaki (2006) afirma que é quase impossível levantar todos os requisitos de um sistema multi-agente no início do desenvolvimento devido à complexidade deste tipo de sistema. Este autor destaca que um sistema multi-agente pode operar como uma organização aberta o que complica ainda mais a compreensão de todos os requisitos funcionais e não funcionais nas fases iniciais do desenvolvimento.

Werneck (2007) acrescenta que o aumento do uso de sistemas multi-agentes gerou desafios não estudados ainda, incluindo a maneira como adaptar a elicitação de requisitos para lidar com propriedades de agentes como autonomia, sociabilidade e proatividade.

Finalmente, Papasimeon (2003) afirma que a engenharia de software de sistemas multi-agentes demanda a especificação dos comportamentos de agente necessários para fornecer requisitos documentados para as fases de projeto e implementação.

1.2 Objetivos

Para contornar a ausência de mecanismos para a modelagem de requisitos funcionais em sistemas multi-agentes nas linguagens derivadas da UML para o projeto deste tipo de software, nos propomos a:

- Desenvolver um metamodelo UML criando novas metaclasses para permitir a modelagem específica de requisitos funcionais em sistemas multi-agentes, por meio do diagrama de casos de uso, identificando os papéis de agente que comporão o sistema, bem como seus objetivos, planos, percepções e ações;
- Modelar os requisitos de projetos de sistemas multi-agentes utilizando o metamodelo proposto;
- Evoluir o metamodelo adaptando-o para a proposta de Vicari (2007), criando estereótipos que permitam representar papéis de agente cognitivos do tipo UAM, PS e SMI, além de permitir a modelagem dos requisitos de cada papel de agente dentro das camadas da arquitetura de agentes sugerida;
- Procurar demonstrar como os conceitos do metamodelo proposto poderiam ser adaptados a outras linguagens de modelagem de sistemas multi-agentes

derivadas a partir da UML, por meio do mapeamento dos conceitos deste metamodelo para conceitos das outras linguagens, como, por exemplo, as linguagens MAS-ML e AML;

- Validar o metamodelo proposto por meio da comparação de diagramas produzidos pela UML padrão com diagramas produzidos por meio do metamodelo aplicados na modelagem de requisitos de sistemas multi-agentes.

1.3 Contribuições Esperadas

Com este trabalho esperamos criar um metamodelo UML adaptando e estendendo esta linguagem de maneira a permitir a modelagem de requisitos funcionais específicos para sistemas multi-agentes de acordo com as necessidades particulares deste tipo de software.

Consideramos nosso trabalho pertinente, uma vez que, de acordo com o que foi estudado até o momento sobre as linguagens de modelagem derivadas da UML para projetar sistemas multi-agentes – especificamente AUML (ODELL, 2004), (HUGET, 2003a), (HUGET, 2003b), (HUGET, 2004); AML (CERVENKA, 2005a), (CERVENKA, 2005b), (TRENCANSKY, 2005); AORML (WAGNER, 2003a), (WAGNER, 2003b), (TAVETER, 2005); MAS-ML (SILVA, 2004a), (SILVA, 2004b), (SILVA, 2008) e o metamodelo de agência proposto por (SILVA, 2007) – não houve preocupação em criar mecanismos que possam modelar os requisitos funcionais necessários a um sistema multi-agente, uma fase considerada crucial para o desenvolvimento de um bom projeto de software.

Nos trabalhos estudados não foram desenvolvidos mecanismos que permitissem a modelagem de requisitos específicos para sistemas multi-agentes, o que impediria o levantamento e análise dos requisitos necessários a um sistema deste tipo, como costuma-se observar na grande maioria dos ciclos de vida do sistema descritos em (PRESSMAN, 1995), (BEZERRA, 2002), (PAULA FILHO, 2003) e (SOMMERVILLE, 2003) e em metodologias de desenvolvimento de software, como, por exemplo, o Unified Process (UP – Processo Unificado) (BEZERRA, 2002), (WAZLAWICK, 2004) (ARLOW, 2005), onde suas primeiras etapas preocupam-se em identificar os requisitos necessários ao software que irá ser desenvolvido.

Dentro deste contexto, diversos autores recomendam o uso do Diagrama de Casos de Uso da UML para a modelagem de requisitos, como (RODRIGUEZ, 2011), (BERENBACH, 2009), (SOMMERVILLE, 2003), (PAPASIMEON, 2003), (BEZERRA, 2002), (MELO, 2002), (BOOCH, 2000) e (REGNEL, 1999). A função principal deste diagrama é o de identificar os atores que podem interagir com o sistema, as funcionalidades oferecidas pelo software e quais atores podem utilizar quais funcionalidades.

Neste diagrama, não há grande preocupação com o detalhamento das funcionalidades identificadas, a documentação de um caso de uso costuma conter apenas os passos gerais referentes a uma funcionalidade, o detalhamento mais profundo destas funcionalidades é desempenhado por outros diagramas da UML, o diagrama de casos de uso se atém essencialmente em identificar os requisitos necessários ao software. Rodriguez (2011) destaca que este diagrama tem demonstrado bons resultados na representação de requisitos funcionais e é uma boa forma de comunicação com os stakeholders. Isto é corroborado por Papasimeon (2003) que afirma que a análise de

casos de uso provou ser útil e bem sucedida para a especificação de requisitos em sistemas orientados a objetos.

Assim, com o objetivo de preencher a lacuna identificada nas linguagens de modelagem de sistemas multi-agente estudadas, no que se refere a recursos para a modelagem de requisitos funcionais específicos para este tipo de sistema, esperamos que este metamodelo permita adaptar o diagrama de casos de uso, fornecendo mecanismos que permitam a modelagem de papéis de agente cognitivos e reativos, bem como de suas ações, percepções, objetivos e planos em um nível de levantamento e análise de requisitos.

Seguindo o escopo do diagrama de casos de uso, pretendemos identificar os papéis de agente que comporão o sistema, quais as percepções e ações que eles deverão possuir, quais os seus objetivos e quais os planos necessários para atingí-los, apresentando uma visão geral dos requisitos funcionais necessários ao software, mas sem detalhar as funcionalidades por meio dos conceitos do metamodelo propriamente dito, posto que existem várias linguagens já derivadas da UML que podem ser empregadas para esse fim.

Esperamos ainda que este metamodelo possa contribuir com as outras linguagens derivadas a partir da UML para o projeto de sistemas multi-agentes, uma vez que, como já foi dito anteriormente, até onde estudamos, estas linguagens não enfocam a modelagem de requisitos.

1.4 Estratégia de Pesquisa

Neste trabalho nós adotamos a abordagem de pesquisa construtiva ou pesquisa de projeto (LUKKA, 2003), que busca produzir construções inovadoras, com o objetivo de solucionar problemas no mundo real e dessa forma contribuir para as áreas na qual ela é aplicada. Todos os artefatos humanos, como modelos, diagramas, planos, produtos comerciais e sistemas de informação, são considerados construções. Tais construções caracterizam-se pelo fato de que elas foram desenvolvidas, e não descobertas.

Kasanen (1993) afirma que este tipo de estratégia é amplamente adotada em ciências técnicas, matemáticas, medicina clínica, entre outras. Este autor afirma que a pesquisa construtiva pode ser encontrada até mesmo na filosofia nos casos em que o mundo é construído, passo a passo, a partir de elementos supostamente básicos como objetos, fatias de espaço-tempo, observações, pensamentos ou relações lógicas. Kasanen (1993) afirma ainda que a criação de linguagens artificiais, como linguagens de programação, são exemplos de construções. Outros exemplos de construções em outras áreas, como a medicina, são a produção de novos remédios ou novos tratamentos.

March e Smith (1995) propõem quatro tipos de saídas para a abordagem de pesquisa construtiva, construções, modelos, métodos e instanciações. Neste contexto, construções são o vocabulário conceitual do domínio de problema. Elas surgem durante a conceitualização do problema e são refinadas ao longo do ciclo do projeto. Com relação às construções, um modelo especifica os relacionamentos entre elas. Métodos fornecem uma forma de manipular as construções visando à obtenção de um modelo de solução. Finalmente, uma instanciação concretiza construções, modelos e métodos em um ambiente.

Kasanen (1993) sumariza, em termos de processo, as principais fases da abordagem de pesquisa construtiva:

1. Encontre um problema relevante em termos práticos que também tenha o potencial de investigação.
2. Obtenha um entendimento geral e abrangente do tema.
3. Inove, ou seja, construa uma idéia de solução.
4. Demonstre que a solução funciona.
5. Mostre as conexões teóricas e a contribuição de pesquisa do conceito da solução.
6. Examine o escopo de aplicabilidade da solução.

Kasanen (1993), acrescenta que estas fases podem ser aplicadas em ordens diferentes, variando caso a caso, destacando, porém, que a fase de inovação é o elemento central de um estudo construtivo. Este autor declara ainda que a pesquisa construtiva pode ser vista como um tipo de estudo aplicado, caracterizando-se pela produção de novos conhecimentos na forma de aplicações normativas.

Lukka (2003) sumariza os aspectos centrais da abordagem de pesquisa construtiva, conforme demonstrado na figura seguinte.

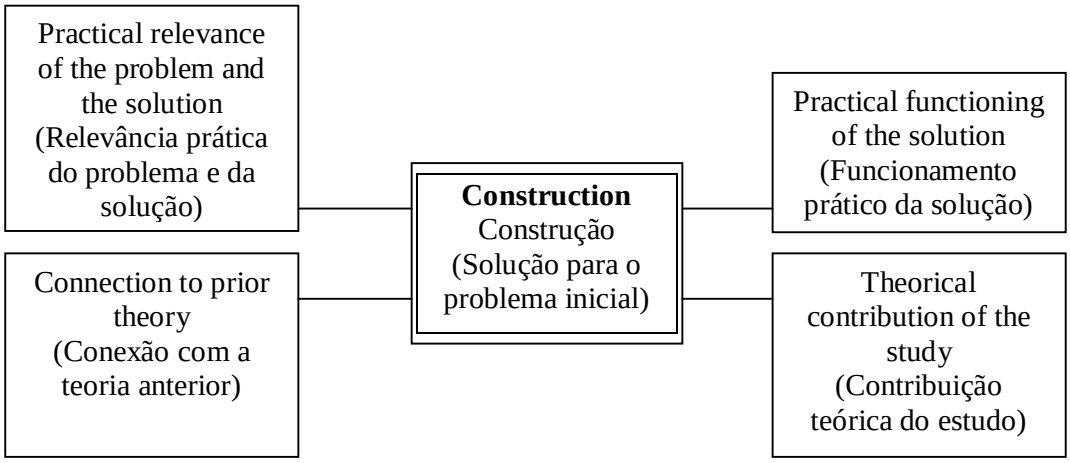


Figura 1.4.1: Elementos Centrais da Abordagem de Pesquisa Construtiva (LUKKA, 2003).

Levando em conta estas afirmações, consideramos a pesquisa construtiva adequada ao nosso trabalho, uma vez que o metamodelo proposto em nossa pesquisa não foi descoberto e sim desenvolvido, tratando-se, portanto, de uma construção. O referido metamodelo tem como objetivo fornecer uma solução para um problema do mundo real, de forma a permitir a modelagem de requisitos em projetos de sistemas multiagentes, contribuindo dessa forma para a área da Engenharia de Software Orientada a Agentes.

1.4.1 Fases da Pesquisa

1.4.1.1 Fase A

Nesta fase possuíamos um problema relevante em termos práticos e procuramos obter um entendimento geral e abrangente do tema.

Consideramos como problema relevante a lacuna encontrada nas linguagens pesquisadas que foram derivadas da UML para serem aplicadas ao projeto de sistemas

multi-agentes no que se refere à modelagem de requisitos, uma vez que a elicitação e análise de requisitos é considerada por muitos autores como de grande importância para o sucesso de um projeto de software e não identificamos mecanismos nas linguagens estudadas que permitam a modelagem de requisitos específicos para sistemas multi-agentes.

Durante esta fase realizamos uma pesquisa sobre metodologias AOSE, o que produziu dois trabalhos individuais; uma pesquisa sobre aplicações da UML e da AUML na modelagem de sistemas multi-agentes e suas limitações, o que gerou um artigo publicado na FP-UML 2009 dentro do ER 2009; e uma pesquisa sobre linguagens de modelagem derivadas da UML aplicadas ao projeto de sistemas multi-agentes, que serviu como base para identificar o problema relevante, além de ser utilizada em 5 outras publicações (SEKE 2010, IFIP AI 2010, QUATIC 2010, SEKE 2011 e EPIA/MASTA 2011). Também pesquisamos sobre metamodelos, modelos e perfis UML, sobre as funções das metaclasses utilizadas pelo diagrama de casos de uso e sobre engenharia de requisitos.

1.4.1.2 Fase B

Nesta fase tentamos construir uma idéia de solução para o problema identificado na fase anterior.

Para preencher a lacuna referente aos mecanismos para modelagem de requisitos não cobertos pelas linguagens estudadas, desenvolvemos um metamodelo UML criando novas metaclasses e estereótipos com o objetivo de permitir a modelagem de conceitos considerados por nós como requisitos importantes que devem ser levados em consideração no projeto de sistemas multi-agentes.

As metaclasses e estereótipos de nosso metamodelo foram criadas para serem utilizadas no diagrama de casos de uso de maneira a permitir que este possa identificar papéis de agentes reativos e cognitivos, bem como identificar as percepções e ações necessárias aos ditos papéis de agentes, além de seus objetivos, os planos para atingí-los e as condições para que estes sejam disparados em um nível de análise de requisitos. O metamodelo em questão foi descrito nas cinco publicações citadas anteriormente.

1.4.1.3 Fase C

Nesta fase procuramos demonstrar que a solução proposta funciona, para isto já modelamos os requisitos de três sistemas multi-agentes aplicando o metamodelo proposto, a saber, os sistemas MCOE, AME-A e APA descritos em detalhes neste trabalho. Estes estudos de caso foram publicados, conforme já citado, nos anais dos eventos SEKE 2010, IFIP AI 2010, QUATIC 2010, SEKE 2011 e MASTA@EPIA 2011.

1.4.1.4 Fase D

Aqui visamos demonstrar as conexões teóricas e a contribuição de pesquisa do conceito da solução, além de examinar o escopo de aplicabilidade da solução.

Atualmente já adaptamos o metamodelo proposto aos princípios de projeto sugeridos por Vicari (2007), onde criamos estereótipos que permitam representar papéis de agente cognitivos do tipo UAM, PS e SMI, além de permitir a modelagem dos requisitos de cada papel de agente dentro das camadas da arquitetura de agentes sugerida e apresentamos a aplicação do metamodelo dentro da arquitetura proposta.

Além disso, já realizamos um mapeamento de modelos construídos utilizando o metamodelo proposto para as linguagens MAS-ML e AML, com isso pretendemos demonstrar as compatibilidades conceituais entre estas linguagens e o metamodelo proposto e também que este metamodelo pode complementar e ser complementado por estas linguagens.

Após isso, produzimos uma proposta de validação do metamodelo, comparando modelos produzidos por meio do diagrama de casos de uso original da UML com modelos produzidos por meio do metamodelo proposto. Também produzimos uma validação por justificativa dos conceitos representados pelas metaclasses de nosso metamodelo, tentando demonstrar a importância e validade da criação das mesmas.

Finalmente, procuraremos nessa fase demonstrar a aplicabilidade da solução, discutindo os estudos de caso produzidos por meio dela e suas compatibilidades com outras linguagens aplicadas ao projeto de sistemas multi-agentes, além das inovações acrescidas ao diagrama de casos da UML por meio do metamodelo proposto.

1.5 Cronograma

Atividade	Semestre/Ano	Situação
Cumprimento dos créditos exigidos pelo programa de doutorado (Sistemas Tutores Inteligentes, Trabalho Individual I)	2/2007	Concluída
Pesquisa sobre Metodologias AOSE	2/2007	Concluída
Cumprimento dos créditos exigidos pelo programa de doutorado (Sistemas Multiagentes, Engenharia de Software, Trabalho Individual II)	1/2008	Concluída
Proficiência em Língua Estrangeira	1/2008	Concluída
Pesquisa sobre Metodologias AOSE	1/2008	Concluída
Exame de Qualificação	2/2008	Concluída
Pesquisa sobre Aplicações da UML e da AUML na Modelagem de Sistemas Multi-Agentes e suas Limitações	2/2008	Concluída
Publicação: “Applying AUML and UML 2 in the Multi-agent Systems Project”. In: FP-UML 2009 – Fifth International Workshop on Foundations and Practices of UML In conjunction with the ER - 2009 - 28th International Conference on Conceptual Modeling, 2009, Gramado. Brasil. Advances in Conceptual Modeling – Challenging Perspectives. LNCS (Lecture Notes in Computer Science). Berlin/Heidelberg : Springer-Verlag, 2009. v. 5833. ISSN 0302-9743, p. 106-115.	1/2009	Concluída
Participação no Seminário de Andamento	1/2009	Concluída
Proposta de Tese	1/2009	Concluída
Pesquisa sobre Linguagens de Modelagem Derivadas da UML Aplicadas ao Projeto de Sistemas Multi-Agentes	1/2009	Concluída
Proposta de Tese	2/2009	Concluída
Pesquisa sobre Linguagens de Modelagem Derivadas da UML Aplicadas ao Projeto de	2/2009	Concluída

Sistemas Multi-Agentes		
Pesquisa sobre Linguagens de Modelagem Derivadas da UML Aplicadas ao Projeto de Sistemas Multi-Agentes	1/2010	Concluída
Proposta de Tese	1/2010	Concluída
Desenvolvimento da Tese	1/2010	Concluída
Desenvolvimento de um Metamodelo UML para a Modelagem de Requisitos em Projetos de Sistemas Multi-Agentes	1/2010	Concluída
Modelagem dos Requisitos de Sistemas Multi-Agentes Aplicando o Metamodelo Proposto (Sistemas MCOE e AME-A)	1/2010	Concluída
Publicação: "A UML Profile Oriented to the Requirements Collecting and Analyzing for the Multi-Agent Systems Project". Proceedings of SEKE – 2010 – Twenty-Second International Conference on Software Engineering & Knowledge Engineering. Redwood City, California, EUA. Skookie, Illinois: Knowledge Systems Institute Graduate School, 2010, Volume 1, ISBN 9781891706264, p. 28-33.	1/2010	Concluída
Publicação: "A UML Profile Oriented to the Requirements Modeling in Intelligent Tutoring Systems Projects". Proceedings of The Third IFIP International Conference on Artificial Intelligence in Theory and Practice – IFIP AI, Brisbane, Australia.	1/2010	Concluída
Publicação: "Applying a UML Profile in the Requirements Modeling to Multi-Agents Systems". Proceedings of 7th International Conference on the Quality of Information and Communications Technology – Quality in ICT Requirements Engineering – Quatic 2010, Porto, Portugal.	1/2010	Concluída
Adequação do Metamodelo aos Princípios de Projeto Propostos por Vicari (2007)	2/2010	Concluída
Proposta de Tese	2/2010	Concluída
Desenvolvimento da Tese	2/2010	Concluída
Mapeamento de Modelos Utilizando o Metamodelo Proposto para as Linguagens MAS-ML e AML	2/2010	Concluída
Validação do Metamodelo Proposto	1/2011	Concluída
Modelagem dos Requisitos do Sistema APA	2/2011	Concluída
Publicação/Submissão dos Resultados Finais	1/2012	Concluída
Conclusão da Tese	1/2012	Concluída
Defesa da Tese	1/2012	

2 REVISÃO DE LITERATURA

2.1 UML, Metamodelos e Perfis

De acordo com OMG (2007a), a UML (Unified Modeling Language – Linguagem de Modelagem Unificada) é uma linguagem visual para especificar, construir e documentar os artefatos de sistemas. Ela é uma linguagem de modelagem de propósito geral que pode ser aplicada a todos os domínios de aplicação.

A UML tornou-se, nos últimos anos, a linguagem-padrão de modelagem adotada internacionalmente tanto nos meios acadêmicos como pela indústria de engenharia de software, porém a UML é uma linguagem de modelagem totalmente independente, não estando vinculada a nenhum processo de desenvolvimento específico e menos ainda a qualquer linguagem de programação.

De acordo com a OMG (2007a), a especificação UML é definida usando uma abordagem de metamodelagem. Quando se está metamodelando, estabelece-se uma distinção entre metamodelos e modelos. Um modelo tipicamente contém elementos que são instanciados a partir um meta-modelo. O papel típico de um metamodelo é definir a semântica para a forma de modelar elementos dentro de um modelo sendo instanciado.

Um modelo captura uma visão de um sistema físico. Um modelo é uma abstração do sistema com um certo propósito, como descrever aspectos estruturais ou comportamentais do software. Este propósito determina o que deve ser incluído no modelo e o que é irrelevante. Assim o modelo descreve completamente aqueles aspectos do sistema físico que são relevantes ao propósito do modelo, no nível apropriado de detalhe.

Um metamodelo define uma linguagem para expressar modelos. O papel de um metamodelo é o de definir a semântica para modelar elementos dentro de um modelo sendo instanciado, dessa forma um modelo é uma instância de um metamodelo.

Um perfil permite a adaptação da UML de forma a adequá-la a certos domínios, métodos e/ou tecnologias. Um Perfil é um tipo de Pacote que estende um metamodelo de referência. A construção de extensão primária é o Stereotype (estereótipo), que é definido como parte dos Perfis. Um perfil introduz diversas restrições sobre a metamodelagem ordinária por meio do uso das metaclasses definidas neste pacote.

Um estereótipo é um tipo limitado de metaclasse que não pode ser utilizada diretamente, mas deve sempre ser usada em conjunção com uma das metaclasses que ele estende. Cada estereótipo pode estender uma ou mais classes através de extensões como parte de um perfil. Similarmente, uma classe pode ser estendida por um ou mais estereótipos.

Dentro do contexto de modelo, um estereótipo permite atribuir novas características a um elemento UML. Existem estereótipos de texto, que não modificam o desenho padrão do elemento e estereótipos gráficos, que modificam o desenho padrão.

Dentro do contexto de metamodelos e perfis, um estereótipo define como uma metaclasse pode ser estendida, atribuindo-lhe novas características e/ou restrições.

No momento em que uma nova metaclasse é derivada a partir de uma metaclasse anterior e utiliza o estereótipo “stereotype”, esta nova metaclasse permite modelar instâncias da metaclasse original contendo um estereótipo com o nome da nova metaclasse. Ou seja, a partir deste momento pode-se modelar elementos estereotipados em um modelo com características que os diferenciem de alguma forma dos elementos originais.

2.2 Agent Unified Modeling Language (AUML)

Uma vez que agentes de software possuem particularidades específicas em relação aos softwares mais tradicionais, como por exemplo, autonomia, habilidade social, reatividade e pro-atividade (JENNINGS, 1995), algumas tentativas foram feitas para adaptar a UML a estas características, o que acarretou no surgimento da AUML - Agent UML, cuja principal documentação pode ser vista em (ODELL, 2004), (HUGET, 2003a) e (HUGET, 2003b), além de algumas aplicações em (BERGAMASCHI, 2003), (BEER, 2003), (HUGET, 2004) e (CERNUZZI, 2004). A AUML tem sido ou foi utilizada total ou parcialmente (ou foi proposta sua utilização) em alguns métodos AOSE como no caso de Gaia (CERNUZZI, 2004a), (CERNUZZI, 2004b), Prometheus (PADGHAM, 2002) (CERVENKA, 2003) ou Roadmap (JUAN, 2002).

A linguagem AUML, porém, suporta atualmente apenas fracas noções de agência, representando os agentes como classes e objetos, utilizando Diagramas de Máquina de Estados para modelar seu comportamento e Diagramas de Interação estendidos para modelar suas comunicações e seus atos comunicativos (VICCARI, 2007), não suportando abstrações cognitivas ou abstrações sociais. Caire (2002) comenta que embora esta notação seja útil, ela não possui o conceito de agente em seu núcleo, afirmando que especificar um comportamento de objeto em termos de protocolos de interação não torna isto um agente.

Uma das principais contribuições da AUML é a definição do metamodelo de superestrutura para o Diagrama de Classes (ODELL, 2004), conforme pode ser observado na figura 2.2.1. Cervenka (2004), identificou algumas deficiências com relação a este metamodelo, apresentando algumas sugestões para contorná-las, sendo este o último documento oficial da AUML (AUML, 2007), sobre este tópico específico.

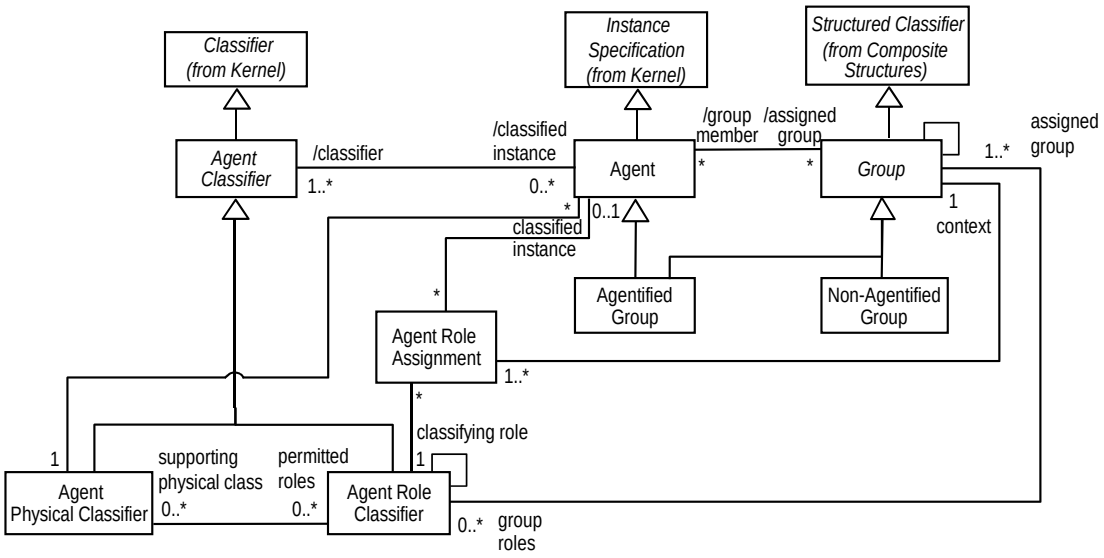


Figura 2.2.1: Metamodelo de Superestrutura para o Diagrama de Classes da AUML (ODELL, 2004).

Existe também uma proposta de definição do Diagrama de Classes para a AUML em (HUGET, 2003a), que se encontra em uma fase incipiente e parece estar muito incompleto, onde muitos dos tópicos a serem tratados contêm apenas os títulos, sem nenhum texto sobre os mesmos.

Finalmente, há um documento maior sobre os Diagramas de Interação em (HUGET, 2003b), que procura estender os diagramas da UML para apoiar atos comunicativos (communicative acts) e modelar a comunicação entre os agentes. Em seu formato original, este documento também propunha alguma notação para representação de escolhas alternativas e paralelismo. No entanto, como pode ser observado em (HUGET, 2004) e no próprio (HUGET, 2003b), cujo documento foi atualizado, esta simbologia foi deixada de lado em favor de uma alternativa proposta pela própria UML 2, como pode ser visto em (OMG, 2007b), onde recursos tais como o uso de fragmentos combinados do tipo “par” (de parallel) e “alt” (de alternatives) foram empregados no diagrama de sequência.

Percebendo a superioridade da notação proposta na nova versão da UML, Huget, (2004) propôs a adaptação desta nova notação para a AUML, adicionando algumas pequenas inovações para os diagramas de interação, como a notação lifeline para a representação de agentes (como objetos). Bauer (2005) apresenta uma comparação entre o diagrama de seqüência original da AUML (esquerda) e as novas melhorias deste diagrama na UML versão 2.0 (direita), como podemos observar na figura 2.2.2: É importante destacar, contudo, que os Diagramas de Interação da AUML estão entre os mais utilizados pelos métodos AOSE que se baseiam na AUML.

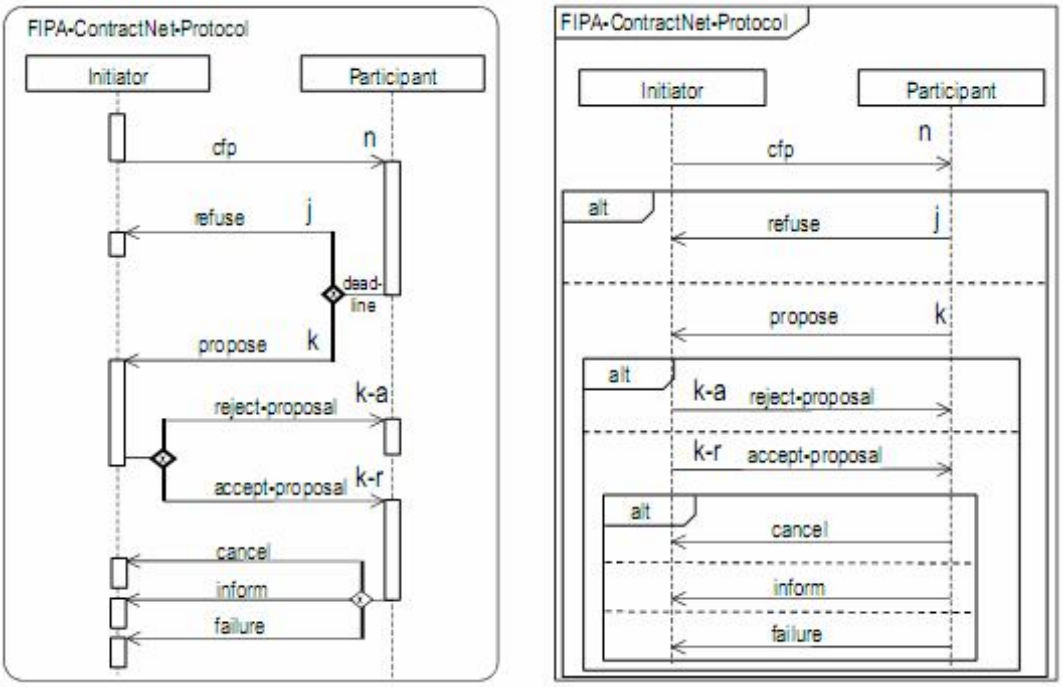


Figura 2.2.2: Comparação entre o Diagrama de Seqüência da AUML Original (esquerda) e as Novas Melhorias deste Diagrama Incorporadas a partir da UML em sua Versão 2.0 (direita) (ODELL, 2004).

Peres (2005) destaca a existência de pouquíssima documentação a respeito da AUML e ainda menor número de atualizações, afirmando também que os artigos encontrados sobre a linguagem repetem sempre os mesmos exemplos encontrados na documentação original. Peres (2005) salienta ainda que as definições de diagramas não são muito exatas em estabelecer até onde é seguido ou quebrado o paradigma UML em relação às particularidades dos Sistemas Multiagentes.

Além da pouca quantidade de exemplos de aplicação da AUML, não encontramos exemplos de aplicação do diagrama de casos de uso para a representação de requisitos de sistemas multi-agentes. Não foi feita nenhuma adaptação do referido diagrama para que este possa ser aplicado neste tipo de função, imaginamos que, como a AUML é baseada na UML, o diagrama de casos de uso seria aplicado em sua forma tradicional, porém nesse caso, este diagrama não poderia ser utilizado para modelar agentes (exceto agentes externos ao sistema), nem suas ações, percepções, objetivos ou planos.

Finalmente, a AUML está atualmente inativa, como pode ser visto em (AUML, 2007), onde uma nota informa que isto ocorre entre outros motivos devido ao lançamento da UML 2.1, contendo muitas das características relacionadas a agentes que a AUML requeria.

2.3 AORML – Agent-Object-Relationship Modeling Language – Linguagem de Modelagem Agente-Objeto-Relacionamento

Segundo Wagner (2003a) e Wagner (2003b), a linguagem AORML pretende suportar o projeto de alto nível de sistemas de informação orientados a agente. De acordo com Wagner (2003a), a AORML baseia-se sobre o metamodelo AOR – Agent-Object-Relationship, cujo propósito é fornecer uma metodologia genérica para a análise e projeto de sistemas de informação baseados em agentes.

Wagner (2003b) afirma que um modelo AOR consiste de um modelo AOR externo, correspondente a um modelo de análise de domínio, e um modelo AOR interno, correspondente ao modelo de projeto, como mostrado na figura seguinte, onde o modelo AOR geral é representado como um pacote UML e os modelos externo e interno como sub-pacotes contidos no pacote do modelo AOR geral.

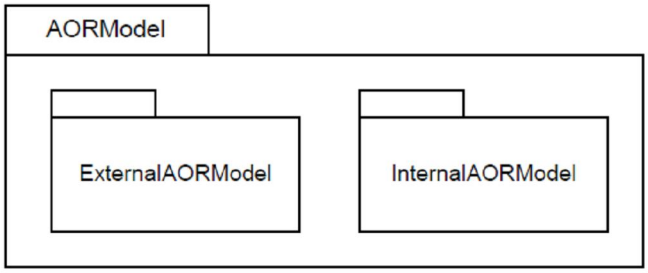


Figura 2.3.1: Modelos AOR externo e interno (WAGNER, 2003b).

Ainda segundo Wagner (2003b), um modelo AOR externo apresenta uma visão externa de um protótipo dos agentes e seus relacionamentos no domínio do problema sob consideração. Taveter (2005) complementa, afirmando que adota-se a perspectiva de um observador externo que visualiza os agentes prototípicos e suas interações. Wagner (2003b) declara que modelos AOR externos focam-se em um agente ou grupo de agentes para o qual se deseja desenvolver um modelo de estado e comportamento.

De acordo com Wagner (2003b), um modelo AOR interno descreve o mundo como ele pode ser representado no estado mental do agente em foco. Se o agente em foco é uma organização, o modelo AOR interno representa sua visão do mundo, e pode ser usado para projetar seu sistema de informação.

Taveter (2005) afirma que em um modelo AOR interno adota-se a visão interna em primeira pessoa de um agente particular a ser modelado. Um modelo de domínio de negócio corresponde a um modelo externo, enquanto um modelo de projeto para um sistema de informação agentificado específico corresponde a um modelo interno que pode ser derivado a partir do modelo externo.

2.3.1 Objetos e Agentes

De acordo com Taveter (2005), AORML baseia-se em uma distinção ontológica entre entidades passivas (objetos) e ativas (agentes). O termo agente inclui elementos artificiais, ou seja, agentes de software e robôs; naturais, ou seja, criaturas vivas como seres humanos; e sociais/institucionais, como grupos e organizações.

Na AORML, uma entidade é um agente, um evento, uma ação, uma reivindicação, um compromisso ou um objeto. Só agentes podem comunicar, perceber, agir, se comprometer e satisfazer reivindicações. Objetos são entidades passivas, sem essas

capacidades. Segundo Wagner (2003b), os tipos de objeto são representados como retângulos planos, como classes em diagramas de classes UML padrão.

Taveter (2005) afirma que, além de agentes humanos e artificiais, AORML também inclui o conceito de agentes institucionais, que são compostos por outros agentes que agem em seu interesse. Wagner (2003a) complementa afirmando que exemplos de tipos de agente humano são Pessoa, Empregado, Aluno, Enfermeira ou Paciente. Exemplos de agentes institucionais são organizações como bancos, hospitais ou unidades organizacionais. Um agente institucional consiste de um certo número de agentes internos que percebem eventos e realizam ações em nome da instituição, pela interpretação de certos papéis. Já como exemplo de agentes artificiais, Wagner (2003a) cita agentes de software, sistemas embarcados ou robôs.

Ainda de acordo com Wagner (2003a), em diagramas AOR, uma classe de agente é representada como um retângulo com os cantos arredondados. Porém, segundo Taveter (2005), esse formato foi abandonado e adotou-se o símbolo de pacote da UML para a representação de agentes.

2.3.2 Compromissos e Reivindicações

De acordo com Wagner (2003a). compromissos e reivindicações são componentes fundamentais de processos de interação social. Compromissos são tratados como relacionamentos binários entre dois agentes.

Segundo Wagner (2003b), certos atos de comunicação tipicamente provocam o surgimento de compromissos para realizar certas ações ou para garantir que certas condições sejam satisfeitas, por exemplo, a confirmação de um pedido de compra implica a criação de um compromisso para entregar os itens pedidos na data de entrega especificada. Um compromisso de a1 (o devedor) para com a2 (o credor) para realizar a ação x é espelhado como uma reivindicação de a2 em relação à a1 para criar o evento de ação x.

Wagner (2003a) afirma que há dois tipos de compromissos: compromissos para realizar uma ação e compromissos para ver se alguma condição se aplica. O primeiro tipo é chamado compromisso a-realizar e o segundo de compromisso a-verificar.

Wagner (2003b) declara que compromissos/reivindicações são conceitualmente acoplados com o tipo de evento de aceitação que eles se referem. Isto é graficamente representado por um retângulo-seta com uma linha pontilhada no topo do retângulo do evento de ação que ele se refere, como demonstrado na figura 2.4.2.

2.3.3 Modelos AOR externos

Segundo Wagner (2003a), em modelos AOR externos foi adotada a visão de um observador externo que observa os agentes prototípicos e suas interações no domínio do problema sob consideração. Um modelo AOR externo foca-se em um agente ou grupo de agentes para os quais se deseja desenvolver um modelo de estado e comportamento. Nesta visão-observador-externo, o mundo, ou seja, o domínio de aplicação, consiste de vários tipos de agentes, eventos de ação comunicativa e não-comunicativa, eventos de não-ação, compromissos/reivindicações entre dois tipos de agente, objetos ordinários, relacionamentos designados (tais como mensagens e ações) e associações ordinárias.

Taveter (2005) sumariza afirmando que um diagrama AOR externo, descreve os tipos e instâncias de agente (se aplicável) de um domínio de problema, junto com seus

tipos e instâncias de agentes internos, suas crenças sobre objetos e agentes externos e os relacionamentos entre agentes e/ou objetos.

De acordo com Wagner (2003b), na visão de um observador externo, ações são também eventos e compromissos são também reivindicações. Portanto, um modelo AOR externo contém, além do agente e tipos de objeto de interesse, classes de evento de ação e classes de compromisso/reivindicação que são necessárias para descrever a interação entre o agente em questão e os outros tipos de agentes.

Tipos de objeto, em um modelo AOR externo, pertencem a um ou mais agentes e definem contâineres para crenças. Se uma classe de objeto pertence exclusivamente a um agente ou classe de agente, o retângulo correspondente é desenhado dentro do retângulo do agente. Se uma classe de objeto representa crenças que são compartilhadas entre dois ou mais agentes, o retângulo de classe de objeto é desenhado fora dos retângulos do agente.

A figura a seguir mostra os elementos centrais de modelagem externa da AORML em sua versão original.

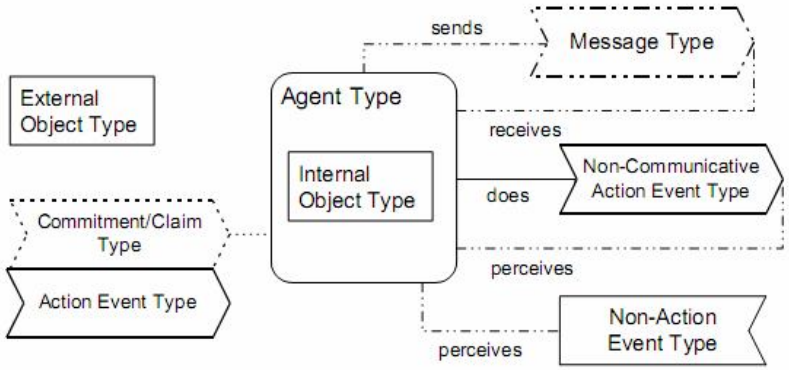


Figura 2.3.2: Elementos Centrais de Modelagem Externa AOR – Modelo Original (WAGNER, 2003b).

No entanto, segundo Taveter (2005), de maneira a restaurar a compatibilidade com a notação de diagrama da UML 2.0, o formato gráfico para um tipo de agente foi alterado de um retângulo com as bordas arredondadas, que hoje é usado como símbolo para ação/atividade na UML, para o formato mostrado na figura seguinte. Foi utilizado o estereótipo <<AgentType>> neste símbolo, que representa um pacote ou sub-sistema na UML 2, de forma que um tipo agente seja visto como uma classe UML.

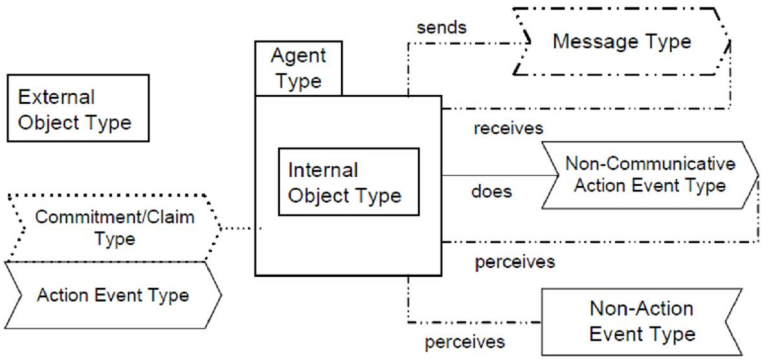


Figura 2.3.3: Elementos Centrais de Modelagem Externa AOR – Modelo Atual (TAVETER, 2005)

Os elementos centrais mostrados nesta figura são utilizados para a modelagem da estrutura de estado mental dos diagramas AOR externos. Esta modelagem pode ser definida como um perfil UML, ou seja, é uma extensão conservadora da modelagem de classes UML.

De acordo com Wagner (2003b), um modelo AOR externo pode incluir um ou mais dos seguintes diagramas:

- Diagramas de agente, descrevendo os tipos de agente do domínio, certos tipos de objetos relevantes e os relacionamentos entre eles.
- Diagramas de Quadro de Interação, descrevendo as classes de evento de ação e as classes de compromisso/reivindicação que determinam as possíveis interações entre dois tipos de agente.
- Diagramas de Seqüência de Interação, descrevendo instâncias prototípicas de processos de interação.
- Diagramas de Padrão de Interação, enfocando padrões de interação gerais expressados por meio de um conjunto de regras de reação definindo um tipo de processo de interação.

2.3.4 Ações

Segundo Wagner (2003b), na perspectiva de um observador externo, todas as ações de agentes são ao mesmo tempo também eventos que podem ser percebidos por outros agentes. Há muitos eventos que são criados por ações de agentes correspondentes. No entanto, há também eventos que não são criados por ações. Assim, existe uma distinção entre eventos de ação e eventos de não-ação.

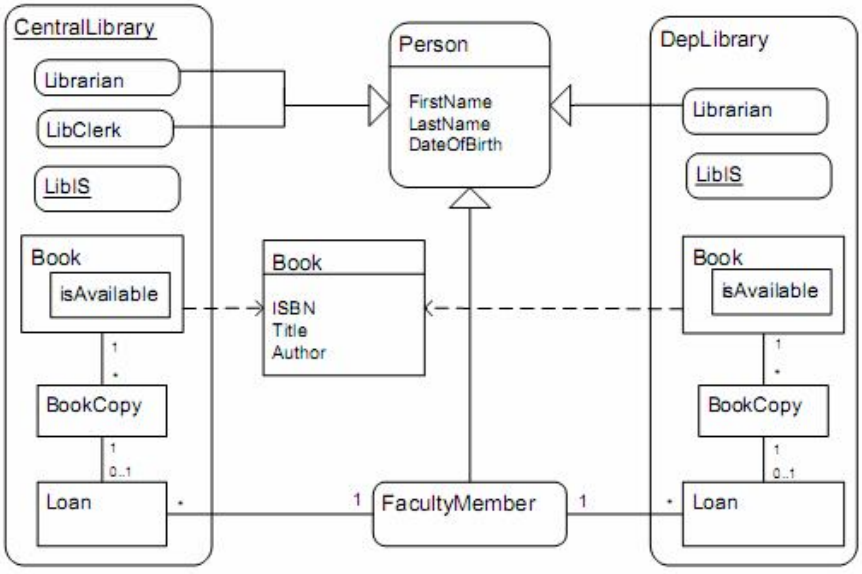


Figura 2.3.4: Um Diagrama de Agente AOR para o Domínio de Bibliotecas de Universidade (WAGNER, 2003b).

Na figura 2.3.4, a biblioteca central e as bibliotecas de departamento são agentes institucionais (utilizando o formato original), tendo bibliotecários como agentes humanos internos (Librarian) e um sistema de informação de biblioteca como um agente interno artificial (LibIS). FacultyMember é uma classe de agente neste domínio. Tipos de objetos importantes são Book, BookCopy e Loan, ou seja, as bibliotecas têm crenças

sobre seus livros, suas cópias de livros e seus empréstimos. A subclasse Book.isAvailable é formada por todos os livros que satisfazem o predicado de status isAvailable, ou seja, para aqueles em que há no mínimo uma cópia de livro disponível.

Em um diagrama AOR Externo, um tipo de evento de ação é graficamente representado por uma seta especial onde um lado é uma seta de entrada ligada ao agente que realiza este tipo de ação e o outro lado é uma seta de saída ligada ao agente que percebe este tipo de evento. Retângulos de evento de ação comunicativa tem uma linha pontilhada. No caso de um evento de não-ação, o retângulo de evento correspondente não tem uma seta de saída. A figura seguinte apresenta um exemplo destes símbolos.

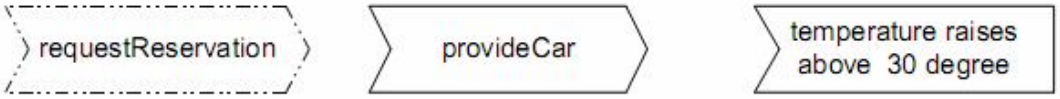


Figura 2.3.5: Um Evento de Ação Comunicativa, um Evento de Ação Não Comunicativa e um Evento de Não-Ação (WAGNER, 2003b).

2.3.5 Diagramas de Quadro de Interação

Wagner (2003b) afirma que no modelo AOR externo, há quatro tipos de relacionamentos, representados por estereótipos de associação, entre agentes e eventos de ação: envios e recebimentos que são associações que relacionam um agente com eventos de ação comunicativa e ações e percepções que são associações que relacionam um agente com eventos de ação não-comunicativa. Há ainda dois tipos de associações entre agentes e compromissos/reivindicações chamados hasCommitment e hasClaim. Estes estereótipos de associação são vistos com tipos de conectores particulares, um exemplo disso é apresentado na próxima figura.

Um diagrama de quadro de interação é um modelo AOR externo que descreve as interações possíveis entre dois tipos de agentes. Ele consiste de eventos de ação comunicativa, eventos de ação não-comunicativa, compromissos/reivindicações acoplados com os tipos correspondentes de eventos de ação e eventos de não-ação.

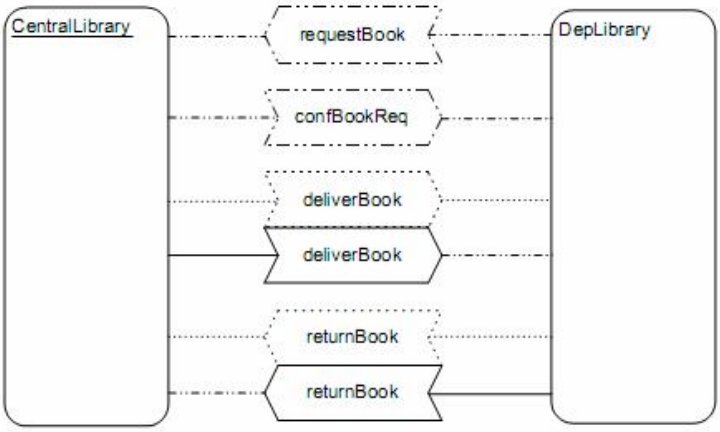


Figura 2.3.6: O Quadro de Interação Entre a Biblioteca Central e as Bibliotecas de Departamento (WAGNER, 2003b).

Nesta figura, uma biblioteca de departamento pode requisitar um livro da biblioteca central (os agentes CentralLibrary e DepLibrary aqui representados utilizam o formato original); quando a requisição tiver sido confirmada pela biblioteca central haverá um compromisso de entregar o livro requisitado, representado pelo retângulo seta de linha tracejada deliverBook. Normalmente tal comprometimento leva a uma ação

correspondente, representada pelo retângulo seta de linha sólida deliverBook; depois que o livro tiver sido entregue, há um compromisso de retorná-lo no devido tempo.

2.3.6 Diagramas de Seqüência de Interação

Segundo Wagner (2003a), Um processo de interação é uma seqüência de eventos de ação e de eventos de não-ação, realizados e percebidos por agentes, e que seguem um conjunto de regras ou protocolo que especificam o tipo do processo de interação. Um diagrama de seqüência de interação descreve uma instância de um processo de interação. Agentes podem interagir com seu ambiente inanimado ou eles podem interagir entre si. A figura a seguir exemplifica a primeira situação, onde um agente humano (utilizando o formato original) percebe que escureceu e age ligando a luz.

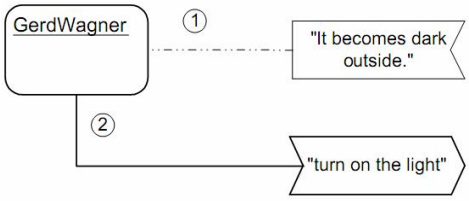


Figura 2.3.7: Um processo de Interação Não-social Envolvendo um Agente e seu Ambiente Inanimado (WAGNER, 2003a).

Um processo de interação social é um conjunto temporário de eventos de ação e de eventos de não-ação coerente e ordenado, envolvendo no mínimo um evento de ação comunicativa, realizado e percebido por agentes e seguindo um conjunto de regras ou protocolo que é governado por normas e que especifica o tipo do processo de interação. Um exemplo de um processo de interação social é mostrado na figura seguinte. Normas sociais implicam, por exemplo, que depois de ter confirmado uma requisição de livro, a biblioteca está comprometida a entregar o livro requisitado.

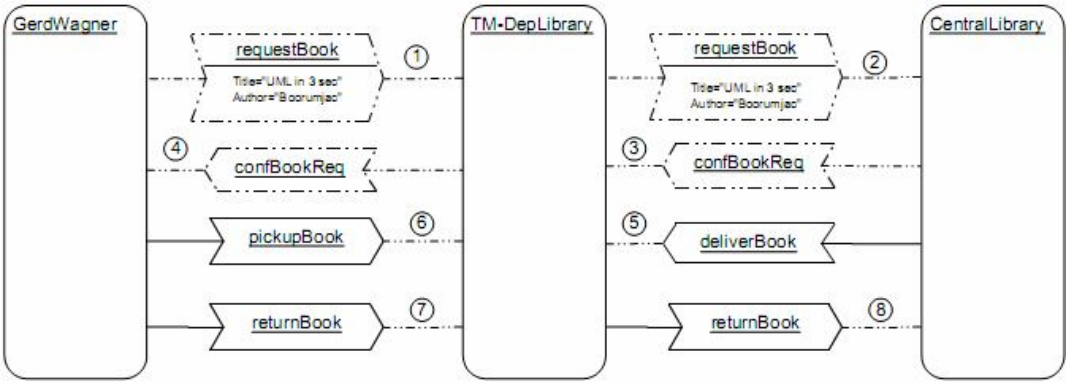


Figura 2.3.8: Um Processo de Interação Social Envolvendo Três Agentes – Formato Original (WAGNER, 2003a).

Nesta figura a requisição do agente GerdWagner de empréstimo do livro “UML in 3 sec” (1) é enviada pelo agente TM-DepLibrary para o agente CentralLibrary (biblioteca central) (2). O pedido é confirmado pela biblioteca central para a biblioteca de departamento (TM-DepLibrary) (3) e esta confirmação é repassada para GerdWagner (4). A biblioteca central entrega o livro para a biblioteca de departamento (5). GerdWagner pega o livro (6) e mais tarde o devolve a biblioteca de departamento (7), e esta por sua vez a devolve à biblioteca central (8).

2.3.7 Modelos AOR Internos

Em um modelo AOR interno, Wagner (2003a), adota a visão interna de um agente a ser modelado. Nesta visão, o domínio da aplicação consiste de outros agentes; ações; compromissos para outros agentes realizarem certas ações; eventos; reivindicações em relação a outros agentes, de que certos eventos de ação irão acontecer; objetos ordinários; relacionamentos designados e associações ordinárias. Estes tipos de meta-entidade de modelagem AOR interna estão apresentados na figura a seguir.

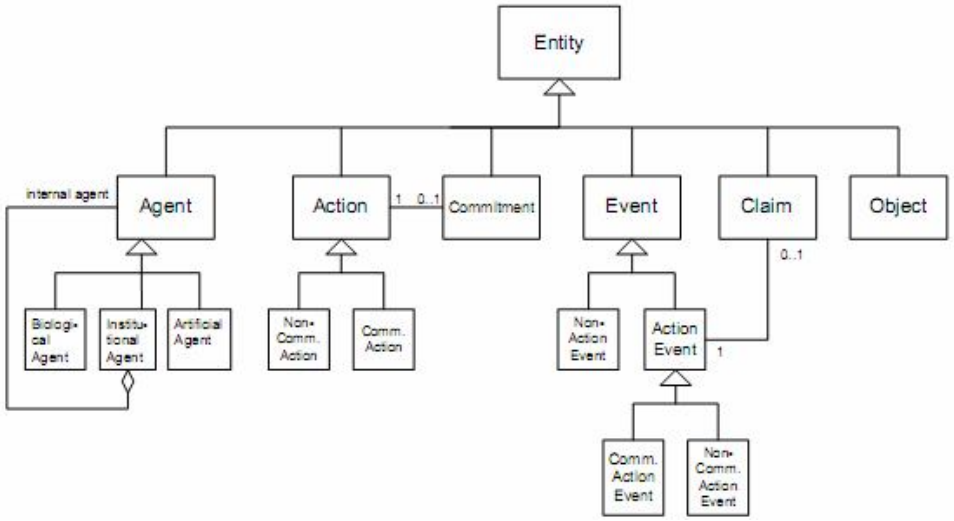


Figura 2.3.9: Tipos de Meta-Entidade da Modelagem AOR Interna (WAGNER, 2003a).

Um modelo AOR interno descreve o ambiente como ele pode ser representado no estado mental do agente em questão. Se o agente é uma organização, o modelo AOR interno representa sua visão do mundo, e pode ser usado para projetar seu sistema de informação. Um modelo AOR interno pode conter um ou mais dos seguintes diagramas:

- Diagramas de Quadro de Reação, descrevendo outros agentes ou tipos de agentes, e os tipos de ações e eventos, bem como os tipos de compromissos e reivindicações que determinam as possíveis interações com eles.
- Diagramas de Seqüência de Reação, descrevendo instâncias prototípicas de processos de interação na perspectiva interna.
- Diagramas de Padrão de Reação, enfocando os padrões de reação do agente sob consideração expressado por meio de regras de reação.

2.3.8 Agentes Externos e Internos

De acordo com Wagner (2003a), em um diagrama AOR interno para um agente institucional, os agentes internos e/ou tipos de agente do agente institucional a ser modelado podem aparecer no nível superior, como o tipo de agente interno SalesDepartment na figura a seguir, onde eles se distinguem dos agentes externos devido às linhas dos retângulos serem tracejadas. Os agentes internos de outros agentes internos, como SalesPerson dentro de SalesDepartment são também agentes internos.

Na figura a seguir, do ponto de vista de uma empresa, Customer é um tipo de agente externo, enquanto SalesDepartment é um agente interno e SalesPerson é um tipo de agente interno de SalesDepartment. Cada SalesPerson tem um certo número de clientes associados. O formato utilizado para representação de agentes é o original.

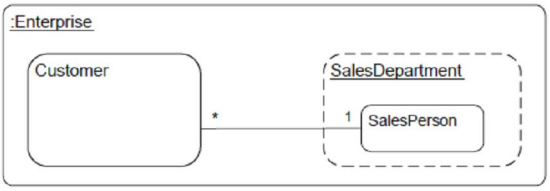


Figura 2.3.10: Tipo de Agente Externo, Agente Interno e Tipo de Agente Interno – Formato Original (WAGNER, 2003a).

Na perspectiva de um agente institucional, somente as ações de agentes internos realizadas em nome da instituição contam como ações, enquanto as ações de agentes externos são consideradas eventos.

2.3.9 Quadros de Reação

Wagner (2003a) afirma que em um modelo AOR interno, o comportamento reativo do agente em consideração com respeito a outro agente "A" é descrito por meio de eventos de comunicação, ou messagens de chegada, criadas pelos atos de comunicação de instâncias de A; atos de comunicação, ou mensagens de saída, dirigidas a instâncias de A; reivindicações em relação a instâncias de A; eventos não-comunicativos criados pelas ações de instâncias de A; compromissos com instâncias de A; ações não-comunicativas que podem ser realizadas de forma a preencher compromissos correspondentes a instâncias de A, ou em resposta a eventos dentro do quadro de interação.

A figura seguinte mostra um Diagrama de Quadro de Reação para uma biblioteca central descrevendo o comportamento reativo com respeito ao tipo de agente DepLibrary (formato original). Há o tipo de mensagem de entrada requestBook, o tipo de mensagem de saída confBookReq, o tipo de compromisso deliverBook acoplado com o tipo de ação correspondente e o tipo de reivindicação returnBook acoplado com o tipo de evento correspondente.

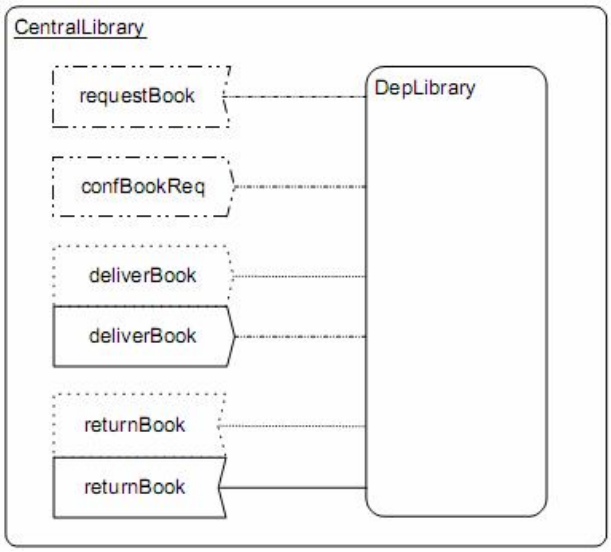


Figura 2.3.11: Diagrama de Quadro de Reação (WAGNER, 2003a).

Em um Diagrama AOR Interno, não é possível ver padrões de interação completos envolvendo dois ou mais agentes, uma vez que o seu escopo é um agente único. O comportamento do agente sob consideração é modelado pela identificação de seus padrões de reação e por sua expressão na forma de regras de reação, como

mostrado na figura seguinte que ilustra um Diagrama de Padrão de Reação para a biblioteca central descrevendo a etapa do processo onde a biblioteca central tem de reagir a uma requisição de livro da biblioteca de departamento (representadas como agentes no formato original).

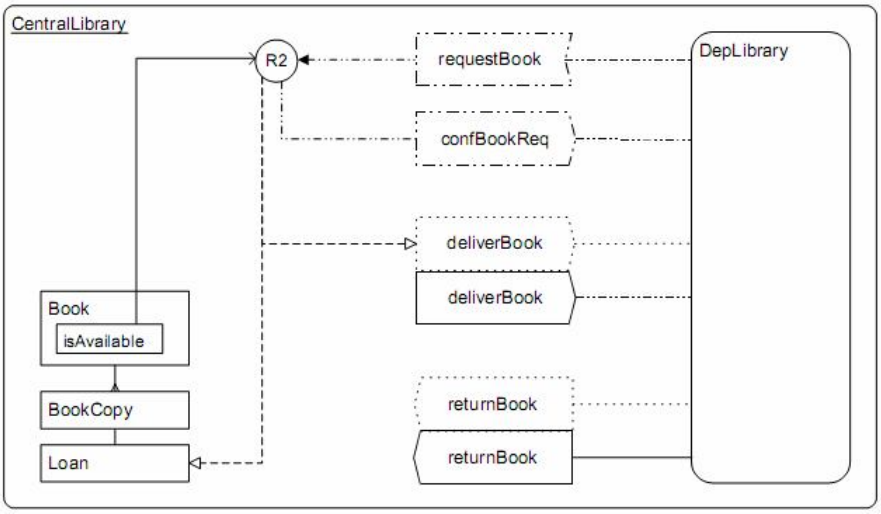


Figura 2.3.12: Diagrama de Padrão de Reação (WAGNER, 2003a).

2.3.10 Modelagem dos Direitos e Deveres dos Agentes Internos

Segundo Wagner (2003a), direitos e deveres correspondem a relacionamentos designados entre tipos de agentes internos e tipos de compromisso, reivindicação, evento e ação. Eles são visualizados com a ajuda de conectores especiais tendo um círculo ao lado do retângulo do tipo de agente interno, estando este vazio quando for um direito ou preenchido quando for um dever, como demonstra a figura a seguir.

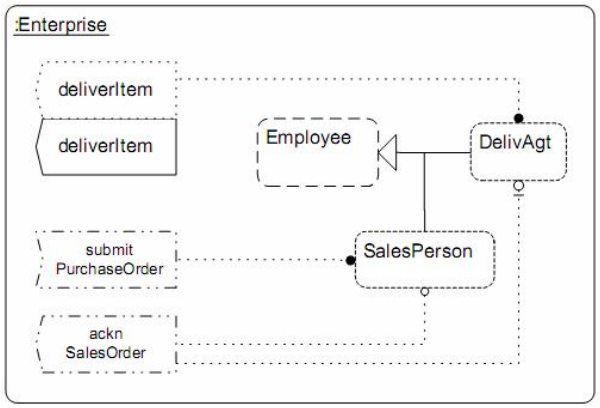


Figura 2.3.13: Direitos e Deveres (WAGNER, 2003a).

Nesta figura, o agente SalesPerson tem o dever de reagir a mensagens submitPurchaseOrder e tem o direito de enviar mensagens acknSalesOrder. Já o agente DelivAgt tem o dever de executar o compromisso deliverItem, o que implica no direito de realizar ações deliverItem. Além disso DelivAgt não tem o direito de enviar mensagens acknSalesOrder. Os agentes aqui representados utilizam o formato original.

2.3.11 Regras de Reação

Segundo Taveter (2005), os elementos de modelagem de comportamento mais importantes da AORML são as regras de reação, que são usadas para expressar padrões

de interação. A figura a seguir mostra um diagrama de padrão de interação que especifica a regra de reação R1 para o compartamento de um Vendedor (Seller) em resposta a percepção de um evento de ação comunicativa do tipo requisição de ordem de compra (PurchaseOrder). Tanto a condição de estado mental como o efeito mental da regra de reação são apresentados como expressões na Linguagem de Restrição de Objeto (OCL) da UML 2 (OMG 2007b) ligadas às setas correspondentes.

A pré-condição para fornecer os produtos requisitados ao Comprador (Buyer) é a disponibilidade do Item de Produto (ProductItem) cuja verificação no banco de dados interno do Vendedor é especificada com a ajuda do predicado de status correspondente isAvailable e a expressão OCL productID = globalProductIdentifier. A pós-condição que representa o efeito mental é expressada como inventory = inventory@pre - requestedQuantity. A pós-condição afeta a representação do Item de Produto correspondente no banco de dados interno do Vendedor pelo decréscimo de seu atributo inventory pelo valor do atributo quantidadeRequisitada (requestedQuantity) da mensagem recebida.

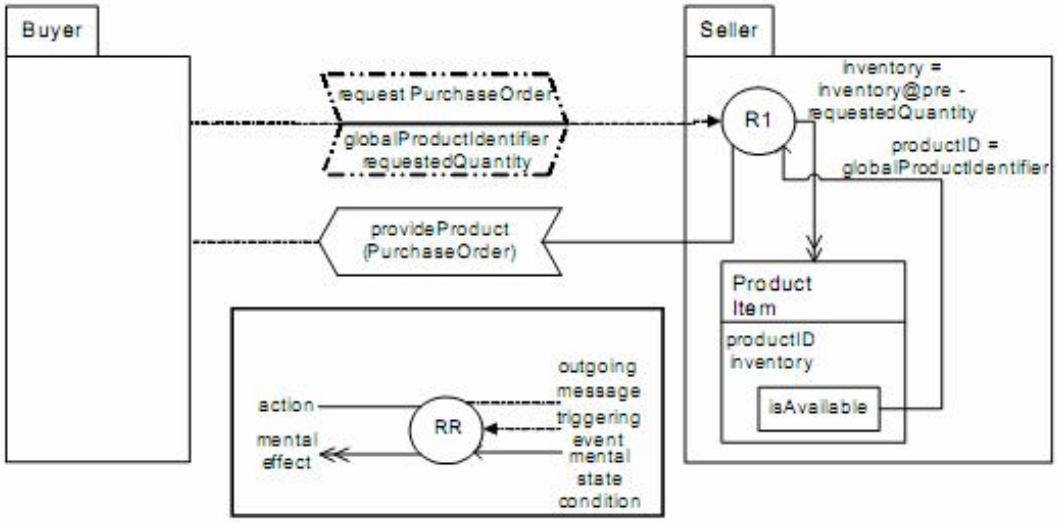


Figura 2.3.14: Um Padrão de Interação (TAVETER, 2005).

Quando recebe uma Ordem de Compra, o Vendedor fornece o produto ao Comprador de acordo com a Ordem de Compra se os produtos estiverem disponíveis.

2.3.12 Considerações Finais

A linguagem AORML concentra-se fortemente na questão da modelagem de interação entre agentes (modelos externos) e como estes reagem a estímulos (modelos internos). Em alguns exemplos, consideramos os diagramas de interação apresentados um pouco difíceis de interpretar em termos da ordem em que as interações ocorrem, posto que às vezes não é muito simples determinar quais eventos ocorrem primeiro. Acreditamos também que muitas das informações mostradas pelos diagramas de interação, senão todas, poderiam ser igualmente representadas por diagramas de seqüência UML estendidos para tal fim.

Embora a AORML modele os compromissos e reivindicações, bem como direitos e deveres de um agente, além de representar crenças como objetos, ela não modela questões como desejos e como estes podem se tornar intenções, ou seja, com exceção da representação de crenças como objetos, não enfoca a questão de agentes baseados no modelo BDI.

Há representação das percepções e ações dos agentes, mas estas são modeladas no detalhamento de interações e não são identificadas como requisitos funcionais do software. Além disso, há pouco suporte para a representação de agentes cognitivos, os agentes modelados apresentam basicamente características reativas e os diagramas utilizados nos modelos internos são essencialmente baseados na reação dos agentes.

Com relação à modelagem de requisitos, embora o modelo externo da AORML apresente uma visão mais geral e externa do sistema, onde se identificam, de certa forma, ações e percepções dos agentes, os autores não o utilizam especificamente para a identificação dos requisitos do software. Wagner (2003a) propõe explicitamente, porém não demonstra, o uso do diagrama de atividade para a representação de requisitos, embora nós acreditemos que este diagrama seja mais aplicável na modelagem de algoritmos do que na de requisitos ou, no caso de sistemas multi-agentes, mais especificamente no detalhamento de planos.

Acreditamos que a modelagem de requisitos seja melhor desempenhada por meio do diagrama de casos de uso, porém Wagner (2003a) não utiliza este diagrama. Em Taveter (2005) aplica-se o diagrama de casos de uso padrão da UML, porém não foi feita verdadeiramente nenhuma adaptação deste diagrama, atores normais são utilizados para representar agentes externos ao sistema, casos de usos normais representam as funcionalidades oferecidas pelo sistema (que também é considerado um agente), embora os autores utilizem essas funcionalidades para também representar as interações entre os agentes, caracterizados por um agente humano externo (representado pelo ator) e um agente representado pelo sistema propriamente dito, modelado por meio da fronteira do sistema, como pode-se observar na figura a seguir.

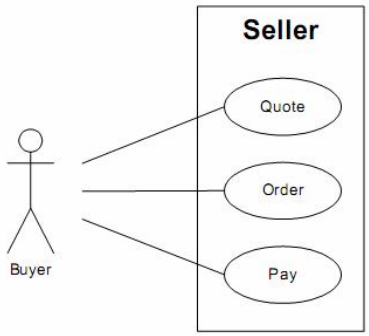


Figura 2.3.15: Diagrama de Casos de Uso Representando as Interações entre Agente por meio de Casos de Uso e Agentes por meio de um Ator e pela Fronteira do Sistema (TAVETER, 2005).

Nesta figura, Buyer é um agente humano externo representado por um ator normal e Seller representa o software propriamente dito, mas também um agente, enquanto as funcionalidades do sistema são representadas por casos de uso normais que também representam as interações entre os dois agentes.

Todavia, não foi feita nenhuma adaptação do diagrama de casos de uso para modelar agentes ou seus papéis, e nem os objetivos, percepções, ações ou planos a eles associados.

2.4 Agent Modeling Language (AML)

Cervenka (2005a) e Trencansky (2005) declaram que a linguagem de modelagem de agente (AML) é uma linguagem visual semi-formal para especificar, modelar e documentar sistemas que incorporam características da teoria de sistemas multi-agentes. Ela é especificada como uma extensão da UML 2.0 de acordo com os principais frameworks de modelagem OMG. O principal objetivo da AML é fornecer uma linguagem de modelagem adequada para o desenvolvimento de software comercial baseado em tecnologias multi-agente.

Cervenka (2005a) afirma que a AML é definida em dois níveis distintos, Metamodelo e Notação AML e Perfis AML, já Trencansky (2005) destaca que a sintaxe abstrata, semântica e a notação da linguagem estão definidas no nível Metamodelo e Notação AML. O Metamodelo AML é novamente estruturado em dois pacotes principais: Kernel AML e Extensão UML para AML, conforme demonstra a figura a seguir:

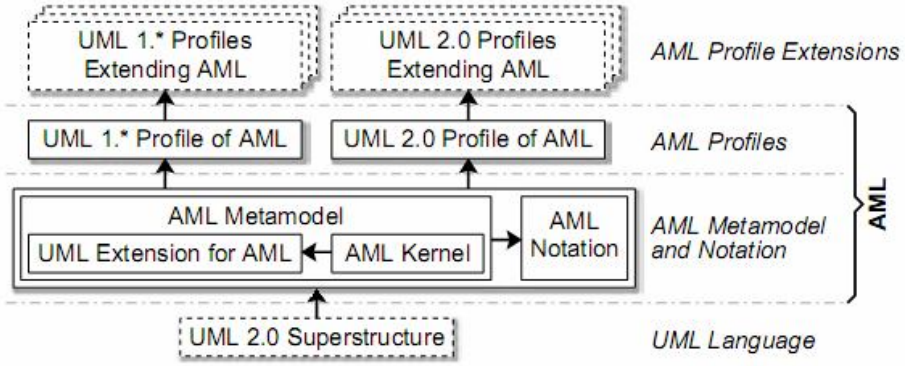


Figura 2.4.1: Níveis de Definição AML (CERVENKA, 2005a).

Segundo Cervenka (2005a), o nível de linguagem UML contém a Superestrutura UML 2.0 definindo a sintaxe abstrata, semântica e notação da UML. A AML usa este nível como a fundação sobre a qual define as construções de modelagem específicas para sistemas multi-agentes. O nível de Metamodelo e Notação AML define a sintaxe abstrata, semântica e notação AML, estruturada em dois pacotes: Kernel AML e Extensões UML para AML.

Trencansky (2005) afirma que o Kernel AML é uma extensão conservadora da UML 2, abrangendo a especificação de todos os elementos de modelagem AML. Ele é logicamente estruturado em diversos pacotes, cada um dos quais contém a especificação de elementos de modelagem dedicados para a modelagem de aspectos específicos de sistemas multiagentes. Cervenka (2005a) destaca que o pacote Kernel AML é o centro da AML onde elementos específicos de modelagem da AML são definidos.

Cervenka (2005a) afirma ainda que a extensão UML para o pacote AML adiciona meta-propriedades e restrições estruturais para os elementos UML padrão. Ela é uma extensão não-conservativa da UML e, portanto, é uma parte opcional da linguagem, porém, segundo Trencansky (2005), as extensões são simples e podem ser facilmente implementadas em ferramentas CASE baseadas em UML.

Trencansky (2005) afirma que a AML define diversos conceitos de metamodelagem abstrata auxiliar chamados semi-entidades e seus tipos. Tipos de semi-

entidade são classes UML especializadas utilizadas para especificar conjuntos coerentes de características, logicamente agrupadas de acordo com os aspectos particulares de sistemas multiagentes.

2.4.1 Semi-Entidades da AML

Trencansky (2005) classifica as semi-entidades AML em:

- Semi-entidades comportamentais, que representam elementos que podem possuir capacidades, perceber e/ou afetar seu ambiente, fornecer e usar serviços e podem ser (de)compostos em fragmentos de comportamento.
- Semi-entidades sociais, que representam elementos que podem formar sociedades, podem participar de relacionamentos sociais e podem possuir propriedades sociais.
- Semi-entidades mentais, que representam elementos que podem ser caracterizados em termos de suas atitudes mentais, como por exemplo, em que elas crêem e quais são seus objetivos.

2.4.2 Entidades Fundamentais da AML

De acordo com Trencansky (2005), o tipo Agente (Agent) é utilizado para especificar agentes, que são definidos em AML como entidades auto-contidas que são capazes de interações, observações (percepções) e comportamento autônomo dentro de seu ambiente.

O tipo Recurso (Resource) é usado para modelar recursos dentro do sistema, que são entidades físicas ou de informação com as quais o principal interesse é sua disponibilidade em termos de sua quantidade, direitos de acesso, condições de uso/consumo, etc.

O tipo Ambiente (Environment) é usado para modelar um ambiente interno do sistema, ou seja, os arredores lógicos ou físicos de entidades que fornecem condições sob as quais as entidades existem e funcionam.

Todos estes tipos de entidade são classes UML especializadas, dessa forma, podem utilizar todas as características definidas para classes UML. As instâncias dos tipos de entidade podem ser modeladas por meio de especificações de instâncias UML classificadas de acordo com os tipos correspondentes.

Cervenka (2005a) complementa, afirmando que, em AML, todos os tipos de entidade são chamados coletivamente entidades comportamentais. Elas podem possuir capacidades, ser decompostas em fragmentos de comportamento, fornecer e utilizar serviços e possuir perceptores e causadores de efeito. Adicionalmente, tipos de agente e ambiente são entidades autônomas que podem ser caracterizadas em termos de suas atitudes mentais, ou seja, podem possuir características, participar em vários tipos de relacionamentos, ser estruturadas internamente em partes, possuir comportamentos, etc.

Na figura a seguir, Trencansky (2005) apresenta uma definição de uma classe abstrata chamada 3DObject que representa objetos espaciais contidos dentro de um espaço que possuem uma forma e posição. Um tipo ambiente abstrato chamado 3DSpace, representa um espaço tridimensional, que, por ser um 3DObject especial pode conter outros objetos espaciais. O ambiente 3DSpace fornece um serviço chamado Motion, para os objetos contidos dentro dele. Três 3DObjetcs concretos, um tipo agente, Person, um tipo recurso, Ball e uma classe Goal são definidos como 3DObjects especializados. Finalmente, um tipo ambiente concreto chamado Pitch, representando

um lance de jogo de futebol contendo dois objetivos e uma bola foi especializado a partir do ambiente 3DSpace.

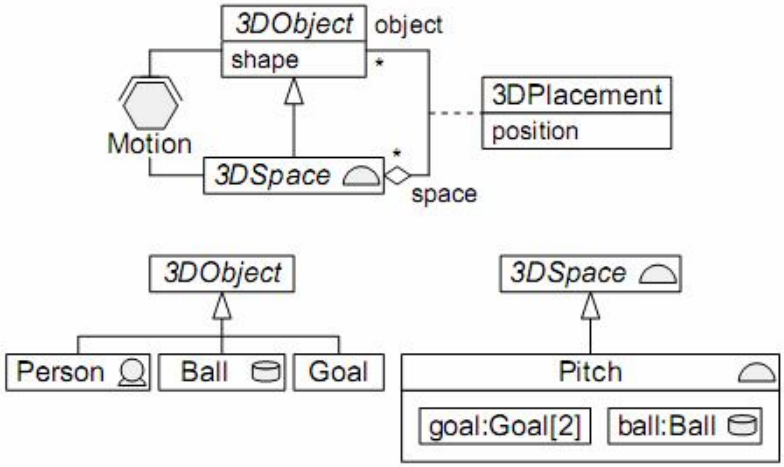


Figura 2.4.2: Exemplo de Entidades, seus Relacionamentos e Fornecimento e Uso de Serviços (TRENCANSKY, 2005).

2.4.3 Aspectos Sociais

De acordo com Cervenka (2005a), a AML define diversos elementos para modelar os aspectos sociais de sistemas multi-agentes, incluindo características estruturais de entidades socializadas e certos aspectos de seu comportamento social. Segundo Trencansky (2005), a partir da perspectiva social os seguintes aspectos são comumente considerados em modelos de sistemas multi-agentes:

- Estruturas sociais, que se preocupam com a identificação de sociedades que possam evoluir dentro do sistema, com a especificação de suas propriedades, estruturas e a identificação dos papéis que abrangem, entidades individuais que podem participar de tais sociedades, quais os papéis que possam exercer, quais sejam seus relacionamentos mútuos, etc.
- Comportamento social, cobrindo fenômenos como dinâmica social (habilidade de uma sociedade de reagir a eventos internos e externos), normas, interações sociais (como indivíduos e/ou sociedades interagem com outros de forma a trocar informação, coordenar suas atividades, etc) e atividades sociais de entidades individuais e sociedades (como elas mudam suas atitudes, papéis que executam, relacionamentos sociais).
- Atitudes sociais, enfocando as tendências individuais e/ou comuns (motivações, necessidades, desejos, intenções, objetivos, crenças, comprometimentos, etc) para qualquer coisa de valor social

Trencansky (2005) afirma que, de forma a acomodar necessidades especiais para modelar aspectos sociais, a AML utiliza conceitos de unidades de organização, relacionamentos sociais, papéis de entidade e propriedades de papéis.

Dentro deste contexto, Cervenka (2005b) utiliza o conceito de entidades sociais, definindo-as como entidades possuidoras de habilidades sociais, ou seja, entidades capazes de participar em sociedades e estabelecer relacionamentos sociais, manifestar comportamento social e ter atitudes sociais. Cervenka (2005b) acrescenta que as entidades agentes e unidades de organização são exemplos de entidades sociais.

2.4.4 Unidades de Organização

Trencansky (2005) afirma que o tipo unidade de organização é um tipo de ambiente especializado que herda características de tipos de semi-entidades comportamentais, sociais e mentais. Eles são usados para especificar o tipo de sociedades que evoluem dentro do sistema tanto a partir de perspectivas externas como internas.

De uma perspectiva externa, unidades de organização representam entidades autônomas coerentes, que podem ser caracterizadas em termos de suas atitudes mentais e sociais, podem ter comportamentos, participar em diferentes tipos de relacionamentos sociais, podem observar e interagir com seu ambiente, oferecer e usar serviços, executar papéis, etc. A partir de uma perspectiva interna, unidades de organização são tipos de ambiente que especificam os arranjos sociais de entidades em termos de estruturas, interações, papéis, restrições, normas, etc.

Cervenka (2005b) acrescenta que unidades de organização são usadas para modelar sociedades, como grupos, organizações e instituições. Elas não são consideradas estáticas, por que suas propriedades, estrutura, comportamento, atitudes sociais, papéis por elas englobados, entidades participantes e suas características podem mudar ao longo do tempo.

A figura a seguir à esquerda (a) mostra um exemplo de um diagrama de classes descrevendo uma estrutura de organização genérica de um projeto de desenvolvimento de software. As equipes de projeto (ProjectBoard, TechnicalTeam, Analysis Team, Implementation Team e Testing Team) são modelados por meio de tipos de unidade de organização, seus relacionamentos sociais por meio de associações sociais e um papel de projeto (ProjectManager) por meio do tipo de papel de entidade. Ao observarmos a figura em questão percebemos que a unidade de organização ProjectBoard supervisiona o papel de entidade ProjectManager (Gerente de Projeto) e este por sua vez gerencia as equipes técnicas que cooperam entre si,

A estrutura interna do tipo de unidade de organização SoftwareDevelopmentProject está modelada na figura à direita (b). Ela define as partes que representam papéis de entidade abrangidos (pm) e unidades de organização de baixo nível (pb, ana, imp, e tst). Os conectores entre eles declaram instâncias das associações sociais especificadas ao nível de classe.

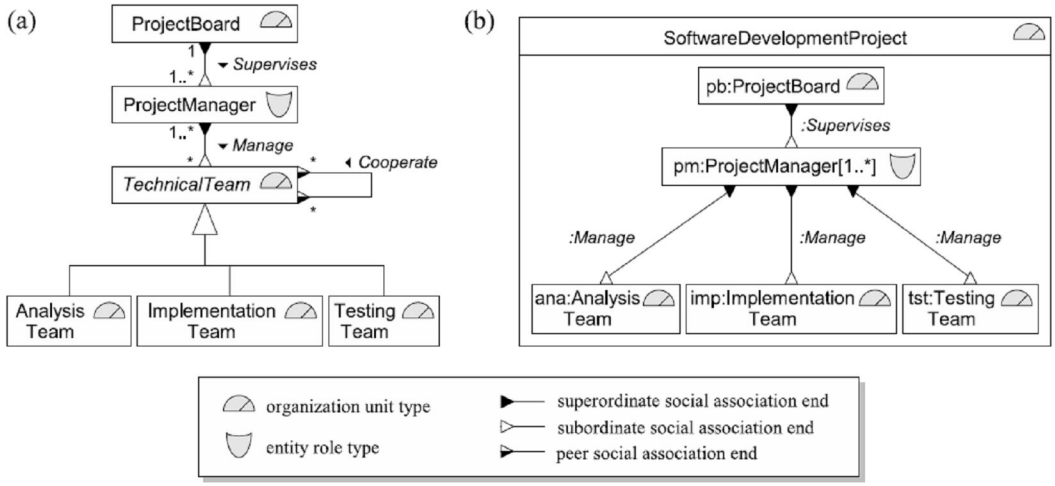


Figura 2.4.3 – Exemplo de Modelo de Estrutura de Organização em Nível de Classe (CERVENKA, 2005b).

2.4.5 Relacionamentos Sociais

De acordo com Trencansky (2005), relacionamento social é um tipo particular de conexão entre entidades sociais relacionadas. Para modelar tais relacionamentos, a AML define um tipo especial de propriedade UML, chamada propriedade social.

Cervenka (2005b) acrescenta que a propriedade social permite especificar a categoria de papel social do relacionamento. A AML suporta duas categorias pré-definidas de relacionamentos sociais, par-a-par (peer-to-peer) e superior-a-subordinado, e as categorias de papel social análogas: par (peer), superior e subordinado. O conjunto de categorias de papel social suportado pode ser estendido se necessário.

A figura 2.4.3 (a) mostra associações superior-a-subordinado entre ProjectBoard (superior – mostrado como um triângulo preenchido colocado no fim da associação) e ProjectManager (subordinado – mostrado como um triângulo vazio colocado no fim da associação) e ProjectManager (superior) e TechnicalTeam (subordinado). Equipes técnicas particulares estão relacionadas pelos relacionamentos par-a-par que são representadas pela associação social Cooperate ligada ao tipo de unidade organizacional TechnicalTeam (a propriedade social par é mostrada como um triângulo preenchido pela metade colocado no final da associação. Os conectores representando instâncias das associações sociais definidas anteriormente estão descritos na figura 2.4.3(b).

2.4.6 Papéis e Propriedades de Papéis

Trencansky (2005) declara que papéis são usados para definir um repertório comportamental normativo de entidades e assim fornecer os blocos de construção básicos de sociedades multiagente. Para modelar papéis, a AML fornece o tipo de papel de entidade, um tipo de semi-entidade comportamental, social e mental especializado.

Tipos de papel de entidade são usados para modelar abstrações de conjuntos coerentes de características, capacidades, comportamentos, observações (percepções), relacionamentos, participação em interações e serviços oferecidos ou requeridos por entidades que participam de um contexto particular. Uma instância de um tipo de papel de entidade é chamada papel de entidade e existe somente enquanto alguma entidade comportamental a executa.

Cervenka (2005a) complementa afirmando que uma entidade de papel representa a execução de comportamentos, utilização de características e/ou participação em interações como definido pelo tipo de papel de entidade particular.

A possibilidade de execução de um papel de entidade por uma entidade comportamental é modelada pela propriedade papel (role) e a associação play. Mecanismos são oferecidos também para modelar mudanças dinâmicas de papéis, raciocínio sobre papéis interpretados, expressão de multiplicidades e restrições sobre papéis de entidade interpretados e navegação pelas características e capacidades estruturais de tipos de papéis de entidade a partir das entidades que os executam.

Segundo Trencansky (2005), para modelar a habilidade de uma entidade para executar um tipo de papel de entidade, AML fornece propriedades de papel. Uma propriedade de papel é uma propriedade UML especializada, usada para especificar que uma instância de seu possuidor (uma entidade comportamental) pode executar um ou mais papéis de um tipo de papel de entidade particular. Cervenka (2005b) acrescenta que uma instância de um possuidor de propriedade de papel é chamado interpretador (player) de papel de entidade e uma instância do tipo de propriedade de papel representa o papel de entidade interpretado.

Trencansky (2005) afirma que uma entidade pode a cada vez executar diversos papéis de entidade. Estes papéis de entidade podem ser do mesmo tipo ou de tipo diferentes. A multiplicidade definida para uma propriedade de papel restringe o número de papéis de entidade de determinado tipo que a entidade particular pode executar concorrentemente.

Para permitir manipulação explícita de papéis de entidade em atividades e máquina de estado UML, a AML define um conjunto de ações para criação e disposição de papéis de entidade, como ação de papel e ação de disposição de papel.

A figura 2.4.4(a) contém o diagrama descrevendo um agente do tipo Person que pode interpretar papéis de entidade do tipo Player, Captain, Coach e Referee. A possibilidade de executar papéis de entidade de um tipo particular é modelada por associações play. A figura 2.4.4(b) descreve uma unidade de organização Soccer Match, que compreende três árbitros (do tipo de papel de entidade Referee) e dois times (do tipo de unidade de organização SoccerTeam). O próprio SoccerTeam consiste de um a três técnicos e onze a quinze jogadores dos quais um é capitão. Os jogadores são pares entre si (como demonstra o conector cooperate) e subordinados aos técnicos (o conector manage) e ao capitão (o conector lead). Os dois times são subordinados aos árbitros (o conector control).

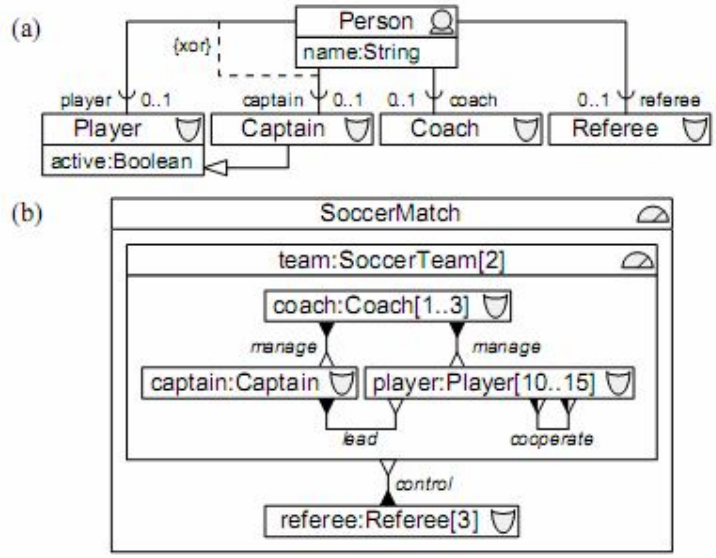


Figura 2.4.4: Exemplo de Modelagem de Estrutura Social (TRENCANSKY, 2005).

A figura 2.4.5 mostra a instanciação dos tipos definidos anteriormente, onde o agente Lampard, do tipo Person, executa o papel de entidade jogador (player), e o agente Terry, também do tipo Person, executa o papel de entidade capitão (captain) e lidera Lampard. O agente Mourinho, executa o papel de entidade técnico (coach) e dirige os jogadores Lampard e Terry.

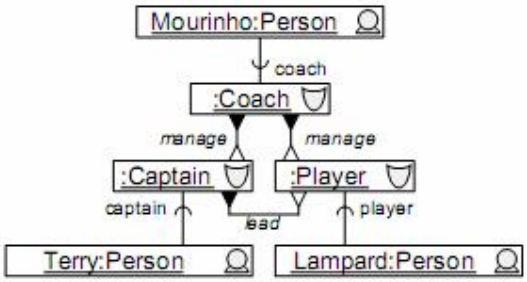


Figura 2.4.5: Exemplo de Instanciação e Execução de Papel de Entidade (TRENCANSKY, 2005).

Dentro deste contexto, Cervenka (2005b) define a existência de papéis sociais, que são abstrações de características, comportamento, atitudes, participação em interações e serviços requeridos ou fornecidos a outros papéis ou entidades sociais. Papéis sociais são modelados por tipos de papel de entidade e podem ser usados para especificar estruturas sociais, posições (conjunto de papéis executados por um agente) e estruturas requeridas, características comportamentais e de atitude de seus constituintes.

Cervenka (2005b) apresenta, na figura a seguir, outro exemplo de especificação de tipos de papel, seus atributos e associações sociais. Nesta figura também podemos perceber que o agente Person (Pessoa) pode interpretar diversos papéis por meio da associação play, como gerente de projeto (ProjectManager), Analista (Analyst), Programador (Programmer) e Testador (Tester).

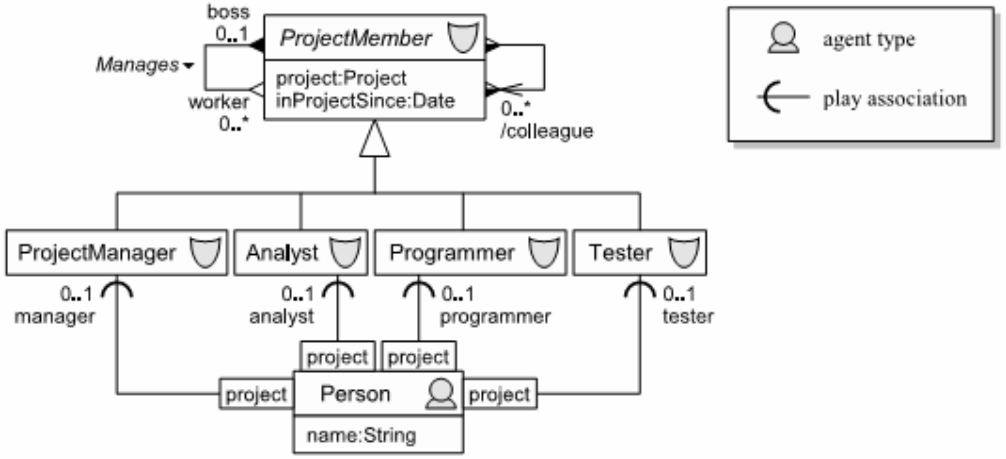


Figura 2.4.6: Tipos de papéis de entidade e associações play (CERVENKA, 2005b).

A figura seguinte mostra a instanciação dos tipos definidos anteriormente, onde o agente Alan do tipo Person, interpreta dois papéis de entidade, testador e analista, enquanto o agente John também do tipo Person interpreta o papel de gerente de projeto (pm) e é o chefe de Alan. Este exemplo também demonstra a habilidade da AML de especificar explicitamente vagas e links sociais de papéis de entidade interpretados sob certas circunstâncias.

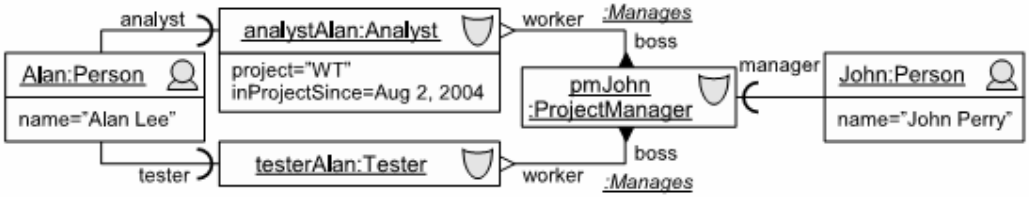


Figura 2.4.7: Exemplo de instanciação e interpretação de papel de entidade (CERVENKA, 2005b).

2.4.7 Modelagem de Interações

Segundo Trencansky (2005), para suportar a modelagem de interações em sistemas multiagentes, a AML fornece um certo número de extensões UML, que podem ser logicamente subdivididas em extensões genéricas para interações UML, extensões baseadas em atos de fala para interações UML, observações e interações de efeito e serviços.

2.4.7.1 Extensões Genéricas para Interações UML

De acordo com Trencansky (2005), as extensões genéricas para interações UML fornecem meios para modelar interações entre grupos de entidades por meio de multi-mensagem e multi-lifeline; mudança dinâmica de atributos de objetos para expressar mudanças na estrutura interna de unidades de organização, em relacionamentos sociais ou papéis de entidade interpretados, induzidos por interações; modelagem de mensagens e sinais não explicitamente associados com a invocação de métodos e recepções correspondentes; mecanismos para modificação de papéis de interação de entidades induzidas por interações; modelar as ações de envio e recepção de mensagens desacopladas em atividades.

Cervenka (2005a) afirma que, em adição a estes conceitos genéricos, AML também modela atos comunicativos baseados em interações usadas comumente em sistemas multi-agente, particularmente mensagens agentificadas especializadas (atos comunicativos), interações UML especializadas usadas para modelar atos comunicativos baseados em interações (interações comunicativas) e interações comunicativas parametrizadas usadas para modelar padrões de interações (protocolos de interação).

Trencansky (2005) apresenta os seguintes conceitos utilizados para a modelagem de interações em AML:

Multi-mensagem (multi-message) é uma mensagem UML especializada que é utilizada para modelar uma comunicação particular entre múltiplos participantes.

Multi-lifeline é um lifeline UML especializado usado para representar múltiplos participantes em interações.

Mensagem desacoplada (Decoupled message) é uma multi-mensagem especializada utilizada para modelar o envio e recepção assíncrono de uma mensagem que não apresente uma especificação do comportamento invocado por parte do receptor.

Mudança de atributo, é um fragmento de interação usado para modelar as mudanças de valores de atributo de entidades produzidas pela interação. Permite expressar adição, remoção ou modificação de valores de atributo, além dos valores de atributo adicionados pelas sub-lifelines. É utilizada na modelagem de mudança dinâmica de papéis de entidade executados por entidades comportamentais em interações.

Subconjunto (Subset) é uma dependência UML especializada entre ocorrências de evento possuídas por duas lifelines distintas, uma agindo como superconjunto e outra como subconjunto, usadas para especificar que, desde que a ocorrência de evento esteja sobre a lifeline superconjunto, algumas das instâncias que ela representa são também representadas por outra, a lifeline subconjunto.

A dependência de união (join dependency) é uma dependência UML especializada usada para especificar que um subconjunto de instâncias que tenha sido, até a ocorrência de evento de subconjunto, representada pela lifeline de subconjunto seja, depois da ocorrência de evento de subconjunto, representada pela lifeline de união.

Ação de Mensagem Desacoplada de Envio (Send decoupled message action) é uma ação de objeto de envio UML especializada usada para modelar a ação de despachar uma mensagem desacoplada. E ação de mensagem desacoplada de aceitação é uma ação de evento de aceitação UML especializada usada para modelar a recepção de uma ação de mensagem desacoplada.

Uma interação simplificada entre entidades que participam de uma substituição de jogador é descrita na figura 2.4.8. Neste exemplo, o técnico principal (Coach) decide substituir o jogador p1 pelo jogador p2. Primeiramente ele notifica o jogador p2 para ficar pronto e então pede ao árbitro principal (Referee) permissão para fazer a substituição. O árbitro principal por sua vez replica por meio de uma resposta. Se a resposta é “sim” (situação representada por um fragmento combinado do tipo opt), o processo de substituição espera até o jogo ser interrompido. Quando isso ocorre, o técnico instrui o jogador p1 a sair e p2 a entrar. O jogador p1 então deixa a partida e junta-se ao grupo de jogadores inativos e p2 junta-se ao jogo e ao grupo de jogadores ativos, isso é feito paralelamente, como demonstra o fragmento combinado do tipo par.

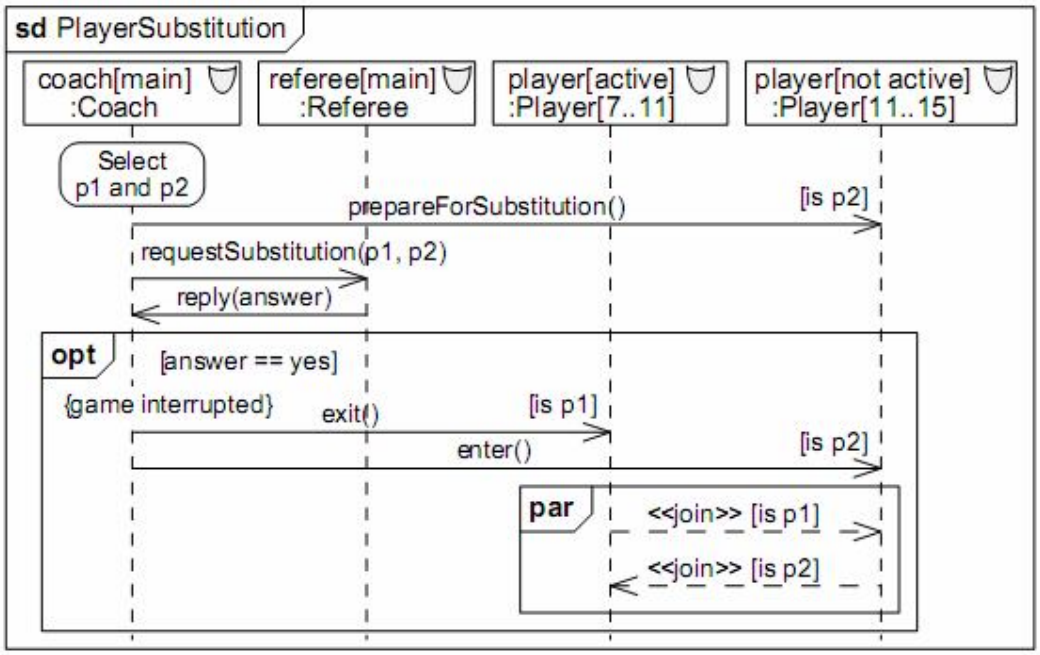


Figura 2.4.8: Exemplo de uma Interação Comunicativa (TRENCANSKY, 2005).

A figura 2.4.9 mostra um exemplo de interação comunicativa na qual os elementos de mudança de atributo são usados para modelar mudanças de papéis de entidade executados pelos agentes. O diagrama modela o cenário de uma mudança de capitão causada pela substituição do capitão original (player2).

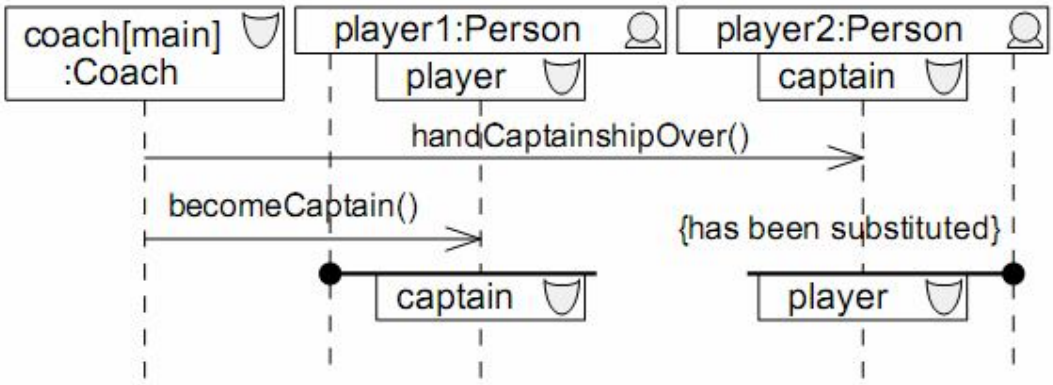


Figura 2.4.9: Exemplo de uma Interação Social com Mudança de Papel de Entidade (TRENCANSKY, 2005).

No início deste cenário o agente player2 é o capitão. Durante a substituição, o técnico principal dá ao player2 ordem para deixar de ser o capitão (handCaptainshipOver) e ao player1 para tornar-se o capitão (becomeCaptain). Depois de receber estas mensagens, o player2 pára de executar o papel de entidade capitão assumindo o papel de entidade jogador (player) e o player1 muda de jogador simples para capitão (captain).

2.4.8 Extensões Específicas de Atos de Fala para Interações UML

Segundo Trencansky (2005), estas extensões compreendem a modelagem de atos de fala (speech-acts) ou seja, mensagens de comunicação, atos de fala baseados em interações (interações comunicativas), padrões de interações (protocolos de interação) e a modelagem das ações de envio e recepção de mensagens baseadas em atos de fala em atividades.

2.4.9 Observações (Percepções) e Interações de Efeito

Na AML as percepções são chamadas observações. Cervenka (2005a) afirma que a AML define diversas construções para modelar observações (percepções), que definem a habilidade de uma entidade observar (perceber) as características de outras entidades. A AML também modela interações de efeito, que definem a habilidade de uma entidade de manipular ou modificar o estado de outras entidades.

De acordo com Trencansky (2005), observações são realizadas por meio de perceptores, que são portas UML especializadas. Tipos de perceptores são usados para especificar as observações que um possuidor de um perceptor daquele tipo pode fazer.

Atos de percepção são operações UML especializadas que podem ser possuídas por tipos perceptores e assim usadas para especificar quais percepções seus possuidores ou perceptores de dado tipo podem realizar.

A especificação de quais entidades podem observar outras é modelada por uma dependência de percepções. Para modelar aspectos comportamentais de observações, a AML fornece uma ação de percepção especializada.

Aspectos diferentes de interações de efeito são modelados analogamente por meio de causadores de efeito (effectors), tipos de causadores de efeito, atos de efeito, dependências de efeito e ações de efeito.

A figura 2.4.10 mostra um tipo de papel de entidade Jogador (Player) com dois olhos (eye) como perceptores e duas pernas (leg) como causadores de efeito. Os olhos

são usados para ver outros jogadores, a partida e a bola e para fornecer informação de localização para as partes internas de um jogador. As pernas são usadas para mudar a posição do jogador dentro da partida e para manipular a bola.

2.4.10 Modelagem de Capacidades e Comportamento

Uma capacidade é basicamente um comportamento que pode ser executado por uma entidade. De acordo com Trencansky (2005), uma capacidade é uma especificação abstrata de um comportamento que permite raciocínio sobre operações na especificação.

Um fragmento de comportamento é um tipo de semi-entidade comportamental especializada usada para modelar um fragmento de comportamento coerente e re-utilizável, além de características comportamentais e estruturais relacionadas. Isto permite a decomposição de um comportamento complexo em fragmentos executáveis mais simples e possivelmente concorrentes, bem como a modificação dinâmica de um comportamento de entidades em tempo de execução.

A figura 2.4.10(a) mostra a decomposição do comportamento do tipo de papel de entidade Player em uma estrutura de fragmento de comportamento, onde ele consegue perceber a partida e a bola por meio do perceptor olhos, que influencia o fragmento Localization, raciocinar sobre isso (fragmento PlayerReasoning) e com base neste raciocínio alterar a posição do jogador e tentar afetar o estado da bola (fragmentos Mobility e Ball Handling) por meio do causador de efeito pernas. Na parte (b) os fragmentos Mobility e Ball Handling, estão descritos em termos de suas capacidades, por exemplo o fragmento Mobility tem as capacidades de se virar (turn), caminhar (walk), correr (run) e parar (stop).

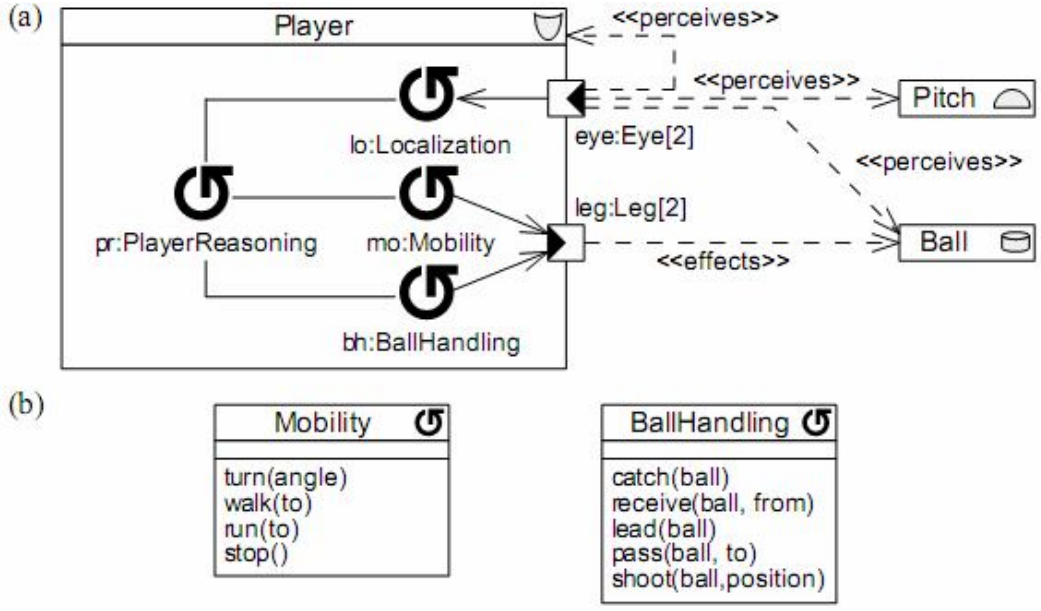


Figura 2.4.10: Exemplo de Fragmentos de Comportamento, Observações, Interações de Efeito e Capacidades (TRENCANSKY, 2005).

2.4.11 Modelagem de Comportamento Social

Segundo Cervenka (2005b), comportamento social é o comportamento de uma entidade social que influencia ou é influenciado pelo estado ou comportamento de outras entidades sociais. Comportamento social cobre dinâmica social, interações sociais e atividades sociais.

Dinâmicas sociais são modeladas como máquinas de estado. A incorporação de ações específicas da AML às máquinas de estado da UML permite a modelagem explícita de formação ou abolição de sociedades, a entrada ou retirada de uma entidade em uma sociedade e a aquisição, disposição ou mudança de um papel por uma entidade.

Já para modelar interações sociais, a AML define construções de modelagem especializadas para modelar atos de fala baseados em interações, observações e interações de efeito. Extensões genéricas para interações UML modelam interações entre grupos de entidades usando multi-mensagem e multi-lifeline, mudança dinâmica de atributos de objeto induzida por interações e modelagem de mensagens e sinais não explicitamente associados com uma invocação de métodos e recepções correspondentes. Extensões específicas de interação comunicativa compreendem a modelagem de atos de fala por meio de atos comunicativos, atos de fala baseados em interações por meio de interações comunicativas e padrões de interações por meio de protocolos de interação.

A figura a seguir mostra um exemplo da interação comunicativa na qual o elemento de mudança de atributo é usado para modelar mudança de papéis de entidade interpretados por agentes.

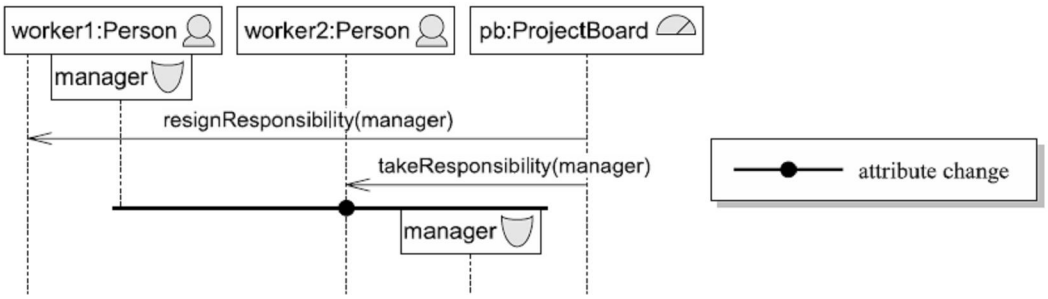


Figura 2.4.11: Exemplo de Interação Social com Mudanças de Papel de Entidade (CERVENKA, 2005b).

Este diagrama apresenta o cenário de substituição de um gerente de projeto por outra pessoa. Um agente worker1 é um gerente, como demonstra sua propriedade de papel manager. Depois de receber uma mensagem resignResponsability (renunciar responsabilidade) do conselho de projeto (project board) (pb) ele pára de executar o papel de gerente de projeto. Ao mesmo tempo, outra pessoa, worker2, assume a responsabilidade, como resultado de ter recebido a mensagem takeResponsability (assuma responsabilidade) enviada pelo conselho de projeto, e inicia a interpretar o papel do gerente de projeto, modelado pela propriedade manager de worker2.

Já com relação à modelagem de atividades, Cervenka (2005b) afirma que atividades UML podem ser usadas, porém, para permitir o desenvolvimento de modelos mais concisos e compreensivos, a AML oferece diversos conceitos e mecanismos de modelagem adicionais.

Para permitir a modelagem de modificação de características sociais, como relacionamentos sociais, papéis interpretados e atitudes sociais, elas foram modeladas como características estruturais de entidades. Isto permite fazer uso de algumas ações de manipulação UML com características estruturais, para modelar a modificação de estruturas sociais, o raciocínio sobre papéis de entidade interpretados, o acesso e raciocínio sobre atitudes sociais, executar comportamento social, etc.

Além disso, a AML define ações específicas para criar/interpretar e dispor papéis de entidade, enviar e receber mensagens de interações comunicativas sociais, perceber e

gerar efeitos em outras entidades, comprometer-se e cancelar compromissos com objetivos, etc.

A figura a seguir mostra um exemplo de um diagrama de atividade que descreve o cenário de substituição do gerente de projeto por outra pessoa. O comitê de projeto, que é representado pela unidade de organização pb, decide sobre a substituição e informa ao gerente de projeto atual, representado pelo papel de entidade pm, junto com um novo gerente de projeto potencial (agente potentialPM) sobre a decisão. O gerente de projeto atual pára de interpretar o papel de gerente de projeto (expressado pela ação de disposição de papel Resign) e o novo gerente de projeto inicia a interpretar seu novo papel de entidade (especificado pela ação de criação de papel Become a new Project manager que cria um novo papel de entidade newPM).

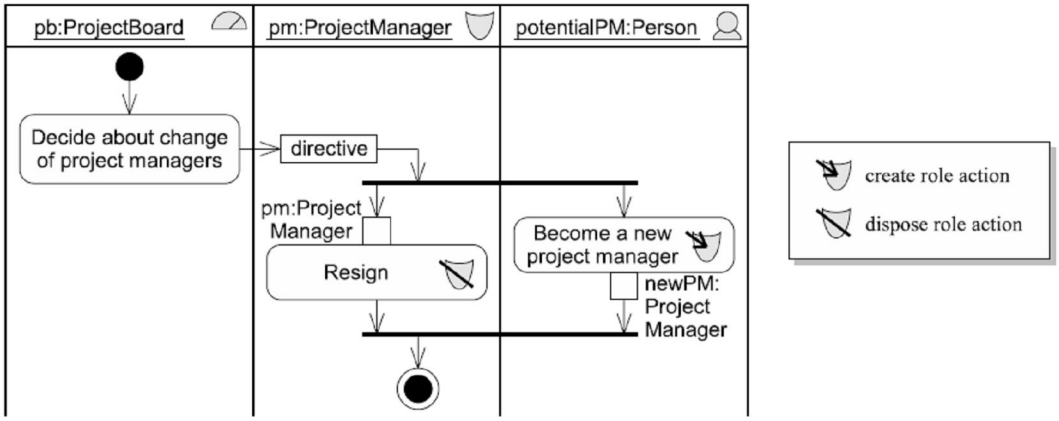


Figura 2.4.12: Exemplo de Atividade Compreendendo as Ações Sociais Induzindo Mudanças de Papéis de Entidade (CERVENKA, 2005b).

2.4.12 Modelagem de Aspectos Mentais

Segundo Cervenka (2005a), a AML fornece construções usadas para modelar crenças, objetivos, planos e relacionamentos mentais. Estas construções podem ser usadas para enriquecer a modelagem de casos de uso por meio de expressão de intencionalidade, modelagem de requisitos baseados em objetivo e decomposição de problema; ou para modelar atitudes mentais de entidades autônomas que representam seus estados de informação, de motivacão e deliberativos. Porém, com relação aos casos de uso, não há exemplos de como isto pode ser realizado.

Trencansky (2005) afirma que semi-entidades mentais podem ser caracterizadas em termos de suas atitudes mentais, ou seja, motivações, necessidades, desejos, intenções, objetivos, crenças e comprometimentos. Para permitir a modelagem disto, AML provê: objetivos, crenças, planos, relacionamentos de contribuição, propriedades e associações mentais, restrições mentais e ações de comprometimento ou cancelamento (commit/cancel) para com objetivos. Isto é detalhado a seguir:

Objetivo (Goal) é uma classe UML especializada usada para modelar objetivos, ou seja, condições ou estados do mundo com os quais a principal preocupação é sua atingibilidade ou manutenção. Objetivos podem assim ser usados para representar objetivos, necessidades, motivações e desejos.

Crença (Belief) é uma classe UML especializada para modelar um estado de coisas, proposição ou outra informação relevante para o sistema e seu modelo mental.

Plano (Plan) é uma atividade UML especializada usada para modelar planos pré-definidos, ou fragmentos de comportamento a partir dos quais os planos podem ser compostos.

Restrição mental é uma restrição UML especializada usada para especificar propriedades de crenças, objetivos e planos possuídos que podem ser usadas dentro de processos de raciocínio de semi-entidades mentais. Tipos suportados de restrições mentais são pré e pós-condições, condições de compromisso (commit), condições de cancelamento e invariantes.

Contribuição é um relacionamento UML especializado usado para modelar relacionamentos lógicos entre objetivos, crenças e planos e suas restrições mentais.

Finalmente, também é possível modelar compromissos e des-compromissos de objetivos por meio de ações especializadas em diagramas de atividades.

A figura seguinte mostra um exemplo de um determinado momento do modelo mental de um time de futebol e seus jogadores. O time de futebol, representado pelo tipo de unidade de organização SoccerTeam, tem o objetivo de ganhar uma partida (modelada pelo objetivo WinMatch). Quando a partida terminar e o time tiver feito mais gols do que recebeu o objetivo terá sido atingido, como é expressado pela contribuição da crença {match.isOver and team.scoredGoals > team.concededGoals} para a pós-condição do objetivo WinMatch. Os jogadores, representados pelo tipo de papel de entidade Player, podem ter o objetivo de pontuar (ScoreGoal) o que é factível se comprometer, quando eles tiverem a chance de pontuar. Isto é expressado pela contribuição da crença ScoringChance para a pré-condição do objetivo ScoreGoal.

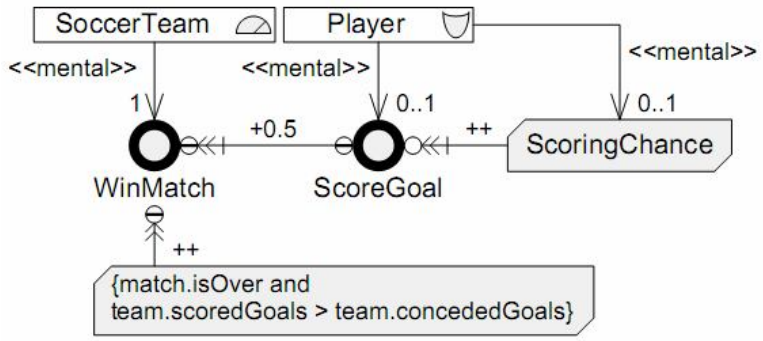


Figura 2.4.13: Exemplo de um Modelo Mental (TRENCANSKY, 2005).

Cervenka (2005b) refere-se ao termo atitude social e afirma que para modelar tipos de atitudes sociais, a AML utiliza as construções de modelagem mental já citadas, particularmente crenças, desejos, planos e relacionamentos mentais, usados pra representar relacionamentos lógicos entre estados mentais tais como meio-fim (means-end), decomposição, correlação e contribuição. Atitudes sociais de entidades são modeladas por meio do tipo especial de propriedade UML chamado propriedade mental. Ele pode ser usado ou na forma de um atributo mental possuído ou com o fim de uma associação mental.

Dois tipos de atitudes sociais podem ser reconhecidas, atitudes sociais compartilhadas por diversas entidades dentro de uma sociedade, crenças e objetivos comuns, planos que incluem a colaboração de diversas entidades, etc; e atitudes sociais de entidades individuais para qualquer coisa de valor social, como o compromisso de realizar uma ação social, crenças sobre fatos sobre outras entidades.

A figura a seguir descreve um trecho de modelo, mostrando um novo exemplo de modelo mental social, desta vez utilizando os tipos de unidades de organização cooperativas Equipe de Análise (AnalysisTeam) e Equipe de Implementação (ImplementationTeam). Eles compartilham o objetivo comum SatisfyQualityCriteria (satisfazer critérios de qualidade) e a crença que "a análise produzida/consumida é completa e consistente". Além das atitudes mentais comuns, as equipes têm também propósitos específicos modelados como objetivos que podem ser decididos. Assim, a equipe de análise tem o AnalyzeSystem e a equipe de implementação tem o ImplementSystem. A figura também apresenta a contribuição positiva necessária para atingir o objetivo geral do projeto, que é desenvolver o sistema (DevelopSystem).

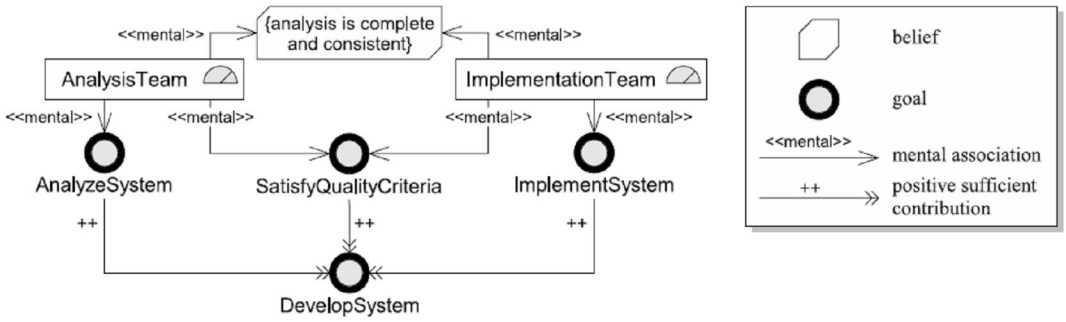


Figura 2.4.14: Exemplo de Crenças e Objetivos Individuais e Gerais (CERVENKA, 2005b).

2.4.13 Considerações Finais

A linguagem AML apresenta bons mecanismos para representar agentes e os papéis por eles interpretados, inclusive permite modelar a troca de papéis por estes dinamicamente durante um processo. Além disso, AML representa as percepções e as ações que os agentes, quando interpretando um papel, podem realizar, modelando os sensores e os causadores de efeito possuídos pelos papéis de entidade.

Segundo os autores, a linguagem permite representar os estados mentais de um agente ou de um papel de entidade, porém estes apresentam exemplos um tanto simples disto. Um objetivo é apoiado por contribuições de crença, que são um tipo de restrições representadas por meio de notas explicativas. Por meio destas contribuições de crença é possível estabelecer as condições para que um objetivo se torne uma intenção. Acreditamos que isso possa tornar o diagrama um pouco poluído, quando um objetivo necessitar basear-se em muitas crenças.

Além disso, não encontramos exemplos onde as crenças descritas nas contribuições de crenças estejam representadas e organizadas e, embora os autores falem em representar planos por meio de diagramas de atividade, não encontramos uma ligação explícita entre um objetivo e os possíveis planos a ele associados.

Não foi encontrada uma maneira de diferenciar agentes reativos de cognitivos e nem exemplos de como modelar os requisitos funcionais do sistema. Uma vez que AML é baseada na UML acreditamos que o diagrama de casos de uso seja utilizado para isso em sua forma tradicional, ou seja, representando os requisitos externos do software, mas não encontramos exemplos disso publicados.

Cervenka (2005a) comenta ser possível aplicar estados mentais para enriquecer a modelagem de casos de uso por meio da representação de requisitos baseada em objetivos, porém não encontramos exemplos de como isso poderia ser feito em seu trabalho. Em conversa informal com os autores fomos informados que este

enriquecimento se caracterizaria pela associação de classes-objetivo aos casos de uso, porém nenhum exemplo disso foi publicado até nosso último contato. Além disso, nesse tipo de modelagem, seriam identificados somente objetivos associados a casos de uso externos, não há representação de percepções, ações ou planos como requisitos funcionais do sistema.

Finalmente, nesta linguagem não foram criadas metaclasses a partir dos conceitos utilizados para a modelagem de casos de uso de forma a serem aplicadas na modelagem de requisitos funcionais de sistemas multi-agentes como o proposto em nosso trabalho.

2.5 MAS-ML – Multi-Agent System Modeling Language – Linguagem de Modelagem de Sistema Multi-Agente

Segundo Silva (2008) MAS-ML foi construída sobre a superestrutura da UML 2.1, estendo-a por meio de diversos novos conceitos de modelagem, representados por novas metaclasses e estereótipos, considerados apropriados para capturar as características típicas de sistemas multi-agentes. As extensões feitas ao metamodelo UML são aditivas, mantendo a MAS-ML de acordo com os frameworks de modelagem da OMG.

As extensões propostas incorporam os conceitos do modelo abstrato TAO (SILVA, 2003), que define um amplo conjunto de conceitos orientados a agente. Segundo Silva (2004a), o papel principal do framework TAO é fornecer um arcabouço (framework) conceitual para compreender abstrações distintas e seus relacionamentos de forma a suportar o desenvolvimento de sistemas multiagentes em larga escala. O arcabouço proposto extrai uma ontologia que conecta abstrações consolidadas, tais como objetos e classes com outras abstrações, tais como agentes, papéis e organizações, que são os fundamentos da engenharia de software baseada em agentes e objetos.

De acordo com Silva (2004a), dois tipos de modificações foram introduzidas, o primeiro tipo se refere aos conceitos de agente que são parcialmente cobertos pelas metaclasses UML já existentes, para este caso, estereótipos foram usados para aumentar suas semânticas. Já para os conceitos que não são cobertos ou são inapropriadamente cobertos pela UML, foram adicionadas novas metaclasses.

2.5.1 Definição de Entidades de Sistemas Multi-Agentes

Segundo Silva (2008), os conceitos básicos orientados a agente descritos em TAO são agente, organização, ambiente e papel (tanto papéis de agente como papéis de objeto). Agentes, organizações e ambientes são entidades estruturadas interativas que têm objetivos, planos e ações. Papéis especificam características e restrições de agente ou objeto. Estes são conceitos chave no desenvolvimento de software orientado a agentes, classificando-os como conceitos de classes-primárias em modelos de sistemas multi-agentes. Assim, para cada conceito, uma nova metaclasse foi adicionada, como demonstra a figura a seguir.

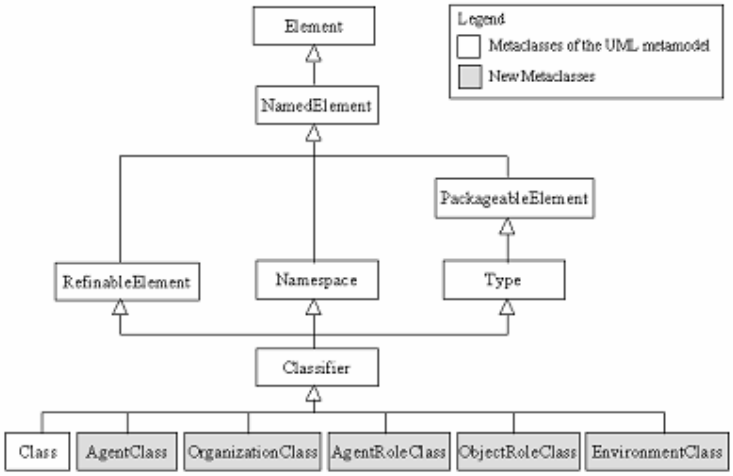


Figura 2.5.1: Metamodelo MAS-ML: Novas Metaclasses (parte I) (SILVA, 2008).

Na figura seguinte, Silva (2008) descreve os relacionamentos estruturais entre as metaclasses introduzidas, além das metaclasses definidas para representar os novos relacionamentos de MAS-ML.

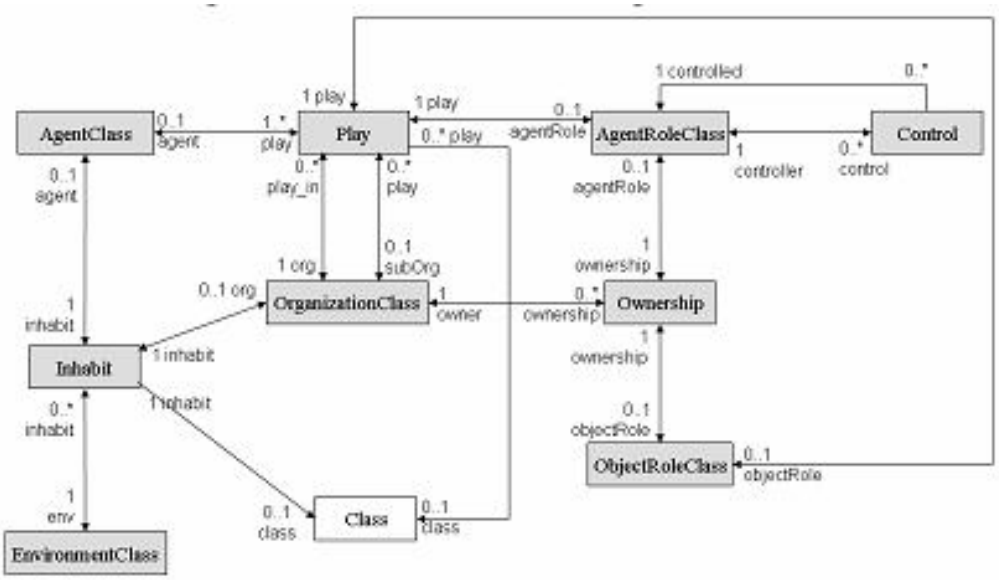


Figura 2.5.2: Metamodelo MAS-ML: Novas Metaclasses (parte II) (SILVA, 2008).

2.5.1.1 AgentClass

De acordo com Silva (2004b), o metamodelo TAO define um agente como um elemento autônomo, adaptativo e interativo no domínio cujas instâncias são expressas por meio de componentes mentais tais como crenças, objetivos, planos e ações. Um agente atua como um processador para planos de ações que são executadas, usando suas crenças para realizar um objetivo. Um objetivo é uma meta que o sistema deve atingir e uma crença é algum tipo de conhecimento sobre o sistema.

Silva (2004b) afirma ainda que se optou por estender o metamodelo UML de maneira a descrever agentes, criando a nova metaclasse AgentClass. Esta metaclasse não foi estendida a partir da metaclasse UML Class e sim da metaclasse Classifier e está associada com as metaclasses StructuralFeature e BehavioralFeature. Assim não há

ligação entre a metaclasse class da UML e, portanto, suas características não são atributos e operações.

Segundo Silva (2008), a AgentClass foi introduzida para descrever um conjunto de agentes que compartilham a mesma especificação de características, restrições e semânticas. Ela é um classificador que estende a metaclasse Classifier.

A metaclasse AgentClass foi criada uma vez que a definição da metaclasse Class fornecida pela especificação UML não pode ser usada para definir agentes. A especificação UML afirma que uma classe é um tipo de classificador cujas características são atributos e operações que podem ser invocadas por um objeto. Agentes não definem características comportamentais que possam ser invocadas por outros elementos.

As características estruturais da AgentClass são objetivos e crenças e suas características comportamentais são planos e ações. Uma vez que objetivos e crenças representam dados sobre o agente, eles foram descritos como estereótipos da metaclasse Property. De forma a iniciar sua execução, um agente deve ter no mínimo um objetivo que ele tentará atingir.

Planos e ações são características orientadas a agente que não tem contra-partes similares no metamodelo UML. Assim novas metaclasses foram adicionadas. Um agente deve ter no mínimo um plano, porque objetivos podem somente ser atingidos através da execução de planos.

Um agente também está imerso em um ambiente e está relacionado à, no mínimo, um papel. A AgentClass tem um relacionamento “Inhabit” com a EnvironmentClass e um relacionamento “Play” com a AgentRoleClass. Um agente deve habitar (inhabit) no mínimo um ambiente e deve executar no mínimo um papel.

De acordo com Silva (2004a) uma AgentClass é mostrada como um retângulo arredondado com três divisões separados por linhas horizontais. A primeira divisão contém o nome da AgentClass, a divisão do meio contém a lista de objetivos, podendo estes estar associados a planos que os possam atingir, e crenças do agente e a última divisão contém a lista de ações e planos, conforme apresentado pela figura seguinte.

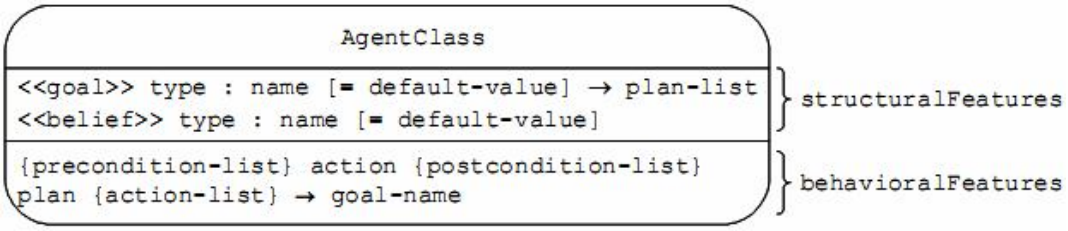


Figura 2.5.3: AgentClass (SILVA, 2004a)

2.5.1.2 OrganizationClass

De acordo com Silva (2004a), a metaclasse OrganizationClass foi criada para representar organizações. Segundo a autora, organizações estendem a noção de agentes descrevendo propriedades e relacionamentos adicionais. Uma organização define um conjunto de axiomas, ou seja, regras que os agentes e sub-organizações devem obedecer, desempenhando a função de restrições globais da organização. Uma organização também define papéis que devem ser interpretados pelos agentes e sub-organizações.

Silva (2004b) acrescenta que o metamodelo TAO define uma organização como um elemento no domínio cujas instâncias agrupam agentes, objetos e sub-organizações. Além de axiomas, uma organização também possui objetivos e crenças. No mínimo uma organização deve habitar o ambiente, chamada organização-principal.

De acordo com Silva (2008), a OrganizationClass foi introduzida para descrever um conjunto de organizações que compartilham as mesmas especificações de características, restrições e semântica. Ela é um classificador estendido a partir da metaclasse Classifier.

Uma organização é usada para modelar a estrutura social de um sistema multi-agente. Semelhante a agentes, organizações são entidades interativas e autônomas que tem objetivos, crenças, planos e ações e que estão imersas em ambientes e podem executar papéis. Diferente de agentes, organizações podem definir papéis de agente e papéis de objeto para ser executadas em organizações. Portanto, classes de organização estabelecem relacionamentos de domínio com estas classes de papel.

Uma vez que organizações descrevem estruturas sociais, elas podem também estabelecer um relacionamento "parte-de" com outras classes de organização, ou seja, uma organização pode ser parte de outra organização. Este relacionamento permite a modelagem da natureza hierárquica de organizações de agente. Uma organização pode também ser parte de mais de uma organização.

2.5.1.3 EnvironmentClass

De acordo com Silva (2004b), no metamodelo TAO, um ambiente é um elemento no domínio cujas instâncias são o habitat para agentes, objetos e organizações, um ambiente pode ser passivo como um objeto ou ativo como um agente.

Silva (2004a) declara que a metaclasse EnvironmentClass é uma classe abstrata criada para representar ambientes. Esta metaclasse não define qualquer propriedade já que elas dependem das características do ambiente, porém define associações entre ela e outras metaclasses MAS-ML.

Silva (2008) afirma que a classe Environment foi introduzida para descrever um conjunto de ambientes que compartilham a mesma especificação de características, restrições e semântica. É um classificador que estende a metaclasse Classifier.

O ambiente fornece o contexto lógico para os agentes, organizações e objetos realizarem suas ações, para comunicar-se e para adquirir informações sobre o problema que eles têm de resolver. O ambiente incorpora as regras de um domínio de problema e os agentes reagem a estas regras.

O ambiente é extensivo e ilimitado, enquanto os agentes são limitados e localizados no ambiente. Uma classe de ambiente relaciona-se com classes de agente, classes e classes de organização por meio do relacionamento inhabit de forma a representar que as instâncias daquelas classes podem habitar ambientes que são instâncias da classe de ambiente.

Em MAS-ML, um ambiente é modelado como um retângulo com 4 compartimentos, sendo que o quarto contém as classes cujas instâncias podem habitar as instâncias da classe de ambiente.

2.5.1.4 AgentRoleClass

De acordo com Silva (2004b), o metamodelo TAO declara que um papel de agente guia e restringe o comportamento de um agente por meio da descrição de um conjunto

de objetivos, crenças e ações. Um papel de agente define deveres, direitos e protocolos que restringem uma instância de agente. Um dever define uma ação que deve ser executada por uma instância de agente; um direito define uma ação que pode ser executada por uma instância de agente; e um protocolo define uma interação entre um papel de agente e outros elementos.

Segundo Silva (2004a), um papel de agente preocupa-se em guiar e restringir o comportamento autônomo de um agente ou organização pela descrição de seus objetivos, crenças e ações que o agente deve e que pode realizar quando estiver interpretando o papel. Para representar os papéis interpretados pelos agentes criou-se a metaclasse AgentRoleClass.

Segundo Silva (2008), a classe AgentRole foi introduzida para descrever um conjunto de papéis de agente que compartilham a mesma especificação de características, restrições e semântica. É um classificador que estende a metaclasse Classifier.

As duas propriedades de papéis mais importantes são sua definição no contexto de uma organização (relacionamento ownership) e o fato que uma instância de papel dever ser interpretada por um agente, por um objeto ou por uma sub-organização (relacionamento play). Uma classe de papel de agente é definida no escopo de uma classe de organização por meio do uso do relacionamento ownership. As instâncias de papel são interpretadas por agentes ou sub-organizações (relacionamento play) executadas em instâncias da classe de organização onde a classe de papel de agente foi definida.

As características estruturais da AgentRoleClass são objetivos, deveres, direitos e protocolos. Os objetivos de um papel de agente estão relacionados aos objetivos da entidade que irá executar o papel. De forma a tentar atingir seus objetivos, a entidade compromete-se a executar um papel que define objetivos similares. Ao executar o papel a entidade tenta atingir no mínimo um sub-conjunto de seus objetivos.

Deveres e direitos especificam que ações um papel de agente deve ou pode fazer no ambiente. Um protocolo especifica, estaticamente, como agentes que executam o papel podem interagir para atingir um ou mais de seus objetivos.

De acordo com Silva (2004a) uma AgentRoleClass é representado por um retângulo sólido com uma curva em sua parte de baixo contendo três divisões. A primeira armazena o nome da AgentRoleClass, a segunda os objetivos e crenças associados com o papel e a divisão de baixo contém os deveres, direitos e protocolos, como mostra a figura seguinte.

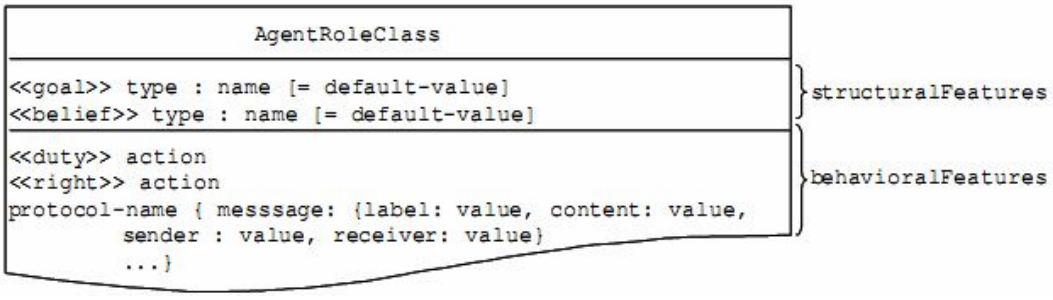


Figura 2.5.4: AgentRoleClass (SILVA, 2004a)

2.5.1.5 ObjectRoleClass

De acordo com Silva (2004b), o metamodelo TAO declara que um papel de objeto guia e restringe o comportamento de um objeto por meio da descrição de um conjunto de características que são vistas por outros elementos. Um papel de objeto pode restringir o acesso ao estado e comportamento de uma instância de objeto, mas pode também adicionar informações, comportamentos e relacionamentos à instância de objeto que executa o papel.

Silva (2008) afirma que a classe ObjectRole foi introduzida para descrever um conjunto de características vistas pelas entidades que tentam acessar o objeto que executa o papel. É um classificador estendido a partir da metaclasse Classifier.

Papéis de objeto são definidos pela estrutura organizacional, assim uma organização possui a definição de um papel (relacionamento ownership). Uma classe de papel de objeto restringe o acesso a alguns atributos e operações definidas na classe que está relacionada à classe de papel de objeto pelo relacionamento play. Uma classe de papel de objeto pode também definir atributos e operações adicionais para tais classes.

2.5.2 Definição de Características Estruturais de Sistemas Multi-Agentes

De forma a modelar as duas características estruturais de objetivo e crença não definidas na UML e requeridas pela TAO, Silva (2008) estendeu a metaclasse Property definindo dois estereótipos, como demonstra a figura seguir.

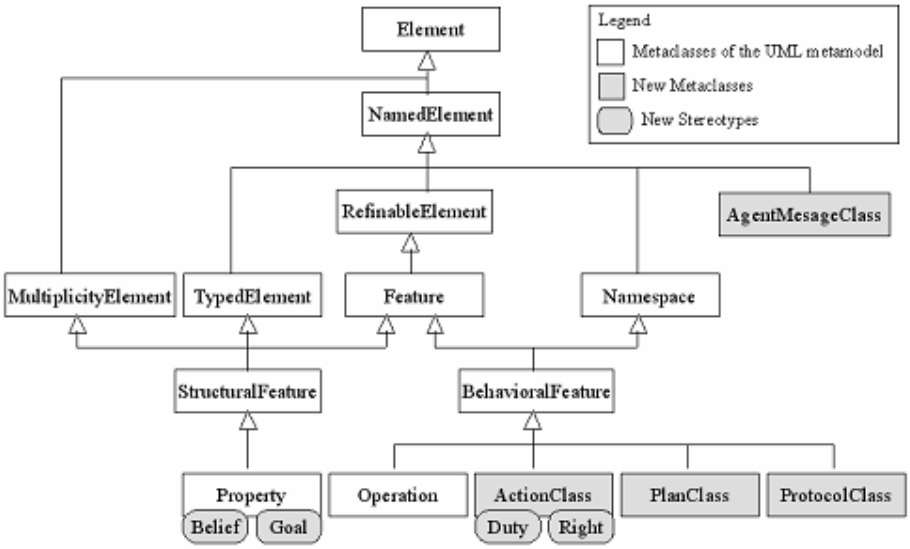


Figura 2.5.5: Metamodelo MAS-ML: Novas Características Comportamentais (SILVA, 2008).

2.5.2.1 Goal

Segundo Silva (2008), objetivos (goals) de agentes e organizações consistem de estados ou desejos futuros que um agente ou uma organização gostaria de atingir ou satisfazer. Os objetivos de um papel de agente definem os objetivos que o agente ou a organização, ao se comprometerem a interpretar o papel, devem atingir.

Um objetivo é uma característica estrutural de uma classe de agente, classe de papel de agente ou classe de organização. Ele relaciona uma instância da c1asse de agente, classe de papel de agente ou classe de organização para um valor ou coleção de valores

do tipo do objetivo. É um estereótipo que pode ser aplicado a uma instância de uma metaclasse Property.

Silva (2004a) afirma que, uma vez que objetivos e crenças são estereótipos de Property e, portanto podem ser descritos como um atributo, suas notações são similares à notação de um atributo. Objetivos e crenças são descritos pelos seus nomes, tipos e valores default. Além disso, um objetivo também identifica os planos para atingir o objetivo (exceto nas classes de papéis de agente). No caso em que um objetivo é sub-dividido em sub-objetivos, estes devem ser igualmente descritos como atributos.

2.5.2.2 Belief

Segundo Silva (2008), uma crença (belief) é uma característica estrutural de uma classe de agente ou de organização. Ela relaciona uma instância da classe de agente ou de organização a um valor ou coleção de valores do tipo da crença. Ela é um estereótipo que pode ser aplicado a uma instância de uma metaclasse Property.

O estereótipo belief especifica a definição de Property de forma a representar as crenças ou conhecimentos que um agente ou organização tem sobre o ambiente, sobre ele mesmo e sobre outras entidades. As crenças incluem o que um agente ou organização sabe, suas memórias e suas percepções sobre o que acontece no sistema.

2.5.3 Aspectos Estáticos da MAS-ML

Segundo Silva (2004b), as definições dos conceitos de objeto e evento do metamodelo TAO são semelhantes aos conceitos das meta-classes Class e Event da UML, assim sua notação foi preservada na MAS-ML. No entanto, foi necessário criar novas definições de meta-classes para outros conceitos TAO, tais como as classes AgentClass, OrganizationClass, EnvironmentClass, ObjectRoleClass e AgentRoleClass. Seguindo a terminologia UML, estes novos elementos de diagrama são compostos de três compartimentos separados por linhas horizontais. O compartimento de cima contém o nome da classe que deve ser única. O compartimento do meio contém a lista de características estruturais e o compartimento de baixo contém a lista de características comportamentais. Além disso, os novos elementos de diagrama foram associados com novos relacionamentos definidos no metamodelo TAO que não existem em UML. Estes relacionamentos são Inhabit, Ownership, Play e Control.

A MAS-ML estende o diagram de classes para incluir informação de modelagem sobre agentes. Este diagrama mostra os relacionamentos de associação, agregação e especialização do metamodelo TAO. Além disso, MAS-ML define ainda os diagramas de Organização e de Papel.

2.5.4 Aspectos Dinâmicos da MAS-ML

De acordo com Silva (2004b), a MAS-ML propõe uma extensão do diagrama de sequência UML para modelar os aspectos dinâmicos baseados no metamodelo TAO. O diagrama de sequência MAS-ML possui três novos elementos para representar conceitos referentes a agente, organização e ambiente, além de propor novas formas para definir caminhos de nomes que identificam instâncias de elemento. Diagramas de sequência ilustram compromissos com papéis e mudanças de papéis de agentes e organizações; envio e recebimento de mensagens por agentes e organizações; execução de ações por agentes e organizações; métodos de chamada de elementos de um objeto; execução de métodos por objetos e criação e destruição de elementos.

Para modelar estas novas características, foram criados novos estereótipos e estendida a definição dos estereótipos UML <<create>> e <<destroy>>, como detalhado a seguir:

create: este estereótipo foi especializado para representar a criação de agentes, organizações e ambientes; o estabelecimento de uma associação de uma instância de papel com outras instâncias de papel e organização; e a criação de um objeto e a associação de um papel para o objeto.

destroy: este estereótipo foi especializado para representar a destruição de agentes, organizações e ambientes, bem como de todas as instâncias de papel associadas com estes elementos.

role_commitment: este estereótipo foi criado para representar o compromisso com um papel por um agente, organização ou objeto.

role_cancel: este estereótipo representa o cancelamento de um papel, quando um elemento pára de interpretar o papel quando um agente ou organização deixa outra organização.

role_deactivate: este estereótipo muda o estado de um papel que um agente ou organização está interpretando de ativo para inativo.

role_activate: já este estereótipo faz exatamente o contrário do anterior.

role_change: este estereótipo representa a mudança de papel por um agente ou organização.

2.5.5 Definição de Características Comportamentais de Sistemas Multi-Agentes

Segundo Silva (2008), uma característica comportamental é uma característica de um classificador que especifica um aspecto do comportamento de suas instâncias. No metamodelo UML uma característica comportamental especifica que uma instância de um classificador responderá a uma requisição designada pela invocação de um comportamento.

A operação é o único tipo de característica comportamental da UML, no entanto, uma operação não serve para especificar um comportamento de agente. Uma operação especifica o nome, tipo, parâmetros e restrições para invocar um comportamento associado. Entidades orientadas a agente não são executadas pela invocação de operações, elas executam planos, ações e protocolos. Estes conceitos devem ser conceitos de primeira classe em modelos de sistemas multi-agentes. Assim para cada conceito uma nova metaclasse foi adicionada, além de dois novos estereótipos, como pode ser visto na figura 2.5.6.

2.5.5.1 ActionClass

Silva (2008) declara que esta classe descreve um conjunto de ações de uma classe de agente ou organização, especificando o nome, tipo, parâmetros e restrições para executar um comportamento associado.

A UML fornece uma metaclasse Action. No entanto, uma ação é a unidade fundamental de funcionalidade executável. A execução de uma ação representa alguma transformação ou processamento no modelo do sistema. Assim, uma ação é uma entidade de nível muito baixo, usada em Diagramas de Atividade (OMG 2007b).

Silva (2008) define uma ação de agente como uma unidade de comportamento que um agente pode realizar. Silva (2004a) declara que uma ação é expressa como uma lista

de pré-condições, seu nome e uma lista de pós-condições. Os parâmetros das pré e pós-condições são seu tipo, nome e valor e são descritos entre chaves ({}), caso não haja parâmetros as chaves são deixadas vazias.

2.5.5.2 PlanClass

Segundo Silva (2008), esta classe foi introduzida para identificar um conjunto de ações a ser executadas para atingir objetivos. Uma nova metaclasse foi criada para representar planos uma vez que não há metaclasse no metamodelo UML com o significado de plano.

Silva (2008) afirma que a metaclasse Operation (OMG 2007b) não pode ser utilizada por que uma operação pode ser invocada por uma entidade enquanto planos de agente não podem ser invocados. Além disso, a metaclasse Activity também não pode ser usada por que uma atividade especifica a coordenação de execuções de comportamentos subordinados modelados como nós de atividades. Uma vez que um nó pode representar a execução de uma operação e um plano não é simplesmente uma sequência de invocações de método, esta metaclasse não pode ser usada para definir planos. Além disso, um plano deve sempre estar relacionado ao classificador AgentClass e pode ocorrer que algumas atividades não possam ser atribuídas a um classificador. Finalmente, um plano sempre está relacionado a no mínimo um objetivo que informa ao agente sobre o propósito do plano.

Silva (2004a) declara que um plano é definido pelo seu nome, por um conjunto de ações associadas ao plano e o objetivo que o plano tenta atingir.

2.5.5.3 ProtocolClass

Silva (2004a) declara que esta classe foi criada para descrever os protocolos que o agente deve obedecer enquanto interpreta um papel e interage com outras entidades. Ela define o conjunto de mensagens que um agente pode enviar e receber de outros agentes em uma interação.

Silva (2008) acrescenta que esta classe tem por objetivo definir as interações entre papéis através da especificação das mensagens. Um protocolo define o conjunto de mensagens que um agente tem permissão de enviar a outros agentes em uma interação e as mensagens que ele pode receber deles.

2.5.5.4 AgentMessageClass

Silva (2004a) afirma que mensagens enviadas e recebidas por um agente são diferentes de mensagens enviadas e recebidas por um objeto, por este motivo, para diferenciar mensagens normais de mensagens de agentes, esta metaclasse foi criada para representar mensagens especificadas em protocolos.

Silva (2008) afirma que o objetivo desta metaclasse é diferenciar mensagens de agente (alto nível, comunicação de ato de fala) de mensagens de objeto (invocação de método). As propriedades desta metaclasse são baseadas na linguagem FIPA ACL1.

2.5.5.5 Deveres e Direitos (Duty e Right)

Segundo Silva (2008), deveres identificam ações que devem ser executadas pelo agente que está executando o papel. O agente é uma instância de uma classe de agente associada pelo relacionamento play ao papel que é uma instância da classe de papel de agente que define o dever. É um estereótipo que pode ser aplicado a uma instância de uma metaclasse ActionClass.

Já direitos identificam ações que o agente pode realizar quando executando o papel, descrevendo permissões associadas com ações. O agente é uma instância de uma classe de agente associada pelo relacionamento play ao papel que é uma instância da classe de papel de agente que define o direito. É também um estereótipo que pode ser aplicado a uma instância de uma metaclasse ActionClass.

2.5.6 Definição de Relacionamentos de Sistemas Multi-Agentes

De acordo com Silva (2008), um relacionamento referencia um ou mais elementos relacionados. Há diversas classes que estendem a metaclasse Relationship, mas três relacionam classes de objeto: Association, Dependency e Generalization. No entanto, em MAS-ML foram criados novos relacionamentos para entidades orientadas a agente como agentes, organizações e ambientes, como demonstra a figura a seguir,

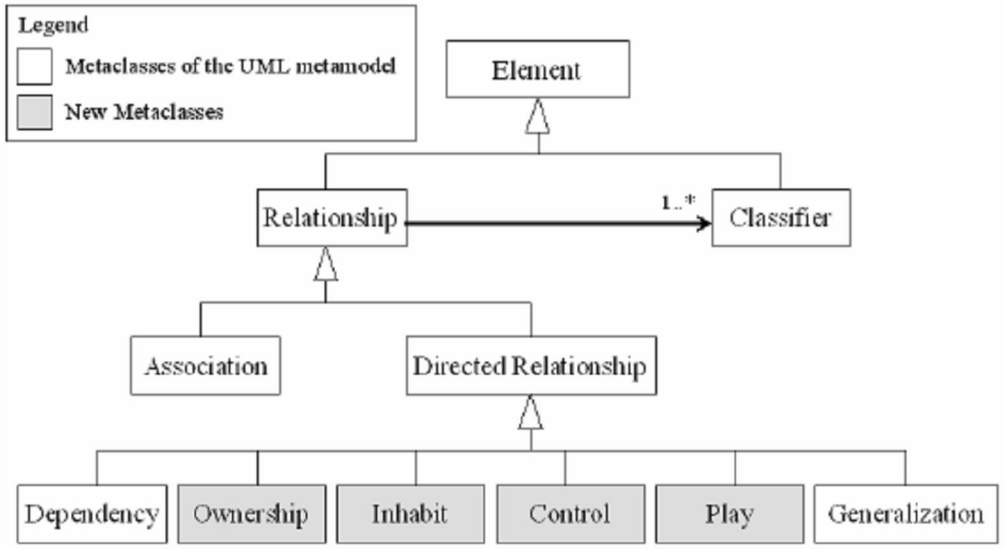


Figura 2.5.6: Metamodelo MAS-ML: Novos Relacionamentos (SILVA, 2008).

2.5.6.1 Play

De acordo com Silva (2004a) este relacionamento especifica que uma entidade está relacionada a um papel, o que significa que uma instância da entidade pode interpretar uma instância de papel. Uma instância de agente ou sub-organização devem interpretar ao menos uma instância de papel.

Silva (2008) acrescenta que um relacionamento play especifica um relacionamento semântico que pode ocorrer entre instâncias de um agente ou objeto, de um papel e de uma organização. O relacionamento play define os papéis a serem executados, as entidades que irão executar os papéis e as organizações onde os papéis serão executados.

2.5.6.2 Ownership

Silva (2004a) afirma que este relacionamento especifica que uma entidade membro está definida dentro do escopo da entidade possuidora e que o membro deve obedecer um conjunto de restrições globais definidas por seu possuidor. O membro não existe fora do escopo de seu possuidor. Os possuidores conhecem seus membros e cada membro conhece seu possuidor. Este relacionamento é aplicado a classes de papéis (membros) e a classes de organizações (possuidores). Um papel só pode ser interpretado por entidades na organização que definiu o papel.

Silva (2008) acrescenta que ownership especifica um relacionamento semântico que pode ocorrer entre instâncias de um papel e de uma organização. Indica que tipo de papel está definido em uma organização. Um ownership é necessário para permitir a modelagem de relacionamentos play.

2.5.6.3 Inhabit

De acordo com Silva (2004a), este relacionamento especifica que a instância de entidade é criada e destruída no habitat e que esta entidade é capaz de deixar habitats e de entrar em habitats, respeitando as permissões dos habitats. Não é possível habitar dois habitats ao mesmo tempo. O habitat conhece todos os cidadãos (entidades que o habitam) e cada cidadão conhece seu habitat.

Silva (2008) acrescenta que este é um relacionamento semântico que pode ocorrer entre instâncias de entidades (agente, organização ou objeto) e de um ambiente. Indica quais entidades estão imersas no ambiente. Um inhabit declara que pode haver links entre instâncias de agentes, organizações e objetos do ambiente. Um link indica que a entidade pode executar no ambiente. Links podem ser criados e destruídos se entidades movem-se no sistema.

2.5.6.4 Control

Segundo Silva (2004a) este relacionamento define que a entidade controlada deve fazer qualquer coisa que a entidade controladora requeira. A entidade controladora conhece as entidades que ela controla e cada entidade controlada conhece as entidades que a controlam. Este relacionamento pode ser usado entre dois papéis de agente. O controlador e as entidades controladas serão os agentes ou organizações que interpretarão os papéis.

Silva (2008) acrescenta que um control especifica um relacionamento semântico muito específico que pode ocorrer entre instâncias de papéis de agente. Um control é usado para modelar um relacionamento social do tipo mestre/escravo entre papéis de agente. O relacionamento control anula a autonomia do agente, uma vez que um papel controlador pode controlar os agentes que assumem o papel controlado e o papel controlado não pode dizer não às requisições feitas pelo controlador.

2.5.7 Diagramas Estruturais da MAS-ML

De acordo com Silva (2008) MAS-ML propõe o uso de três diagramas estruturais para modelar os aspectos estruturais estáticos de um Sistema Multi-Agente.

2.5.7.1 Diagrama de Classes

Silva (2008) afirma que o diagrama de classes MAS-ML é uma extensão do diagrama de classe da UML para modelar os aspectos estruturais de classes, agente, organização, ambientes e os relacionamentos entre estas entidades. O diagrama de classes MAS-ML estende o diagrama de classes UML definindo novos elementos de diagrama para modelar as novas classes e os relacionamentos entre estes elementos.

Silva (2004a) declara que o principal objetivo de estender o diagrama de classes foi modificá-lo para representar os relacionamentos entre as classes e outras entidades de sistemas multi-agentes. O diagrama de classes estendido representa os relacionamentos entre as classes e ambientes, classes e agentes, classes e organizações e os relacionamentos entre agentes, entre ambientes e entre organizações.

Silva (2008) acrescenta que o diagrama de classes permite ao projetista descrever os ambientes que os agentes podem habitar (inhabit), os recursos (objetos) que os agentes e as organizações podem manipular e as organizações e suas sub-organizações. Os relacionamentos entre agentes e objetos existem somente se agentes de tais classes de agente podem acessar os objetos independentemente do papel que eles estão executando. Se há qualquer restrição para o papel sendo executado, o relacionamento não deveria ser modelado em diagramas de classes, mas em diagramas de papéis. O mesmo é verdadeiro para organizações.

Neste diagrama os classificadores EnvironmentClass, OrganizationClass, AgentClass e Class podem ser instanciados. Quanto aos relacionamentos, podem ser usados todos os definidos na UML, mas alguns possuem funções extras, como o relacionamento de associação, que é usado para modelar links entre agentes ou organizações e objetos. Uma associação indica os recursos do ambiente que agentes e organizações podem usar, além de poder também ligar ambientes para indicar que agentes podem se mover entre ambientes ligados.

Há também o relacionamento de agregação, que é usado para modelar hierarquias de organizações, ou seja, para identificar as organizações que são partes de outra organização. Finalmente, o relacionamento de especialização é usado para modelar relacionamentos de especialização entre classes de agentes ou organizações.

Na figura a seguir, Silva (2008) mostra o diagrama de classes que modela um sistema de gerenciamento de conferência. A organização que representa a conferência é composta por oficinas e tutoriais. Os recursos disponíveis no sistema são artigos (Paper), avaliações gerais (Overall_evaluation) e avaliações (Review). Os agentes que interagem com tais recursos são alunos (Student), pesquisadores (Researcher) e professores.

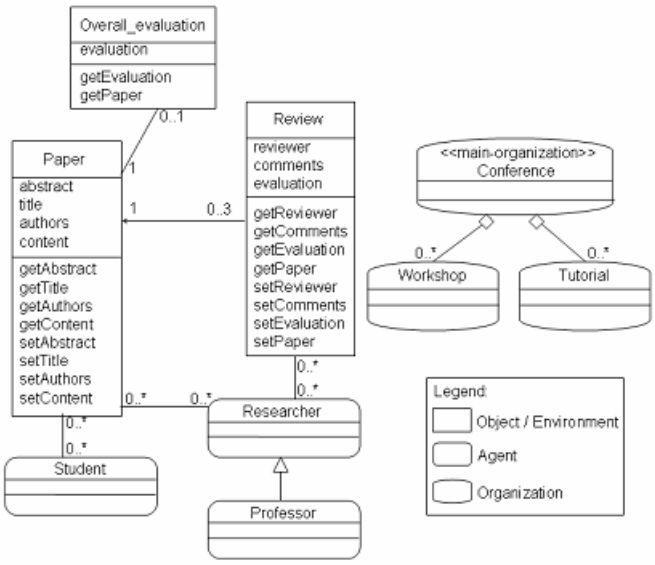


Figura 2.5.7: Exemplo de Diagrama de Classe (SILVA, 2008).

2.5.7.2 Diagrama de Organização

De acordo com Silva (2004b), este diagrama modela as organizações do sistema identificando seus habitantes, os papéis que eles definem e os elementos que interpretam estes papéis. Este diagrama utiliza os relacionamentos TAO ownership, play e inhabits e, segundo Silva (2008), permite instanciar os classificadores

OrganizationClass, AgentClass, AgentRoleClass, Class, ObjectRoleClass e EnvironmentClass. Silva (2004a) acrescenta que os relacionamentos de especialização entre as organizações são modelados em diagramas de classes.

Segundo (Silva, 2008), um diagrama de organização modela as organizações do sistema detalhando como tais organizações estão relacionadas a outras entidades do sistema. Portanto, deve haver um diagrama de organização para cada classe de organização. O diagrama mostra a classe de ambiente que pode fornecer os ambientes para ser o habitat das organizações, as classes de papel possuídas pela classe de organização e as classes de agente, classes e classes de (sub-)organização cujas instâncias executarão os papéis.

As figuras 2.5.8 e 2.5.9 exemplificam o uso de diagramas de organização. Ambos os diagramas modelam a organização da conferência (Conference). A figura 2.5.8 mostra as classes de papel de agente possuídas pela conferência que pode ser instanciada para ser executada pelos agentes das classes de agente relacionadas a tais classes de papel de agente. Por exemplo, a classe de papel de agente Chair pode ser instanciada para ser executada pelos pesquisadores (Researchers) e professores, uma vez que Professor especializa Researcher como ilustrado no diagrama de classes da figura 2.5.9. Por outro lado, alunos não podem nem ser os Chairs de conferências nem membros do PC, mas podem ser os autores de artigos (Papers). Esta figura também ilustra as sub-organizações que são parte de Conference: Workshop e Tutorial.

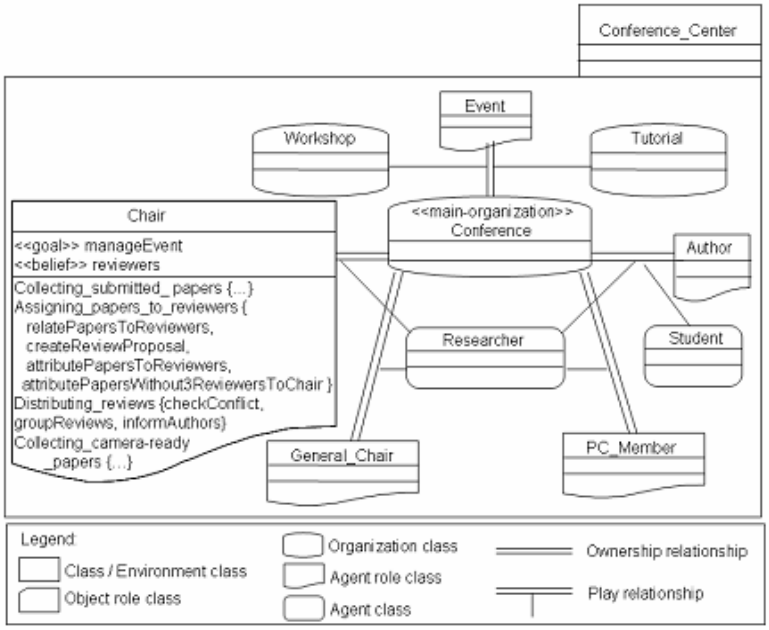


Figura 2.5.8: Exemplo de Diagrama de Organização (parte I) (SILVA, 2008).

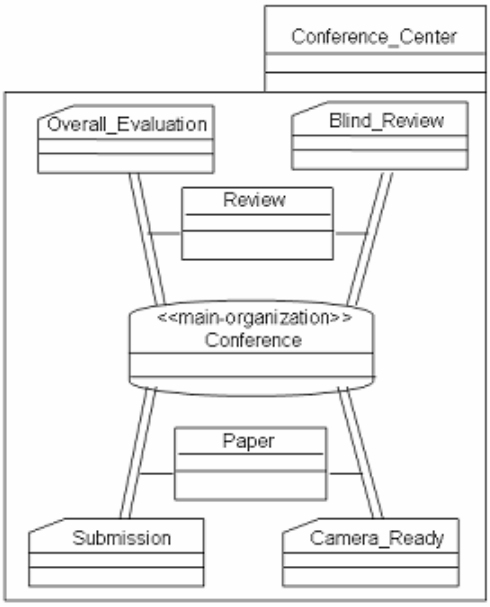


Figura 2.5.9: Exemplo de Diagrama de Organização (parte II) (SILVA, 2008).

A figura 2.5.9 descreve as classes de papel de objeto possuídas pela Conference e as classes cujos objetos podem executar os papéis. Papers (artigos) podem ser vistos como submissões ou como Camera-Readies e avaliações (Reviews) podem ser vistas como avaliações cegas.

2.5.7.3 Diagrama de Papel

Silva (2004b) destaca que o diagrama de organização não descreve os relacionamentos entre os papéis, por isso é complementado por este diagrama. Segundo Silva (2004a), o diagrama de papel é responsável por ilustrar os relacionamentos entre os papéis de agente e os papéis de objeto identificados nos diagramas de organização. Este diagrama também identifica as classes acessadas pelos papéis de agente e de objetos. As interações entre os agentes e organizações do sistema são descritas baseadas nos relacionamentos entre os papéis ilustrados nos diagramas de papel.

Silva (2008) acrescenta que este diagrama é responsável por modelar os aspectos estruturais de papéis de agente e papéis de objeto definidos nas organizações e também os relacionamentos entre os papéis e entre os papéis e os objetos no ambiente. Neste diagrama é possível instanciar os classificadores AgentRoleClass, ObjectRoleClass e Class. É também possível utilizar os relacionamentos control, dependency, association e specialization.

O relacionamento control é usado entre classes de papel de agente, enquanto o dependency é usado entre classes de papel de objeto; entre classes de papel de agente e classes de papel de objeto; e entre classes de papel de agente.

O relacionamento de association é usado entre classes de papel de objeto, o que significa que um objeto que executa um papel tem acesso limitado a outro objeto e que as políticas de acesso são definidas pelo papel sendo executado pelo outro objeto; entre classes de papel de agente e classes de papel de objeto, que declara que agentes que executam tais papéis tem acesso limitado aos objetos que executam o outro papel e as políticas de acesso são definidas pelo papel de objeto; entre classes de papel de agente, o que significa que agentes que executam tais papéis podem interagir; e entre classes de

papel de agente e classes, o que significa que um agente que executa tal papel tem total acesso ao objeto.

O relacionamento de specialization é usado entre classes de papel de objeto e entre classes de papel de agente.

A figura a seguir apresenta um diagrama de papel contendo as classes de papel de agente e as classes de papel de objeto definidas na organização Conference. Neste diagrama, o relacionamento entre as classes de papel de objeto Author e Blind_Review restringe o acesso de agentes que executam o papel Author para revisores (instâncias da classe Review). Autores não podem acessar avaliações (reviews) diretamente uma vez que não há uma associação entre as classes Author e Review no diagrama de classes. A única forma que autores podem acessar as avaliações é usando os papéis de objeto blind reviews que não fornecem métodos para modificar as avaliações ou obter o nome dos revisores, que é um atributo de Review.

As possíveis interações entre os agentes são modeladas por meio de relacionamentos entre os papéis que irão ser executados por estes agentes, como o relacionamento entre as classes de papel de agente Chair e Author que define que agentes que executam instâncias de tais papéis podem interagir. Ao observar o diagrama de organização pode-se perceber que alunos podem executar Author mas não podem executar Chair. Assim o sistema não torna possível a interação entre dois alunos.

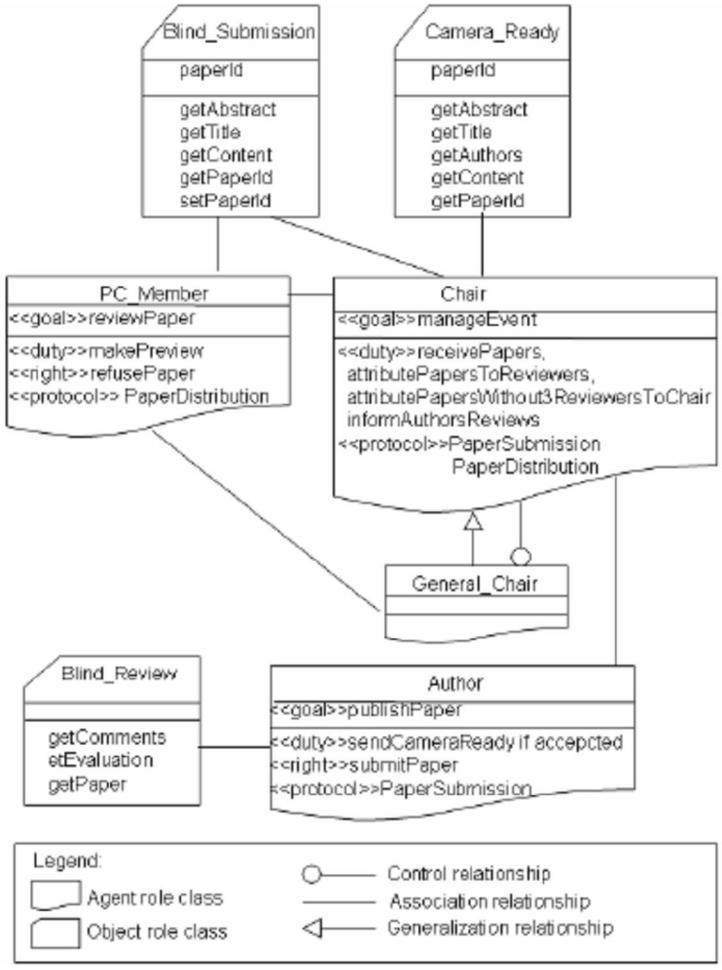


Figura 2.5.10: Exemplo de Diagrama de Papel (SILVA, 2008)

2.5.8 Diagramas Dinâmicos da MAS-ML

Silva (2008) afirma que MAS-ML define dois diagramas dinâmicos diferentes estendendo os diagramas de sequência e atividade da UML.

2.5.8.1 Diagrama de Sequência

De acordo com Silva (2004a), o propósito de estender o diagrama de sequência da UML foi o de representar as interações entre as instâncias de sistemas multi-agentes e as intra-ações definidas por cada instância.

Estender esse diagrama exigiu a criação de novos caminhos de nomes (pathnames) e ícones para representar instâncias de sistemas multiagentes como agentes, organizações e ambientes. Além disso, para representar as interações entre agentes, organizações, ambientes e objetos, foi estendido o conceito de mensagem usado na UML para representar entidades que enviem e recebam mensagens e não somente chamem métodos de outras entidades. Também foram criados e redefinidos alguns estereótipos associados com mensagens para representar a criação e destruição de instâncias de agentes e para representar as interações entre agentes, organizações, objetos e seus papéis.

O diagrama de seqüência também foi estendido para representar a execução de planos e ações na modelagem de intra-ações relacionadas a agentes, organizações e ambientes; e ainda para ilustrar os protocolos descritos pelos papéis de agentes. Quando este diagrama é utilizado para modelar um protocolo, o diagrama representa os papéis de agentes envolvidos no protocolo e a sequência de mensagens definidas pelo protocolo.

Segundo (Silva, 2008), um diagrama de sequência identifica as entidades propriamente ditas, os papéis que elas estão executando, as organizações nas quais os papéis estão sendo executadas e também os ambientes onde eles estão imersos.

Silva (2008) exemplifica que, de forma a identificar completamente o contexto da execução de um agente é importante destacar o agente (instância), o papel que um agente está executando enquanto realiza as ações, a organização onde o agente está executando tal papel e o ambiente onde o agente está inserido.

A figura a seguir ilustra a interação entre três agentes: Rob executando o papel Reviewer; Charles executando o papel Chair; e Ana executando o papel Author. De forma a identificar o contexto da execução do agente Charles é necessário destacar o papel que ele está executando (AOSEChair), a organização onde ele está executando tal papel (workshop) e o ambiente onde ele está imerso (Conference_Center). No caso de uma identificação não ambígua, parte do pathname pode ser suprimido, como é o caso do pathname dos dois outros agentes.

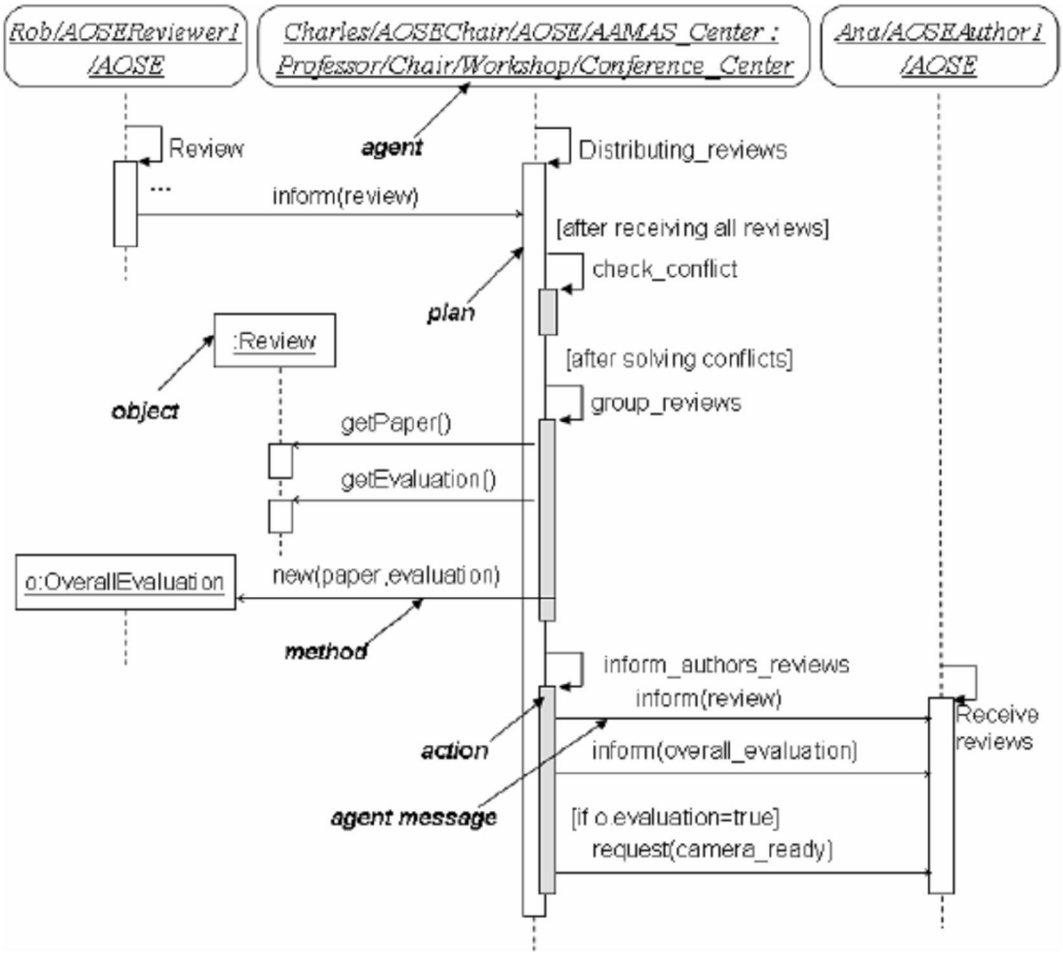


Figura 2.5.11: Modelagem de Planos e Ações - Diagrama de Sequência (SILVA, 2008).

Com relação a interações, embora objetos interajam pela chamada de métodos de outros objetos, agentes e organizações interagem pelo envio e recebimento de mensagens. Dessa forma, o diagrama de sequência MAS-ML torna possível a modelagem de envio e recebimento de mensagens destacando as performativas (elementos interpretativos) das mensagens e indicando o tipo dos conteúdos. A figura 2.5.11 ilustra o chair Charles recebendo uma avaliação de um revisor e enviando-a para um autor.

O diagrama de sequência MAS-ML pode também ser usado para modelar os protocolos de interação que são usados pelos agentes e sub-organizações para comunicação. A figura seguinte ilustra o protocolo de interação “Paper Submission” que é usado entre agentes que executam os papéis author e chair.

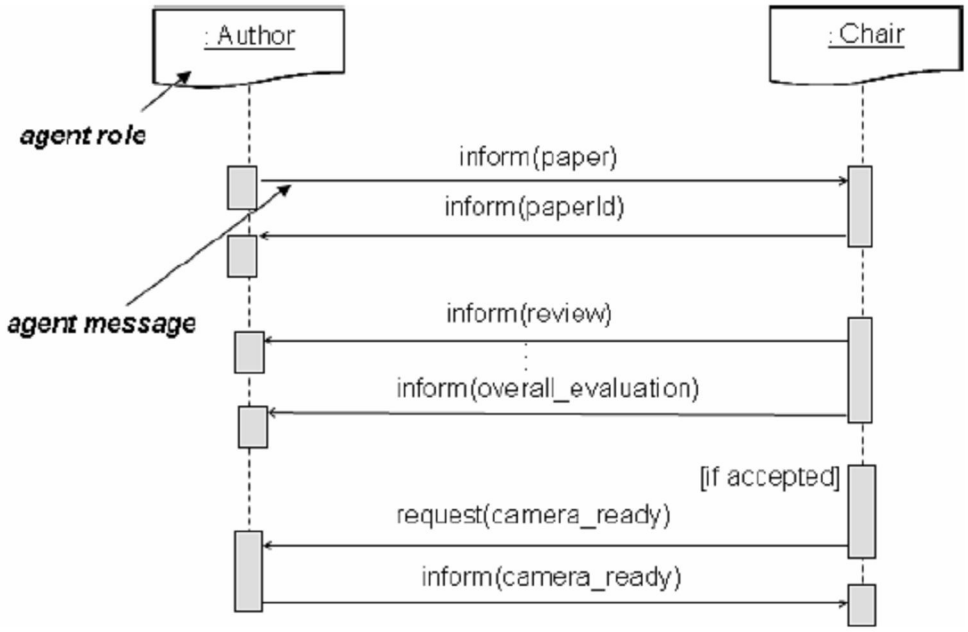


Figura 2.5.12: Modelagem de um Protocolo - Diagrama de Sequência (SILVA, 2008).

Segundo Silva (2008), o diagrama de sequência MAS-ML precisa ser capaz de modelar planos e ações e não somente métodos, porque, diferente de objetos, agentes e organizações não tem métodos, eles executam planos e ações de forma a atingir seus objetivos. Durante a execução dos planos e ações, agentes e organizações podem interagir e acessar recursos.

A figura 2.5.11 mostra o chair da oficina AOSE executando o plano "Distribute reviews". O chair recebe as avaliações dos avaliadores e, depois de verificar e resolver os conflitos, distribui as avaliações para os autores. Enquanto realiza essa distribuição, o chair usa o protocolo "Paper Submission" ilustrado na figura 2.5.10.

Uma vez que agentes podem mudar seus papéis, é importante representar agentes comprometidos em mudar e cancelar seus papéis no diagrama de seqüência. Portanto, o diagrama de seqüência foi estendido com um conjunto de estereótipos para mostrar o comportamento dinâmico relacionado a papéis. Estes estereótipos podem ser vistos na tabela a seguir.

Tabela 2.5.2: Estereótipos de Papel

Role_commitment	Compromete-se a um novo papel
Role_cancel	Cancela um de seus papéis
Role_change	Muda de um papel para outro
Role_deactivate	Desativa um de seus papéis
Role_activate	Ativa um papel inativo

Fonte: SILVA, 2008

De forma a exemplificar o uso de tais estereótipos Silva (2008) ilustra o plano "Assign papers to reviewers" executado pelos chairs. Enquanto executa o plano, o chair tenta alocar 3 revisores a cada artigo submetido. No entanto, se não há avaliadores suficientes, o chair pode contatar novos avaliadores ou o próprio chair pode realizar as avaliações. A figura a seguir descreve o agente executando o papel chair, tornando-se

um avaliador. Depois de se comprometer a executar o papel de revisor, o agente Charles está executando dois papéis ao mesmo tempo: chair e reviewer (avaliador).

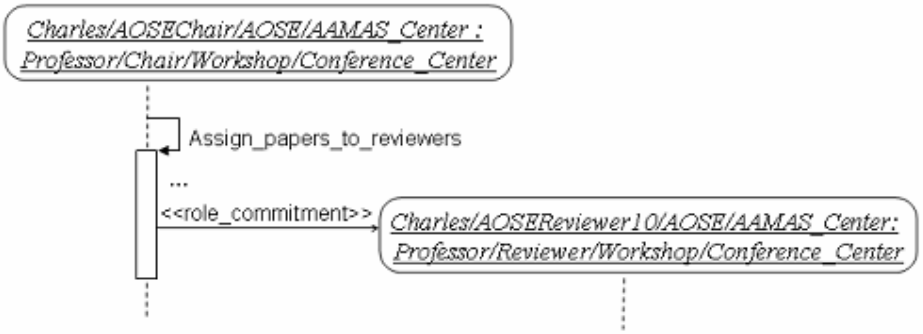


Figura 2.5.13: Agente comprometendo-se com um papel (Diagrama de Sequência) (SILVA, 2008).

2.5.8.2 Diagrama de Atividade

De acordo com Silva (2008), MAS-ML estende o diagrama de atividade da UML 2.1 para modelar os planos e ações de agentes e organizações. Uma vez que a definição de planos e atividades são similares, diagramas de atividade podem ser usados para modelar planos. Planos são compostos de ações e definem a ordem na qual eles serão executados. Silva (2008) propõe descrever uma ação usando uma notação independente de domínio.

A figura a seguir modela o plano “Assign papers to reviewers”. As ações que compõem tal plano são: relacionar artigos a avaliadores; criar proposta de revisão; e atribuir artigos sem 3 avaliadores ao chair. O plano “Review paper” está sendo modelado como um plano invocado pelo “Assign papers to reviewers”. O plano é invocado quando o prazo para decidir os avaliadores dos artigos expirou e há artigos sem avaliadores.

Os objetivos de um agente são atingidos por meio da execução de um plano. Enquanto tenta atingir um objetivo, o agente seleciona um dos planos disponíveis que podem ser usados para atingir o objetivo. Portanto, durante a modelagem de planos, os objetivos associados a eles devem ser identificados. Silva (2008) propõe o uso de um novo estereótipo <<goal>> para descrever o objetivo relacionado ao plano. A figura 2.5.14 relaciona o plano “Assign papers to reviewers” ao objetivo “to have papers assigned to reviewers”. As outras estratégias ou planos podem ser relacionadas ao mesmo objetivo e que não necessariamente todos os estados finais do diagrama de atividade que representam um plano atingem o objetivo.

Na UML entidades de diagramas de atividade podem se comunicar pelo envio de sinais (modelados pelo uso da metaclasse SendSignalAction) e pelo recebimento de eventos (modelados pelo uso da metaclasse AcceptEventAction). Silva (2008) definiu o estereótipo <<message>> baseado nestas metaclasses de maneira a usá-las para modelar o envio e recebimento de mensagens pelos agentes. Esta autora também propôs o uso da FIPA ACL (FIPA, 2004) para prover detalhes sobre as mensagens. Alguns dos parâmetros definidos na ACL podem ser usados para descrever em mais detalhes as mensagens. A figura 2.5.14 ilustra as duas representações de mensagens propostas: uma identificação de mensagens pelo uso do novo estereótipo (a mensagem “Asking for new reviewers”) e uma descrição de mensagem usando alguns dos parâmetros definidos pela ACL (a mensagem “Proposals”).

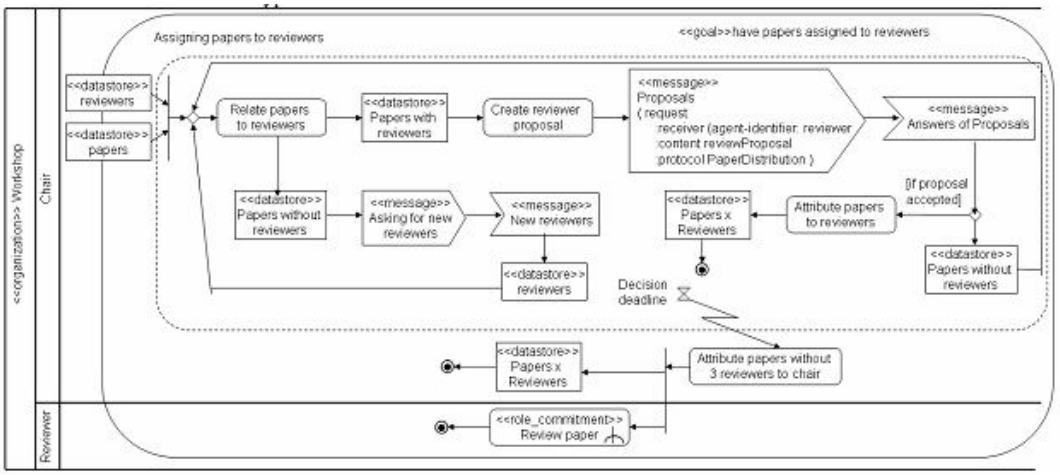


Figura 2.5.14: Diagrama de Atividade (SILVA, 2008).

Enquanto estão em execução, os agentes estão sempre interpretando no mínimo um papel. Se a execução de um plano sempre ocorrer enquanto o agente está interpretando o mesmo papel, é possível associar o plano ao papel pelo uso do estereótipo <<role>>. No entanto, um agente pode mudar seu papel ou se comprometer com outro papel durante a execução do plano. Um agente pode se comprometer com um novo papel, pode cancelar um de seus papéis, pode temporariamente parar de executar um papel ou pode ativar um papel que tinha parado temporariamente de interpretar.

Para representar estas situações foram propostos o uso de cinco estereótipos: o estereótipo <<role_commitment>>, para modelar um agente comprometendo-se com um novo papel; o estereótipo <<role_cancel>>, para modelar um agente cancelando um papel; o estereótipo <<role_deactivate>>, para modelar um agente que pára temporariamente de executar um papel; o estereótipo <<role_activate>> para modelar um agente ativando um papel; e o estereótipo <<role_change>> para modelar um agente parando de executar um papel e se comprometendo a executar outro papel.

Nas situações onde não é possível associar um único papel com o plano todo é necessário relacionar cada ação do plano ao papel que está sendo executado. Silva (2008) propõe o uso de partições de atividades (OMG 2007b) que podem agrupar as ações pelos papéis que serão executados.

A figura 2.5.14 ilustra o compromisso com um papel. Se o prazo para escolher os avaliadores dos artigos tiver expirado, o agente que executa o papel chair compromete-se com o papel de avaliador para avaliar artigos. A partir deste momento o agente executa concorrentemente ambos os papéis: chair e reviewer.

No exemplo apresentado na figura 2.5.14, o plano sendo modelado é executado no contexto de organizações de Workshop. Os chairs dos workshops podem executar o plano “Assign papers to reviewers” para atingir seus objetivos. Silva (2008) usa o estereótipo <<organization>> para identificar a partição que define a organização onde o plano será executado.

Neste exemplo, o agente se compromete a executar o papel de avaliador na mesma organização onde ele estava executando o papel chair. No entanto, pode acontecer do agente comprometer-se a executar um papel em outra organização, cancelar um papel sendo executado em outra organização ou mesmo ativar ou desativar um papel em outra organização. A figura 2.5.15 ilustra o uso de partições para modelar um agente movendo-se de uma organização para executar um papel em outra organização.

Alguns agentes tem a habilidade de se mover de um ambiente para outro. De forma a modelar agentes móveis é necessário representar os ambientes onde os agentes estão executando. O diagrama de atividade usa partições para delimitar ambientes e identifica cada partição pelo uso do estereótipo <<environment>>.

Uma vez que agentes não podem estar em dois ambientes ao mesmo tempo, agentes precisam parar de executar papéis nas organizações do ambiente onde se encontravam e iniciar a executar papéis nas organizações dos ambientes de destino. A figura 2.5.15 ilustra a habilidade de um agente que é chair de um workshop no (ambiente) AAMAS_Center tem de mover-se para o ambiente CAiSE_Center e tornar-se avaliador de um tutorial.

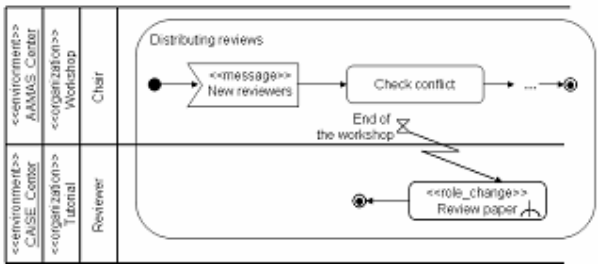


Figura 2.5.15: Movendo de Organização e Ambiente (SILVA, 2008).

2.5.9 Considerações Finais

MAS-ML parece ser, dentre as linguagens estudadas, a mais completa. Como já foi descrito, possui diagramas estruturais e comportamentais por meio dos quais é possível representar agentes, seus ambientes e os recursos que os agentes podem manipular, bem como suas crenças, objetivos e planos, além das possíveis organizações do sistema e os papéis que podem ser interpretados pelos agentes. Contudo, alguns destes conceitos não possuem uma notação própria, como os objetivos que são modelados dentro de classes Agent ou AgentRole que são representados como uma espécie de atributo estereotipado, não possuindo um símbolo próprio.

A linguagem também permite identificar a troca de mensagens entre os agentes ou seus papéis por meio de diagramas de sequência estendidos, bem como detalhar planos por meio de diagramas de atividade.

No entanto, a linguagem MAS-ML, como as outras abordagens já relatadas, não se preocupa com a questão de modelagem de requisitos, não possuindo mecanismos para representar requisitos funcionais. Além disso, nenhuma tentativa foi feita para utilizar, adaptar ou estender o Diagrama de Casos de Uso para este fim.

2.6 Metamodelo de Agência proposto por Silva (2007)

Segundo Silva (2007), o método Tropos (Bresciani, 2004), um método AOSE, baseia-se na notação i* para descrever tanto requisitos como projeto arquitetural, porém esta notação não é adequada como linguagem de descrição arquitetural, uma vez que possui limitações para expressar algumas informações requeridas para projetar arquiteturas de sistemas multiagentes, como portas, conectores, protocolos e interfaces. Dessa forma, Silva (2007) estendeu o metamodelo UML para suportar conceitos de agência, dividindo-o nas categorias intencional e de interação, conforme demonstrado nas figuras 2.6.1 e 2.6.2.

De acordo com Silva (2007), cada agente pode interpretar um ou mais papéis de agente (AgentRoles) e um agente que execute um papel de agente tem a intenção de

atingir os objetivos do papel. Quando um agente compromete-se com um papel, compromete-se a atingir objetivos e executar planos associados ao papel em questão.

Para representar este compromentimento, Silva (2007) criou a metaclasse Intention que possui a propriedade commit, que possui por sua vez duas estratégias de comprometimento: single-minded, onde um agente pode abandonar compromissos quando ele acredita que eles não podem mais ser atingidos, não importando as mudanças em seus objetivos; e open-minded, onde um agente pode abandonar compromissos quando ele acredita que eles não possam mais ser atingidos ou quando os objetivos não são mais desejados. Depois de se comprometer com um objetivo e um plano associado, um agente inicia a realização do plano. Os conceitos de Right e Intention são estensões da metaclasse AssociationClass da UML, uma vez que eles representam informações que aparecem somente quando há um relacionamento entre dois outros elementos e esta informação necessita ser encapsulada em uma classe.

Silva (2007) criou também metaclasses para representar objetivos (Goal), planos (Plan) e objetivos-soft (softgoal, um conceito de Tropos que se refere a um objetivo nebuloso (Bresciani, 2004)) que representam condições ou estado de coisas no mundo que o Agente gostaria de atingir, mas diferente do conceito de objetivo, há níveis de atingibilidade.

Estes conceitos estendem a metaclasse Class, uma vez que eles representam informações que precisam ser encapsuladas em uma classe. Planos podem contribuir positiva (+ ou ++) ou negativamente (- ou --) ou não ter influência na satisfação de um objetivo-soft, o grau de influência de um plano sobre a atingibilidade de um objetivo-soft é chamado contribuição. O conceito de contribuição é uma extensão da metaclasse AssociationClass, uma vez que ela representa informações que aparecem no relacionamento entre um plano e um objetivo-soft e, portanto, precisam ser encapsuladas em uma classe. A contribuição pode ser de cinco tipos, plusplus, plus, neutral, minunsminus and minus, definidas na classe de enumeração ContribType.

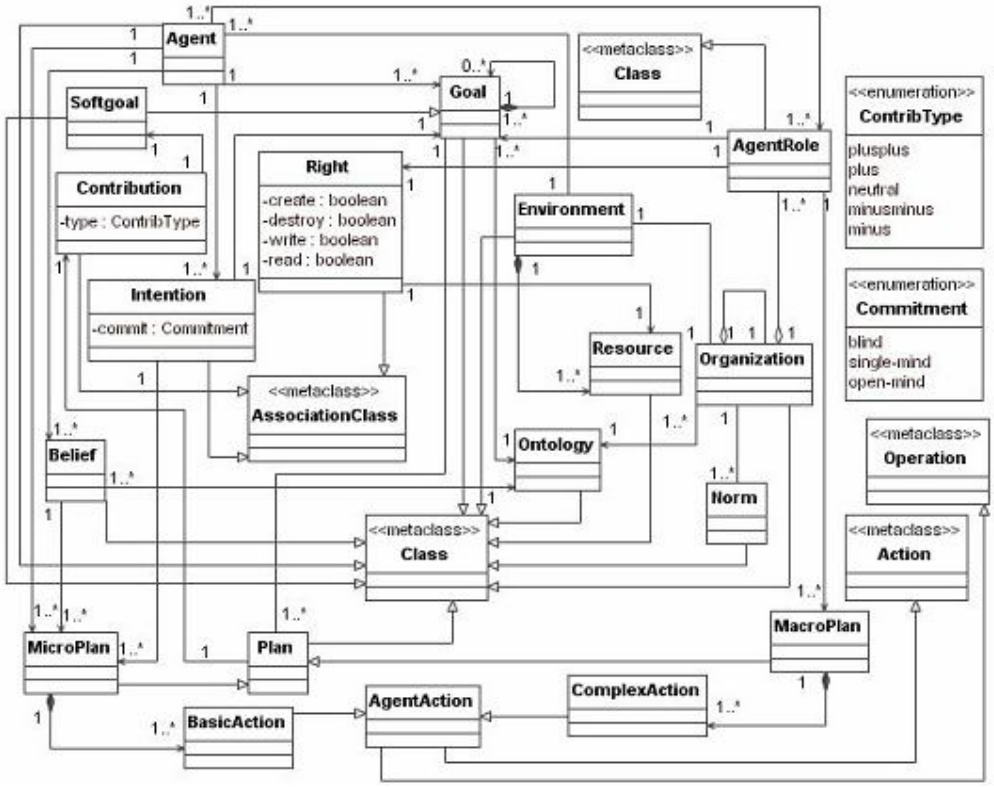


Figura 2.6.1: Metamodelo de Agência Refletindo Conceitos Intencionais (SILVA, 2007).

Uma ação de agente (AgentAction) determina os passos para realizar um plano e estende as classes Action e Operation da UML. A ação de agente (AgentAction) tem duas sub-classes: a ComplexAction que pode ser novamente refinada e a BasicAction que não pode ser decomposta.

Um plano tem duas sub-classes, MacroPlan se o plano é definido pelo papel do agente e um MicroPlan se o plano é definido pelo Agente. A diferença entre MacroPlan e MicroPlan é que o primeiro é um plano parcial e, portanto, composto de ações complexas e o último é o plano final completo, composto de ações básicas. Crenças representam as informações que o Agente tem sobre si mesmo e seu ambiente. Estas são pré-condições para executar seus planos e ordenar as ações dos microplanos. Estes conceitos estendem a metaclasse Class, já que eles representam informações que precisam ser encapsuladas em uma classe. A descrição tanto das crenças como objetivos devem concordar com a ontologia usada na organização, ou seja, o vocabulário de conceitos que pertencem ao domínio do sistema. Silva (2007) definiu a ontologia de sistemas multiagentes estendendo a metaclasse Class.

Segundo Silva (2007), um Dependum define um consentimento de fornecimento de serviço entre dois papéis de agente que executam os papéis de Depender e Dependee. Quando um papel de agente depende de outro para atingir um objetivo-soft está implícito que o papel de agente confia no outro e delega a ele tais atividades, assim a metaclasse Dependum tem a propriedade trusty, que pode ser de três tipos: full, medium e none, como definido na classe de enumeração TrustKind. Na figura 2.6.2 são apresentados os conceitos de interação do metamodelo proposto por Silva (2007).

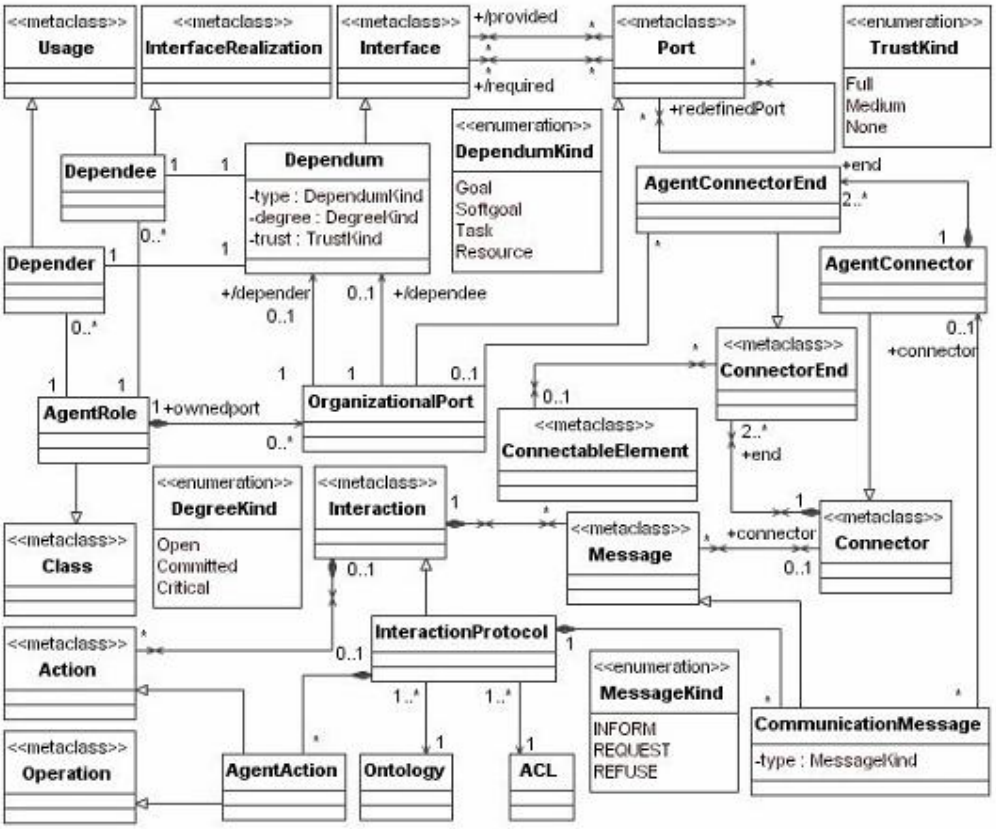


Figura 2.6.2: Metamodelo de Agência Refletindo Conceitos de Interação (SILVA, 2007).

Papéis de Agente precisam trocar sinais através de um conector de agente (AgentConnector) para concluir o acordo contratual de serviço fornecido entre os papéis de agente. Uma porta organizacional (OrganizationalPort) especifica um ponto de interação distinto entre o papel de agente e seu ambiente. Um Fim de Conector de Agente (AgentConnectorEnd) é um ponto de fim de um conector de agente, o qual liga o conector de agente para uma porta organizacional. Cada papel de agente pode interagir com outros papéis de agente de acordo com um protocolo de interação (InteractionProtocol).

Um protocolo de interação descreve uma seqüência de mensagem de comunicações (CommunicationMessages) que podem ser enviadas ou recebidas pelos agentes executando certos papéis de agente através da execução de suas ações de agente. A linguagem de comunicação de agente (ACL) define o formato de cada mensagem de comunicação que compõe o protocolo de interação (InteractionProtocol). O conceito de ACL estende a metaclasse class, enquanto o conceito de protocolo de interação estende a metaclasse Interaction. O conceito de mensagem de comunicação estende a metaclasse Message e pode ser de diversos tipos como REQUEST, INFORM e REFUSE, entre outros definidos pela FIPA (FIPA 2004).

2.6.1 Perfil de Agência

Silva (2007) criou um perfil UML para adaptá-la ao domínio de agência. A figura a seguir apresenta algumas extensões feitas para o metamodelo de agência definido nas figuras anteriores, uma extensão é usada para indicar que as propriedades de uma metaclasse são estendidas através de um estereótipo.

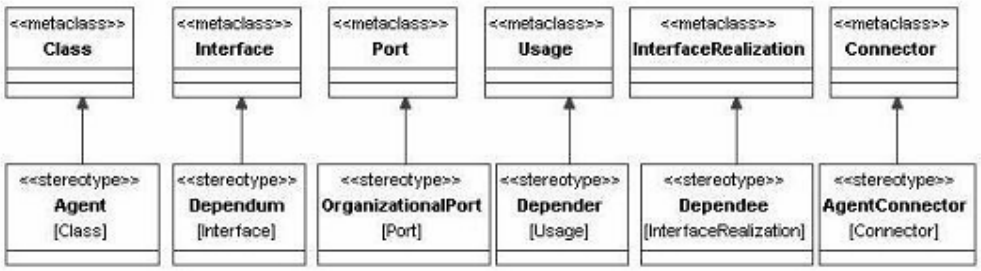


Figura 2.6.3: Estereótipos de Agência (SILVA, 2007).

Baseada no perfil criado, Silva (2007) propôs seis visões de projeto de sistemas multiagentes: arquitetural, comunicação, ambiental, intencional, racional e de ação.

2.6.2 Diagrama Arquitetural (Architectural Diagram)

Este diagrama estende o diagrama de classe da UML (OMG 2007b). O diagrama arquitetural é definido em termos de papéis de agente (AgentRoles) que possuem objetivos atingíveis por planos. Um papel de agente possui portas organizacionais (OrganizationalPorts) que permitem a troca de mensagens com outros agentes através de conectores de agente (AgentConnectors) para realizar algum Dependum.

A figura seguinte apresenta o papel de agente Provider, responsável por realizar o serviço definido no Dependum. Este papel de agente visa realizar o objetivo ServicePerformed executando o MacroPlan PerformPlan, que por seu turno consiste em realizar a ação complexa (ComplexAction) service().

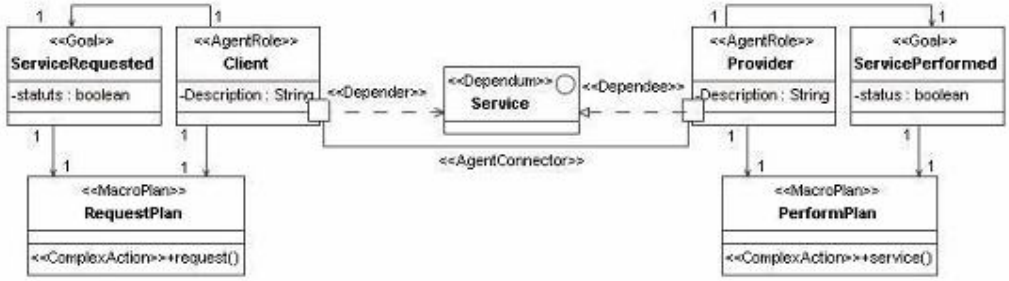


Figura 2.6.4: Diagrama Arquitetural (SILVA, 2007).

O papel de agente Client visa realizar o objetivo ServiceRequest executando o MacroPlan RequestPlan, que por sua vez, consiste em realizar a ação complexa (ComplexAction) request(). Portanto, o papel de agente Client é responsável por requerer o serviço definido no Dependum. Tanto a mensagem para requerer a execução do serviço como a mensagem para confirmar se o serviço foi concluído com sucesso são enviadas através do conector de agente.

2.6.3 Diagrama de Comunicação (Communication Diagram)

Este diagrama estende o diagrama de seqüência. O diagrama de comunicação é definido em termos de agentes executando determinados papéis de agente e mensagens trocadas entre eles para executar um serviço. A figura a seguir mostra uma interação envolvendo agentes que executam os papéis de agente Client e Provider. A interação especificada no diagrama de comunicação é assíncrona.

Figura 2.6.5: Diagrama de Comunicação (SILVA, 2007).

Nesta figura o agente que interpreta o papel de cliente solicita ao agente que interpreta o papel de fornecedor que realize um serviço por meio de uma mensagem FIPA e, após concluir o serviço, o agente que interpreta o papel de fornecedor informa que o serviço foi realizado ao agente que interpreta o papel de cliente.

2.6.4 Diagrama Ambiental (Environmental Diagram)

Estende o diagrama de classes para modelar os papéis de agente que compõem uma organização situada em um ambiente. Este ambiente é composto de recursos, que são acessados pelos papéis de agente de acordo com seus direitos de forma a realizar suas responsabilidades. A figura seguinte apresenta o papel de agente Provider que faz parte da organização Org que está situada no ambiente Env.

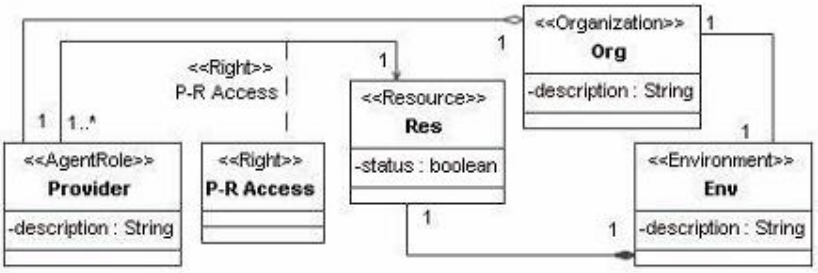


Figura 2.6.6: Diagrama Ambiental (SILVA, 2007).

O papel de agente Provider precisa acessar o recurso Res disponível no ambiente Env para realizar suas responsabilidades. O papel de Agente Provider pode somente ler o recurso Res de acordo como seu direito de acesso leia (read) Provider-Res.

2.6.5 Diagrama Intencional (Intentional Diagram)

Estende o diagrama de classes (OMG, 2007b) para modelar papéis de agente, agentes, suas crenças, objetivos, planos, bem como as normas e a ontologia usada na organização. A figura seguinte mostra o papel de agente Provider compondo a organização Org que deve cumprir com a norma OrganizationalNorm. O agente AgentY que executa o papel de agente Provider acredita que uma mensagem de requisição foi recebida, descrita como a crença (belief), RequestReceived.

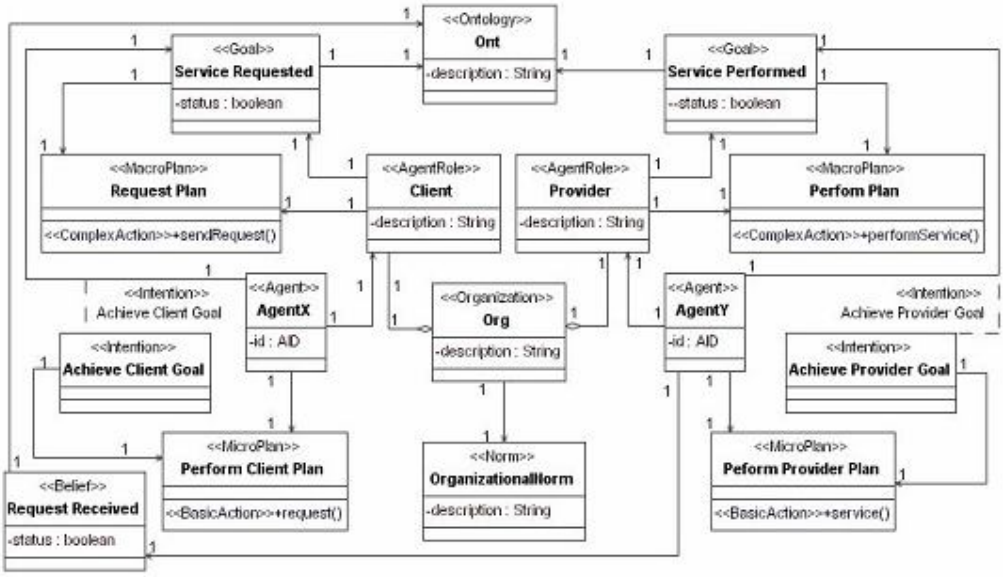


Figura 2.6.7: Diagrama Intencional (SILVA, 2007).

2.6.6 Diagrama Racional (Rational Diagram)

Estende o diagrama de classe para modelar objetivos-soft, a contribuição de planos para satisfazer estes objetivos-soft, objetivos, planos para atingir estes objetivos e papéis de agente. A figura a seguir apresenta um exemplo deste diagrama, onde é mostrado o objetivo-soft "Fast Service Supply", em que um agente executando o papel de agente Client é responsável por satisfazê-lo. O papel de agente Client possui o macroPlan Request Plan para realizar o objetivo (goal) Service Requested. Este plano tem alguma influência na satisfação do objetivo-soft (softgoal) Fast Service Supply.

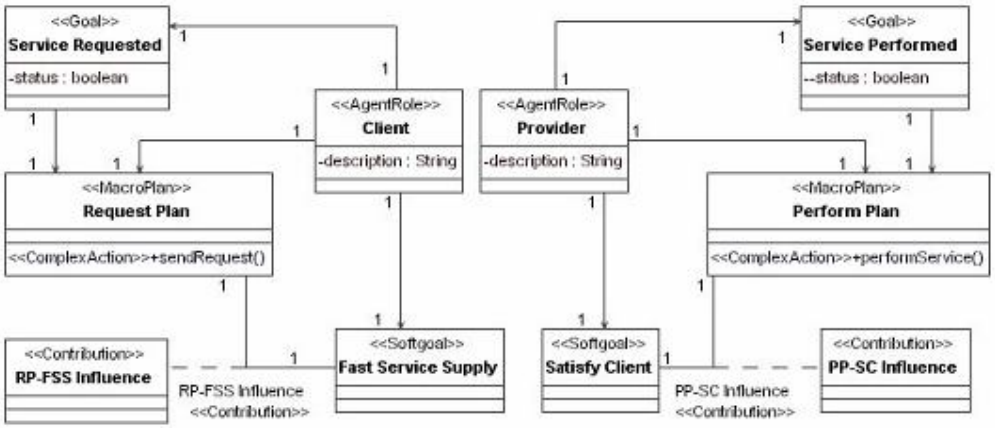


Figura 2.6.8: Diagrama Racional (SILVA, 2007).

2.6.7 Diagrama de Plano (Plan Diagram)

Estende o diagrama de atividade (OMG, 2007b) para modelar a seqüência de ações que compõem os planos para atingir um objetivo. Através deste diagrama pode-se ver quais ações podem ser realizadas em paralelo e quais ações são dependentes sob outras ações. Este diagrama pode ser usado para modelar a seqüência de ações em macroplanos ou microplanos.

2.6.8 Considerações Finais

O metamodelo proposto por Silva (2007) abrange vários aspectos da modelagem de sistemas multi-agentes, permitindo a representação de papéis, organizações, ambientes, crenças, objetivos e planos, bem como a comunicação entre agentes. No entanto a representação de percepções e ações é um pouco limitada, restringindo-se a representação de envio e recebimento de mensagens entre os papéis de agente.

Embora possa ser usado independentemente, o metamodelo de Silva (2007) é uma forma de mapeamento da notação utilizada pela metodologia Tropos para a notação UML. Além disso, algumas metaclasses foram derivadas diretamente a partir da metaclasse Class e não da metaclasse mais genérica Classifier, como por exemplo, as metaclasses Goal e AgentRole, uma vez que não é possível retirar quaisquer restrições de uma metaclasse quando esta é especializada, temos algumas dúvidas se esta foi a escolha mais apropriada.

Finalmente, apesar deste metamodelo suportar diversos diagramas para a modelagem de sistemas multi-agentes, nenhum dos diagramas propostos enfoca a questão de representação de requisitos, funcionais ou não, e nenhuma tentativa foi feita para utilizar ou adaptar o Diagrama de Casos de Uso para este tipo de modelagem.

2.7 Metodologias AOSE – Agent Oriented Software Engineering – Engenharia de Software Orientada a Agente

Neste capítulo falaremos brevemente sobre algumas das principais metodologias AOSE existentes atualmente e discutiremos sucintamente como elas enfocam a modelagem de requisitos em sistemas multi-agentes. Cumpre destacar novamente que nosso enfoque está concentrado nas linguagens derivadas a partir da UML para a modelagem de sistemas multi-agentes e que o metamodelo proposto nesse trabalho não se trata de uma metodologia e sim uma estensão da linguagem UML para a modelagem de requisitos para este tipo específico de software e, como tal, pode ser aplicado por qualquer metodologia que assim o desejar.

2.7.1 Gaia

Zambonelli (2003) define os objetivos e limitações da metodologia Gaia:

- Gaia não lida diretamente com técnicas de modelagem particulares. Ela propõe, mas não se compromete com, técnicas específicas para modelagem como por exemplo, papéis, ambiente, interações.
- Gaia não lida diretamente com questões de implementação. O resultado do processo Gaia é uma especificação detalhada, mas de tecnologia neutra.
- Gaia não lida explicitamente com as atividades da captura e modelagem de requisitos, especialmente com a engenharia de requisitos iniciais. A suposição básica em Gaia é que a fase de análise pode contar com a saída de uma fase de engenharia de requisitos inicial, que analisa as características e os objetivos do sistema, na medida em que eles emergem das necessidades dos usuários e do ambiente operacional específico.

A metodologia GAIA produz artefatos basicamente textuais, a metodologia não possui uma notação gráfica particular, embora recomende o uso da AUML para preencher essa lacuna. Esta metodologia demonstra pouca preocupação com a fase de

elicitação e análise de requisitos, que é o principal enfoque de nosso trabalho e tampouco aplica o diagrama de casos de uso para esse fim.

2.7.2 MaSE – Multiagent Systems Engineering – Engenharia de Sistemas Multiagentes

Para o propósito de DeLoach (2001), proponente deste método, agentes são simples processos de software que interagem uns com os outros a fim de atender a um objetivo geral do sistema. Segundo DeLoach (2001), freqüentemente surge o caso em que agentes múltiplos, mas não-complexos, possam interagir de tal forma que o sistema inteiro venha a exibir um comportamento aparentemente inteligente, assim DeLoach (2001), encara os agentes meramente como abstrações convenientes, que podem ou não possuir inteligência, manejando deste modo igualmente os componente do sistemas, tanto os inteligentes como os não-inteligentes, dentro do mesmo arcabouço (framework) conceitual de trabalho.

DeLoach (2001), procura responder ao sexto desafio identificado por Sycara (1998), ou seja, como criar a engenharia para construir sistemas multi-agentes práticos e igualmente como criar um arcabouço (framework) sobre o qual se possam descobrir soluções para os primeiros cinco desafios de Sycara (1998), empregando as abstrações fornecidas pelos sistemas multi-agentes com o fito de desenvolver sistemas inteligentes de software distribuído. A fim de atingir este objetivo, MaSE emprega um certo número de modelos graficamente embasados a fim de descrever os tipos de agentes encontrados em um sistema e suas interfaces com outros agentes, do mesmo modo que uma definição independente de arquitetura sobre o desenho interno de tais agentes. A figura 2.7.1 apresenta as etapas da metodologia MaSE.

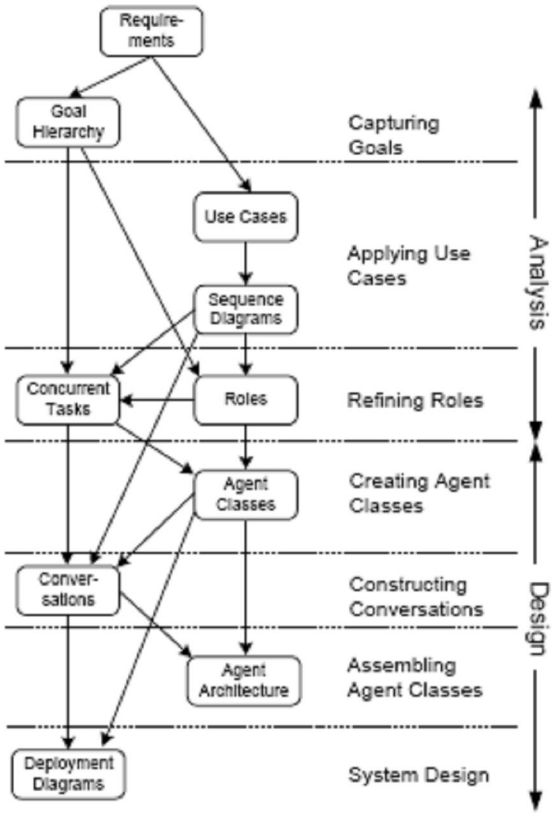


Figura 2.7.1 – Etapas da Metodologia MaSE (DELOACH, 2001)

Segundo DeLoach (200l) e DeLoach(2006) a metodologia MaSE define os sistemas multi-agentes em termos de classes de agentes e de sua organização. A organização é definida em termos de quais agentes podem se comunicar usando conversação. Há duas fases básicas em MaSE, análise e projeto. A primeira fase inclui três etapas: captura de objetivos, aplicação de casos de uso e refinamento de papéis.

A primeira etapa, Captura de Objetivos, toma os requisitos do usuário e os transforma em objetivos de alto nível do sistema. Depois de definir os objetivos a nível de sistema, são extraídos casos de uso também a nível de sistema e definidos Mapas Seqüenciais na etapa da Aplicação de Casos de Uso. Esta etapa define um conjunto inicial de funções de sistema e caminhos de comunicação. Empregando os objetivos e as funções do sistema identificadas por meio dos casos de uso, o conjunto inicial de papéis é refinado e estendido e definidas as tarefas de modo a realizar cada objetivo na etapa do Refinamento de papéis.

Na fase de Projeto, DeLoach (2001), transforma os modelos da análise em construtos que se demonstrem úteis para a implementação real do sistema multi-agente. A fase de Projeto apresenta quatro etapas: criação de classes de agentes, construção de conversações, montagem de classes de agentes e projeto do sistema. Na primeira etapa, a criação de classes de agentes, são definidas classes específicas de agentes a fim de preencher as funções definidas na fase da Análise. Então, depois de determinado o número e os tipos de classes de agentes do sistema, pode-se ou construir conversações entre estas classes de agentes ou definir os componentes internos que são abrangidos pelas classes de agentes. O analista pode desempenhar estas tarefas em paralelo, durante as etapas de construção de conversações e de montagem de classes de agentes. Assim que se tiver definido a estrutura completa do sistema, passa-se a determinar como o sistema deve ser distribuído. Durante esta etapa, o projetista define o número de agentes individuais, suas localizações e outros itens específicos do sistema.

2.7.2.1 Captura de Objetivos

Segundo DeLoach (2001), a primeira fase da metodologia MaSE consiste em Capturar os Objetivos, tomando a especificação inicial do sistema e a transformando em um conjunto estruturado de objetivos para o dito sistema, descrito como um Diagrama de Hierarquia de Objetivos, conforme apresentado na figura 2.7.2. Um objetivo MaSE é sempre definido como um objetivo a nível de sistema. Construtos de nível inferior podem herdar ou ser responsáveis por objetivos, mas todas as metas sempre apresentam um contexto a nível de sistema.

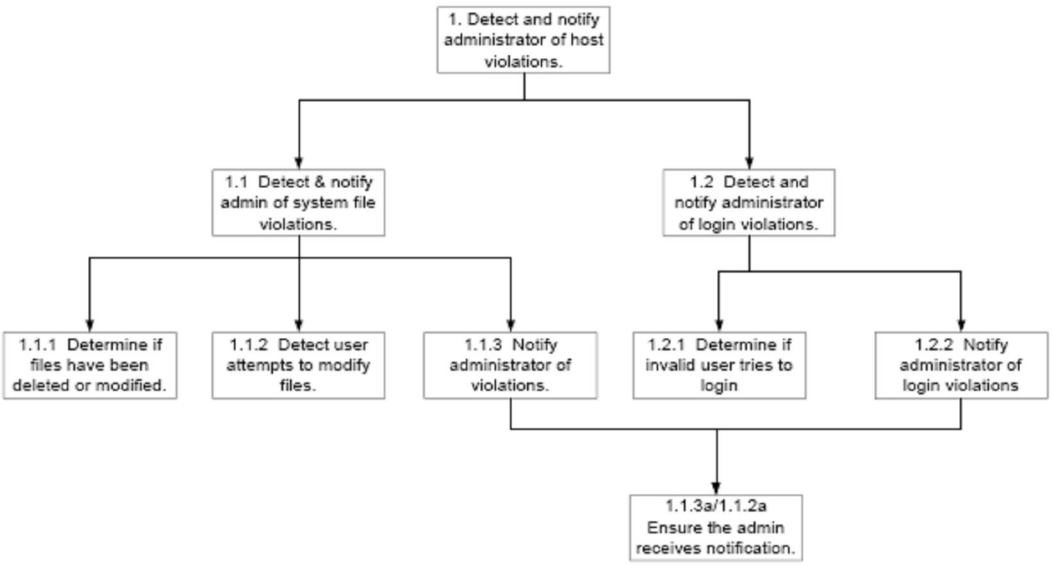


Figura 2.7.2 — Diagrama de Hierarquia de Objetivos (DELOACH, 2001)

Há duas etapas na Captura de Objetivos: identificação dos objetivos e sua estruturação. Um analista pode identificar os objetivos destilando a essência do conjunto de requisitos. Estes requisitos podem incluir documentos técnicos detalhados, históricos de usuários ou especificações formais. A seguir, o analista realiza a análise e estrutura os objetivos identificados em um Diagrama de Hierarquia de Objetivos. Em um Diagrama de Hierarquia de Objetivos, o analista organiza os objetivos de acordo com sua importância. Cada nível da hierarquia deverá conter objetivos aproximadamente iguais em seus âmbitos. O analista também identifica sub-objetivos necessários para satisfazer os objetivos geradores. Eventualmente, o analista associa cada objetivo com um papel e um conjunto de classes de agentes responsáveis para satisfação desse objetivo.

2.7.2.2 Aplicação de Casos de Uso

A etapa da Aplicação de Casos de Uso constitui um passo crucial na tradução dos objetivos em papéis e tarefas a eles associadas. O analista retira casos de uso a partir dos requisitos do sistema e de seus usuários. Casos de Uso são descrições narrativas de uma seqüência de eventos que define um comportamento desejado para o sistema. São exemplos de como o sistema se deverá comportar em uma determinada situação. A fim de ajudar na determinação das comunicações reais requeridas dentro de um sistema multi-agente, o analista re-estrutura os casos de usos em Diagramas de Seqüencia, conforme mostrado na figura 2.7.3. Um Diagrama de Seqüencia descreve uma seqüência de eventos entre papéis múltiplos e, como resultado, define a comunicação mínima que deverá ocorrer entre os papéis. Os papéis identificados nesta etapa formam o conjunto inicial de regras empregado para definir integralmente os papéis do sistema na próxima etapa, em que o analista utilizará os eventos aqui identificados para ajudar a definir tarefas e, eventualmente, conversações.

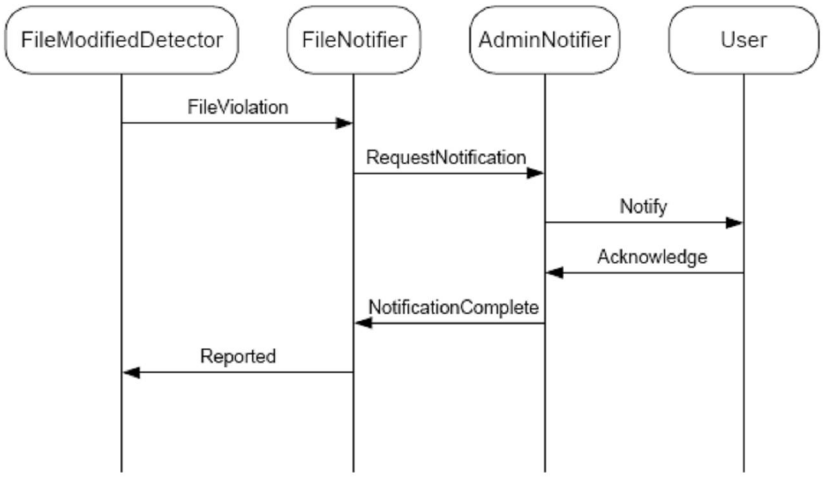


Figura 2.7.3 — Diagrama de Seqüência (DELOACH, 2001)

2.7.2.3 Refinamento de Papéis

A terceira etapa do MaSE é a de garantir que foram identificados todos os papéis necessários e desenvolver as tarefas que definem o comportamento dos papéis e os padrões de comunicação. Os papéis são identificados a partir dos Diagramas de Seqüência desenvolvidos durante a etapa da Aplicação dos Casos de Uso, do mesmo modo que a partir dos objetivos do sistema definidos durante a etapa da Captura de Objetivos.

A associação de um papel específico a cada objetivo, que é eventualmente executada no projeto final por, no mínimo, um dos agentes garante que todos os objetivos do sistema estejam satisfeitos. Cada objetivo é em geral mapeado para um único papel. Todavia, existem situações em que é útil combinar objetivos múltiplos em um único papel para conveniência ou eficiência. DeLoach (2001) baseia estas decisões em conceitos padronizados de engenharia de software tais como coesões funcional, comunicativa, procedural ou temporal. Outros fatores incluem a distribuição natural de recursos ou problemas especiais de interface. Os papéis são capturados em um Modelo de Papéis tal como mostrado na 2.7.4.

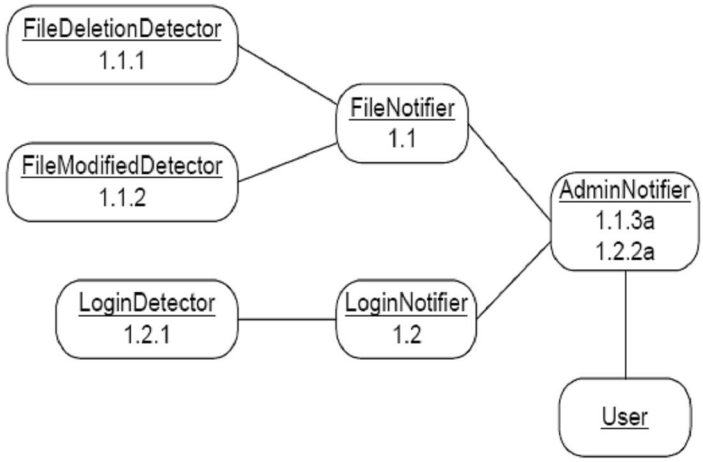


Figura 2.7.4 – Modelo de Papéis (DELOACH, 2001)

Assim que os papéis tiverem sido definidos, criam-se tarefas. Segundo DeLoach (2001), a parte mais interessante e também a mais difícil da metodologia MaSE é a

transformação dos papéis em classes de agentes e a definição das conversações e dos comportamentos internos dos agentes. Como auxiliar para a realização desta tarefa, é preciso ser capaz de definir quais tarefas de alto nível podem ser transformadas em funcionalidades específicas para os agentes. Esta funcionalidade ajuda a definir os componentes internos dos agentes, do mesmo modo que os detalhes das conversações de que participam tais agentes. A Figura 2.7.5 mostra uma versão detalhada do modelo de papéis MaSE. As ovais sob cada papel denotam tarefas que a função deve executar a fim de realizar suas metas. As linhas entre os nodos indicam protocolos entre as tarefas. Estes protocolos definem uma série de mensagens entre as tarefas que lhes permitem trabalhar cooperativamente. As flechas nas linhas protocolares apontam da iniciadora do protocolo para a tarefa respondente.

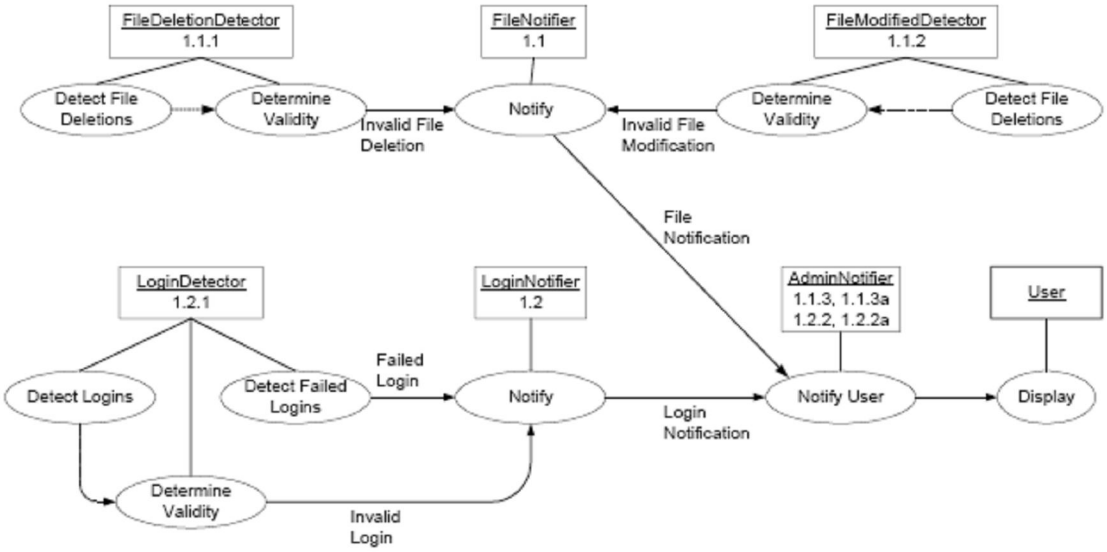


Figura 2.7.5 – Modelo de Papéis MaSE (DELOACH, 2001)

DeLoach (2001) define estas tarefas concorrentes como um autômato de estado finito que especifica as mensagens enviadas entre os papéis e tarefas. As tarefas concorrentes também permitem especificar o processamento interno via atividades nos estados. Através do uso de tarefas concorrentes pode-se definir protocolos de interação complexos e de alto nível que requeiram coordenação entre agentes múltiplos. Um exemplo de Diagrama MaSE para Tarefas Concorrentes, que define a tarefa Notificar Usuário do papel Notificador de Administrador, é mostrado na figura 2.7.8.

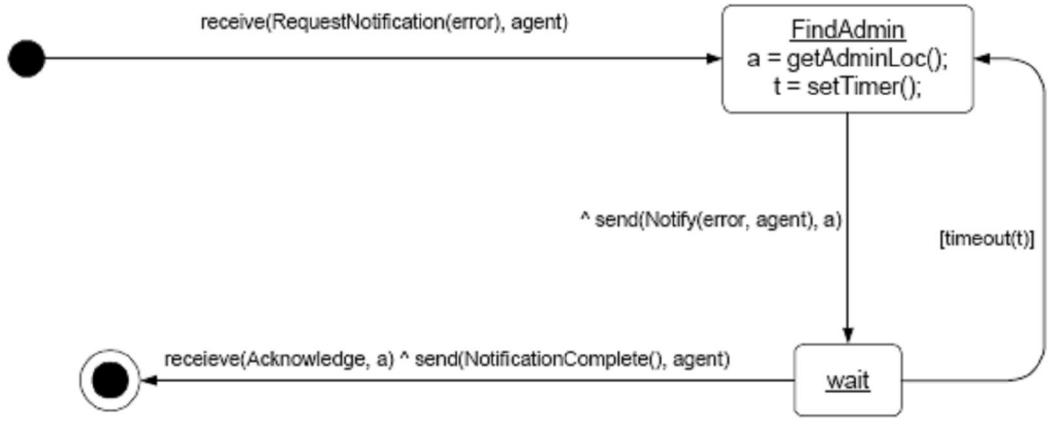


Figura 2.7..8 — Diagrama de Tarefas Concorrentes (DELOACH, 2001)

A sintaxe de uma transição segue a notação abaixo: trigger(args1) [guard] / transmission (args2). DeLoach (2001) interpreta que esta transição diz que, se um desencadeador de evento for recebido com um número de argumentos args1 e a condição de guarda se aplica, então a transmissão da mensagem é enviada com o conjunto de argumentos args2. Todos os itens são optativos. Por exemplo, uma transição com uma única condição de guarda é permitida, do mesmo modo que outra que apresente uma condição de mensagem recebida e transmissão, trigger/transmission. Eventos de transmissão múltipla também são permitidos e sua separação representada por sinais de ponto e vírgula [;]. As ações podem ser realizadas e escritas como funções matemáticas.

2.7.2.4 Criação de Classes de Agentes

Na etapa de Criação de Classes de Agentes, as classes de agentes são definidas a partir dos papéis e documentadas em um Diagrama de Classes de Agentes, como mostrado na Figura 2.7.9. Os Diagramas de Classes de Agentes descrevem as classes de agentes como retângulos e as conversações entre elas por meio de linhas interligando as classes de agentes.

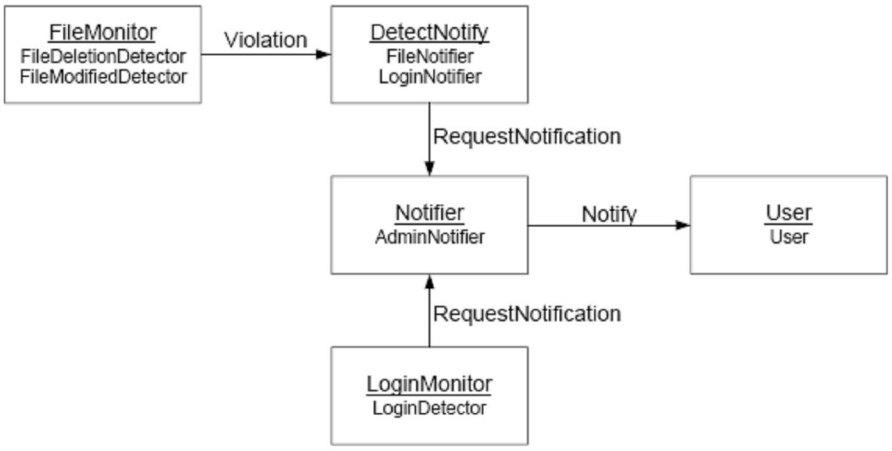


Figura 2.7.9 – Diagrama de Classes de Agentes (DELOACH, 2001)

Do mesmo modo que transcorre com os objetivos e papéis, geralmente define-se, além das classes de agentes, um mapeamento de um-para-um entre os papéis que são listadas sob o nome de classe de agente. Todavia, o projetista pode combinar papéis múltiplos em uma única classe de agentes ou mapear uma única função para ser exercida por classes de agentes múltiplos.

Uma vez que os agentes herdam os caminhos de comunicação entre os papéis, quaisquer caminhos demarcados entre dois papéis se transformam em conversações entre suas respectivas classes. Deste modo, à medida em que o projetista atribui funções a classes de agentes, a organização geral do sistema vai sendo definida. A fim de tornar sua organização mais eficiente, freqüentemente é desejável combinar dois papéis que compartilham de um alto volume de tráfego de mensagens. Ao se determinar quais os papéis a combinar, conceitos como coesão e o volume do tráfego de mensagens se tornam considerações importantes.

2.7.2.5 Construção de Conversações

O projetista pode realizar em paralelo as duas etapas seguintes, Construção de Conversações e Montagem de Agentes. As duas etapas estão intimamente ligadas, uma vez que a arquitetura dos agentes definida em Montagem de Agentes deve implementar as conversações e os métodos definidos em Construção de Conversações. Uma conversação MaSE define um protocolo de coordenação entre dois agentes. Especificamente, uma conversação consiste em dois Diagramas de Classes de Comunicação, um para a iniciadora e outro para a responsiva. Um Diagrama de Classes de Comunicação é um par de máquinas de estado finito que define uma conversação entre duas classes de agentes dela participantes. Um lado da conversação é mostrado na Figura 2.7.10. A iniciadora sempre começa a conversação enviando a primeira mensagem. A sintaxe para os Diagramas de Classes de Comunicação é muito semelhante à dos Diagramas de Tarefas Concorrentes. A diferença principal entre conversações e tarefas concorrentes é que as tarefas concorrentes podem incluir conversações múltiplas entre muitas funções e tarefas diferentes, enquanto as conversações são trocas binárias entre agentes individuais.

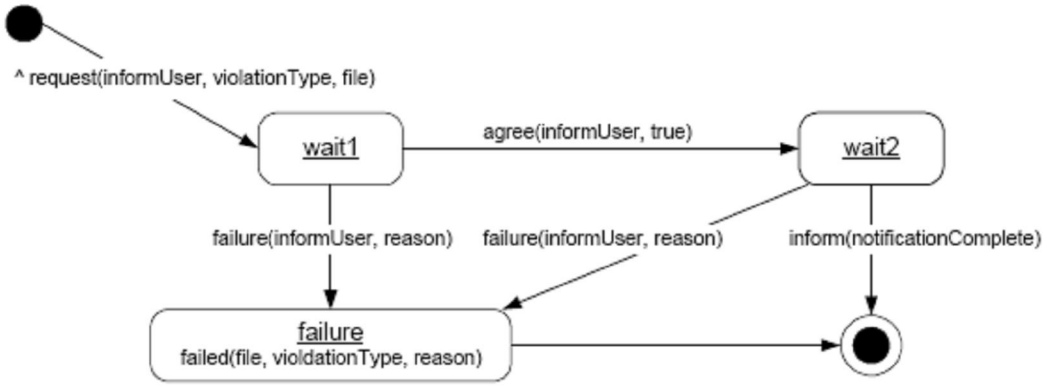


Figura 2.7.10 — Diagrama de Classes de Comunicação (DELOACH, 2001)

2.7.2.6 Montagem de Agentes

Nesta etapa de MaSE são criados os elementos internos das classes de agentes. Este processo é simplificado pelo uso de uma linguagem de modelagem arquitetônica que combina a natureza abstrata das linguagens tradicionais de descrição arquitetural com a Linguagem de Restrições do Objeto, a qual permite ao projetista especificar os detalhes de baixo nível.

2.7.2.7 Localização Dentro do Sistema

A etapa final do MaSE define a configuração do sistema real a ser implementado. Em MaSE, a arquitetura geral do sistema é definida empregando Diagramas de Implantação a fim de mostrar os números, tipos e localizações de agentes no interior de um sistema. A Localização no Sistema também é a etapa em que todas as decisões de implementação previamente indefinidas, tais como qual a linguagem de programação a ser adotada ou moldura (framework) de comunicações devem ser adotada.

2.7.2.8 Considerações Finais

O casos de uso em MaSE são aplicados em sua forma tradicional em termos de funções externas que o sistema deve oferecer. A partir dos casos de uso identificados são definidos papéis de agentes, representando funções que deverão ser desempenhadas

por agentes que os assumam. Não foi feita uma adaptação do diagrama de casos de uso para representar papéis, objetivos, percepções, ações ou planos como proposto em nosso trabalho.

2.7.3 MESSAGE/UML - Methodology for Engineering Systems of Software Agents – Metodologia para Engenharia de Sistemas de Agentes de Software

De acordo com Caire (2002), MESSAGE toma a UML como um ponto de partida e adiciona conceitos de entidade e relacionamento necessários para a modelagem orientada a agente.

A modelagem orientada a agente pega emprestado do estudo de organizações e sociedades humanas, a descrição da maneira como os agentes em um Sistema Multi-Agente trabalham juntos para atingir um propósito coletivo, e da Inteligência Artificial e psicologia cognitiva para descrever os próprios agentes. Estes conceitos adicionais podem ser definidos em termos de conceitos orientados a objeto, mas lidam com idéias e estruturas em um nível conceitual mais alto. Na IA este nível mais alto é muitas vezes referido como "o nível de conhecimento", contrastando conhecimento com dados. Essencialmente, MESSAGE usa a UML padrão como sua linguagem de modelagem de "nível de dados", mas fornece conceitos de "nível de conhecimento" adicionais.

2.7.3.1 Conceitos de Nível de Conhecimento

A maioria dos conceitos de entidade de nível de conhecimento do MESSAGE classificam-se nas categorias principais: EntidadeConcreta (ConcreteEntity), Atividade (Activity) e EntidadeEstadoMental (MentalStateEntitiy). Os principais tipos de EntidadeConcreta de acordo com Caire (2002) são:

Agent (Agente): Um Agent é uma entidade autônoma atômica que é capaz de realizar alguma função. A capacidade funcional é capturada como os serviços do agente. Um serviço é o nível de conhecimento análogo de uma operação de objeto. A qualidade de autonomia significa que as ações do agente não são somente ditadas pelos eventos externos ou interações, mas também por sua própria motivação. Caire (2002), captura esta motivação em um atributo chamado propósito (purpose). O propósito irá, por exemplo, influenciar se um agente concorda com uma requisição para realizar um serviço e também com a forma como ele fornece o serviço. SoftwareAgent (Agente de Software) e HumanAgent (Agente Humano) são especializações de Agent.

Organisation (Organização): Uma Organisation é um grupo de Agents trabalhando juntos para um propósito comum. Ela é uma entidade virtual no sentido que o sistema não tem entidades computacionais individuais correspondentes a uma organização, seus serviços são fornecidos e seus propósitos atingidos coletivamente por seus agentes constituintes. Ela tem uma estrutura expressada através de relacionamentos fortes (por exemplo, relacionamentos superior-subordinado) entre constituintes e mecanismos de comportamento/co-ordenação expressados através de interações entre constituintes.

Role (Papel): A distinção entre papel e agente, segundo Caire (2002) é análoga àquela entre interface e classe (objeto): Um papel descreve as características externas de um agente em um contexto particular. Um agente pode ser capaz de executar diversos papéis, e múltiplos agentes podem ser capazes de executar o mesmo papel (Role). Papéis podem também ser usados como referências indiretas a agentes. Isto é útil na definição de padrões re-utilizáveis.

Resource (Recurso): Recurso é usado para representar entidades não-autônomas tais como bancos de dados ou programas externos usados pelos agentes. Conceitos orientados a objeto padrão são adequados para modelar Resources.

Os principais tipos de Activity (Atividade) são (CAIRE, 2002):

Task (Tarefa): Uma tarefa é uma unidade de atividade de nível de conhecimento com um único realizador principal. Uma tarefa tem um conjunto de pares de Situações descrevendo pré e pós-condições. Se a Task é realizada quando uma pré-condição é válida, então pode-se esperar que a pós-condição associada seja válida quando a Tarefa for completada. Tarefas compostas podem ser expressas em termos de sub-tarefas ligadas causalmente, isto é, em termos de causa e efeito, umas originando outras, (que podem ter realizadores diferentes da Task geradora). Tarefas são StateMachines (Máquinas de Estado), permitindo que diagramas de atividade UML possam ser usados para mostrar dependências temporais de sub-tarefas.

Interaction (Interação) e InteractionProtocol (Protocolo de Interação): Uma interação por definição tem mais do que um participante e um propósito que os participantes devem almejar atingir coletivamente. O propósito tipicamente é atingir uma visão consistente de alguns aspectos do domínio do problema, a fim de concordar com os termos de um serviço ou para mudar para os resultados de um ou mais serviços. Um Protocolo de Interação (InteractionProtocol) define um padrão de troca de Mensagem (Message) associado com uma Interação (Interaction).

Goal (Objetivo): Um objetivo associa um agente com uma Situation (Situação). Se uma instância de objetivo está presente na memória de trabalho do agente, então o agente pretende gerar a situação referenciada pelo objetivo. Alguns objetivos são intrínsicos à identidade do agente e são derivados de seu propósito. Estes objetivos são persistentes através da vida do agente. Outros são objetivos táticos transientes. É muitas vezes útil expressar o propósito em termos de uma função de utilidade que associa “bons valores” com situações. A situação de destino do objetivo é então aquela que é estimada para maximizar utilidade (determinada dinamicamente). Caire (2002) destaca que a base de conhecimento do agente necessita incluir “regras” governando asserção e deleção de objetivos táticos.

Dois outros conceitos simples mas importantes usados em MESSAGE de acordo com Caire (2002) são:

InformationEntity (Entidade de Informação) (Um objeto que encapsula um pedaço de informação) e Message (Mensagem): Caire (2002) informa que o conceito orientado a agente de mensagem difere do conceito orientado a objetos em um grande número de considerações. Em UML, uma mensagem é uma ligação causal em uma série comportamental, indicando que uma Ação realizada por um objeto dispara uma Ação por outro objeto. Em MESSAGE, uma mensagem é um objeto comunicado entre agentes. A transmissão de uma mensagem leva um tempo finito e requer uma Action (Ação) a ser realizada pelo remetente e também pelo destinatário. Os atributos de uma Mensagem especifcam o remetente, destinatário, um ato comunicativo (categorizando a Mensagem em termos da intenção do remetente) e o conteúdo (uma InformationEntity – Entidade de Informação).

A figura 2.7.11 apresenta uma visão geral informal centralizada no agente de como estes conceitos são inter-relacionados, mostrando seus relacionamentos ao conceito de agente.

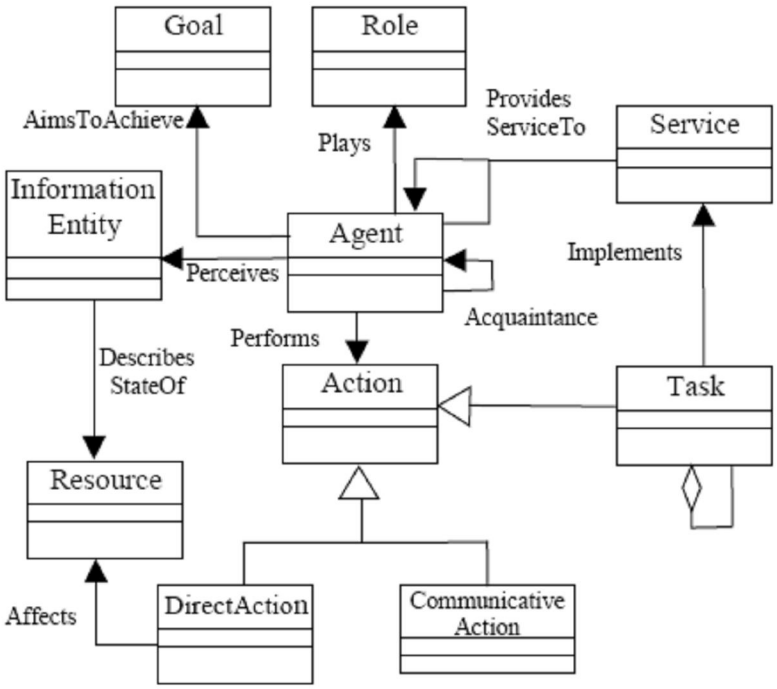


Figura 2.7.11 – Conceitos MESSAGE centralizados no Agente (CAIRE, 2002)

2.7.3.2 Visões de Modelo de Análise

Um modelo de análise é uma rede complexa de classes e instâncias inter-relacionadas derivadas de conceitos definidos no metamodelo MESSAGE/UML. MESSAGE define um grande número de visões que focam-se sobre sub-conjuntos de conceitos de entidade e relacionamento sobrepostos:

Visão de Organização (Organisation view – OV): Esta mostra ConcreteEntities – Entidades Concretas (Agentes, Organizações, Funções e Recursos) existentes no sistema e no seu ambiente, além dos relacionamentos de natureza ampla entre elas (relacionamentos de agregação, energia e conhecimento). Um relacionamento de conhecimento indica a existência da ocorrência de, no mínimo, uma interação entre as entidades referidas.

Visão de Objetivo/Tarefa (Goal/Task view (GTV)): Esta mostra Objetivos, Tarefas, Situações e as dependências entre eles. Tanto Objetivos como Tarefas tem atributos do tipo Situação, assim eles podem ser ligados por dependências lógicas para formar grafos que mostrem, por exemplo, decomposição de Objetivos de alto nível em sub-objetivos e como Tarefas podem ser realizadas para atingir Objetivos. Grafos mostrando dependências temporais podem também ser desenhados e os Diagramas de Atividade da UML podem ser úteis neste ponto, segundo Caire (2002).

Visão de Agente/Papel (Agent/Role view – AV): Esta foca-se sobre Agentes e Papéis individuais. Para cada agente/papel a visão utiliza conjuntos de esquemas suportados por diagramas relativos às suas características tais como por quais Objetivos ela é responsável, quais eventos ela necessita perceber, quais recursos ela controla, quais Tarefas ela sabe como realizar, “regras de comportamento”, etc.

Visão de Interação (Interaction viem – IV): Para cada interação entre agentes/papéis esta visão mostra o iniciador, os colaboradores, o motivador (geralmente um objetivo pelo qual o iniciador é responsável), a informação relevante suprida/atingida por cada

participante, os eventos que disparam a interação, outros efeitos relevantes da interação (por exemplo, um agente torna-se responsável por um novo objetivo). Séries maiores de interações ao longo do sistema (por exemplo, correspondente a casos de uso) podem também ser consideradas.

Visão de Domínio (Domain view – DV): Mostra os conceitos e relacionamentos de domínio específico que são relevantes para o sistema sob desenvolvimento.

2.7.3.3 Processo de Análise

O propósito da Análise é produzir um modelo (ou conjunto de modelos) do sistema a ser desenvolvido e seu ambiente que é acordado entre o analista e o cliente (e outros interessados). Ele ajuda a comunicação entre o time de desenvolvimento e o cliente, e fornece a base a partir da qual o projeto pode continuar com confiança. Os modelos de análise são produzidos por refinamento por etapas.

Abordagem de Refinamento: O alto nível de decomposição é referido como nível 0. O nível inicial está interessado em definir o sistema a ser desenvolvido com respeito a seus usuários e ambiente. O sistema é visto como um conjunto de organizações que interagem com recursos, atores ou outras organizações. Atores podem ser usuários humanos ou outros agentes existentes. Estágios subseqüentes de refinamento resultam na criação de modelos no nível 1, nível 2 e assim por diante.

No nível 0, o processo de modelagem inicia construindo a Organização e as visões Objetivo/Tarefa. Estas visões então agem como entradas para criar o Agente/Papel e as Visões de Domínio. Finalmente, a visão de interação é construída usando a entrada de outros modelos. O modelo de nível 0 dá uma visão geral do sistema, seu ambiente e sua funcionalidade global. A granularidade de nível 0 foca-se sobre a identificação de entidades e seus relacionamentos de acordo com o metamodelo. Mais detalhes sobre a estrutura interna e o comportamento destas entidades são adicionadas progressivamente nos níveis seguintes.

No nível 1 são definidas a estrutura e o comportamento de entidades tais como organização, agentes, tarefas, entidades de domínio de objetivos. Níveis adicionais podem ser definidos para analisar aspectos específicos do sistema lidando com requisitos funcionais e não funcionais, tais como performance, distribuição, tolerância à falhas e segurança. Deve haver consistência entre níveis subseqüentes. No projeto MESSAGE só o nível 0 e o nível 1 tem sido considerados.

Estratégias de Refinamento de Análise: Diversas estratégias são possíveis para refinar os modelos do nível 0. Abordagens centradas em organização focam-se em analisar propriedades gerais tais como estrutura do sistema, os serviços oferecidos, tarefas e objetivos globais, principais papéis e recursos. Os agentes necessários para atingir os objetivos aparecem naturalmente durante o processo de refinamento. Então pode ser analisada co-operação, possíveis conflitos e resolução de conflito.

Abordagens centradas em agente focam-se na identificação de agentes necessários para fornecer a funcionalidade do sistema. A organização mais adequada é identificada de acordo com os requisitos do sistema. Abordagens orientadas a interação sugerem refinamento progressivo de cenários de interação que caracterizam o comportamento interno e externo da organização e agentes. Estes cenários são a fonte para caracterizar tarefa, objetivo, mensagens, protocolos e entidades de domínio.

Abordagens de decomposição objetivo/tarefa são baseadas sobre decomposição funcional. Papéis, objetivos e tarefas de sistema são sistematicamente analisadas de

forma a determinar as condições de resolução, métodos de solução de problemas, decomposição e tratamento de falha. Pré-condições de tarefa, estrutura de tarefa, saída de tarefa e pós-condição de tarefa podem determinar que Entidades de Domínio são necessárias. Objetivos e tarefas devem ser desempenhados por agentes executando certos papéis. Consequentemente a pesquisa na estrutura geral de objetivo e tarefa nas decisões de visão de objetivo/tarefa pode ser feita sobre os agentes mais apropriados e sobre a estrutura de organização para atingir aqueles objetivos/tarefas.

Segundo Caire (2002), a experiência em MESSAGE mostra que as diferentes visões do sistema deixam o analista livre para escolher a estratégia mais apropriada. Na prática, são utilizadas uma combinção de estratégias de refinamento com freqüentes "laços de retorno" (loop-backs) entre elas. O processo de análise pode iniciar com a OV, então mudar para a AV e continuar com a IV. Os resultados da análise de cenários de interação específicos pode levar a reconsiderar parte da OV, e iniciar novamente a refinar e adaptar constituintes OV.

2.7.3.4 Estudo de Caso

Caire (2002) apresenta um estudo de caso enfocando o desenvolvimento de um sistema para gerenciamento de conhecimento a ser usado por uma equipe de engenheiros de uma companhia operadora de telecom que realiza operações de instalação e manutenção de equipamento. A companhia deseja que cada engenheiro possua um terminal wireless e que o sistema automaticamente notifique os engenheiros sobre os serviços a eles atribuídos; além disso, o sistema deverá trazer a documentação relevante para cada serviço; automaticamente ou quando pedido identificar outros engenheiros que possam ajudar no serviço; e permitir aos engenheiros relatarem o serviço realizado diretamente em um formulário eletrônico, armazenando esta informação em um banco de dados.

Uma vez que a documentação relevante para um serviço deve ser proativamente fornecida para o engenheiro que está indo realizar o serviço, o sistema a ser desenvolvido requer que seus componentes mostrem um alto grau de autonomia. Além disso é quase impossível antever exatamente todas as falhas possíveis que podem acontecer no equipamento a ser mantido e portanto será necessário comportamento orientado a objetivo. Finalmente encontrar um engenheiro com habilidades apropriadas para fornecer assistência em um certo serviço pode requerer alguma forma de negociação e coordenação distribuída.

2.7.3.5 Análise de Nível 0 – Visão de Organização

A partir deste enunciado, Caire (2002) desenvolveu dois diagramas de Visão de Organização no nível 0, apresentando os principais relacionamentos estruturais e de conhecimento, como pode ser visto nas figuras 2.7.12 e 2.7.13.

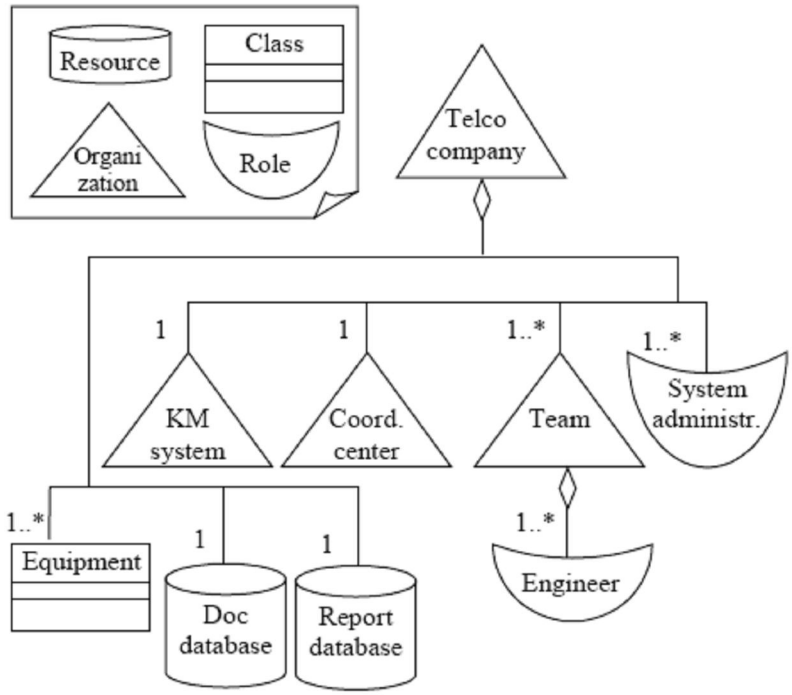


Figura 2.7.12 – Nível 0 – Visão de Organização – Relacionamentos Estruturais (CAIRE, 2002)

A figura 2.7.12 demonstra que o sistema de Gerência de Conhecimento (Knowledge Managemente (KM)) é possuído pela companhia operadora de telecom (Telco Company). Um engenheiro é parte de uma equipe e há diversos times na companhia. Deve-se observar que este diagrama de organização é um diagrama de classes UML onde ícones apropriados foram associados a estereótipos diferentes. No nível 0 o sistema sob desenvolvimento, ou seja, o sistema de gerência de conhecimento (KM) é, em si mesmo, uma organização que irá ser analisada no nível 1.

A figura 2.7.13 mostra os relacionamentos de conhecimento no diagrama de organização no nível 0. O sistema de Gerência de Conhecimento (KM) interage com dois papéis, o Administrador do Sistema e o Engenheiro e com dois sistemas externos (recursos), o Banco de Dados de Documentação Técnica para recuperar documentação e o Banco de Dados Relatório para inserir os relatórios de trabalho preenchidos pelos engenheiros. Além disso ele interage com o Centro de Coordenação para pegar a lista de trabalhos a realizar. Um Engenheiro também interage com outros Engenheiros para pedir ajuda direta. Deve-se notar que um engenheiro não interage diretamente com os Bancos de Dados de Documentação e Relatório. Todas estas interações são realizadas através do sistema de Gerência de Conhecimento (KM).

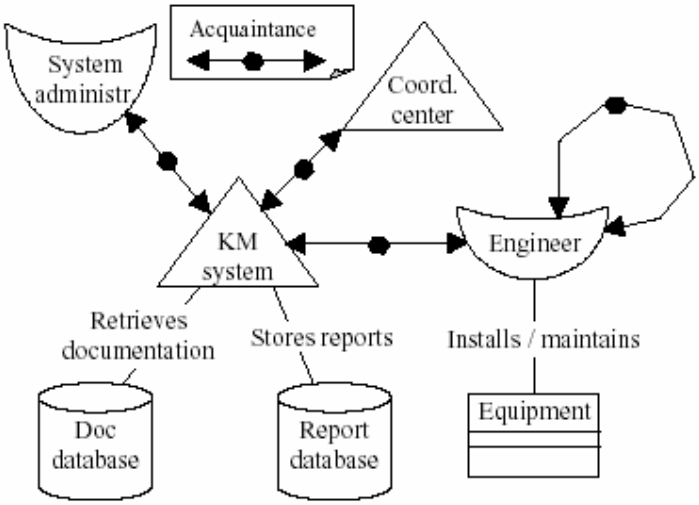


Figura 2.7.13 – Nível 0 – Visão de Organização – Relacionamentos de Conhecimento (CAIRE, 2002)

2.7.3.6 Análise de Nível 0 – Visão de Objetivo/Tarefa

A figura 2.7.14 apresenta um diagrama de implicação de objetivo/tarefa, onde o principal objetivo do sistema, que é dar assistência ao engenheiro (EngineerAssisted) é satisfeito quando a documentação relevante para o serviço corrente é fornecida e o nome de um engenheiro qualificado para o qual solicitar ajuda direta é identificado. O objetivo de fornecer documentação (DocumentationProvided) por sua vez é satisfeito quando o serviço a ser realizado é conhecido, a documentação requerida para realizar o trabalho foi identificada/recuperada, e esta documentação é apresentada para o engenheiro auxiliado.

A figura 3.15 apresenta um diagrama de implicação de objetivo/tarefa, onde o principal objetivo do sistema, que é dar assistência ao engenheiro (EngineerAssisted) é satisfeito quando a documentação relevante para o serviço corrente é fornecida e o nome de um engenheiro qualificado para o qual solicitar ajuda direta é identificado. O objetivo de fornecer documentação (DocumentationProvided) por sua vez é satisfeito quando o serviço a ser realizado é conhecido, a documentação requerida para realizar o trabalho foi identificada/recuperada, e esta documentação é apresentada para o engenheiro auxiliado. A decomposição do objetivo de identificar um engenheiro qualificado (QualifiedEngineerIdentified) não é apresentada (CAIRE, 2002).

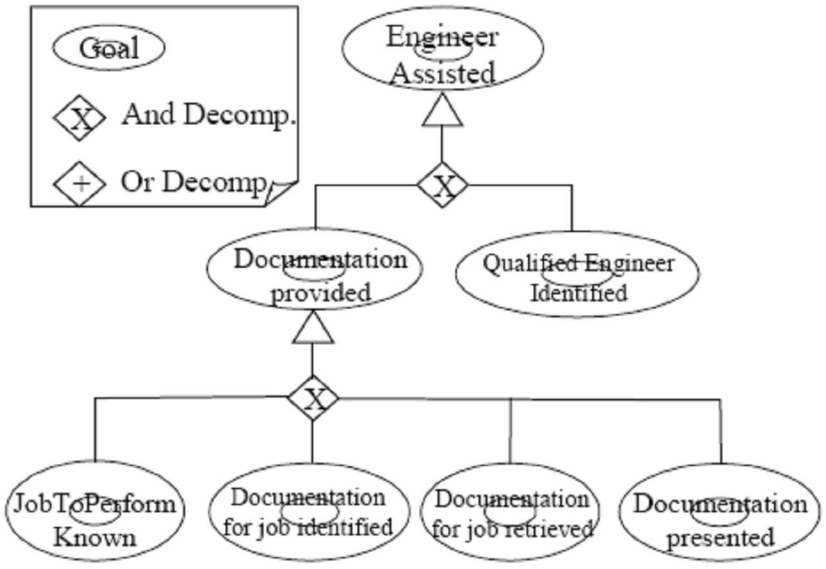


Figura 2.7.14 – Nível 0 – Diagrama de Implicação de Objetivo/Tarefa (CAIRE, 2002)

Alternativamente ou em conjunção com o diagrama de implicação de objetivo/tarefa, é útil para analisar como um dado serviço é realizado por um conjunto de tarefas parcialmente ordenado. A seguinte mostra o fluxo de trabalho (workflow) de tarefas implementando o serviço de Identificação de Engenheiro Qualificado (Identify-Qualified-Engineer) por meio de um diagrama de workflow (um Diagrama de Atividade UML onde tarefas são mostradas ao invés das atividades). O diagrama também mostra as classes que são entrada/saída de tarefas usando fluxos de objeto e os papéis que realizam as tarefas.

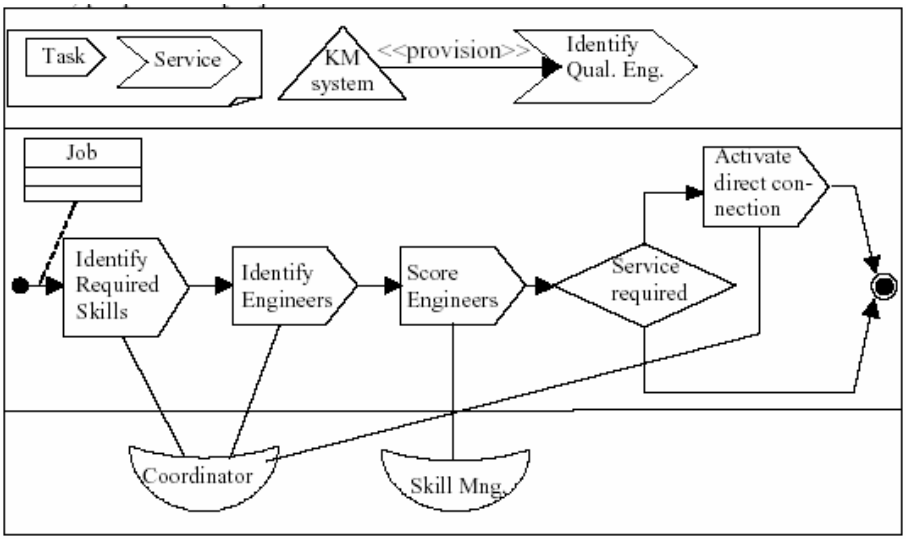


Figura 2.7.15 – Nível 0 – Diagrama de Fluxo de Trabalho (Workflow) (CAIRE, 2002).

2.7.3.7 Análise de Nível 1 – Visão de Organização

A análise de nível 1 foca-se sobre o sistema propriamente dito, identificando de imediato as partes principais de funcionalidade requerida (vistas como papéis e/ou tipos de agentes). A abordagem seguida por Caire (2002), foi considerar inicialmente somente papéis e definir que agentes irão povoar o sistema e que papéis cada agente irá

executar no início do processo de projeto. A figura 2.7.16 apresenta relacionamentos de conhecimento em um Diagrama de Organização de Nível 1.

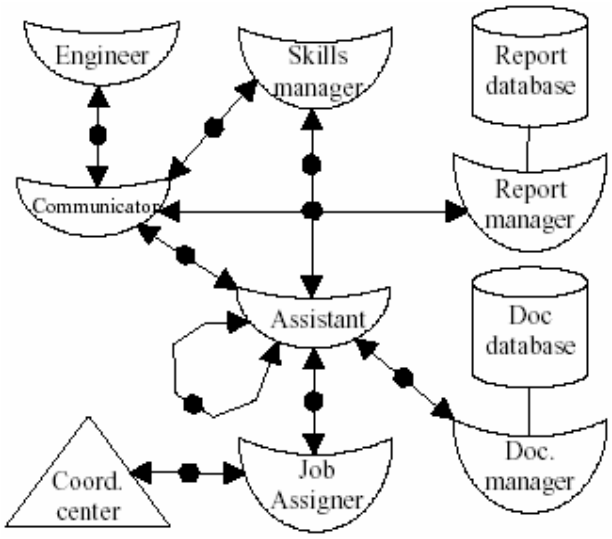


Figura 2.7.16 – Nível 1 – Diagrama de Organização (Relacionamentos de Conhecimento) (CAIRE, 2002).

Como pode-se observar na figura 2.7.16, o Gerente de Habilidades (Skills manager) mantém conhecimento das habilidades do engenheiro baseado nos serviços que ele realiza. As interações entre Assistentes (Assistants) requer uma rede de contrato para identificar outro engenheiro que possua as habilidades corretas para fornecer assistência para um determinado serviço.

2.7.3.8 Análise de Nível 1 – Visão de Agente/Papel

Segundo Caire (2002), delegação, diagramas de estrutura de fluxo de trabalho e esquemas textuais de Agente/Papel são úteis para descrever a visão. Um diagrama de estrutura de delegação mostra como os sub-objetivos obtidos decompondo um objetivo de uma organização são atribuídos aos agentes/papéis incluídos na organização. Claramente este diagrama é estritamente relacionado tanto ao diagrama de decomposição de objetivo mostrando a decomposição do objetivo da organização como o diagrama de organização mostrando os agentes/papéis dentro da organização. A figura seguinte mostra um diagrama de estrutura de delegação.

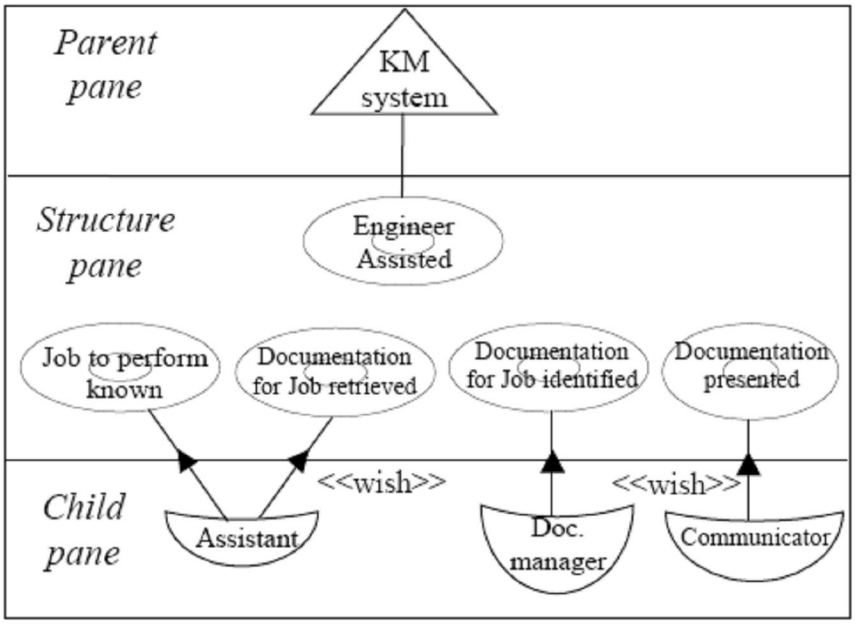


Figura 2.7.17 – Nível 1 – Diagrama de Estrutura de Delegação (CAIRE, 2002)

Nesta figura somente a raiz e as folhas da decomposição do objetivo de organização pai são apresentadas.

Similarmente um diagrama de estrutura de fluxo de trabalho (workflow) mostra os papéis em uma organização que deve realizar as tarefas necessárias para implementar um determinado serviço fornecido pela organização.

Para cada agente/papel há um esquema Agente/Papel que descreve suas características. No nível de análise esta informação é tipicamente muito informal e portanto um texto livre é preferido a uma notação gráfica. A seguir é descrito o esquema do papel Assistente (Assistant).

Tabela 2.7.1 – Esquema de Papel

Role Schema (Esquema de Papel)	Assistant
Goals (Objetivos)	JobToPerformKnown, DocumentationForJobRetrieved
Capability (Capacidade)	Alguma capacidade de aprendizado é necessária para manter o perfil do engenheiro atualizado sobre a base do trabalho completado.
Knowledge, Beliefs (Conhecimento, Crenças)	Um perfil das habilidades do engenherio a ser usado para avaliar se e como o engenheiro pode fornecer ajuda a um colega que pede assistência.
Agent requirements (Requisitos de Agente)	Este papel será executado pelo agente que auxilia o engenheiro atualmente.

Fonte: CAIRE, 2002

2.7.3.9 Visão de Interação

Esta visão destaca qual, por quê e quando agentes/papéis precisam se comunicar deixando todos os detalhes sobre como a comunicação é estabelecida para o processo de projeto.

A visão de interação é tipicamente refinida através de diversas iterações enquanto novas interações são descobertas. Isto pode ser convenientemente expressado por meio de um certo número de diagramas de interação. Estes diagramas são centrados em interação (ou seja, há um diagrama para cada interação) e mostra o iniciador, o respondedor, o motivador (muitas vezes um objetivo do iniciador) de uma interação além de outras informações opcionais, tais como a condição de início (trigger) e as informações atingidas e supridas por cada participante.

A figura seguinte mostra um exemplo de um diagrama de interação descrevendo a Interação de Requisição de Documentação (Documentation Request interaction) entre os papéis Assistente e o Gerenciador de Documentação.

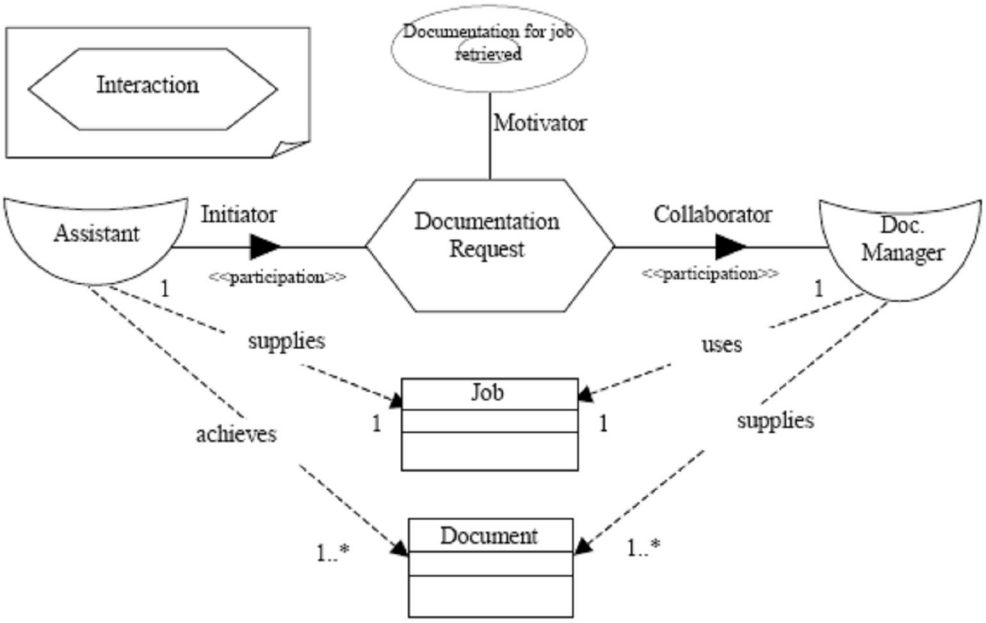


Figura 2.7.18 – Diagrama de Nível de Interação (CAIRE, 2002)

Segundo Caire (2002), os detalhes do protocolo de interação e as mensagens que são trocadas entre os papéis podem ser representadas usando o Diagrama de Seqüência da AUML.

2.7.3.10 Visão de Domínio

Segundo Caire (2002), esta visão pode convenientemente ser representada por meio de típicos diagramas de classe UML, onde classes representam conceitos específicos de domínio e associações representam relações específicas de domínio. Esta visão é tipicamente construída em paralelo com outras visões pela adição de novos conceitos e relações à medida em que eles forem necessários nas outras visões. A figura seguinte mostra um Diagrama de Informação de Domínio.

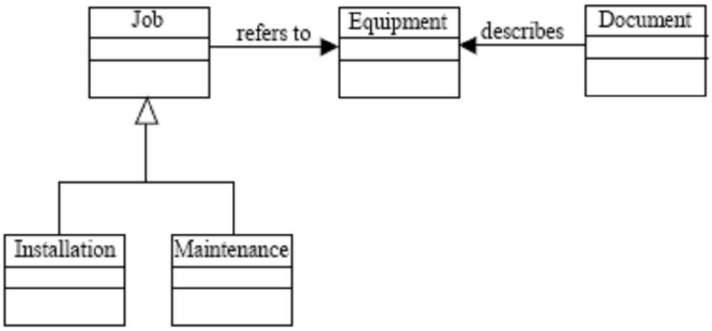


Figura 2.7.19 – Diagrama de Informação de Domínio (CAIRE, 2002)

2.7.3.11 Considerações Finais

Os modelos de análise são produzidos por meio de refinamento por etapas. No nível 0, o processo de modelagem inicia construindo a Visão de Organização e a Visão de Objetivo/Tarefa. Estas visões então agem como entradas para criar a Visão de Agente/Papel e a Visão de Domínio. Finalmente, a visão de interação é construída usando a entrada dos outros modelos. O modelo de nível 0 dá uma visão geral do sistema, seu ambiente e sua funcionalidade global. Mais detalhes sobre a estrutura interna e o comportamento destas entidades são adicionadas progressivamente nos níveis seguintes.

O mais próximo da aplicação de casos de uso em MESSAGE/UML é a visão de objetivo/tarefa, onde é utilizada uma notação um pouco similar, porém o símbolo utilizado (uma elipse dentro de outra) representa somente objetivos gerais do sistema, que posteriormente serão utilizados para produzir papéis.

Esta abordagem difere da sugerida em nosso trabalho, onde tentamos adaptar o diagrama de casos de uso para a modelagem de requisitos funcionais específicos para sistemas multi-agentes de forma a identificar os papéis de agentes por meio de Agent_Role Actors, seus objetivos, percepções, ações e planos. Tentamos manter a adaptação do diagrama de casos de uso o mais semelhante à sua aplicação original, pretendendo assim facilitar sua compreensão por quem já possuísse domínio nesse tipo de modelagem.

2.7.4 PASSI (Process for Agent Societies Specification and Implementation – Processo para Especificação e Implementação de Sociedade de Agentes)

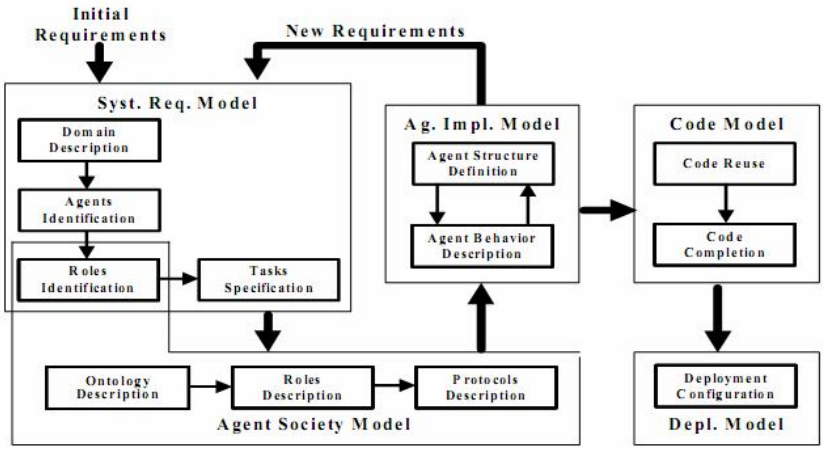
De acordo com Burrafato (2002), PASSI é uma metodologia para projetar e desenvolver sociedades multi-agente integrando modelos de projeto e conceitos de engenharia de software e inteligência artificial usando a notação UML. Os modelos e fases de PASSI abrangem representação antropomórfica de requisitos do sistema, ponto de vista social, arquitetura de solução, produção e reuso de código e implementação de configuração para suporte a mobilidade de agentes.

Em PASSI a UML foi adotada como a linguagem de modelagem uma vez que ela é amplamente aceita e seus mecanismos de extensão (restrições, valores etiquetados – tagged e estereótipos) auxiliam a costumizar as representações de projetos orientados a agente.

A metodologia possui cinco modelos:

- Modelo de Requisitos de Sistema, um modelo antropomórfico dos requisitos do sistema em termos de agência e destino, dividido em quatro passos.
- Modelo de Sociedade de Agente, um modelo das interações e dependências sociais entre os agentes que participam na solução, dividido em quatro passos.
- Modelo de Implementação de Agente, um modelo da arquitetura de solução em termos de classes e métodos.
- Modelo de Código, um modelo da solução ao nível de código.
- Modelo de Implantação, um modelo da distribuição das partes do sistema sobre unidades de hardware.

A figura seguinte apresenta os modelos e fases da metodologia PASSI.



Key

D.D. – Domain Description
A.Id. – Agents Identification
R.Id. – Roles Identification
T.Sp. – Tasks Specification
A.S.D. – Agents Structure Definition
A.B.D. – Agents Behaviour Description
O.D. – Ontology Description
R.D. – Roles Description
P.D. – Protocols Description
C.R. – Code Reuse
C.C. – Code Completion
D.C. – Deployment Configuration

Figura 2.7.20 – Modelos e Fases da Metodologia PASSI (BURRAFATO, 2002)

2.7.4.1 Fase de Descrição de Domínio

Burrafato (2002) afirma que em PASSI os requisitos são descritos em termos de diagramas de casos de uso. A Fase de Descrição de Domínio é uma descrição funcional do sistema composto de uma série hierárquica de diagramas de casos de uso. Cenários dos diagramas de casos de uso detalhados são então explanados usando diagramas de seqüência. A figura seguinte mostra um diagrama de Descrição de Domínio descrevendo a análise de um estudo de caso de uma livraria. Os estereótipos usados aqui vem da UML padrão. A convenção adotada para relacionamentos entre atores externos e o sistema é direcionar setas do iniciador das comunicações para o participante.

O cenário aqui apresentado ocorre sempre que a livraria precisa comprar livros. O cenário é disparado a partir da funcionalidade Predict Students Needs (Prevê Necessidades dos Alunos) que usa a funcionalidade Search Store's Archive (Pesquise o Arquivo do Estoque) de forma a estabelecer se há um número suficiente de itens daquele livro no estoque ou não. Caso não haja, e o livro é necessário, uma nova

compra deve ser feita e portanto a funcionalidade Provide Books (Forneça Livros) é invocada.

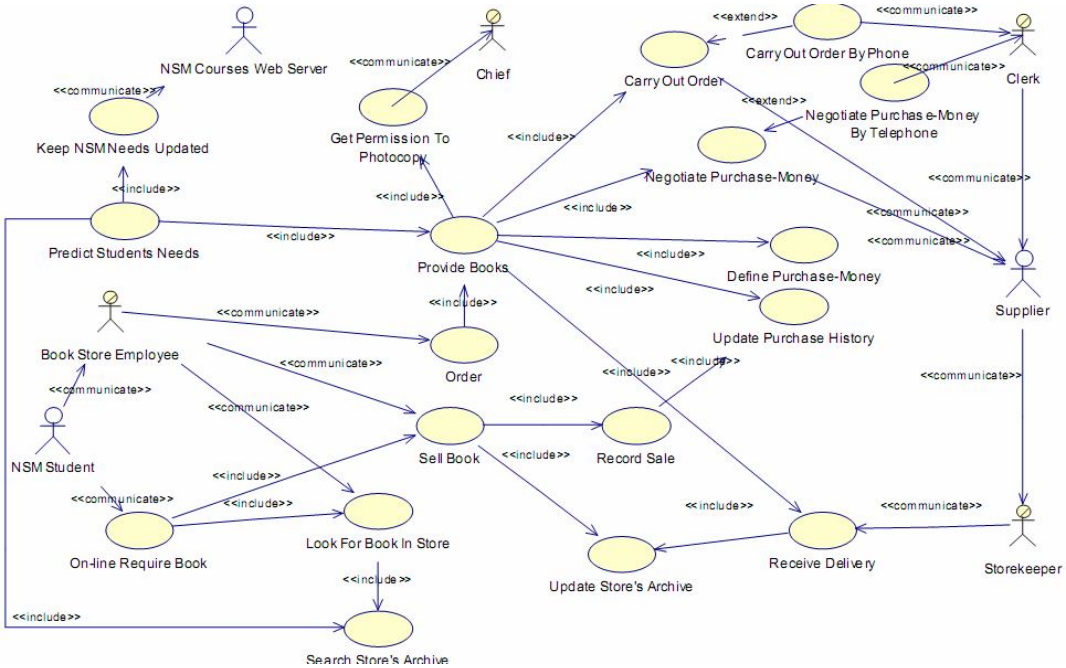


Figura 2.7.21 – Diagrama de Descrição de Domínio para o Estudo de Caso da Livraria (BURRAFATO, 2002)

2.7.4.2 Fase de Identificação de Agente

A identificação dos agentes inicia a partir dos diagramas de caso de uso do passo anterior. De acordo com Burrafato (2002) é possível ver um agente como um caso de uso ou um pacote de casos de uso na decomposição funcional da fase anterior. Começando a partir de um diagrama suficientemente detalhado das funcionalidades do sistema, agrupou-se um ou mais casos de uso em pacotes estereotipados de maneira a formar um novo diagrama. Assim, cada pacote define as funcionalidades de um agente específico. A figura seguinte apresenta a parte do diagrama de Identificação de Agente que compreende somente os agentes envolvidos no cenário que está sendo investigado.

Relacionamentos entre casos de uso do mesmo agente seguem a sintaxe e estereótipos UML padrão, enquanto relacionamentos entre casos de uso de agentes diferentes são estereotipados como “comunicação” (communication). A convenção adotada para este diagrama é a de direcionar relacionamentos de comunicação entre agentes do iniciador para o participante.

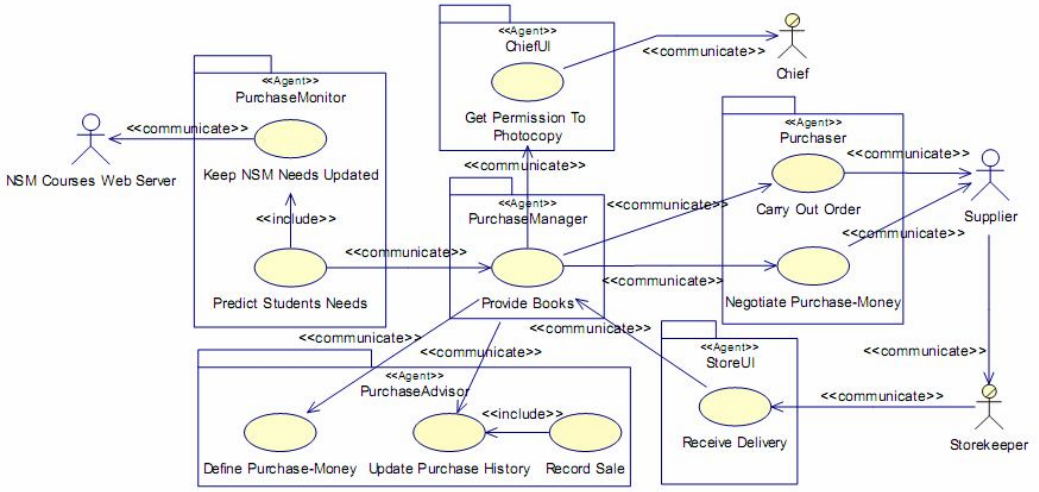


Figura 2.7.22 – Parte do Diagrama de Identificação de Agente Obtida a partir da Fase de Descrição de Domínio (BURRAFATO, 2002)

2.7.4.3 Fase de Identificação de Papéis

Esta fase ocorre primeiramente na análise de requisitos uma vez que nesse momento preocupa-se mais com o comportamento visível externamente do agente ao invés de sua estrutura – somente aproximada nesta etapa. Já que um papel é também um conceito social, considerou-se esta fase para ser também parte do Modelo de Sociedade do Agente. A identificação de papéis é baseada na exploração de todos os caminhos possíveis do diagrama de Identificação de Agentes envolvendo comunicação inter-agente. Um caminho descreve um cenário de agentes interativos trabalhando para atingir um comportamento requerido do sistema. Ele é composto de diversos caminhos de comunicação, que são simplesmente um relacionamento “comunicado” entre dois agentes no Diagrama de Identificação. Cada um deles pode pertencer a diversos cenários, que são detalhados por meio de diagramas de seqüência no quais objetos são usados para simbolizar os papéis.

A figura seguinte mostra o cenário discutido na fase de Descrição de Domínio que ocorre quando uma nova compra é requerida pelo papel Informante (Informer) do agente PurchaseMonitor (Monitor de Compra) para o papel BooksProvider (Fornecedor de Livors) do agente PurchaseManager (Gerente de Compras). Cada objeto no diagrama representa um papel e foi nomeado com a sintaxe <role_name>:<agent_name> (<nome_papel>:<nome_agente>).

Um agente pode participar em cenários diferentes executando papéis distintos em cada um. Ele pode também executar papéis distintos no mesmo cenário. As mensagens no diagrama de seqüência podem ou significar eventos gerados pelo ambiente externo ou comunicação entre os papéis de um ou mais agentes. Uma mensagem especifica o que o papel deve fazer e possivelmente os dados a serem fornecidos ou recebidos.

Burrafato (2002) afirma que pode-se descrever o cenário como segue:

- O informante (Informer) informa ao BooksProvider que a livraria precisa comprar um estoque de livros específico.

- Dada uma lista de fornecedores para os livros necessários, o BooksProvider requer ao Consultante (Consultant) para sugerir condições de compra sobre a base de negócio passado.
- Quando o Consultante retornar um aviso, o BookProvider dá ao Negociador (Negotiator) os dados sobre o fornecedor para negociar e as condições a ser negociadas; ao mesmo tempo ele requer que a negociação seja iniciada. O BookProvider então está pronto para cuidar de outras requisições que podem vir dos papéis dos agentes cooperantes.
- O Negociador negocia e pega a melhor oferta. Ele então a retorna para o BookProvider.
- O BooksProvider estabelece se a oferta é boa o suficiente ou não, de acordo com seu orçamento e considerações tais como o preço do livro e o número de alunos que iria então comprá-lo. Neste cenário assume-se que a oferta seja boa e assim o BookProvider propõe ao OrderPlacer para comprar os livros. Portanto o BooksProvider está pronto para cuidar de outras requisições.
- Quando os livros são entregues uma notificação é encaminhada do DeliveryNotifier (Notificador de Entrega) para o BooksProvider.

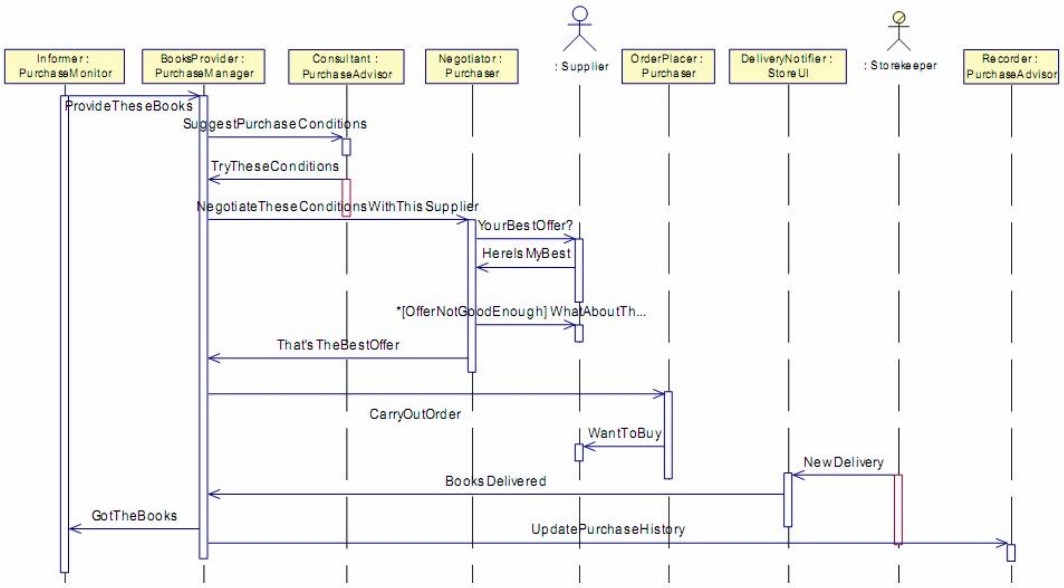


Figura 2.7.23 – Diagrama de Identificação de Papéis para o cenário no qual o PurchaseMonitor Anuncia a Necessidade de Compra de Livros (BURRAFATO, 2002)

2.7.4.4 Fase de Especificação de Tarefa

Nesta etapa, para cada agente foca-se em seu comportamento de maneira a decompô-lo em tarefas. Tarefas geralmente encapsulam alguma funcionalidade que forma uma unidade lógica de trabalho.

Para todo agente no modelo, produziu-se um diagrama de atividade que é formado por duas raias de natação. A da direita contém uma coleção de atividades simbolizando as tarefas do agente, enquanto que a da esquerda contém algumas atividades representando os outros agentes interagentes.

Um diagrama de Especificação de Tarefa sumariza o que o agente é capaz de fazer, ignorando informações sobre papéis que um agente interpreta quando executando tarefas particulares. Relacionamentos entre atividades significam ou mensagens entre tarefas e outros agentes interagentes ou comunicação entre tarefas do mesmo agente. Os últimos não são atos de fala, mas ao invés disso sinais enfocando a necessidade de iniciar uma elaboração, ou seja, disparando uma execução de tarefa ou delegando outro tarefa para fazer alguma coisa. Cada atividade pode ser melhor especificada por diagramas de seqüência, diagramas de atividade ou texto semi-formal com campos para pré-condições, triggers, etc. De maneira a gerar um diagrama de Especificação de Tarefa de agente é preciso examinar todos os diagramas de Identificação de Papéis de agente. Então explora-se todas as interações e ações internas que o agente realiza para completar um propósito de cenário. Para cada diagrama de Identificação de Papel obtém-se uma coleção de tarefas relacionadas. Agrupando-as todas juntas apropriadamente resulta no diagrama de Especificação de Tarefa. Uma tarefa Listener (ouvinte) é necessária de maneira a passar comunicação de entrada para a tarefa apropriada. Novas tarefas são necessárias para manusear todas as mensagens de entrada do cenário de Identificação de Papel. Do mesmo modo, uma tarefa é introduzida para cada mensagem de saída do cenário de Identificação de Papel. Tarefas extras podem ser introduzidas para uma melhor decomposição do agente.

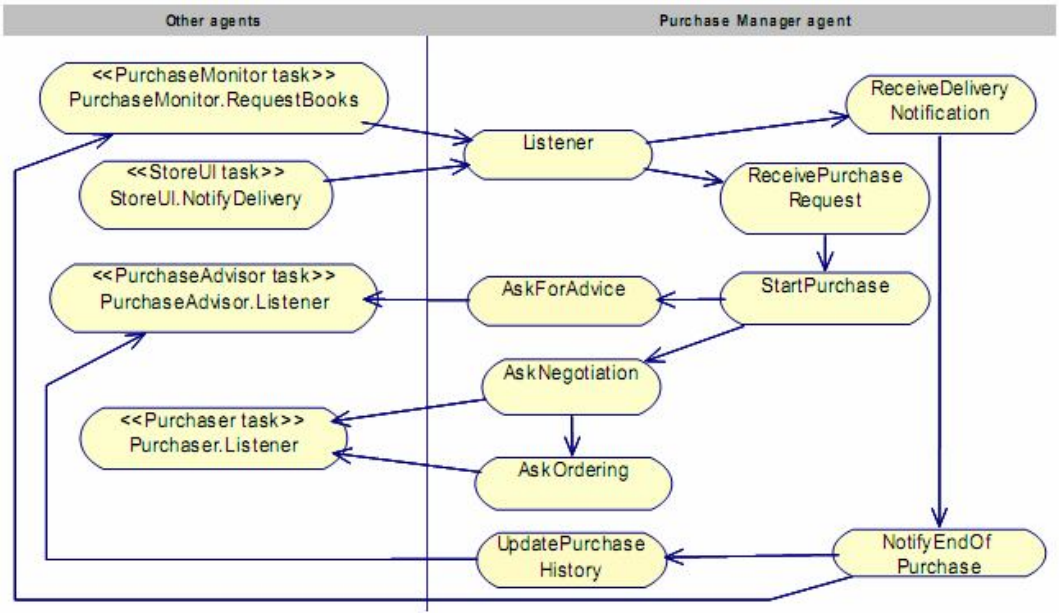


Figura 2.7.24 – As Tarefas do Agente PurchaseManager (BURRAFATO, 2002)

2.7.4.5 Fase de Descrição de Ontologia

Nesta fase descreve-se a sociedade de agentes levando em conta o ponto de visão ontológico. A Descrição de Ontologia gera dois diagramas: Descrição de Ontologia de Domínio e Descrição de Ontologia de Comunicação; no primeiro a ontologia de domínio é representada como um esquema XML enquanto que no último a comunicação é explanda por um diagrama de classe. A figura seguinte mostra o diagrama de Descrição de Ontologia de Domínio; os elementos definidos aqui serão usados para definir as peças de conhecimento de domínio. Já a figura 2.7.26 contém um diagrama de Descrição de Ontologia de Comunicação que descreve tanto o

conhecimento dos agentes, representado como atributos, como a ontologia de sua comunicação.

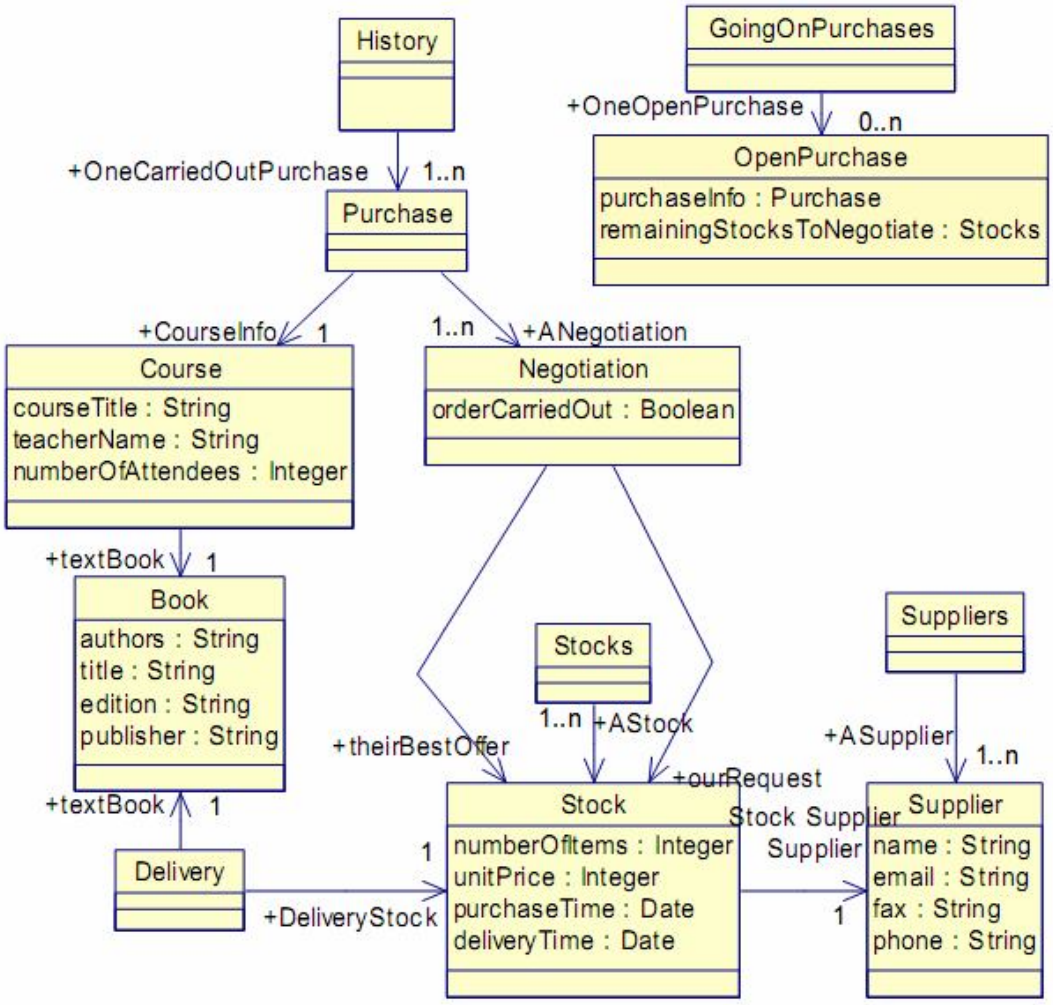


Figura 2.7.25 – Diagrama de Ontologia de Domínio (BURRAFATO, 2002)

Burrafato (2002), afirma que para uma mensagem fazer sentido, ele precisa espeficar um protocolo. Além disso, sem uma linguagem e uma ontologia o conteúdo de uma mensagem não seria entendido. Assim é preciso utilizar classes associativas no diagrama de Descrição de Ontologia de Comunicação de maneira a ligar protocolo, linguagem e ontologia a cada comunicação. Uma comunicação é desenhada a partir do iniciador para o participante da conversação. Cada uma é deduzida do diagrama de Identificação de Papel. Na figura 2.7.26 o agente PurchaseManager inicia uma conversação com o agente PurchaseAdvisor. A Conversação contém a ontologia Course (Curso), o protocolo de consulta e a linguagem RDF. Isto significa que o PurchaseManager quer realizar um ato de fala baseado sobre o protocolo de consulta FIPA de maneira a solicitar ao PurchaseAdvisor um conselho sobre como comprar suprimentos, por exemplo, fornecidos pela Informação de Curso.

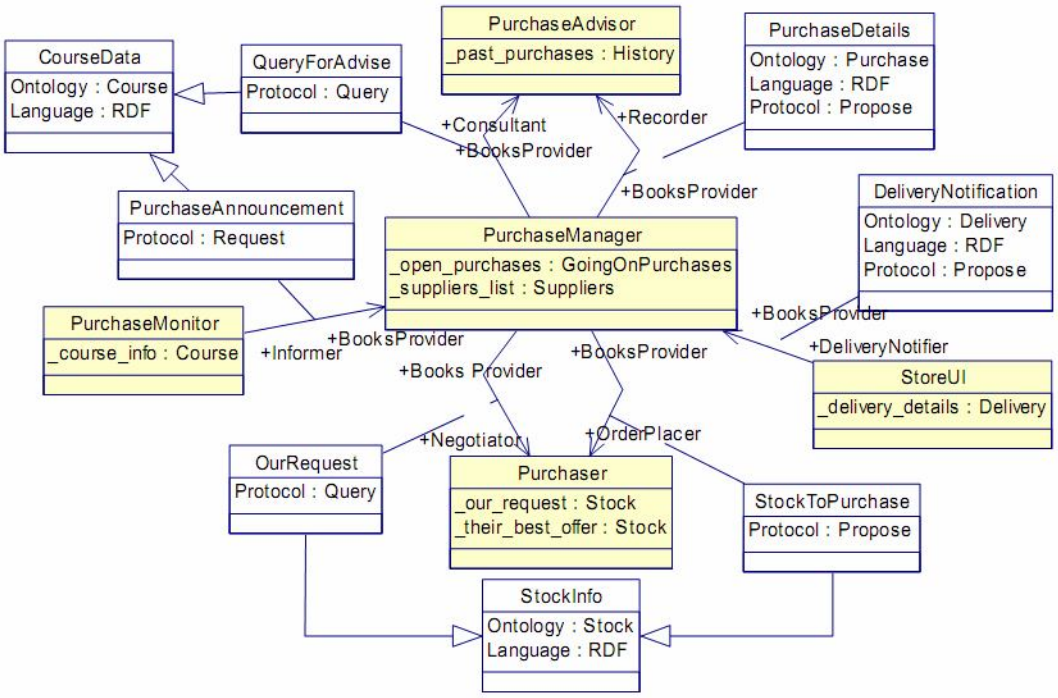


Figura 2.7.26 – Diagrama de Ontologia de Comunicação (BURRAFATO, 2002)

2.7.4.6 Fase de Descrição de Papéis

Esta fase modela o ciclo de vida de um agente levando em conta seus papéis, as colaborações necessárias e as convesações que ele está envolvido. Nesta fase pode-se também introduzir as regras sociais da sociedade de agentes e as leis comportamentais que podem ser expressadas em OCL, por exemplo.

A fase de Descrição de Papéis gera uma coleção de diagrama de classe na qual classes são usadas para representar papéis. O agente é simbolizado por um pacote contendo clsses de papel, como demonstra a próxima figura. Cada papel é obtido pela composição de diversas tarefas em um comportamento resultante. Esta é a razão porque colocou-se tarefas no compartimento de operação da classe de papel relacionada. Isto torna claro do que um papel é capaz e pode ser útil na identificação de padrões reusáveis. Um diagrama de Descrição de Papel pode também mostrar conexões entre papéis do mesmo agente, representando uma mudança de papel (uma linha tracejada com o nome [ROLE CHANGED]). Esta conexão é descrita como um relacionamento de dependência porque deseja-se demonstrar a dependência do segundo papel sobre o primeiro. Algumas vezes a condição de disparo não é explicitamente gerada pelo primeiro papel mas sua aparência precedente no cenário justifica a consideração que ela é necessária para preparar a situação que permite o segundo papel iniciar. Conversações entre papéis são indicadas por linhas sólidas.

Burrafato (2002) também considerou dependências entre agentes. Agentes são autônomos, por isso eles podem recusar fornecer um serviço ou recurso. Por esta razão, o projeto precisa de um esquema que expresse tais questões de maneira a explorar formas alternativas para atingir os objetivos.

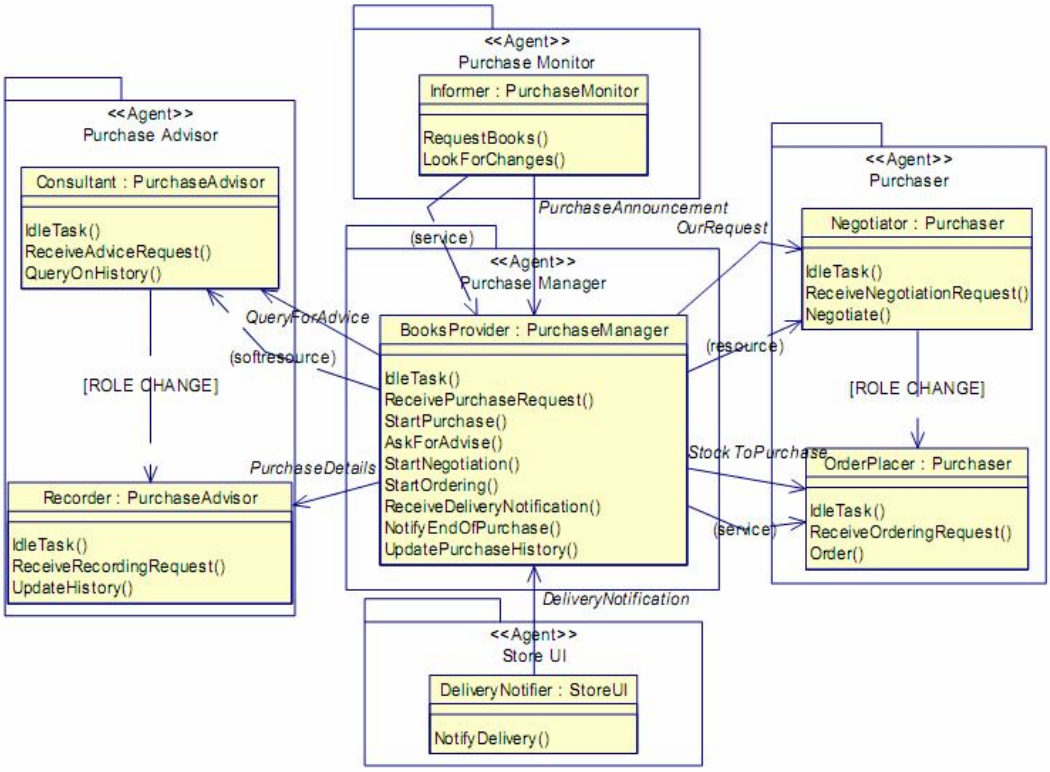


Figura 2.7.27 – Diagrama de Descrição de Papéis para o Cenário de Compra de Livros (BURRAFATO, 2002)

De maneira a compreender tal esquema, Burrafato (2002) introduziu no diagrama de Descrição de Papéis alguns relacionamentos adicionais que expressam os seguintes tipos de dependência:

- Dependência de Serviço: Um papel depende de outro para atingir um objetivo. Essa dependência é representada por uma linha tracejada com o nome “(service)” (serviço).
- Dependência de Recurso: Um papel depende de outro para a disponibilidade de uma entidade. Essa dependência é representada por um linha tracejada com o nome “(resource)” (recurso).
- Dependência Soft-Service e Soft-Resource: O serviço/recurso requisitado é útil ou desejável, mas não essencial para atingir um objetivo de papel. Essa dependência é representada por uma linha tracejada com os nomes “(soft-service)” e “(soft-resource)”.
- No exemplo da figura 2.7.27 a dependência entre o papel BookProvider e o papel Consultant é nomeado (softresource) já que o BookProvider ficaria feliz em ser aconselhado pelo Consultant. Mas se o Consultant de alguma maneira não puder satisfazer a consulta, o BookProvider continuará sem ele.

2.7.4.7 Fase de Descrição de Protocolos

Burrafato (2002) afirma que um protocolo é usado para cada comunicação e que cada protocolo é um protocolo padrão FIPA. Geralmente a documentação relacionada é feita por meio de diagramas de seqüência AUML. Conseqüentemente o projetista não

precisa especificar protocolos por si próprio. Em alguns casos, no entanto, os protocolos FIPA não são adequados e portanto alguns protocolos precisam ser projetados, o que pode ser feito usando a abordagem de documentação FIPA.

2.7.4.8 Fase de Definição de Estrutura de Agentes

Esta fase influencia e é influenciada pela fase de Descrição de Comportamento de Agente uma vez que um nível duplo de iteração ocorre entre eles. A fase de Definição de Estrutura de Agente produz diversos diagramas de classes logicamente subdivididos nas visões multi-agente e agente-único. Na primeira, enfoca-se a arquitetura geral do sistema onde se determinam os agentes e suas tarefas. Na última, concentra-se sobre a estrutura interna de cada agente, revelando todos os atributos e métodos da classe de agente junto com suas classes de tarefas internas. Na visão de Definição de Estrutura Multi-Agente, um diagrama de classes representa o sistema multi-agente como um todo, como demonstra a figura seguinte. O diagrama apresenta classes, cada uma simbolizando um dos agentes identificados na fase de Identificação de Agente. Compartimentos de atributos podem ser usados para representar o conhecimento do agente enquanto que compartimentos de operações são usadas para representar as tarefas do agente. Na visão de Definição de Estrutura de Agente-Único, um diagrama de classes, apresentado na figura 2.7.29, é usado para cada agente para ilustrar a estrutura interna do agente por meio de todas as clases que formam o agente, que são a classe principal do agente junto com as classes internas que identificam suas tarefas. Neste ponto define-se os atributos e métodos de ambas as classes de agente e as classes de tarefas. O resultado deste estágio produz uma estrutura detalhada do software, pronto para ser implementado.

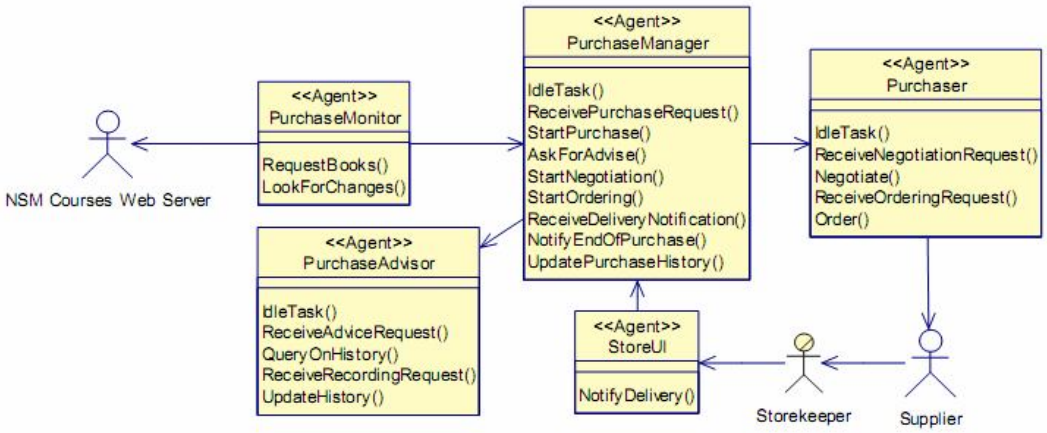


Figura 2.7.28 – Parte do diagram de Definição de Estrutura Multi-Agente para o Estudo de Caso de Livraria (BURRAFATO, 2002)

2.7.4.9 Fase de Descrição de Comportamento de Agentes

A fase de Descrição de Comportamento de Agente produz diversos diagramas que são subdivididos nas visões multi-agente e agente-único. Na primeira, desenhou-se o fluxo de eventos pela invocação de métodos e a troca de mensagens. Na última, caracterizou-se os métodos acima. Na visão de Descrição de Comportamento Multi-Agente, um ou mais diagramas de atividade são desenhaedos para mostrar o fluxo de evnetos entre e dentro tanto das classes de agentes principais como de suas classes internas (representando suas tarefas). Burrafato (2002) descreveu uma raia de natação

para cada agente e para cada tarefa. As atividades dentro das raias de natação indicam os métodos das classes relacionadas.

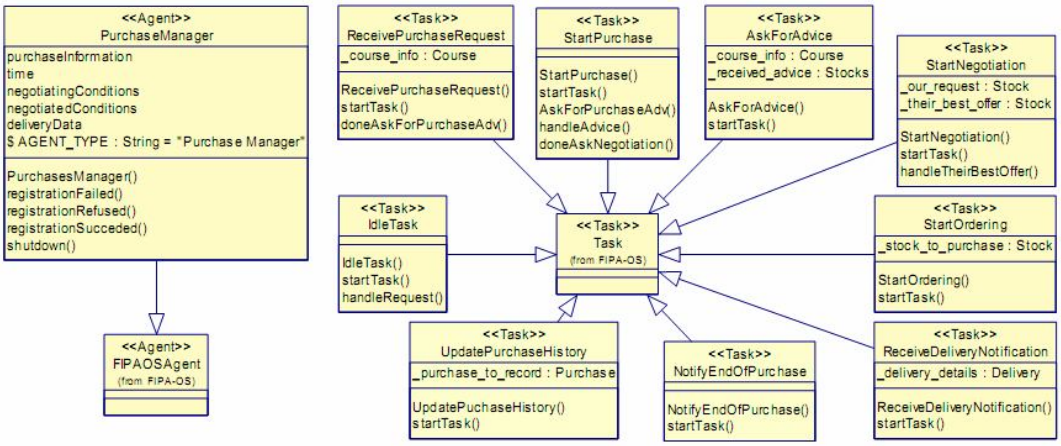


Figura 2.7.29 – Diagrama de Definição de Estrutura de Agente-Único para o Agente PurchaseManager (BURRAFATO, 2002)

Transições usuais da UML padrão são aqui descritas para representar ou eventos ou invocação de métodos. Uma transição é desenhada para cada mensagem reconhecida nas fases precedentes. Neste tipo de transição indica-se o performativo da mensagem como é especificado no diagrama de Descrição de Ontologia de Comunicação e o conteúdo da mensagem como descrito no diagrama de Descrição de Ontologia de Domínio. Isto resulta em ter uma descrição compreensiva da comunicação incluindo os métodos exatos envolvidos. A figura seguinte mostra um exemplo de descrição de comportamento de agentes. A tarefa StartPurchase (Incia Compra) do agente PurchaseManager instancia a tarefa StartNegotiation (Inicia Negociação) pela invocação do método da super-classe newTask (nova tarefa). Isto tem de ser feito de maneira a solicitar ao agente Purchaser (Comprador) para realizar uma negociação com o fornecedor. A invocação da tarefa StartNegociação implica que o método startTask (inica tarefa) será invocado. O que o método startTask faz é justamente enviar uma mensagem para o agente Purchaser. Este método contém a requisição performativa eo conteúdo OurRequest (Nossa Requisição). O método handleRequest (manuseia requisição) da tarefa IdleTask (tarefa ocios) do Purchaser recebe a comunicação de entrada e a envia para a tarefa ReceiveNegotiationRequest (Requisição de Negociação Recebida) depois que esta tenha sido instanciada. Quando uma tarefa completa seu trabalho o método é invocado.

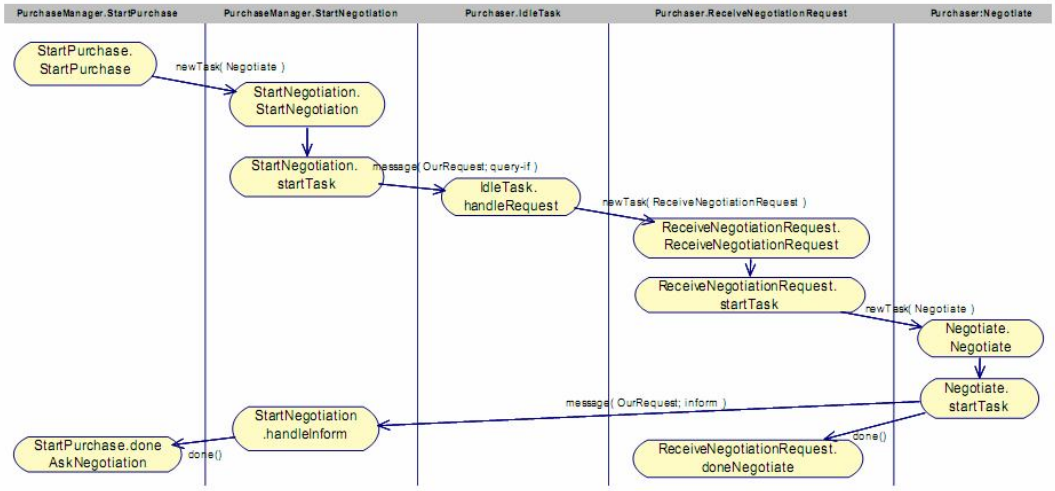


Figura 2.7.30 – Diagrama de Descrição de Comportamento Multi-Agente
(BURRAFATO, 2002)

2.7.4.10 Fase de Reuso de Código

Nesta fase tenta-se reutilizar padrões predefinidos de agentes e tarefas. Burrafato (2002) estendeu a ferramenta CASE UML Rational Rose desenvolvendo um Add-In para suportar PASSI. Isto tem se provado bastante flexível em produzir padrões, graças a sua ligação de elementos de projeto a código. O resultado tem sido uma coleção de peças de código reutilizáveis documentas pelos diagramas de Comportamento Multi-Agente e Estrutura de Agente-Único; o primeiro descreve comportamentos por meio do fluxo de eventos e métodos invocados; o último descreve estruturas de software.

2.7.4.11 Fase de Acabamento de Código

Segundo Burrafato (2002), esta fase representa o trabalho clássico do programador, que necessita completar o corpo dos métodos gerada neste ponto, levando em consideração os diagramas de projeto.

2.7.4.12 Fase de Configuração de Implantação

De acordo com Burrafato (2002) esta fase representa um aspecto chave na evolução de PASSI. Esta fase foi concebida para concordar com os requisitos de detalhamento das posições dos agentes em sistemas distribuídos ou em contextos de agentes móveis.

Os diagramas de Configuração de Implantação ilustram a localização dos agentes (as unidades de processamento onde eles se localizam), seu movimento e seu apoio de comunicação. Burrafato (2002) afirma que a UML padrão é útil para representar unidades de processamento e agentes (por meio de componentes), o que a UML não suporta é a representação da mobilidade do agente o que foi representado por meio de uma extensão de sintaxe consistindo de uma linha tracejada com um estereótipo “move_to” (mova para).

2.7.4.13 Considerações Finais

Em PASSI, os requisitos são descritos por meio de diagramas de casos de uso e estes são detalhados por meio de diagramas de seqüência.

A identificação dos agentes se inicia a partir dos diagramas de casos de uso. Em PASSI, um agente é visto como um caso de uso ou um pacote de casos de uso. Após a

identificação das funcionalidades do sistema, um ou mais casos de uso são agrupados em pacotes estereotipados, de forma que cada pacote defina as funcionalidades (externas) de um agente específico.

A identificação de papéis baseia-se na exploração de todos os caminhos possíveis envolvendo a comunicação inter-agente. Um caminho (path) descreve um cenário de agentes que interagem para atingir um comportamento requerido. Um caminho de comunicação é um relacionamento do tipo "communicate" entre dois agentes, que podem pertencer a diversos cenários detalhados por meio de diagramas de seqüência nos quais objetos são usados para simbolizar papéis. Estes objetos são instâncias dos pacotes de casos de uso que identificam os agentes.

Apesar de PASSI identificar as tarefas (tasks) de cada agente (por meio de classes estereotipadas), não identifica seus objetivos, pois, segundo os autores (BURRAFATO, 2002) uma tarefa é algo que o agente é capaz de fazer (o que é diferente de um objetivo). Além de não modelar objetivos graficamente, nem as condições para que estes se tornem intenções, a metodologia também não identifica as percepções sobre o ambiente ou as ações que o agente ou papel pode realizar sobre o mesmo e tampouco identifica os planos de um agente ou papel, embora as tarefas de um agente possam ser detalhadas por meio de diagramas de atividade.

2.7.5 Prometheus

Segundo Padgham (2004) e KHALLOUF (2009) esta metodologia consiste de três fases: A Fase de Especificação do Sistema que foca-se sobre a identificação dos cenários e das funcionalidades básicas do sistema, juntamente com entradas (percepções externas), saídas (ações externas) e qualquer fonte de dados compartilhados importantes. A Fase de Projeto Arquitetural utiliza as saídas da fase anterior para determinar quais agentes o sistema conterá e como eles interagirão. Já a Fase de Projeto detalhado preocupa-se com a parte interna de cada agente e como ele irá realizar suas tarefas dentro do sistema como um todo. A figura 2.7.31 ilustra as fases da metodologia.

Padghan (2002) declara que a metodologia Prometheus suporta especificamente o desenvolvimento de agentes do tipo BDI que utilizam objetivos, crenças, planos e eventos, porém destaca que isto torna Prometheus menos útil para aqueles que não desenvolvem agentes do tipo BDI, todavia Padghan (2002) afirma que os estados iniciais da metodologia são apropriados para o projeto de qualquer tipo de sistema multi-agente.

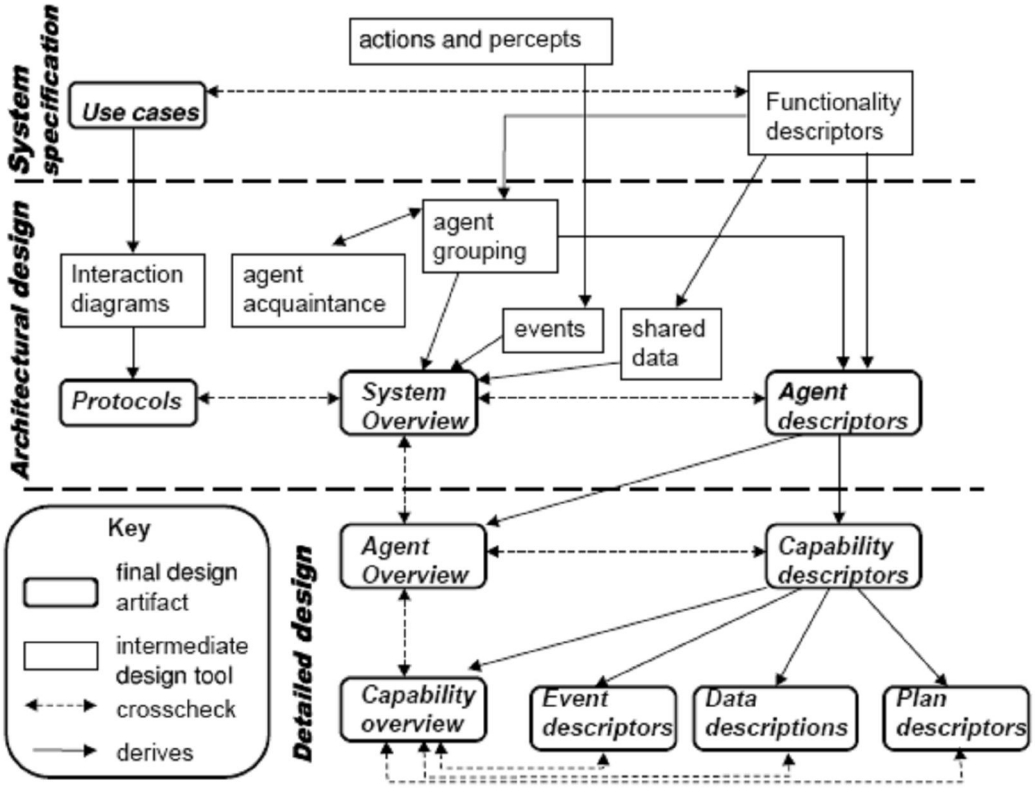


Figura 2.7.32 – Fases da Metodologia Prometheus (PADGHAN, 2002).

2.7.5.1 Especificação do Sistema

Padghan (2006) afirma que nesta fase é necessário definir as interações esperadas com o sistema, os objetivos do sistema, os papéis dentro do sistema a ser projetado, e alguns cenários que fornecem uma noção do que o sistema fará. Também pode ser útil especificar a interface entre o sistema e seu ambiente em termos de percepções e ações.

De acordo com Padghan (2002), consideram-se percepções, nesta metodologia, como a entrada de novas informações do ambiente e ações os mecanismos utilizados para afetar o mesmo. Esta autora destaca ainda a diferença entre percepções e eventos, sendo que um evento é uma ocorrência significativa para o sistema de agente enquanto uma percepção é um dado cru disponível para o sistema de agente. Muitas vezes percepções podem requerer processamento de forma a identificar eventos a que um agente pode reagir. Padghan (2002) exemplifica uma situação em que a câmera de um robô jogando futebol mostra uma bola onde espera-se que esta esteja, neste caso esta percepção não é significante, no entanto se a bola é vista onde não se esperava vê-la, então é significante.

Paralelamente ao descobrir ou especificar as percepções e ações, o desenvolvedor deve iniciar a descrever o que o agente deveria fazer em um senso mais amplo, ou seja, as funcionalidades do sistema, permitindo uma maior compreensão do sistema e seus propósitos. Como funcionalidades, em um sistema de livraria digital, poderia considerar-se algo do tipo “A livraria fornecerá uma interface personalizada aos clientes” ou “A livraria irá manter seu estoque”.

Padghan (2002) destaca ainda a importância de manter as funcionalidades tão restritas quanto possível, lidando com um único aspecto ou objetivo do sistema,

procurando assim evitar equívocos. Ao definir uma funcionalidade é importante também definir a informação que é requerida e a informação produzida por ela. Esta autora recomenda que uma funcionalidade contenha:

- Um nome;
- Uma descrição curta em linguagem natural;
- Uma lista de ações;
- Uma lista de percepções relevantes;
- Dados usados e produzidos;
- Uma breve descrição de interações com outras funcionalidades.

Padghan (2002) fornece um exemplo descrição de funcionalidade referente a um atendente em uma livraria virtual:

Atendente: Fornece uma resposta customizada ao usuário quando ele se loga no site.

Ações: Fornecer link para o status de pedidos existentes, saudar pelo nome, saudar como novo usuário, questionar a satisfação por compras recentes, indicar ofertas especiais relevantes aos interesses.

Percepções: Cliente acessa o site.

Acesso a dados: Lê informação do cliente, ofertas especiais e interações do cliente. Grava interações do cliente.

Interações: Sem interações com outras funcionalidades.

Enquanto as funcionalidades focam-se sobre aspectos particulares do sistema, cenários de casos de uso dão uma visão mais holística do sistema. A idéia básica foi tomada do projeto orientado a objeto. No entanto, aos cenários de caso de uso são dados ligeiramente mais estrutura.

A parte central de um cenário de caso de uso em Prometheus é a seqüência de passos descrevendo um exemplo do sistema em operação. Cada passo é anotado com o nome da funcionalidade responsável, bem como a informação usada ou produzida. Estas anotações permitem a verificação cruzada para consistência com os descritores de funcionalidade.

Os templates (documentação) de caso de uso utilizados contêm um número de identificação, uma breve visão geral em linguagem natural, um campo opcional chamado contexto que indica quando este cenário acontecerá ou o ponto inicial do cenário, o próprio cenário que é uma seqüência de passos, um sumário de toda a informação usada nos vários passos e uma lista de pequenas variações. Devido a um cenário capturar somente uma seqüência particular de passos, ele pode ser utilizado para indicar pequenas variações com uma breve descrição. Qualquer variação maior deve estar em um cenário de caso de uso separado. A figura 2.7.33 apresenta um exemplo de Diagrama de Cenários.

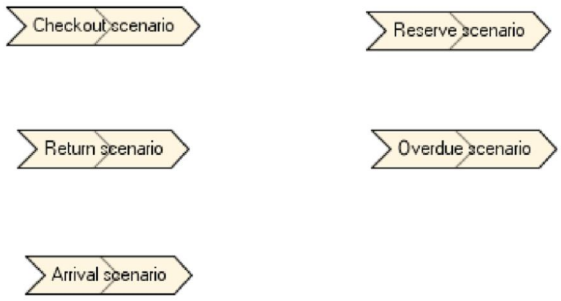


Figura 2.7.33 – Diagrama de Cenários (RMIT, 2004)

Em Winikoff (2004), como pode ser observado na próxima figura, percebe-se que houve algumas mudanças nas fases do método. Na Fase de Especificação do Sistema passou-se a representar também os objetivos do sistema, além dos casos de uso passarem a ser representados no artefato de projeto chamado Cenários e os Descritores de Funcionalidade passarem a ser chamados Descritores de Funcionalidade Inicial. Já na Fase de Projeto Arquitetural, o que em Padghan (2002), era identificado como "Eventos" passou-se chamar "Mensagens". Finalmente na Fase de Projeto Detalhado, passou-se a identificar também Processos a partir dos Protocolos identificados na fase anterior.

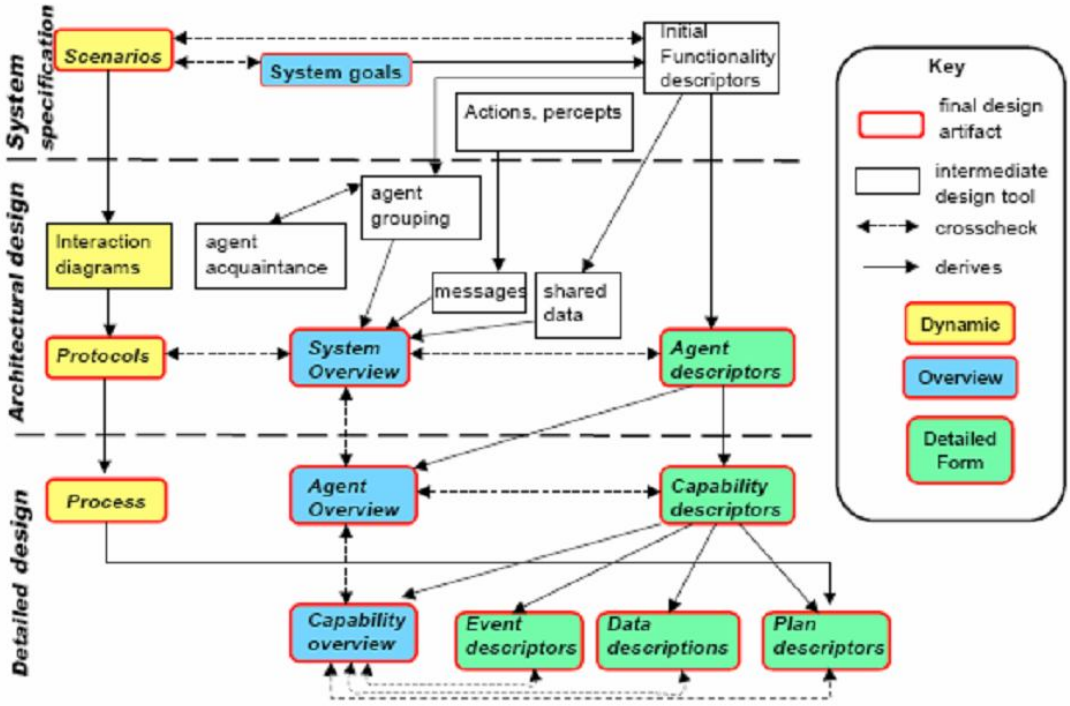


Figura 2.7.34 – Fases da Metodologia Prometheus incluindo os objetivos e processo do sistema (WINIKOFF, 2004)

Cheong (2005) explica que os objetivos do sistema são definidos por meio de um Diagrama de Visão Geral de Objetivo que apresenta todos os objetivos e os relacionamentos objetivo-subobjetivo. Os objetivos são usados como a base para identificar funcionalidades, ou seja, pedaços específicos de comportamento formados pelo agrupamento de objetivos relacionados. Estas funcionalidades são agrupadas para

formar tipos de agentes. A figura seguinte apresenta um exemplo de Diagrama Geral de Objetivos.

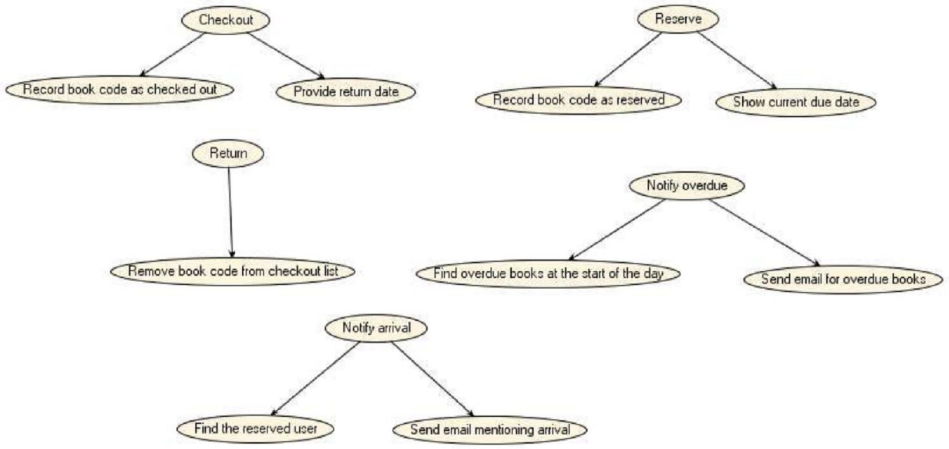


Figura 2.7.35 – Diagrama de Visão Geral de Objetivos (RMIT, 2004)

Após a criação dos diagrama de Visão Geral de Objetivos e Diagrama de Cenários pode-se produzir o Diagrama de Papel do Sistema. Para produzir este diagrama, agrupam-se as diferentes funcionalidades do sistema em diferentes papéis e em seguida as percepções, objetivos e ações são ligadas aos papéis no Diagrama de Papéis do Sistema (RMIT, 2004), conforme demonstra a próxima figura.

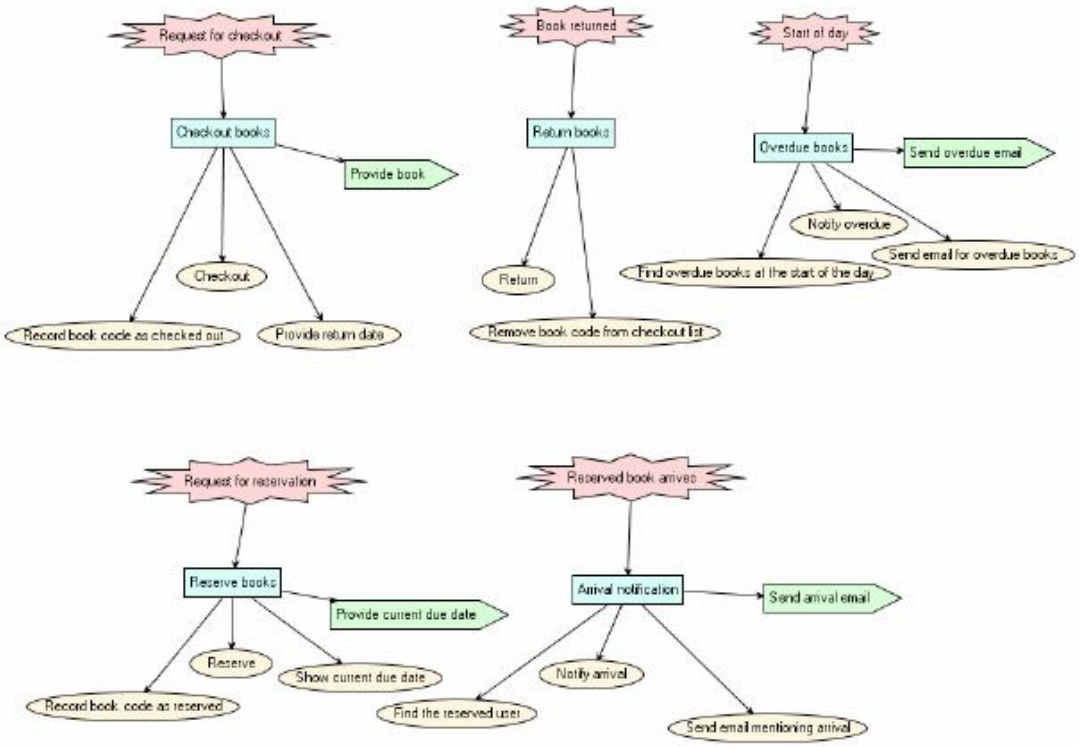


Figura 2.7.36 – Diagrama de Papéis do Sistema (RMIT, 2004)

Em Padghan (2006) percebe-se que posteriormente passou-se a oferecer mais um diagrama na fase de Especificação do Sistema, o Diagrama de Visão Geral de Análise.

O objetivo deste diagrama é desenvolver a visão de alto nível dos requisitos do sistema, sendo usado para:

- Especificar atores chave externos que utilizarão o sistema e cenários chave em que eles estarão envolvidos.
- Ligar os atores aos cenários em que eles estarão envolvidos usando percepções e ações.

Nesta situação Percepções são sinais do ator para o sistema e Ações são respostas do sistema, possivelmente dirigidas a um ator em particular. Este diagrama pode ser usado muito livremente para esboçar um projeto do sistema, onde a grande maioria das entidades (ícones que representam por exemplo percepções e ações, agentes e papéis) utilizadas na metodologia estão disponíveis para serem utilizadas como o usuário desejar, podendo este fazer ligações entre eles sem muitas validações, permitindo que se elabore um rascunho do sistema. Segundo Padghan (2006), uma forma de usar este diagrama é desenvolver um diagrama Ator-Cenário, porém é possível especificar vários outros diagramas e relacionamentos. A figura a seguir apresenta um exemplo de Diagrama de Visão Geral de Análise, onde foi identificado um ator externo que interage com o sistema, através do cenário de Empréstimo. O ator é identificado como “Sócio Biblioteca”.

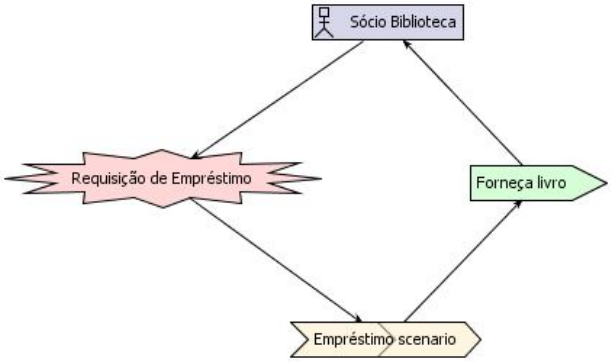


Figura 2.7.37 – Diagrama de Visão Geral de Análise (RMIT, 2004)

2.7.5.2 Projeto Arquitetural

Padghan (2002) afirma que a principal decisão a ser feita durante o projeto arquitetural é definir quais agentes devem existir. Como critério de decisão, utiliza-se a atribuição de funcionalidades aos agentes. O processo de identificar agentes por meio do agrupamento de funcionalidades envolve analisar as razões a favor e contra o agrupamento de funcionalidades particulares. Se funcionalidades usam os mesmos dados, isto é uma indicação para agrupá-las, já que é significante a interação entre elas. Uma razão contra o agrupamento pode ser uma funcionalidade claramente não relacionada. Geralmente procura-se por agentes que tenham forte coerência e fraco acoplamento.

De maneira a avaliar um agrupamento potencial para acoplamento, utiliza-se um “Diagrama de Familiaridade de Agente”. Este diagrama simplesmente liga cada agente com cada um dos agentes com os quais ele interage, conforme pode ser visto na figura 2.7.38, à direita. Um projeto com menos ligações é menos altamente acoplado e, portanto preferido. Prometheus utiliza uma notação consistente para descrever agentes, eventos, planos, capacidades, dados/crenças, etc. conforme pode ser observado na figura 2.7.38 à esquerda.

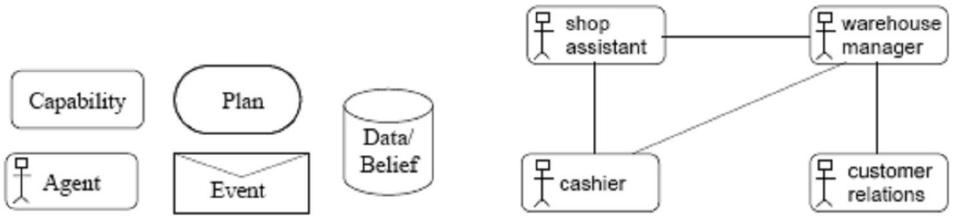


Figura 2.7.38 – Diagrama de Familiaridade de Agente (Agent Acquaintance Diagram) (PADGHAN, 2002).

Uma vez que se tenha decidido quais agentes o sistema deverá conter é possível iniciar a descrever a informação necessária sobre os agentes. A informação de alto nível sobre cada agente é contida no formulário de um descritor de agente, semelhante aos descritores de funcionalidade. Questões que precisam ser resolvidas sobre agentes neste estágio incluem: Quantos agentes de determinado tipo existirão? Qual é o tempo de vida do agente? Se eles são criados ou destruídos durante a operação do sistema, o que acarreta isto? O que necessita ser feito durante a iniciação do agente? Quando da destruição do agente, o que necessita ser feito? Que dados este agente necessita manter? A quais eventos este agente irá reagir?

Descritores de agentes contêm, além destas informações, o nome e a descrição do agente, uma breve descrição das maneiras que ele interagirá com outros agentes e uma lista das funcionalidades que estão incorporadas dentro deste agente.

Com relação aos dados necessários ao agente, pode-se utilizar um Diagrama de Acoplamento de Dados, através do qual identificam-se os tipos de dados que necessitam ser armazenados no sistema como crenças em um repositório de dados, em seguida liga-se este repositório a papéis (funcionalidades) que serão executados pelos agentes (RMIT, 2004). Um exemplo de Diagrama de Acoplamento de Dados pode ser observado na figura a seguir.

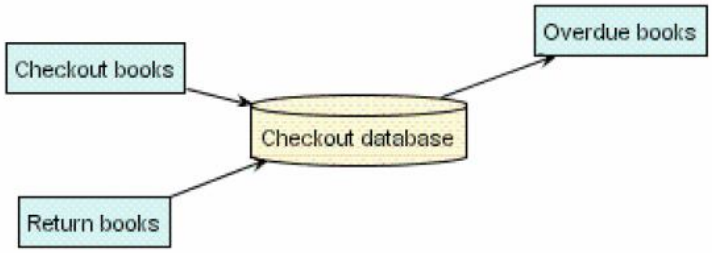


Figura 2.7.39 – Diagrama de Acoplamento de Dados (RMIT, 2004).

Em seguida, os papéis são agrupados e identificados os agentes que irão realizá-los. Isto é feito por meio do Diagrama de Ligação de Papel de Agente, que pode ser observado na próxima figura.

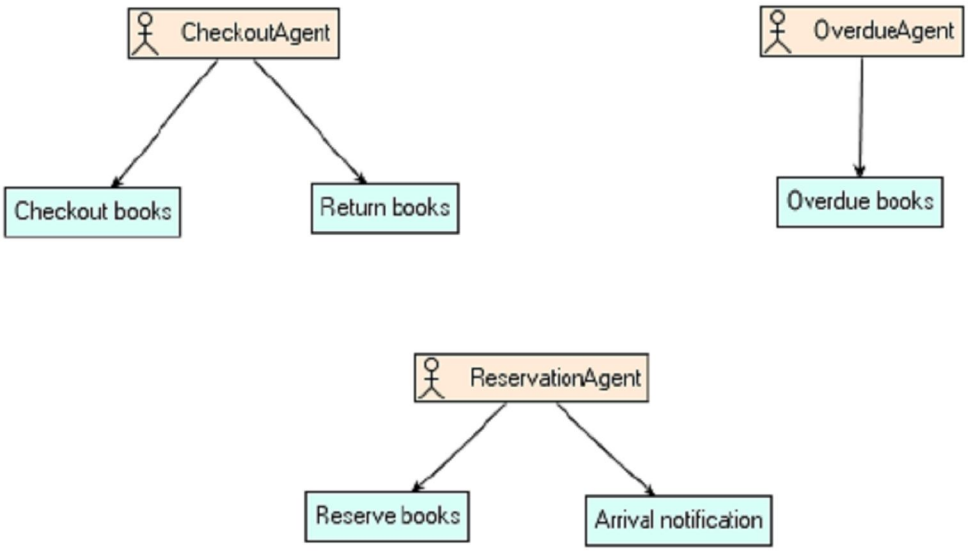


Figura 2.7.40 – Diagrama de Ligação de Papel de Agente (RMIT, 2004).

O Diagrama de Visão Geral do Sistema une os agentes, eventos e objetos de dados compartilhados. Este é possivelmente o artefato mais importante de todo o processo de projeto, embora ele não possa ser totalmente compreendido isoladamente. As figuras 2.7.41 e 2.7.42 apresentam exemplos de Diagramas de Visão Geral do Sistema.

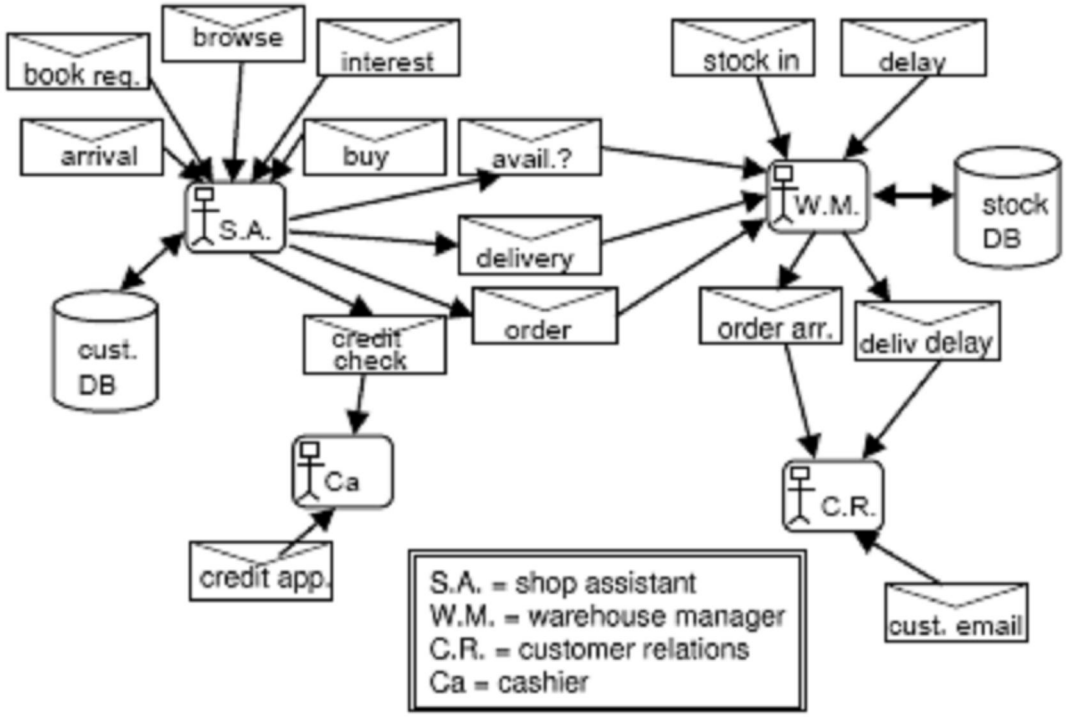


Figura 2.7.41 – Diagrama de Visão Geral do Sistema (PADGHAN, 2002).

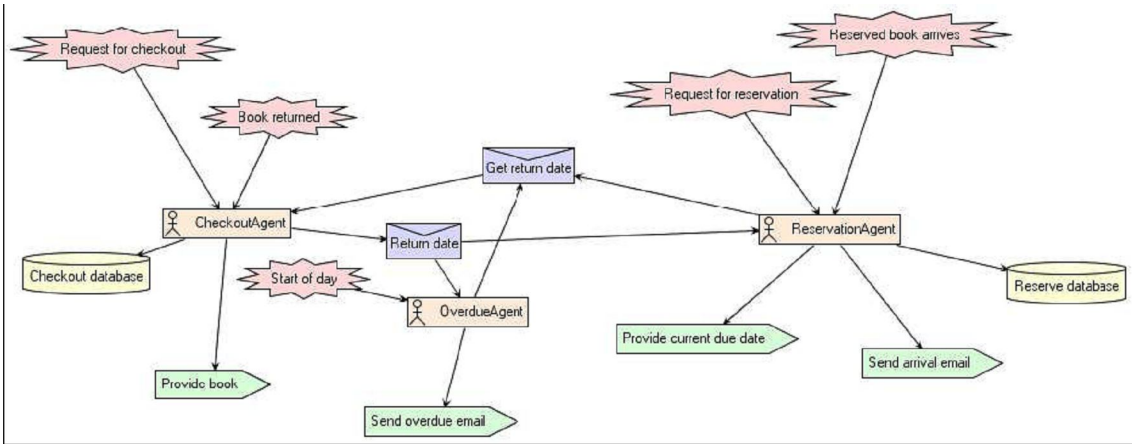


Figura 2.7.42 – Diagrama de Visão Geral do Sistema (RMIT, 2004).

O aspecto final do projeto arquitetural é especificar totalmente a interação entre agentes. Os Diagramas de Interação são usados como uma ferramenta inicial para realizar isto, enquanto protocolos de interação totalmente especificados contituem-se no artefato de projeto final.

Estes diagramas foram tomados diretamente do projeto orientado a objetos, demonstrando interações entre agentes ao invés de objetos. Um dos principais processos para desenvolver Diagramas de Interação é tomar os casos de uso desenvolvidos na fase de especificação e construir os diagramas de interação correspondentes. Sempre que existir um passo no caso de uso que envolva uma funcionalidade de um novo agente, deve haver alguma interação de um agente anteriormente envolvido com o novo agente participante. A figura 2.7.43 à esquerda demonstra um Diagrama de Interação para um cenário de compra de um livro em uma livraria eletrônica.

De maneira a ter um sistema definido precisamente progrediu-se dos Diagramas de Interação para protocolos de interação que definem precisamente que seqüências de interação são válidas dentro do sistema. A figura 2.7.43 (à direita) demonstra um exemplo de Diagrama de Protocolo. Devido aos protocolos terem de mostrar todas as variações, eles são muitas vezes maiores que os Diagramas de Interação correspondentes e podem precisar ser divididos em pedaços menores. A notação AUML é utilizada para especificar protocolos.

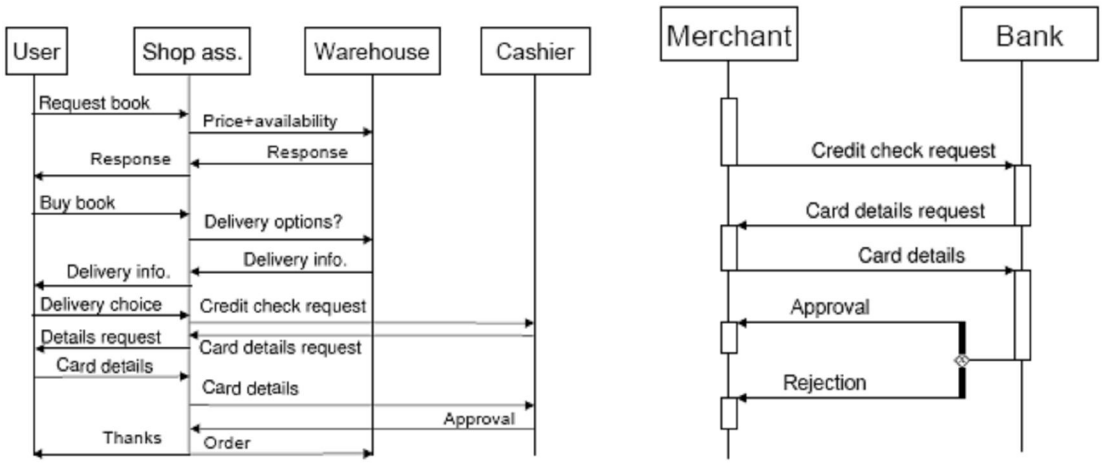


Figura 2.7.43 – Diagrama de Interação (à esquerda) e Diagrama de Protocolo (à direita) (PADGHAN, 2002).

2.7.5.3 Projeto Detalhado

Segundo Padghan (2002), o projeto detalhado foca-se no desenvolvimento da estrutura interna de cada um dos agentes e como irá atingir suas tarefas dentro do sistema. É neste estágio do projeto que a metodologia torna-se específica para agentes que utilizam planos definidos por usuário, disparados por objetivos ou eventos, tais como as várias implementações de sistemas BDI. Um grande número de detalhes a respeito da plataforma de implementação também torna-se evidente neste estágio.

O foco da fase de projeto detalhado está em definir capacidades (módulos dentro do agente), eventos internos, planos e estruturas de dados detalhadas. Um processo de refinamento progressivo é usado, iniciando por descrever as partes internas dos agentes em termos de capacidades. A estrutura interna de cada capacidade é então descrita, opcionalmente usando ou introduzindo capacidades adicionais. Estas são refinadas por sua vez até que todas as capacidades tenham sido definidas. Em um nível mais baixo, as capacidades são definidas em termos de planos, eventos e dados.

As funcionalidades da fase de especificação fornecem um bom conjunto inicial de capacidades, as quais podem ser novamente refinadas se for desejado. Algumas vezes há também funcionalidade semelhante a “rotinas de biblioteca” as quais são necessárias em múltiplos lugares – ou dentro de múltiplos agentes, ou dentro de múltiplas capacidades dentro de um único agente. Tal funcionalidade deve também ser extraída em uma capacidade que pode então ser incluída em outras capacidades ou agentes conforme requerido.

É permitido aninhar capacidades dentro de outras e assim este modelo permite muitas camadas dentro do projeto detalhado, de maneira a atingir uma complexidade compreensível a cada nível.

Cada capacidade deveria ser descrita por um descritor de capacidade que contém informação sobre a interface externa da capacidade – quais eventos são entradas e quais eventos são produzidos (como entradas para outras capacidades). O descritor também contém uma descrição em linguagem natural da funcionalidade, um nome descritivo único, informação relativas a interações com outras capacidades, ou inclusões de outras capacidades, e uma referência a dados lidos e escritos pela capacidade.

O Diagrama de Visão Geral do Agente fornece a visão de alto nível das partes internas do agente. Ele é muito similar em estilo ao Diagrama de Visão Geral do Sistema, mas ao invés de agentes dentro de um sistema, ele mostra capacidades dentro de um agente. Este diagrama mostra as capacidades de alto nível do agente e o evento ou fluxo de tarefa entre estas capacidades, bem como os dados internos do agente. Através da leitura de descritores de capacidade relevantes, junto com o diagrama é possível obter uma visão de alto nível clara de como os módulos dentro do agente interagirão para atingir as tarefas gerais do agente como descrito no descritor de agente do projeto arquitetural. As figuras 2.7.44 e 2.7.45 apresentam exemplos de diagramas de Visão Geral de Agente.

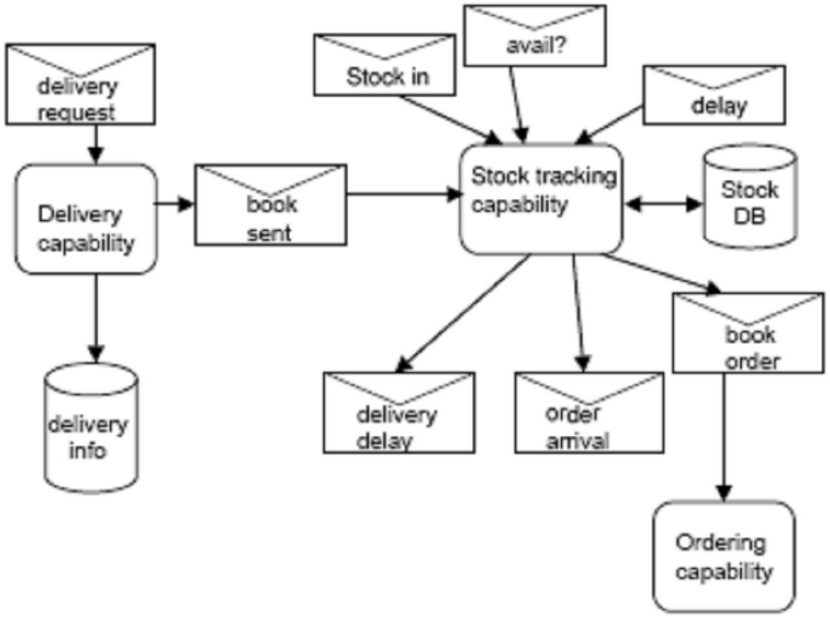


Figura 2.7.44 - Diagrama de Visão Geral de Agente (PADGHAN, 2002)

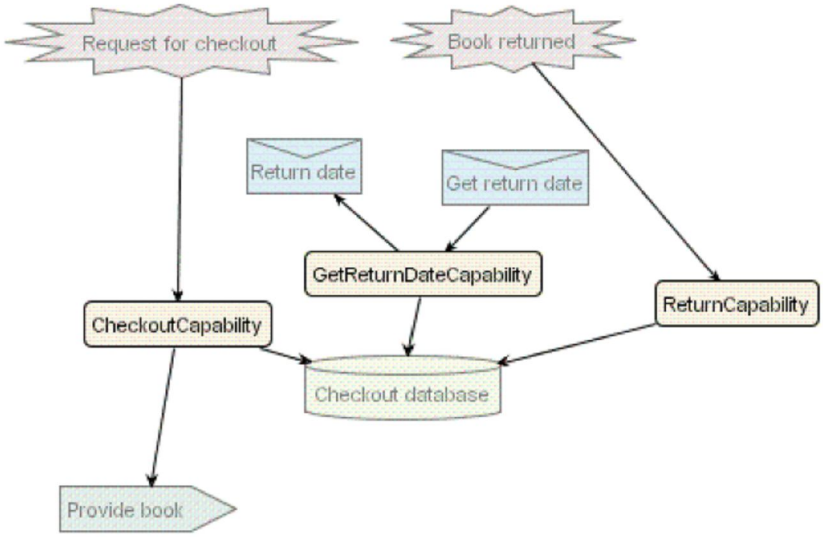


Figura 2.7.45 – Diagrama de Visão Geral de Agente (RMIT, 2004).

Padghan (2002) afirma que um novo nível de detalhe é fornecido por Diagramas de Capacidade que tomam uma única capacidade e descrevem suas partes internas. Em um baixo nível, estas conterão planos com eventos fornecendo as conexões entre planos, bem como entre capacidades e entre agentes. Em níveis intermediários eles podem conter capacidades aninhadas ou uma mistura de capacidades e planos. Estes diagramas são similares em estilo aos Diagramas de Visão Geral de Agente e Visão Geral de Sistema, embora planos sejam obrigados a ter um único evento de entrada. A próxima figura apresenta um exemplo de Diagrama de Capacidade.

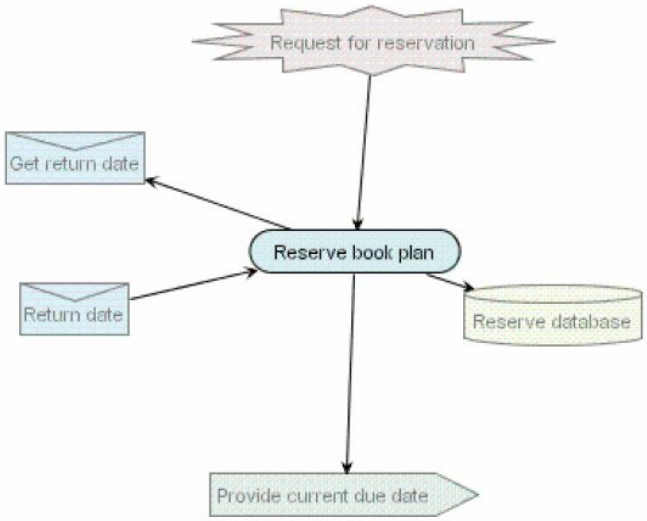


Figura 2.7.46 – Diagrama de Capacidade Contendo Planos (Reserve book plan) (RMIT, 2004).

Da mesma forma que o Diagrama de Visão Geral de Agente deve ser verificado para consistência com a Visão Geral de Sistema (em termos de entrada e saída de eventos), cada Diagrama de Visão Geral de Capacidade deve ser verificado com seu contexto interno – ou a visão geral de agente ou outra visão geral de capacidade.

Os artefatos de projeto final necessários são o plano individual e os descritores de eventos e dados. Estes descritores fornecem os detalhes necessários para passar a implementação. Os detalhes apropriados para estes descritores dependerão de aspectos da plataforma de implementação.

Os descritores de planos utilizados fornecem um identificador, o tipo de evento disparador, os passos do plano, bem como uma curta descrição em linguagem natural, uma especificação de contexto indicando quando este plano deveria ser utilizado e uma lista de dados lidos e escritos.

Os descritores de evento são usados para especificar totalmente todos os eventos, incluindo aqueles inicialmente identificados. O descritor de evento deve identificar o propósito do evento e quaisquer dados que o evento carregue. Também é indicado para cada evento se é esperado que ele seja amparado e se é esperado que ele seja não ambíguo. Um evento é amparado se há sempre no mínimo um plano de manipulação que seja aplicável, ou seja, para qualquer situação, no mínimo um dos planos que combinem terão uma condição de contexto verdadeira. Um evento é não ambíguo se há sempre no mínimo um plano manipulável aplicável.

Descritores de dados devem especificar os campos e métodos de quaisquer classes usadas para armazenamento de dados dentro do sistema. Se estruturas de dados especializadas são fornecidas para manter crenças, estas devem também ser especificadas.

Um artefato adicional que é completado e verificado neste ponto é o dicionário de dados. Este dicionário deve ser iniciado no início do projeto e incrementado a cada estágio. O dicionário de dados é importante para garantir uso consistente de nomes. Uma opção é organizar o dicionário de dados em seções separadas para agentes, capacidades, planos, eventos e dados, organizadas alfabeticamente dentro de seções. A outra opção é ter uma estrutura alfabética de categorias.

Porém, segundo Padghan (2002), o suporte da metodologia Prometheus para o aspecto social de agentes está correntemente focado no denominador comum mais baixo: mensagens e protocolos. Prometheus também não lida muito bem com agentes móveis. Além disso, Prometheus não é baseado em UML, os autores afirmam que isto pode ser um ponto forte ou fraco, forte porque UML é muito centrada em objeto, por outro lado, fraca porque UML é uma notação padrão.

2.7.5.4 Considerações Finais

Originalmente Prometheus utilizava a terminologia “Casos de Uso” e a identificação dos casos de uso era uma etapa da primeira fase. Porém, em Winikoff (2004), esta terminologia foi substituída por “Cenários” (o que é semelhante ao que os casos de uso costumam representar). Em Prometheus a função dos cenários é fornecer uma noção do que o sistema fará. Com a ajuda destes cenários também são identificadas as funcionalidades do sistema Ainda na primeira fase, são ainda identificados os objetivos gerais do sistema e as percepções e ações externas necessárias aos agentes. As chamadas funcionalidades aqui identificadas representam as funções gerais do sistema que são associadas a objetivos gerais.

Embora os cenários e, sob alguns aspectos, as funcionalidades de Prometheus apresentem semelhanças com os casos de uso da UML, a metodologia Prometheus não utiliza a notação UML, tendo desenvolvido uma notação própria. Os autores afirmam que isto pode ser um ponto forte ou fraco, forte porque UML é muito centrada em objetos, por outro lado, fraca porque UML é uma notação padrão.

Dessa forma, embora tenha utilizado alguns conceitos da UML, Prometheus não utiliza e nem adapta o diagrama de casos de uso para a modelagem de requisitos, embora a metodologia identifique os cenários do sistema, o que é um pouco semelhante ao que o diagrama de casos de uso realiza.

Na fase de especificação de Prometheus, que é o equivalente à fase de análise de requisitos, são necessários quatro diagramas diferentes para fazer algo semelhante à nossa proposta.

Porém esta fase esta fase identifica somente os cenários externos, as funcionalidades e objetivos gerais do sistema, os atores externos (mas não os agentes internos propriamente ditos), que utilizarão o sistema e os cenários em que eles estarão envolvidos. As percepções representadas são os eventos causados pelos usuários externos e as ações são as respostas do sistema, não há detalhamento do que acontece dentro de cada cenário. Nesta fase não são identificadas as ações, percepções, objetivos ou planos específicos de cada agente/papel.

Posteriormente em outras fases, os agentes são identificados e ligados às funcionalidades, mas há pouca representação dos agentes com seus objetivos e planos e as condições para que os planos sejam disparados, há diversos diagramas que tratam vários aspectos em separado, mas com tênue ligação entre si.

São identificadas as bases de dados, mas não há detalhamento de como as crenças são armazenadas ou podem ser modificadas. Há pouca representação de como os objetivos se tornam intenções, exceto a identificação de eventos, mas não há maior detalhamento das condições para um objetivo se tornar uma intenção.

Finalmente, Prometheus é complementado pela AUML para o detalhamento de protocolos e não há detalhamento dos passos de cada plano.

2.7.6 Tropos

Segundo Bresciani (2004), a metodologia Tropos pretende suportar todas as atividades de análise e projeto no processo de desenvolvimento de software, desde a análise de domínio da aplicação até a implementação do sistema. Esta metodologia é dividida em cinco fases: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural, Projeto Detalhado e Implementação.

O objetivo da análise de requisitos em Tropos é fornecer um conjunto de requisitos funcionais e não-funcionais para o sistema. A análise de requisitos em Tropos é dividida em duas fases principais: Análise de Requisitos Iniciais e Análise de Requisitos Finais. Ambos compartilham a mesma abordagem conceitual e metodológica. Assim a maioria das idéias introduzidas na análise de requisitos iniciais são utilizadas na análise de requisitos finais. Mais precisamente, durante a primeira fase, o engenheiro de requisitos identifica as pessoas interessadas no domínio e as modela como atores sociais, que dependem um dos outros para atingir objetivos, realizar planos e fornecer recursos. Ao definir estas dependências é possível afirmar o porquê, além de o que e o como, das funcionalidades do sistema e, como um último resultado, verificar como a implementação final combina com as necessidades iniciais.

Na análise de Requisitos Finais, o modelo conceitual é estendido incluindo um novo ator, que representa o sistema e um grande número de dependências com outros atores do ambiente. Estas dependências definem todos os requisitos funcionais e não-funcionais do sistema.

Bresciani (2004) afirma que as fases de Projeto Arquitetural e Projeto Detalhado focam-se na especificação dos sistema, de acordo com os requisitos resultantes das fases anteriores. O Projeto Arquitetural define a arquitetura global do sistema em termos de sub-sistemas interconectados através de fluxos de dados e controle. Sub-sistemas são representados no modelo como atores e as interconexões de dados/controle são representados como dependências. O projeto arquitetural também fornece um mapeamento dos atores do sistema para um conjunto de agentes de software, cada um caracterizado pelas capacidades específicas. A fase de Projeto Detalhado almeja especificar capacidades e interações de agentes. Neste ponto usualmente, a plataforma de implementação já foi escolhida e esta pode ser levada em conta de forma a realizar um projeto detalhado que irá mapear diretamente para o código. A atividade de implementação segue passo a passo, de uma forma natural, a especificação de projeto detalhado dentro da base do mapeamento estabelecido entre os construtos da plataforma de implementação e as noções de projeto detalhado.

2.7.6.1 Conceitos Chave

De acordo com Bresciani (2004), modelos são adquiridos como instâncias de um metamodelo conceitual baseado sobre os seguintes conceitos/relacionamentos:

Ator, que modela uma entidade que tem objetivos estratégicos e intencionalidade dentro do sistema ou do cenário organizacional. Um ator representa um agente físico, social ou de software bem como um papel ou posição. Assume-se agente de software como tendo propriedades tais como autonomia, habilidade social, reatividade e proatividade; e define-se papel como uma caracterização abstrata do comportamento de um ator social dentro de algum contexto ou domínio especilizado de desejo e uma posição representa um conjunto de papéis, tipicamente executados por um agente. Um agente pode ocupar uma posição enquanto uma posição é dita cobrir um papel.

Objetivo, que representa os interesses estratégicos do ator. Bresciani (2004) distingue objetivos sólidos (hard goals) de objetivos nebulosos (soft goals), estes últimos não tem definição clara e/ou critérios para decidir se eles são satisfeitos ou não, são tipicamente usados para modelar requisitos não-funcionais.

Plano, que representa, em um nível abstrato, uma forma de fazer alguma coisa. A execução de um plano pode ser um meio de satisfazer um objetivo.

Recurso, que representa uma entidade física ou informacional.

Dependência, entre dois atores indica que um ator depende do outro para atingir algum objetivo, executar algum plano ou entregar um recurso. O primeiro ator é chamado "depender", o último "dependee" e o objeto em torno do qual a dependência enfoca, "dependum". Em geral, pela dependência de outro ator para um dependum, um ator é capaz de atingir objetivos que ele seria incapaz de atingir sozinho. Ao mesmo tempo, o depender torna-se vulnerável. Se o dependee falha em entregar o dependum, o depender seria adversamente afetado em sua habilidade de atingir seus objetivos.

Capacidade, que representa a habilidade de um ator definir, escolher e executar um plano para o preenchimento de um objetivo, dadas certas condições de mundo e na presença de um evento específico.

Crença, que representa o conhecimento do ator sobre o mundo.

2.7.6.2 Atividades de Modelagem

Segundo Bresciani (2004), várias atividades contribuem para a aquisição de um primeiro modelo de requisito inicial, para seu refinamento e para sua evolução em modelos subseqüentes. Elas são:

Modelagem de ator, que consiste em identificar e analizar os atores do ambiente, os atores do sistema e agentes. Em particular, na fase de requisito inicial, a modelagem de ator foca-se sobre a modelagem dos stakeholders de domínio de aplicação e suas intenções como atores sociais que querem atingir objetivos. Durante o último requisito, a modelagem de ator foca sobre a definição do ator sistema a ser desenvolvido, visto que no projeto arquitetural, foca-se sobre a estrutura do ator do sistema a ser desenvolvido especificando-o em termos de sub-sistemas (atores) inteconectados através de fluxos de dados e controle. Em projeto detalhado, os agentes do sistema são definidos especificando todas as noções requeridas pela plataforma de implementação destino e, finalmente, durante a fase de implementação a modelagem de ator corresponde à codificação do agente.

Modelagem de Dependência, que consiste em identificar atores que dependem um de outro para atingir objetivos, realizar planos e fornecer recursos. Em particular, na fase de requisito inicial, foca-se sobre a modelagem de dependências de objetivo entre atores sociais do ambiente organizacional. Novas dependências são elicitadas e adicionadas ao modelo sobre a análise de objetivo realizada durante a atividade de modelagem de objetivo. Durante a última análise de requisitos, a modelagem de dependência foca-se em analisar as dependências do ator do sistema a ser desenvolvido. Na fase de projeto arquitetural, fluxos de controle e dados entre sub-atores dos atores do sistema a ser desenvolvido são modelados em termos de dependências, fornecendo a base para a modelagem de capacidade. A figura seguinte apresenta um exemplo de diagrama de ator modelando os stakeholders de um projeto. Os atores são representados por círculos, os objetivos concretos por retângulos com as bordas verticais arredondadas e os objetivos nebulosos por nuvens, da mesma forma que os dependum.

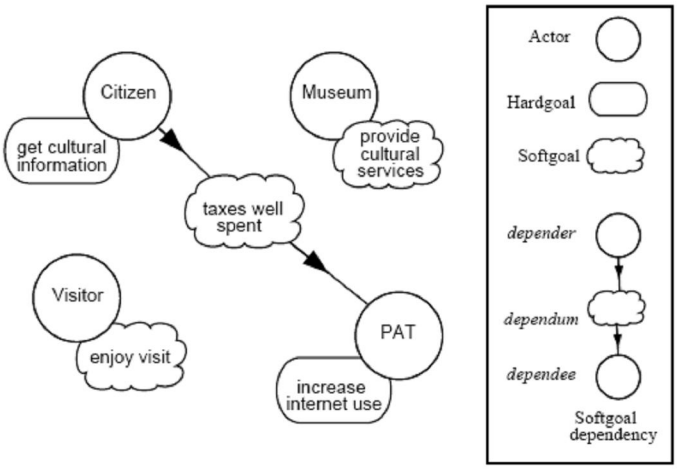


Figura 2.7.47 – Diagrama de Ator modelando os stakeholders de um projeto (BRESCIANI, 2004)

A modelagem de objetivo baseia-se na análise de objetivos de ator, conduzidos do ponto de vista do ator, usando três técnicas básicas de raciocínio: análise de meio-fim, para identificar planos, recursos e objetivos nebulosos que fornecem meios para atingir um objetivo; análise de contribuição, para identificar objetivos que podem contribuir positiva ou negativamente para atingir o objetivo e decomposição e/ou, que decompõe um objetivo raiz em sub-objetivos, modelando uma estrutura de objetivo mais fina. A modelagem de objetivo é aplicada em modelos de requisitos iniciais e finais de forma a refiná-los e para elicitar novas dependências. Durante o projeto arquitetural, ela contribui para motivar a primeira decomposição dos atores do sistema a ser desenvolvido em um conjunto de sub-atores. A figura a seguir apresenta um exemplo de modelagem de objetivo.

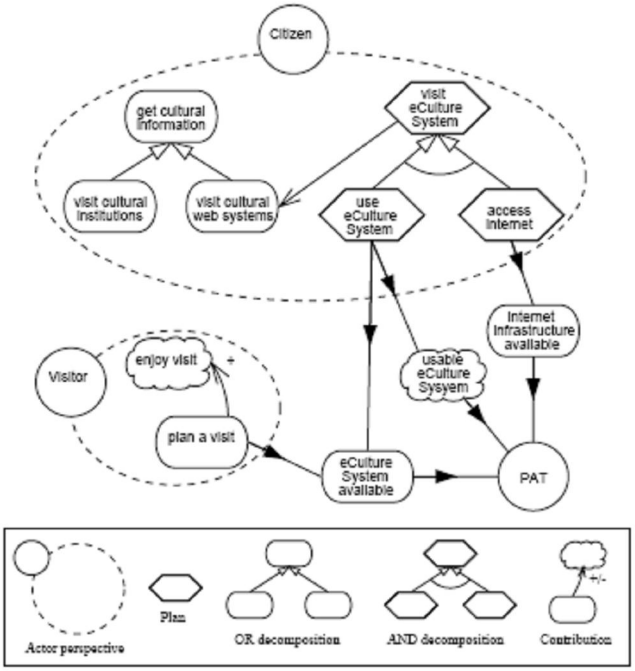


Figura 2.7.48 – Exemplo de Diagrama de Objetivo para Modelagem de Objetivo (BRESCIANI, 2004)

A modelagem de plano pode ser considerada como uma técnica de análise complementar à modelagem de objetivo. Ela baseia-se em técnicas de raciocínio análogas àquelas usadas na modelagem de objetivo. A próxima figura apresenta um exemplo de modelagem de plano.

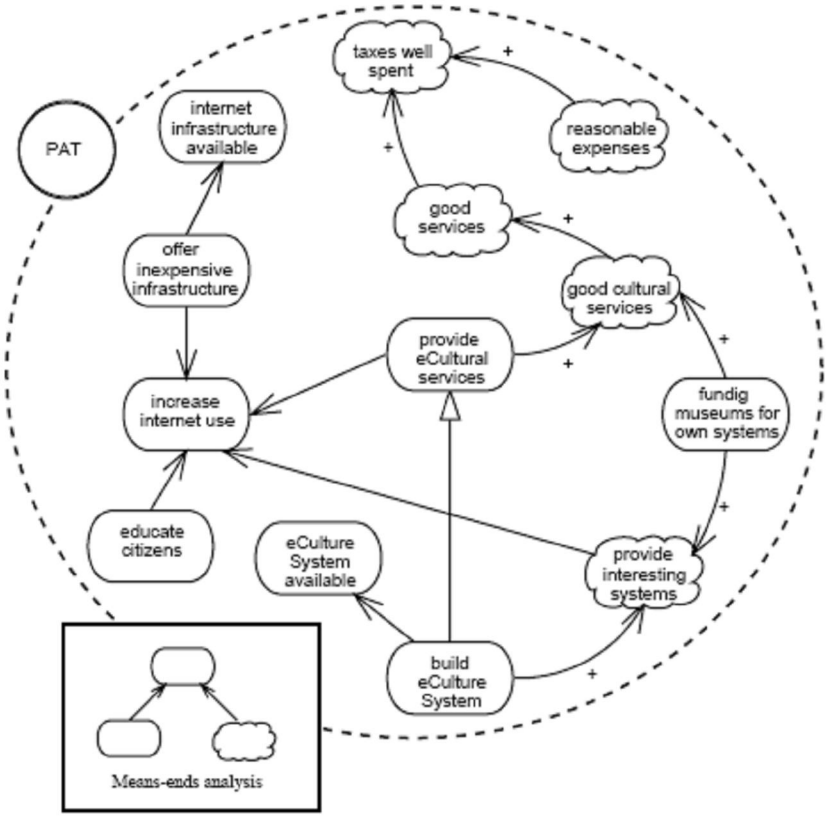


Figura 2.7.49 – Exemplo de Diagrama de Objetivo para Modelagem de Plano (BRESCIANI, 2004).

A Modelagem de Capacidade, inicia-se no final do projeto arquitetural quando os sub-atores de sistema foram especificados em termos de seus próprios objetivos e as dependências com outros atores. De forma a definir, escolher e executar um plano para atingir seus próprios objetivos, cada sub-ator do sistema tem de ser provido com capacidades "individuais" específicas. Capacidades "sociais" adicionais deveriam também ser fornecidas para gerenciamento de dependências com outros atores. Objetivos e planos previamente modelados tornam-se parte integral das capacidades. Em projeto detalhado, cada capacidade de agente é novamente especificada e então codificada durante a fase de implementação. Estas capacidades são representadas graficamente pelos diagramas de capacidade e plano que utilizam diagramas de atividade da UML e diagramas de interação da AUML.

2.7.6.3 Considerações Finais

Tropos permite a modelagem de atores que, nesta metodologia representam entidades que tem objetivos estratégicos e intencionalidade dentro do sistema ou do cenário organizacional. Um ator pode representar um agente físico, social ou de software bem como um papel ou posição. Tropos permite também modelar os objetivos dos atores, tanto hardgoals como soft-goals, estes últimos são objetivos cujas condições

para serem satisfeitos são nebulosas e não totalmente definidas. A metodologia modela ainda dependências, que consistem em identificar atores que dependem uns dos outros para atingir objetivos, realizar planos e fornecer recursos.

Apesar de fortemente centrado na modelagem de requisitos, Tropos não utiliza a notação UML, tendo criado uma notação própria e, portanto não utiliza o diagrama de casos de uso para a modelagem de requisitos. Além disso, apesar de representar os objetivos e planos dos atores, Tropos não representa graficamente, por exemplo, as percepções e ações necessárias a um agente. Os objetivos são identificados, mas não há detalhamento de como eles se tornarão intenções. Além disso, Tropos não faz diferenciação entre atores internos e externos e nem se são cognitivos ou reativos, além de não diferenciar agentes de papéis de agente.

2.7.7 Resumo das Características das Linguagens e Metodologias Estudadas Comparadas ao Metamodelo

Tabela 2.7.1: Resumo das Linguagens e Metodologias Comparadas ao Metamodelo

Linguagem/ Metodologia	Possui mecanismos para representação de requisitos para SMA por meio de uma notação gráfica.	Representação gráfica de agentes/papéis por meio do DCU adaptado ou não (ou por meio de outra notação na fase de AR ou equivalente em metodologias que não utilizam UML)	Representação gráfica de objetivos de agentes /papéis por meio do DCU adaptado ou não (ou por meio de outra notação na fase de AR ou equivalente em metodologias que não utilizam UML)	Representação gráfica de percepções ou ações de agentes por meio do DCU adaptado ou não (ou por meio de outra notação na fase de AR ou equivalente em metodologias que não utilizam UML)	Representação gráfica de planos de agentes por meio do DCU adaptado ou não (ou por meio de outra notação na fase de AR ou equivalente em metodologias que não utilizam UML)
AUML	–	–	–	De certa forma representa ações e percepções se considerarmos o envio e recebimento de mensagens como tal, mas não por meio do DCU e estas percepções e ações não são identificadas como requisitos	–
AORML	Apenas requisitos externos por meio do DCU padrão	Somente atores externos	–	Representa ações e percepções mas não por meio do DCU e estes são aplicados no detalhamento de interações.	–
AML	Casos de uso representando requisitos externos associados a classes-objetivo	Somente atores externos	Representa objetivos por meio de classes estereotipadas e os associa a casos de uso normais	Representa percepções e ações por meio de associações de dependência estereotipadas, mas não por meio do DCU e não as identifica como requisitos	Representa planos por meio do Diagrama de Atividades, mas não por meio do DCU e não os identifica como requisitos.
MAS-ML	–	–	Representa objetivos como um tipo de atributo estereotipado, mas não por meio do DCU e estes não são identificados como requisitos	Representa ações e percepções, mas não por meio do DCU e estas não são identificados como requisitos	Representa planos, mas não por meio do DCU e estes não são identificados como requisitos
Metamodelo de Silva (2007)	–	–	Representa objetivos como classes estereotipadas, mas não por meio do DCU e estes não são identificados como	Representa ações e, de certa forma, percepções, mas não por meio do DCU e estas não são identificadas como	Representa planos como classes estereotipadas e os detalha por meio de diagramas de atividade estendidos, mas não

			requisitos	requisitos	por meio do DCU e estes não são identificados como requisitos
Gaia	–	–	–	–	–
MaSE	Requisitos externos, representados por meio de objetivos gerais do sistema e por casos de uso detalhados por diagramas de seqüência	Adapta a UML mas não utiliza o DCU para identificar papéis. Os papéis são associados aos objetivos gerais do sistema e depois mapeados em classes de agente.	Identifica objetivos gerais a nível de sistema.	Identifica percepções e ações, em termos de recebimento e envio de mensagens, mas não por meio do DCU e não os identifica como requisitos.	–
Message/UML	Identifica somente os objetivos gerais do sistema e não por meio do DCU	Não utiliza o DCU para identificar papéis. Os papéis são inicialmente descritos textualmente e em fases posterioes representados por meio de classes estereotipadas.	Os objetivos são identificados e associados a papéis, porém não por meio do DCU.	Identifica ações de comunicação e ações diretas e, de certa forma, identifica as informações de uma entidade como percepções de um agente, porém não por meio do DCU e não como requisitos.	Não identifica planos, mas sim tarefas, que são utilizadas para atingir objetivos. Um objetivo pode ser atingido pela execução de diversas tarefas, porém não são definidas as condições para que um objetivo se torne uma intenção e as tarefas não são identificadas como requisitos e nem representadas pelo DCU
PASSI	Somente os requisitos externos	Os agentes são identificados como pacotes de casos de uso	Não identifica objetivos, somente as tarefas de um agente/papel	–	Não identifica os planos de um agente ou papel, embora as tarefas de um agente possam ser detalhadas por meio de diagramas de atividade.
Prometheus	Funções gerais do sistema associadas a objetivos gerais, percepções causados por atores externos e ações de resposta do sistema. Identificação de papéis baseados nos objetivos gerais, percepções e ações externas.	Agentes são identificados, mas em fases posteriores, não na fase de especificação que corresponde à de análise de requisitos	Representa somente os objetivos gerais do sistema.	Representa percepções em termos de eventos causados por atores externos e as ações externas do sistema	Não representa planos na fase de especificação de sistema.
Tropos	Identifica atores (agentes físicos, sociais ou de software) e seus objetivos e planos.	Identifica atores que são equivalentes a agentes/papéis, porém não diferencia atores externos de internos nem se são cognitivos ou reativos.	Os objetivos são identficados	–	Identifica os planos associados a um ator, mas não detalha as condições para que um objetivo se torne uma intenção e dispare um plano.

2.8 Princípios de Projeto Propostos por Vicari (2007)

Nesta seção discorreremos sobre os princíos de projeto propostos por Vicari (2007). Nós criamos essa seção devido ao nosso metamodelo ter sido adaptado a partir de sua proposta original para suportar alguns desses princípios.

Segundo Vicari (2007), de uma perspectiva de aplicações STI (Sistemas Tutores Inteligentes), sistemas multiagentes deveriam ser projetados considerando três elementos básicos: que domínio este sistema tentará ensinar, que modelo será usado para compreender os alunos e que relacionamentos pedagógicos deveriam ser estabelecidos ou evitados entre o sistema e os aprendizes.

O domínio de aprendizado constitui o foco de interesse dos aprendizes que são os usuários do sistema. O problema que um STI deve resolver é: “como ajudar aprendizes a aprender tópicos deste domínio?”. Modelos de aprendiz representam o conhecimento que o sistema tem sobre entidades externas que interagem com ele. Relacionamentos pedagógicos podem ser considerados relacionamentos sociais que deveriam ser estabelecidos ou evitados entre o sistema e seus usuários.

Com base nessas afirmações Vicari (2007), sugere diversos princípios de projeto (Design Principles – DP), descritos a seguir:

(DP. 1) Divisão do conhecimento de domínio de aplicação:

O primeiro princípio de projeto define como o domínio de aplicação deveria ser dividido pelo menos em três grandes sub-domínios. Este princípio também estabelece a divisão do ambiente onde o agente irá trabalhar. A figura seguinte mostra os três sub-domínios distintos em que o conhecimento sobre o domínio de aplicação deveria ser dividido, a saber:

- Conhecimento sobre o foco de interesse dos usuários em relação ao domínio de aplicação. Este é o conhecimento relacionado a problemas que se pretende que a aplicação resolva. Este é o conhecimento de solução de problema (Problem Solving – PS) da aplicação.
- Conhecimento sobre entidades que irão interagir com a aplicação. Isto inclui conhecimento sobre os usuários da aplicação (agentes humanos reais) e também de outros agentes artificiais que irão usar ou ser usados (interagir) com agentes da aplicação. Este é o conhecimento de modelagem de usuários e agentes (Users and Agents Modelling – UAM) a ser usado na aplicação.
- Conhecimento sobre o mecanismo de interação social entre agentes. Este conhecimento irá definir como mediar produtivamente as interações sociais entre o usuário e os agentes externos e agentes de aplicação, e entre os agentes internos da aplicação. Este é o conhecimento relacionado às Interações Sociais Mediadas (Social Mediated Interactions – SMI) que ocorrem na aplicação.

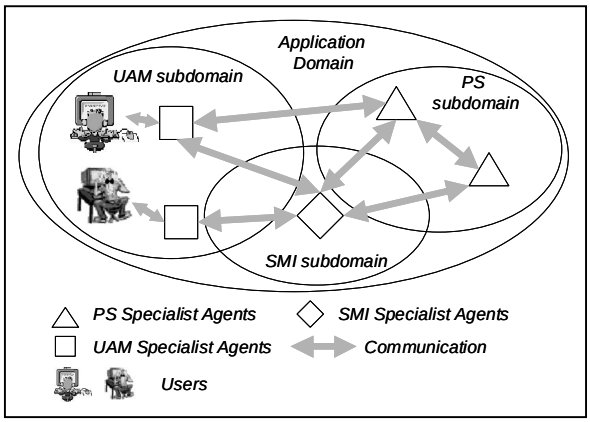


Figura 2.7.1: Sub-domínios de Aplicação e Tipos de Agente (VICARI, 2007).

(DP. 2) Tipos de Agentes:

Vicari (2007), assume que cada um destes sub-domínios deveria estar sob a responsabilidade de um tipo diferente de agente. Assim, um tipo específico de agente irá incorporar o conhecimento de cada um destes sub-domínios:

- O conhecimento sobre o problema que a aplicação irá ajudar a resolver é incorporado nos agentes especialistas PS (Problem Solving).
- O conhecimento sobre a modelagem de usuários e agentes será incorporado por um tipo de agente responsável em compreendê-los e fazer modelos cognitivos de usuários e de outros agentes. Estes são os agentes especialistas UAM (Users and Agents Modelling).
- O conhecimento sobre como mediar interações sociais entre o usuário e o sistema estará sob responsabilidade dos agentes especialistas SMI (Social Mediated Interactions).

(DP. 3) Propósitos do Agente:

A divisão do conhecimento de domínio e a definição de agentes para manipular estas sub-divisões estão diretamente relacionadas à definição de como estes requisitos da aplicação serão satisfeitos por estes agentes.

Pretende-se que o conjunto de requisitos que expressam os propósitos da aplicação seja manipulado pelos agentes. Eles precisam ser divididos em três sub-conjuntos de desejos de alto nível de forma a ser atribuídos a cada tipo de agente estabelecido pelo princípio 2 (DP. 2).

(DP. 4) Conhecimento do Agente

Bases específicas de crenças devem ser atribuídas para sub-domínios PS, UAM e SMI. Cada uma destas bases deveria incluir os seguintes tipos de conhecimento:

- Conhecimento sobre os elementos do sub-domínio.
- Conhecimento sobre as habilidades, ações e percepções do agente a respeito destes elementos.
- Conhecimento sobre planejamento e resolução de problemas.

Isto implica que agentes PS, UAM e SMI irão precisar ter bases de crença customizadas de forma a manipular os diferentes tipos de questões manipuladas em cada sub-domínio. Para agentes trabalhando com o sub-domínio PS, a base de crença deve então incluir:

- Definição de como os elementos do domínio, considerados importantes para os usuários, serão representados.
- Definição de quais habilidades cada agente tem em relação a estes elementos, que ações estes agentes podem realizar sobre estes elementos e o que eles podem perceber deles.
- Especificação de quais problemas ocorrem com estes elementos que são do interesse dos usuários e que estratégias, táticas e heurísticas deveriam ser usadas para fazer planos de forma a resolver estes problemas, e quais são as responsabilidades de cada agente no processo de resolução de problema.

Agentes trabalhando com o sub-domínio UAM precisam de bases de crença que incluam:

- Definição de quais propriedades, incluindo crenças, desejos e intenções serão usadas para modelar usuários e outros agentes.

- Especificação de que percepções serão usadas para detectar estas propriedades e que ações são relacionadas a esta modelagem. Definição das estratégias de raciocínio, táticas e heurísticas necessárias para fazer esta modelagem, ou seja, definição do conhecimento necessário para fazer diagnósticos acurados sobre o estado corrente do modelo.
- Definição sobre que decisões do agente serão afetadas pelos modelos de outros agentes e que mecanismos de planejamento e resolução de problemas serão afetados por estas decisões.

Finalmente, bases de crença dos agentes trabalhando com o sub-domínio SMI precisam incluir:

- Definição sobre que relacionamentos sociais deveriam ser perseguidos (relacionamentos sociais produtivos) e que relacionamentos sociais deveriam ser evitados (relacionamentos sociais negativos) com usuários e agentes externos.
- Especificação de que conhecimento é necessário para diagnosticar o estado corrente destes relacionamentos sociais. A avaliação de um relacionamento social com o usuário ou outro agente externo deveria ser feita levando em consideração o conhecimento de agentes especialistas PS e UAM. Deveria-se considerar o estado corrente do ambiente, como percebido e compreendido pelos agentes PS, e considerar o modelo de usuário corrente ou modelo de agente externo, como percebido e compreendido pelos agentes UAM.
- Especificação de que estratégias, táticas e heurísticas são necessárias para maximizar a chance que relacionamentos sociais produtivos sejam estabelecidos e minimizar a chance que relacionamentos negativos possam ser formados. Da mesma forma como o caso anterior, o conhecimento de agentes PS e UAM deveriam ser usados quando necessário.

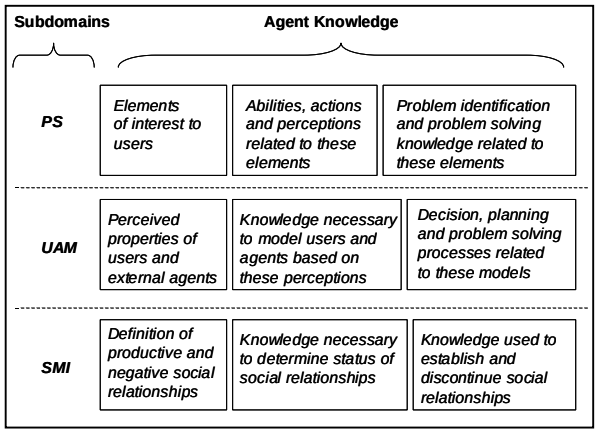


Figura 2.7.2: Divisão de Conhecimento de Agentes (VICARI, 2007)

(DP. 5) Linguagens de Comunicação de Agente

Os princípios de projeto (DP. 2) a (DP. 4) especificam a estrutura básica da arquitetura multiagente do sistema. Eles definem que tipos de agentes deveriam ser usados na arquitetura, que tipos de propósitos deveriam ser atribuídos a cada tipo de agente e o que estes agentes deveriam saber para atingir estes propósitos. No entanto, estes agentes precisam comunicar-se entre si de forma a atingir seus propósitos. Portanto, a arquitetura multiagente não está completa se não estão especificados que princípios guiarão esta comunicação.

Primeiro é necessário definir o meio de comunicação: toda comunicação que ocorre entre agentes do sistema deve ser realizada através de uma linguagem ACL; todo conhecimento será encapsulado e transportado por uma linguagem satisfatória.

A comunicação com usuários e agentes externos também serão modelados pelos atos comunicativos.

(DP. 6) Ontologias

Agentes comunicam crenças para outros agentes, onde mesmo comandos ou requisições tem suas intenções relacionadas, e intenções são também crenças. Nem todas as crenças de agentes tornam-se necessárias comunicar a outros agentes. Da mesma forma, nem todas as crenças informadas pelos outros agentes serão necessariamente compreendida s ou significativas para um agente particular. Assim é necessário definir o subconjunto de crenças que serão compartilhadas e compreendidas em termos de atos comunicativos por cada agente do sistema. Isto inclui toda comunicação que ocorre entre agentes dentro do sistema e toda a comunicação entre o sistema e seus agentes ou usuários externos.

Os subconjuntos de crenças, compartilhadas entre os agentes do sistema, usuários e agentes externos são especificados na ontologia a ser usada no sistema. Estes subconjuntos constituem as crenças compartilhadas do sistema.

Na arquitetura multiagente proposta por Vicari (2007), a ontologia do sistema deveria especificar o conjunto de crenças trocada entre os tipos de agentes SMI, PS e UAM. Ela deveria também definir crenças trocadas entre estes agentes e as interfaces associadas aos usuários e aos agentes externos.

(DP.7) Protocolos de Interação

Segundo Vicari (2007), agentes comunicam informação e conhecimento através da troca de atos comunicativos. Esta troca é governada por meio de protocolos de interação.

Protocolos de interação especificam uma conversação entre dois ou mais agentes. Estes agentes são os participantes do protocolo de interação. Agentes participam em algum protocolo de interação com a intenção de atingir algum propósito ou satisfazer algum desejo. O protocolo de interação é projetado como meio para permitir que os agentes participantes atinjam propósitos específicos. O propósito de cada agente define seu papel no protocolo.

Todas as interações possíveis que ocorrem entre os tipos de agentes SMI, PS e UAM deveriam ser especificadas através de um protocolo de interação apropriado. Da mesma forma, as interações que podem ocorrer entre estes tipos de agentes e as interfaces de agentes externos e usuários deveriam ser especificadas pelos protocolos de interação apropriados.

(DP.8) Arquitetura de agentes individuais

De acordo com Vicari (2007), a definição de como um agente individual pode compreender o conhecimento que recebe ou como ele pode processar este conhecimento para executar ações para atingir seus propósitos, constitui sua arquitetura de agente. As arquiteturas de agente informam como agentes individuais são construídos e como eles trabalham.

Arquiteturas de agente são geralmente divididas em diversas camadas de abstração distintas. Em termos de agentes BDI cognitivos cada camada incorpora crenças, desejos,

intenções e técnicas de solução de problema que podem ser claramente diferenciadas de elementos similares de outras camadas. Desta forma, cada camada fornece uma visão abstrata distinta em termos de projeto de agente, reduzindo a complexidade deste projeto. Estas camadas levam em conta a divisão de conhecimento e processos de raciocínio necessários para:

a) registrar percepções e executar ações elementares (a camada concreta);

b) reconhecer o estado corrente do mundo e planejar ações para um estado desejado de situação; e

c) assumir intenções e tomar decisões apropriadas para atingir um desejo particular.

Seguindo este raciocínio, (Vicari, 2007) baseada no framework de (DILLENBOURG, 1992) propõe que a arquitetura de cada tipo de agente definida no princípio de projeto II possa ser dividida nas seguintes camadas:

- Camada de Julgamento e Decisão (Judgment and Decicion Layer - JDL): a camada mais alta do agente que incorpora os propósitos do agente, que são o conjunto de desejos de mais alta prioridade do agente. Ela acumula os principais processos de decisão do agente em respeito a estes desejos incluindo as atividades de julgamento e diagnóstico relacionadas ao próprio status cognitivo do agente e o que acontece no mundo externo. Decisões tomadas nesta camada tornam-se novas intenções do agente.
- Camada de Avaliação e Planejamento (Assessment and Planning Layer - APL): Uma camada intermediária responsável por planejar e realizar as decisões que o agente toma. Além disso, considerando que o agente deve ser muito bem informado de forma a tomar as decisões corretas, esta camada é também responsável por planejar e executar informações (evidências), acumulando ações necessárias para o nível mais alto. É nesta camada onde a avaliação do status corrente do mundo é realizada.
- Camada de Percepção e Ação (Perception and Action Layer - PAL): O nível mais básico responsável por todas as atividades diretas de interação do agente com o mundo externo. Neste nível, é conhecido o que são as ações elementares e percepções básicas para o agente e como executar estas ações e detectar as percepções.

Não há necessidade de especificar uma camada de abstração social em agentes PS, UAM e SMI porque as questões sociais são manipuladas diretamente pelos propósitos de cada um destes tipos de agentes. Na arquitetura multiagente proposta por Vicari (2007), relacionamentos sociais são estabelecidos entre o sistema e os usuários ou outros agentes externos. Estes estados do relacionamento dependem do conhecimento e informação obtidos pelos agentes especialistas UAM e PS. O estabelecimento, reforço e término destes relacionamentos são responsabilidades dos agentes especialistas SMI.

O fluxo básico de informação entre camadas é cíclico:

1. O agente descobre informação e conhecimento sobre o mundo externo através de percepções e ações realizadas na camada PAL.

2. Parte ou toda esta informação é enviada para a camada APL que, depois de um processo de coleta e classificação, diagnostica a situação e a informa para a camada JDL.
3. O agente na camada JDL faz julgamentos sobre a informação, toma decisões e assume novas intenções.
4. A intenção é enviada para a camada APL que, depois de planejar e organizar, executa-a e define que ações devem ser realizadas para cumprir uma decisão.
5. Muitas dessas ações envolvem interações com elementos do mundo externo como comunicação com outros agentes e, embora eles sejam selecionados na camada APL, é a camada PAL que os realiza.

3 UM METAMODELO UML VOLTADO PARA A MODELAGEM DE REQUISITOS EM PROJETOS DE SISTEMAS MULTI-AGENTES

3.1 Noções Básicas de Agência

Para o propósito deste trabalho, assume-se que um agente é um processo computacional, situado em um ambiente, projetado para atingir um propósito através de um comportamento autônomo e flexível. O ambiente é o domínio da aplicação onde o agente irá trabalhar para atingir seus propósitos. Assume-se ainda que o propósito de um agente pode ser totalmente especificado pela definição de suas crenças e desejos e que o comportamento deste agente é fortemente influenciado por suas intenções (VICARI, 2007).

Jennings (1995) e Franklin (1996) destacam algumas características importantes para um agente de acordo com o contexto do agente e seu papel no sistema:

- Autonomia: um agente é capaz de agir sem intervenção humana direta de acordo com seu próprio controle e estado interno.
- Habilidade Social: um agente é capaz de interagir com outro agente, auxiliando ou pedindo auxílio para seus objetivos.
- Reatividade: um agente pode perceber e reagir a qualquer alteração no ambiente.
- Proatividade: um agente pode agir por conta própria, não precisando de estímulos externos.

Agentes podem perceber eventos que ocorrem sobre o ambiente e realizar ações sobre o mesmo. Agentes podem ser classificados em reativos ou cognitivos. Agentes reativos apenas reagem a eventos, enquanto agentes cognitivos costumam ser mais avançados, possuindo crenças, desejos e intenções sobre o ambiente em que se encontram.

Como nosso grupo de pesquisa concentra-se fortemente em agentes cognitivos e na arquitetura BDI, adotamos a definição de Bratman (1999), o qual afirma que o propósito de um agente pode ser totalmente especificado pela definição de suas crenças e desejos e o comportamento deste agente é influenciado por suas intenções.

Crenças representam o conhecimento do agente. Este conhecimento pode mudar e define o que o agente sabe sobre o mundo. Agentes podem atualizar suas crenças para refletir mudanças detectadas no ambiente. Desejos especificam o conjunto de estado de crença que o agente eventualmente quer produzir. O modelo cognitivo de agentes supõe que o propósito do agente está explicitamente declarado com o conjunto de desejos de prioridade mais alta do agente. As ações do agente são dirigidas por intenções que são

caracterizadas por uma escolha de um estado de coisas a atingir, e um compromisso para fazer esta escolha (COHEN, 1990). O agente escolherá aqueles desejos que são possíveis, de acordo com suas crenças.

O relacionamento do agente com o mundo é realizado pelas ações e percepções. Uma ação é uma mudança causada diretamente pelo agente. Portanto, o agente deveria saber, por meio de crenças, os efeitos produzidos por suas ações e como estas ações estão relacionadas a suas intenções.

As mudanças ocorrem independentemente do agente ou em resposta a algumas ações executadas pelo agente ou pelos outros agentes. Assim, as percepções do agente produzem uma atualização da base de crenças do agente. No entanto, a atualização exata produzida por uma percepção particular depende do estado corrente das crenças do agente.

Os desejos de um agente podem ser traduzidos em objetivos. Estes objetivos possuem planos associados, onde são determinados os passos para atingir estes objetivos. No momento em que o agente acredita que possa realizar um objetivo (desejo) este passa a se tornar uma intenção e são executados os planos associados ao objetivo em questão.

Um agente pode assumir papéis de agente. Um papel de agente representa uma função que o agente pode assumir no sistema. Um agente pode assumir mais de um papel de agente, não necessariamente ao mesmo tempo.

Trencansky (2005) declara que papéis são usados para definir um repertório comportamental normativo de entidades e assim fornecer os blocos de construção básicos de sociedades multiagentes. Já Silva (2004a) afirma que um papel de agente preocupa-se em guiar e restringir o comportamento autônomo de um agente pela descrição de seus objetivos, crenças e ações (nós acreditamos que deveria-se acrescentar também percepções, uma vez que estas ajudam a definir/alterar as crenças de um agente, além de ser por meio delas que o agente consegue determinar o que ocorre no mundo), que o agente deve e que pode realizar quando interpretando o papel. Finalmente, de acordo com Silva (2007), cada agente pode interpretar um ou mais papéis de agente e um agente que execute um papel de agente tem a intenção de atingir os objetivos do papel. Quando um agente compromete-se com um papel, compromete-se a atingir objetivos e executar planos associados ao papel em questão.

3.2 Engenharia de Requisitos

Zave (1997) afirma que engenharia de requisitos é o ramo da engenharia de software que se dedica a determinar os objetivos do mundo real, as funções e as restrições do software, bem como em determinar o relacionamento destes fatores para especificações precisas do comportamento do software e suas evoluções ao longo do tempo e entre as famílias de software.

Xiao (2003), declara que por meio da compreensão das crenças dos interessados (stakeholders), a engenharia de requisitos especifica um problema a ser solucionado, identifica os limites do sistema, analisa propriedades do ambiente e caracteriza os comportamentos do software resultante. Este autor acrescenta que as atividades centrais da engenharia de requisitos são:

- Extração (elicitação) de requisitos, onde procura-se determinar o problema a ser solucionado e identificar os limites do sistema.

- Análise e modelagem de requisitos, onde constrói-se descrições abstratas passíveis de interpretação.
- Comunicação de requisitos, onde os requisitos são comunicados entre os interessados (stakeholders). A forma como os requisitos são documentados desempenha um papel importante em garantir que estes podem ser lidos, analisados, re-escritos e validados.
- Concordância de requisitos, onde tenta-se resolver conflitos entre os interessados, quando estes têm objetivos divergentes.
- Evolução de requisitos, onde é necessário gerenciar a documentação dos requisitos quando estes mudam em novas versões ou mesmo na versão atual, quando o impacto destas mudanças e o risco ao projeto devem ser monitoradas e controladas.

De acordo com Berenbach (2009), estudos indicam que cerca de metade dos fatores associados com o sucesso de projetos estão relacionados a requisitos e que o sucesso de um projeto está diretamente ligado à qualidade dos requisitos. Isto é corroborado por Hull (2005) e Pressman (1995) que afirmam que a definição incorreta dos requisitos do sistema é um dos principais fatores para a falha de um projeto.

Já segundo Regnel (1999), a elicitação, análise e documentação de requisitos em sistemas complexos é uma tarefa crucial e não-trivial. E, segundo Sommerville (2003), casos de uso são de interesse especial na engenharia de requisitos, uma vez que se mostraram valiosos para elicitar, documentar e analisar requisitos. Além disso, segundo Regnel (1999), casos de uso também permitem a rastreabilidade de requisitos nas fases de projeto, implementação e verificação e validação.

Hull (2005) afirma que requisitos são a base de todo projeto, definindo o que os stakeholders necessitam de um sistema e o que o sistema deve fazer de forma a satisfazer esta necessidade. Hull (2005) destaca ainda que requisitos formam a base para planejamento de projeto, gerenciamento de risco, teste de aceitação, tradeoff e controle de mudança.

Segundo Berenbach (2009), a engenharia de requisitos envolve todas as atividades do ciclo de vida devotadas à identificação de requisitos de usuário, análise dos requisitos para derivar requisitos adicionais, documentação dos requisitos como uma especificação e validação dos requisitos documentados de acordo com as necessidades do usuário, bem como os processos que suportam estas atividades.

Ainda segundo Berenbach (2009), as características de um bom requisito são que ele seja possível, válido, unambíguo, verificável, modificável, consistente, completo e que possa ser rastreável.

Um requisito é possível se sua implementação na plataforma planejada é possível dentro das restrições do projeto, ou seja, um requisito é possível se puder ser realizado dados os recursos, orçamento, habilidades, cronograma e tecnologias disponíveis à equipe do projeto.

Um requisito é válido se este é um dos requisitos que o sistema deverá suportar. Neste caso é importante identificar os requisitos que devem ser obrigatoriamente suportados daqueles considerados que seria bom se existissem.

Um requisito é unambíguo, se ele possuir uma única interpretação.

Um requisito é verificável se o sistema final puder ser testado para garantir que ele suporta o requisito.

Uma especificação de requisitos é modificável se sua estrutura e estilo são tais que qualquer mudança em um requisito possa ser feita facilmente, completamente e consistentemente mantendo a estrutura e estilo.

Um requisito é consistente se ele não contradiz ou não está em conflito com quaisquer documentos externos incorporados ou padrões ou outros produtos ou projetos de requisitos.

Uma especificação de requisitos é completa se ela inclui todos os requisitos relevantes corretos e informação suficiente está disponível para o produto ser construído.

Um requisito é rastreável se a fonte do requisito puder ser identificada, qualquer componente que implemente o requisito puder ser facilmente identificado e quaisquer casos de teste para verificar que o requisito tenha sido implementado possam facilmente identificá-lo.

Vários autores como Rodriguez (2011), Berenbach (2009), Sommerville (2003), Papasimeon (2003), Bezerra (2002), Melo (2002), Booch (2000) e Regnel (1999), sugerem a aplicação do diagrama de casos de uso da UML para a modelagem de requisitos. Baseados nessas afirmações desenvolvemos um metamodelo UML no qual criamos novas metaclasses e estereótipos com o intuito de permitir o uso do diagrama de casos de uso para a modelagem de requisitos funcionais no projeto de sistemas multi-agentes.

Segundo Yu (1997) conceitos de agente tem sido introduzidos na engenharia de requisitos primariamente como construções de modelagem para caracterizar elementos ativos no ambiente. Este autor afirma que uma abordagem orientada a agentes para engenharia de requisitos será de particular interesse para ambientes nos quais exista um alto grau de computação distribuída aberta e nos quais as mudanças sejam frequentes. Em frameworks de engenharia de software orientada a agentes, questões cruciais de engenharia de requisitos tais como funcionalidade, qualidade e processo irão ser organizadas em torno de agentes de maneira que elas serão tratadas de uma forma que seja apropriada para ambientes abertos, distribuídos e em constante evolução.

Yu (1997) acrescenta que para que um framework de engenharia de requisitos seja verdadeiramente orientado a agente, a identidade e a existência de um agente como uma cosntrução ao nível de engenharia de requisitos precisa ser determinada dentro deste nível, baseado sobre critérios de engenharia de requisitos.

Ainda segundo Yu (1997) cenários são de muito interesse para a engenharia de requisitos. Um cenário pode ser visto como uma instância única de um processo. Como agentes participam de cenários, em que grau, combinação de ação e reflexão, são questões importantes para investigar em um framework de engenharia de requisitos orientado a agentes.

Segundo Xiao (2003) abordagens orientadas a agentes estão se tornando populares para a modelagem de requisitos; Elas têm sido introduzidas na Engenharia de Requisitos principalmente para caracterizar elementos ativos no ambiente.

Tiryaki (2006) afirma que é quase impossível levantar todos os requisitos de um sistema multi agente no início do desenvolvimento devido à complexidade deste tipo de sistema. Este autor destaca que um sistema multi-agente pode operar como uma

organização aberta o que complica ainda mais a compreensão de todos os requisitos funcionais e não funcionais nas fases iniciais do desenvolvimento. Tiryaki (2006) conclui que é necessária uma abordagem de desenvolvimento incremental e iterativa e que seu nível de implementação seja capaz de licar com a complexidade e a contínua mudança dos requisitos.

Rodriguez (2011) afirma que a modelagem de requisitos em sistemas multi-agentes requerem abstrações, técnicas e notações que tenham sido adaptadas particularmente para este tipo de domínio, enquanto Werneck (2007) destaca que o aumento do uso de sistemas multi-agentes gerou desafios não estudados ainda, incluindo a forma de adaptar a elicitação de requisitos para lidar com propriedades de agentes como autonomia, sociabilidade e proatividade. Finalmente, Papasimeon (2003) declara que a engenharia de software de sistemas multi-agentes demanda a especificação dos comportamentos de agente requeridos para fornecer requisitos documentados para as fases de projeto e implementação.

3.3 Metaclasses Utilizadas na Modelagem de Casos de Uso em UML

Nesta seção apresentaremos as metaclasses originais da UML aplicadas na modelagem de casos de uso e as metaclasses a partir da qual estas foram derivadas, como pode ser observado na figura a seguir. A partir de algumas destas metaclasses derivamos as novas metaclasses do metamodelo aqui proposto.

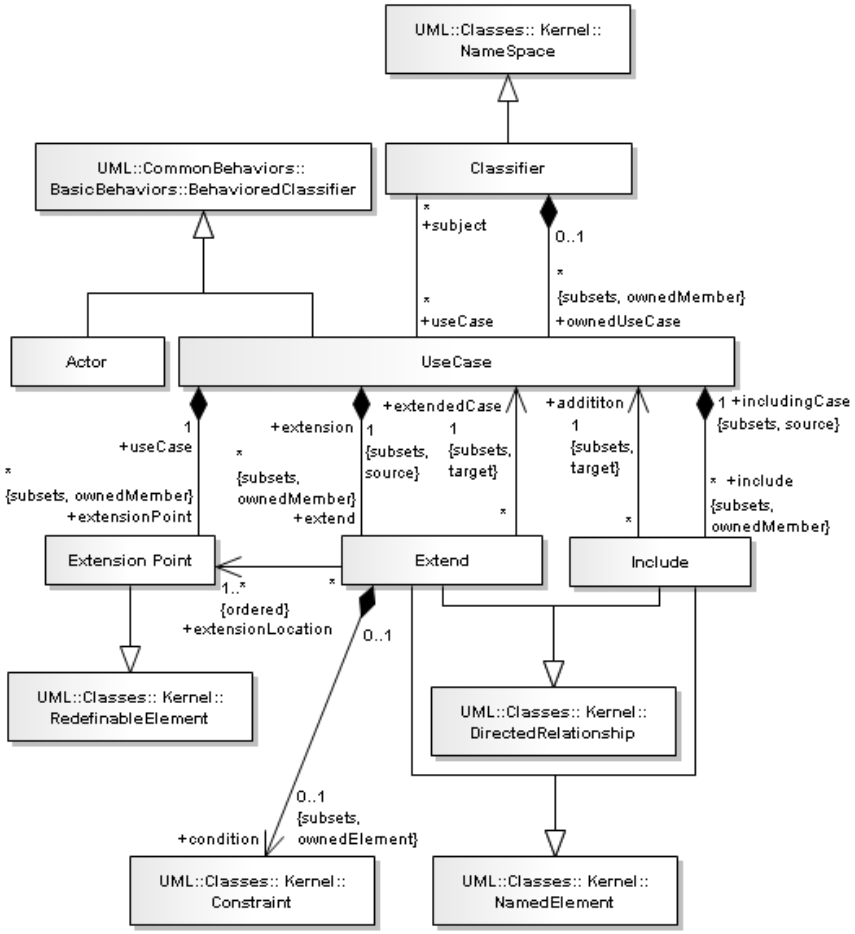


Figura 3.1: Conceitos Usados na Modelagem de Casos de Uso (OMG, 2007b).

A seguir descreveremos sucintamente as metaclasses apresentadas na figura 3.1, e, quando necessário, os pacotes de onde foram derivadas.

3.3.1 Pacote Common Behaviors, Subpacote BasicBehaviors e Metaclasse BehavioredClassifier

De acordo com a OMG (2007b), o pacote Common Behaviors especifica os conceitos centrais requeridos para elementos dinâmicos e fornece a infraestrutura para suportar definições de comportamento mais detalhadas.

O subpacote BasicBehaviors introduz o arcabouço (framework) que será usado para especificar comportamentos, enquanto os subtipos concretos de Comportamento (Behavior) irão fornecer mecanismos diferentes para especificar comportamentos.

O BehavioredClassifier é um classificador que tem especificações de comportamento definidas em seu namespace.

3.3.2 Pacote Classes, Kernel, Metaclasse NameSpace, Metaclasse NamedElement, Metaclasse DirectedRelationship, Metaclasse Constraint e Metaclasse RedefinableElement

Segundo a OMG (2007b), o pacote Classes contém sub-pacotes que lidam com conceitos básicos de modelagem da UML, em particular classes e seus relacionamentos.

O pacote Kernel representa os conceitos de modelagem centrais da UML, incluindo classes, associações e pacotes.

Um namespace (espaço de nome) é um elemento em um modelo que contém um conjunto de elementos nomeados que podem ser identificados pelo nome. É uma metaclasse abstrata.

Um named element (elemento nomeado) representa elementos que podem ter um nome. O nome é usado para identificação do elemento nomeado dentro do espaço de nome no qual está definido. Esta é também uma metaclasse abstrata.

Um directed relationship (relacionamento direcionado) representa um relacionamento entre uma coleção de elementos de modelo fonte e uma coleção de elementos de modelo destino. Esta é igualmente uma metaclasse abstrata.

Uma constraint (restrição) é uma metaclasse abstrata que representa uma condição ou restrição expressada em texto em linguagem natural ou linguagem de programação para o propósito de declarar algumas das semânticas de um elemento. Constraint é uma condição que restringe a extensão do elemento associado além do que é imposto por outras construções de linguagem aplicados naquele elemento.

Um redefinable element (elemento redefinível) é também uma metaclasse abstrata que representa um elemento nomeado que pode ser redefinido no contexto de uma generalização.

3.3.3 Metaclasse Classifier

Um classificador (classifier) é uma classificação de instâncias, ele descreve um conjunto de instâncias que tem características em comum. Um classificador é um espaço de nome cujos membros podem incluir características. É uma metaclasse abstrata. Um classificador é um tipo e pode possuir generalizações, tornando possível definir relacionamentos de generalização para outros classificadores. Um classificador pode especificar uma hierarquia de generalização por meio de referência aos seus

classificadores gerais. Um classificador é um elemento redefinível, o que significa que é possível redefinir classificadores aninhados (OMG, 2007b).

Um classificador é um mecanismo que descreve características comportamentais e estruturais, porém as instâncias de um classificador não são objetos como as instâncias de uma classe, mas elementos UML concretos (que incluem classes). Um classificador não pode ser utilizado em um modelo, somente suas instâncias o podem sê-lo. Classificadores são utilizados apenas em metamodelos.

As metaclasses Actor, UseCase, Extension Point, Exdend e Include não são metaclasses abstratas e serão descritas na seção seguinte.

3.4 O Metamodelo Proposto

Considerando que a UML é uma linguagem de modelagem padrão amplamente aceita e compreendida na área de engenharia de software, que sistemas multi-agentes possuem suas próprias características e que muito poucos trabalhos que aplicaram e/ou derivaram a UML para o projeto de sistemas multi-agentes preocuparam-se em enfocar a questão de levantamento e análise de requisitos, decidiu-se nesse trabalho criar um metamodelo UML no qual acrescentou-se novas metaclasses e estereótipos ao metamodelo em que se baseia o diagrama de casos de uso, preparadas para identificar as funcionalidades particulares deste tipo de sistema, como pode ser visto na próxima figura.

Em um primeiro momento pretendia-se desenvolver apenas um perfil UML estendendo as metaclasses originais utilizadas pelo diagrama de casos de uso, porém logo foi percebido que apenas estender as metaclasses originais não seria suficiente, sendo necessário criar novas metaclasses. A seguir serão descritas cada uma das metaclasses criadas.

Inicialmente, pensou-se em usar a metaclasse Actor original com o objetivo de representar papéis de agente, posto que, de acordo com OMG (2007b), esta metaclasse representa um tipo de papel interpretado por uma entidade que interage com o sistema, mas que é externa a ele. Atores podem representar papéis interpretados por usuários humanos, hardware externo ou outros sistemas. Um ator não necessariamente representa uma entidade física específica, mas meramente uma faceta particular, um papel, de alguma entidade que é relevante para a especificação de seus casos de uso associados. Assim, uma única instância física pode interpretar o papel de diversos atores diferentes e um dado ator pode ser executado por múltiplas instâncias diferentes.

Entretanto, na maioria das vezes, os agentes de software não são externos ao software, ao invés disso, eles costumeiramente estão inseridos no ambiente do sistema e, como eles são independentes, pró-ativos e capazes de interagir com o software de acordo com seus objetivos, os papéis interpretados por estes agentes deveriam ser representados como atores. Assim, como afirma Bauer (2005), é necessário adaptar este conceito, considerando que agentes são internos ao sistema, ou seja, um agente faz parte do sistema. Dessa maneira, para poder representar papéis de agente como atores, estes atores devem ser representados internamente dentro das fronteiras do sistema, uma vez que eles pertencem ao software.

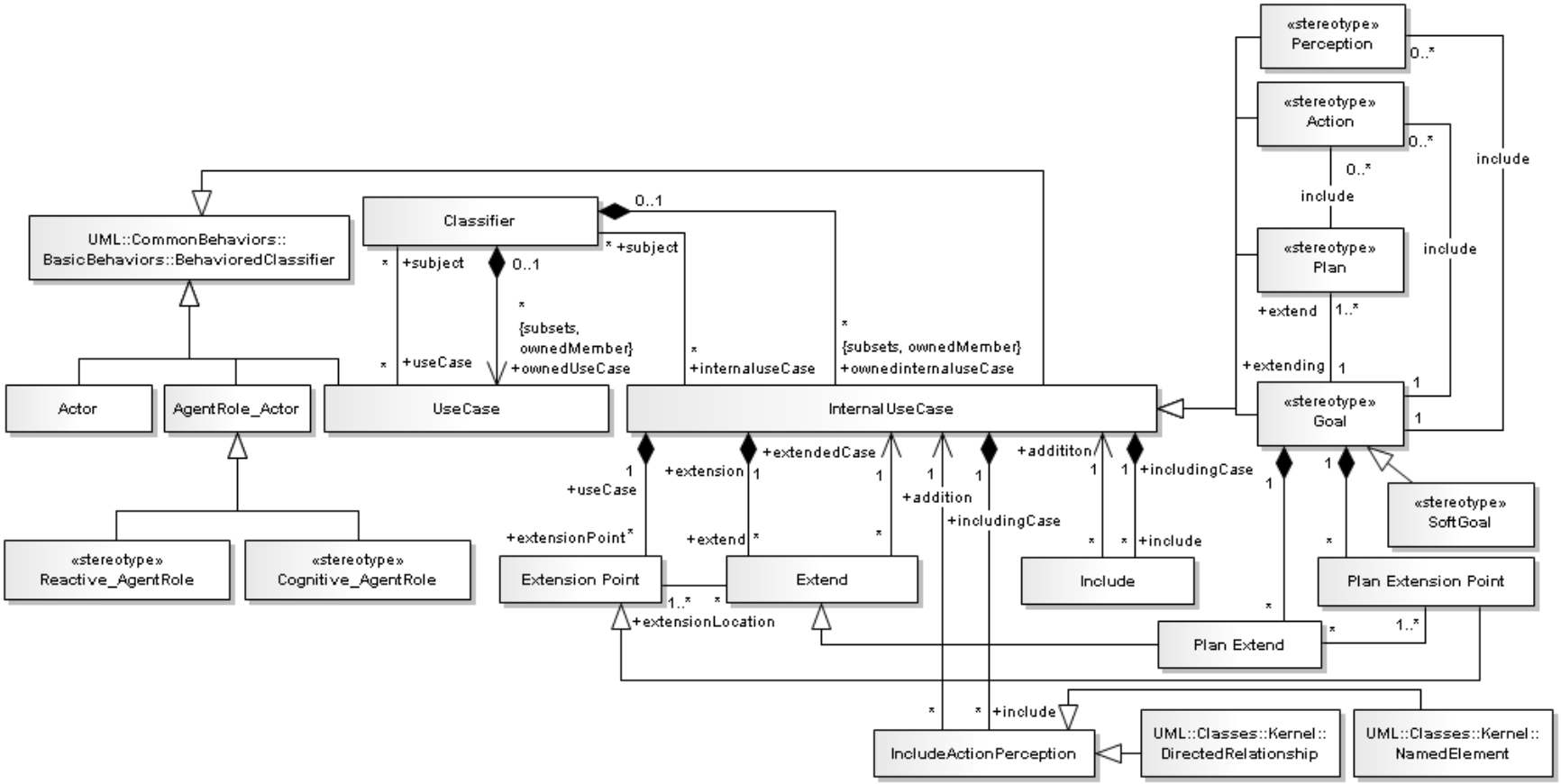


Figura 3.2: Metamodelo UML para a Modelagem de Requisitos Funcionais Específicos para o Projeto de Sistemas Multi-Agentes.

A representação de agentes/papéis como atores UML pode ser vista também em (FLAKE, 2001), no qual agentes representam objetos ativos e são modelados dentro do sistema como atores com cabeça quadrada e em (MYLOPOULOS, 2001), que propôs um mapeamento de conceitos utilizados na metodologia Tropos (BRESCIANI, 2004), para UML, entre os quais o conceito de ator utilizado em Tropos para o conceito de ator da UML. Finalmente, este tipo de representação de agentes como atores pode ser visto também em PAPASIMEON (2003), que criou o estereótipo <<agent>> para representar agentes que sejam parte de um software como atores, representando-os dentro das fronteiras do sistema.

Todavia, (BAUER, 2005) somente sugere que o conceito de ator deveria ser adaptado e nem (FLAKE, 2001), (MYLOPOULOS, 2001) ou (PAPASIMEON, 2003) criaram um metamodelo UML para modelar os requisitos de sistemas multi-agentes. Neste trabalho criou-se explicitamente novas metaclasses a partir das metaclasses originais da UML para permitir a modelagem de requisitos funcionais em sistemas multi-agentes.

Papasimeon (2003) simplesmente criou um estereótipo para ser aplicado sobre atores UML, o que, na nossa opinião, não seria totalmente correto, uma vez que atores representam entidades externas ao sistema e não é possível retirar restrições de uma metaclasse original ao aplicar-lhe um estereótipo. Além disso, este autor utiliza o estereótipo <<agent>> para representar diretamente agentes, enquanto nós representamos papéis de agente, que consideramos mais próximos do conceito de Ator da UML, ou seja, uma entidade assumindo um papel no sistema. Finalmente, Papasimeon (2003) não diferencia agentes reativos de cognitivos, como é proposto neste trabalho.

Em Rodriguez (2011) pode-se verificar outro trabalho que utiliza atores para representar papéis de agentes usando atores com o estereótipo <<role>>, o que é um pouco semelhante à que propõe-se nesse trabalho. Além disso, essa autora representa associações entre os atores contendo os estereótipos <<collaboration>>, <<disposition>> e <<veracity>> para determinar quando os papéis colaboram entre si, a disposição do papel (se é benevolente, auto-interessado ou malevolente) e se o papel é confiável ou não, respectivamente. Os recursos do sistema são representados como classes e os atores são associados a estas classes por meio de associações estereotipadas que determinam quando um agente ao interpretar um papel pode perceber (perceive) ou modificar (modify) um recurso.

Contudo, de acordo com OMG (2007a), não é possível retirar quaisquer restrições aplicadas a um metamodelo em um perfil, o que nos levou a criar novas metaclasses no metamodelo proposto. Assim, ao invés de estender a metaclasse Actor, criou-se uma nova metaclasse derivada da mesma metaclasse a partir da qual foi derivada a metaclasse Actor, como pode ser visto na figura 3.3, que apresenta a metaclasse AgentRole_Actor. Nesta nova metaclasse copiou-se todas as características da metaclasse Actor, incluindo o mesmo símbolo, suprimindo-se apenas a restrição que um ator precisa ser externo ao sistema.

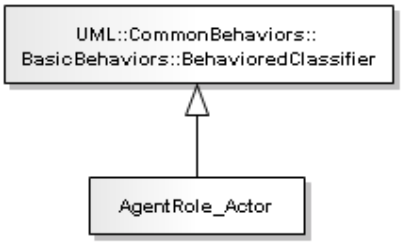


Figura 3.3: Metaclasse AgentRole_Actor

A partir desta nova metaclasse derivou-se as metaclasses Reactive_AgentRole e Cognitive_AgentRole, como pode ser observado, na figura 3.4, foi aplicado o estereótipo “stereotype” em ambas as metaclasses, o que significa que estas metaclasses serão aplicadas como estereótipos sobre AgentRole_Actors e lhes atribuirão características especiais, uma vez que estes AgentRole_Actors serão utilizados em domínios específicos para representar papéis interpretados por agentes cognitivos e reativos. Optou-se por criar estes dois estereótipos para estabelecer uma diferença entre papéis de agente reativos e cognitivos, considerando que cada grupo apresenta características diferentes.

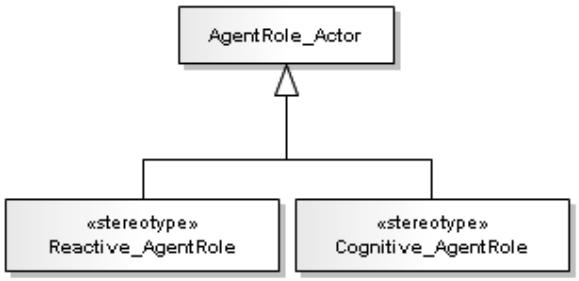


Figura 3.4: Metaclasses Reactive_AgentRole e Cognitive_AgentRole Derivadas a Partir da Metaclasse AgentRole_Actor

Embora no trabalho de Rodriguez (2011) a representação de atores estereotipados para representar papéis seja de alguma maneira semelhante à proposta aqui apresentada, esta autora utiliza atores normais estereotipados, dessa forma eles não podem ser utilizados para representar agentes (ou seus papéis) que façam parte de um sistema, apenas agentes externos a um software. Além disso, no trabalho aqui proposto criou-se também os estereótipos Reactive e Cognitive_AgentRole para diferenciar papéis de agentes reativos de cognitivos, finalmente, Rodriguez (2011) utiliza apenas simples associações estereotipadas para determinar quando um recurso pode ser percebido ou modificado. Na abordagem aqui apresentada são representadas percepções, ações, objetivos e planos por meio de casos de uso internos, como será demonstrado a seguir.

Com relação aos casos de uso, como já foi dito antes, diversos autores recomendam o uso deste recurso para a modelagem de requisitos. De acordo com a OMG (2007b), um caso de uso é a especificação de um conjunto de ações realizadas por um sistema, gerando um resultado observável que tipicamente produz algum valor para um ou mais atores. Cada caso de uso especifica algum comportamento que o sistema pode realizar em colaboração com um ou mais atores.

Assim, com o objetivo de adaptar o conceito de caso de uso para a modelagem de requisitos em sistemas multi-agentes, da mesma forma como procedeu-se com a metaclasse AgentRole_Actor, derivou-se uma nova metaclasse a partir da mesma metaclasse que a metaclasse UseCase foi derivada, sendo esta denominada InternalUseCase, e então foram criados relacionamentos iguais aos existentes entre a metaclasse Classifier e a metaclasse UseCase para a nova metaclasse InternalUseCase, como pode ser visto na figura 3.5.

Esta metaclasse foi criada porque pretendia-se representar ações, percepções, objetivos e planos como casos de uso, porém a semântica da metaclasse UseCase afirma que casos de uso representam requisitos externos, portanto, para adaptar este conceito à modelagem de sistemas multi-agentes, foi criada uma metaclasse similar, modificando sua semântica para representar requisitos internos.

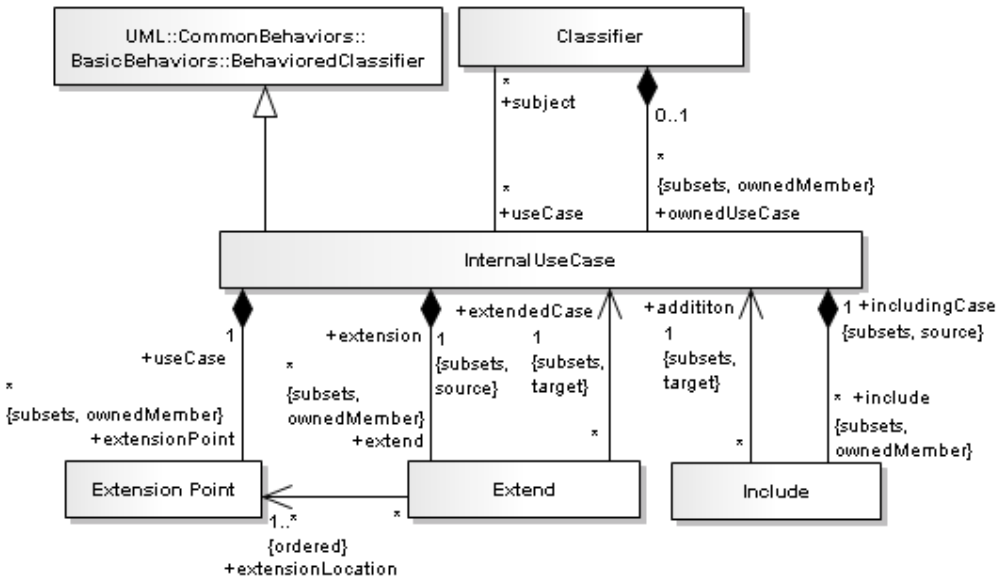


Figura 3.5: Metaclasse InternalUseCase

Obviamente, todos os casos de uso internos, como seu próprio nome diz, são internos ao sistema e não podem ser vistos ou usados por entidades externas. Papasimeon (2003) sugere um conceito um pouco parecido, ao criar um estereótipo para ser aplicado sobre os casos de uso normais, denominado <<agent use case>>. Este estereótipo serviria para representar cenários sobre interações de agentes com o sistema. Cada cenário descreveria uma seqüência de eventos iniciados por agentes. O resultado da seqüência de eventos tem de ser de alguma forma útil para o agente que o iniciou, para outro agente ou para qualquer outro ator.

Entretanto, nós não consideramos possível simplesmente aplicar um estereótipo a casos de usos para representar as funcionalidades que um agente poderia solicitar ao sistema, uma vez que casos de uso representam funcionalidades que os atores externos podem solicitar ao sistema, o que na maioria das vezes não é verdadeiro para as funcionalidades que os agentes utilizam, sendo estas transparentes para os usuários externos. Por este motivo, e, como já foi explicado anteriormente, por não ser possível retirar restrições de uma metaclasse, decidiu-se criar a metaclasse InternalUseCase. Finalmente, o estereótipo criado por Papasimeon (2003) não diferencia entre objetivos, percepções, ações ou planos como é proposto pelo metamodelo aqui proposto e será explicado a seguir.

A partir da metaclasse InternalUseCase, foram estendidas algumas metaclasses para atribuir características especiais aos casos de uso internos; essas metaclasses estendidas serão empregadas como estereótipos. Assim, criou-se as metaclasses Perception e Action para modelar casos de uso internos que contenham os passos necessários para um agente perceber ou realizar uma determinada ação, conforme pode ser observado na figura 3.6. Este é um procedimento também sugerido por (BAUER, 2005) para ser aplicado em casos de uso normais, embora como afirmado anteriormente em (OMG, 2007b), casos de uso normais se refiram a requisitos externos, não aos internos.

Uma terceira metaclasse foi derivada a partir da metaclasse InternalUseCase, para representar objetivos, a metaclasse Goal. Um caso de uso interno empregando o estereótipo Goal conterá uma descrição de um desejo a ser atingido por um agente e as possíveis condições para que o desejo torne-se uma intenção.

Além disso, nós evoluímos a primeira proposta deste metamodelo (na época ainda o denominávamos perfil) apresentado em (GUEDES, 2010a), quando incluímos o conceito de softgoal (objetivo-soft), derivando a metaclasse SoftGoal a partir da metaclasse Goal. O conceito de softgoal é descrito em (BRESCIANI, 2004) e é usado para representar objetivos nebulosos, que não apresentam uma definição clara e/ou critérios para decidir se eles foram satisfeitos ou não. A evolução deste metamodelo, incluindo a nova metaclasse SoftGoal, foi publicada em (GUEDES, 2010b). O metamodelo ainda sofreu algumas alterações, que também foram publicadas em (GUEDES, 2010c). Este metamodelo foi novamente evoluído e publicado em (GUEDES, 2011a) e (GUEDES, 2011b). Todavia sua versão atual é a apresentada na figura 3.2.

Uma proposta um pouco semelhante para representar objetivos como casos de uso pode ser vista em (FLAKE, 2001). Porém, além de criarmos o conceito de casos de uso internos, nós fomos ainda mais além da proposta apresentada por este autor, quando nós consideramos que, da mesma forma que um objetivo representa um desejo que não irá necessariamente tornar-se uma intenção, os passos para sua execução deveriam ser detalhados em um ou mais casos de uso internos, diferentes do caso de uso interno que representaria o objetivo.

Dessa forma, em uma situação onde um caso de uso interno empregando o estereótipo Goal para representar um objetivo de um agente fosse utilizado, nós iríamos somente detalhar neste caso de uso interno aquelas percepções e condições necessárias para que o objetivo torne-se uma intenção. Considerando que um objetivo pode eventualmente possuir mais do que um plano e que este ou estes planos somente serão executados mediante certas condições, nós decidimos derivar uma quarta metaclasse a partir da metaclasse InternalUseCase, chamada Plan, para identificar os planos associados aos objetivos de um papel de agente.

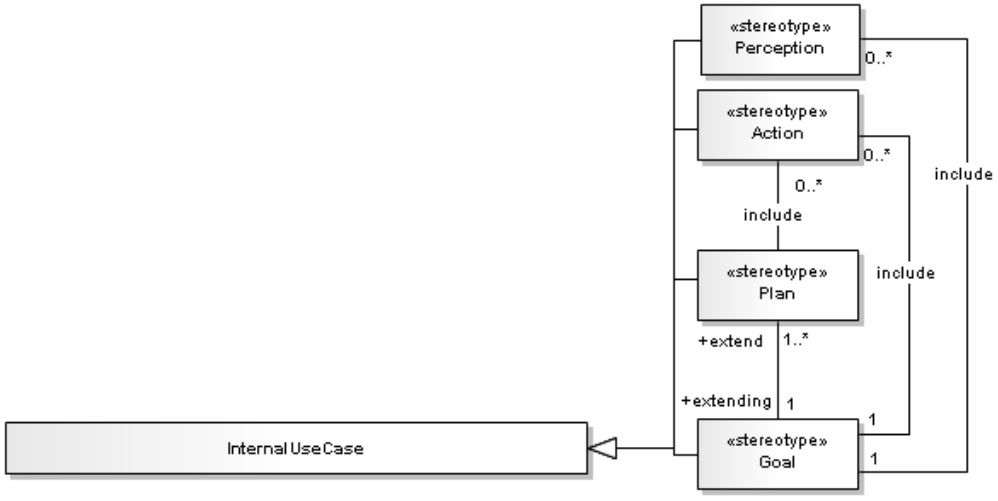


Figura 3.6: Metaclasses Perception, Action, Goal e Plan

A figura 3.6 mostra as metaclasses Perception, Action, Goal e Plan derivadas a partir da metaclasse InternalUseCase. É importante observar que existem associações entre elas que demonstram que um caso de uso interno do tipo objetivo pode incluir percepções em seu comportamento (que poderiam auxiliar a determinar se um objetivo poderia se tornar uma intenção), além de poder incluir também o comportamento de ações, se isto fosse considerado necessário. A metaclasse Goal possui ainda uma associação com a metaclasse Plan, que demonstra que o comportamento de um objetivo pode ser estendido por um ou mais planos.

Além disso há também uma associação entre os casos de usos internos Plan e Action, significando que um plano pode também incluir ações em seu comportamento.

Optou-se por derivar estas metaclasses a partir da metaclasse InternalUseCase com o objetivo de representar explicitamente os objetivos de cada papel de agente, as possíveis percepções associadas a estes objetivos, que poderiam auxiliar a determinar se um objetivo poderia ou não se tornar uma intenção, bem como as ações que o papel poderia executar que de alguma afetassem o ambiente e/ou pudessem ser percebidas por outros agentes, além dos planos associados aos objetivos necessários para atingi-los quando estes objetivos viessem a se tornar intenções. Embora se pudesse argumentar que as percepções ou ações pudessem ser englobadas pelos objetivos ou planos, considerou-se importante representá-las graficamente de forma explícita para tornar claro e visível a necessidade que um determinado papel de agente poderia ter de possuir percepções e ações.

Uma vez que um plano somente é disparado depois que alguma condição particular é satisfeita, decidiu-se estender a metaclasse Plan Extend a partir da metaclasse Extend e associá-la com a metaclasse Goal por meio de uma associação de composição, cuja multiplicidade informa que um Goal pode ter muitos Plan Extend associados a ele, porém um Plan Extend só pode estar associado a um único Goal, conforme é apresentado pela figura 3.7. Procedeu-se desta forma para diferenciar associações de extensão normais de associações de extensão de plano.

Segundo a OMG (2007b), uma extensão (extend) representa um relacionamento entre um caso de uso que estende e um caso de uso estendido, que especifica como e quando o comportamento do caso de uso que estende pode ser inserido no comportamento definido no caso de uso estendido. Se a condição da extensão é verdadeira no momento em que o primeiro ponto de extensão é atingido durante a execução do caso de uso estendido, então o comportamento do caso de uso que estende será também executado. Se a condição é falsa, a extensão não ocorrerá. Assim, este relacionamento exige que uma condição seja satisfeita para que a extensão de comportamento ocorra.

Dessa maneira nós criamos a metaclasse Plan Extend a partir da metaclasse Extend para representar as situações em que o comportamento de um objetivo poderia ser estendido pelo comportamento de um plano para atingi-lo. Além disso este tipo de associação implicaria na satisfação de uma condição para que efetivamente ocorresse.

Nós derivamos ainda a metaclasse Plan Extension Point a partir da metaclasse Extension Point, como demonstra a figura 3.7. De acordo com OMG (2007b), um ponto de extensão identifica um ponto no comportamento do caso de uso onde aquele comportamento pode ser estendido pelo comportamento de algum outro caso de uso, como especificado por um relacionamento de extensão. Esta metaclasse foi derivada somente para estabelecer uma diferença entre pontos de extensão de plano e pontos de extensão normais; além disso, isto pode servir também para estabelecer explicitamente a condição para que um plano seja executado.

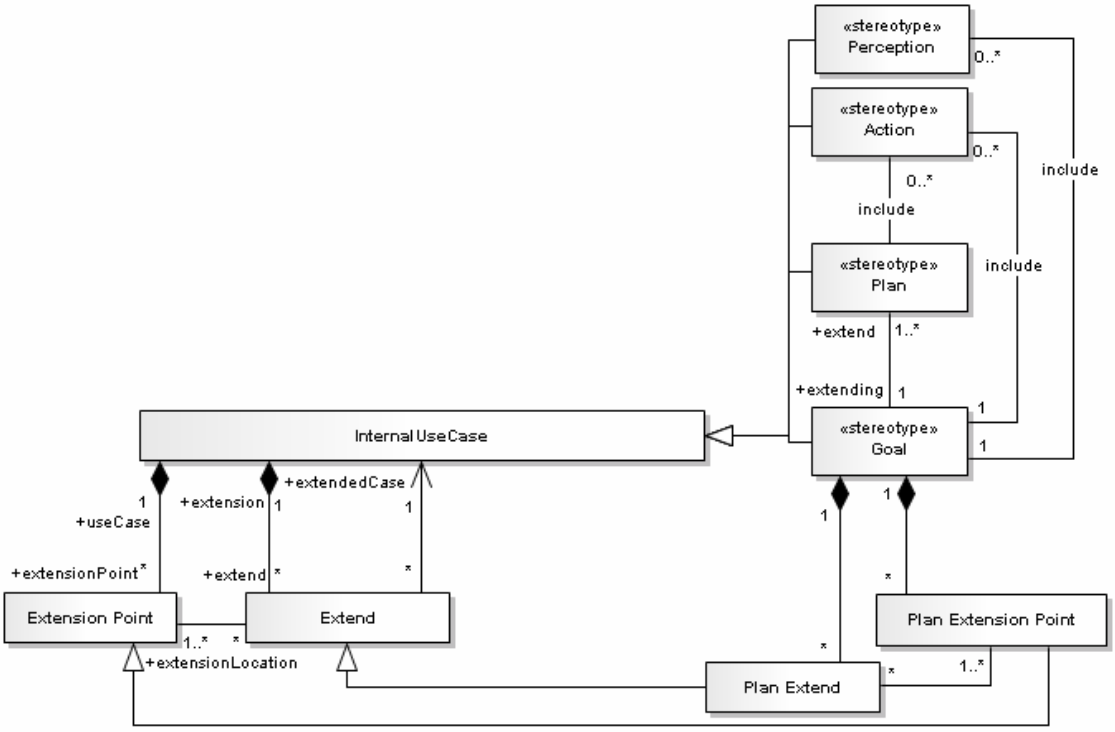


Figura 3.7: Metaclasses Plan Extend e Plan Extension Point

Finalmente, derivou-se a metaclasse IncludeActionPerception a partir das mesmas metaclasses que a metaclasse Include foi derivada, como pode ser observado na próxima figura. De acordo com a OMG (2007b) uma inclusão é um relacionamento direto entre dois casos de uso, que implica que o comportamento do caso de uso incluído é inserido no comportamento do caso de uso que inclui. Este relacionamento indica também uma obrigatoriedade, sempre que o primeiro caso de uso (que inclui) for executado o segundo deverá obrigatoriamente ser executado também.

Pretendia-se utilizar o relacionamento de inclusão diretamente para incluir o comportamento de percepções e ações no comportamento de objetivos e para incluir o comportamento de ações em planos. Porém a semântica da metaclasse Include afirma que se pretende que este relacionamento seja usado quando há trechos de comportamento iguais no comportamento de dois ou mais casos de uso. Nessas situações costuma-se tomar o trecho comum e colocá-lo em um caso de uso à parte, incluindo-o no comportamento dos casos de uso que o utilizam por meio de associações de inclusão.

Embora papéis de agente possam compartilhar percepções e ações, isto não é uma regra, dessa maneira optou-se por criar uma nova metaclasse na qual copiou-se as mesmas características da metaclasse Include, porém retirando a recomendação de que ela seja usada quando há comportamentos comuns a mais de um caso de uso. Dessa maneira, é possível incluir o comportamento de uma ou mais percepções ou ações dentro do comportamento de um objetivo ou o comportamento de uma ou mais ações dentro do comportamento de um plano.

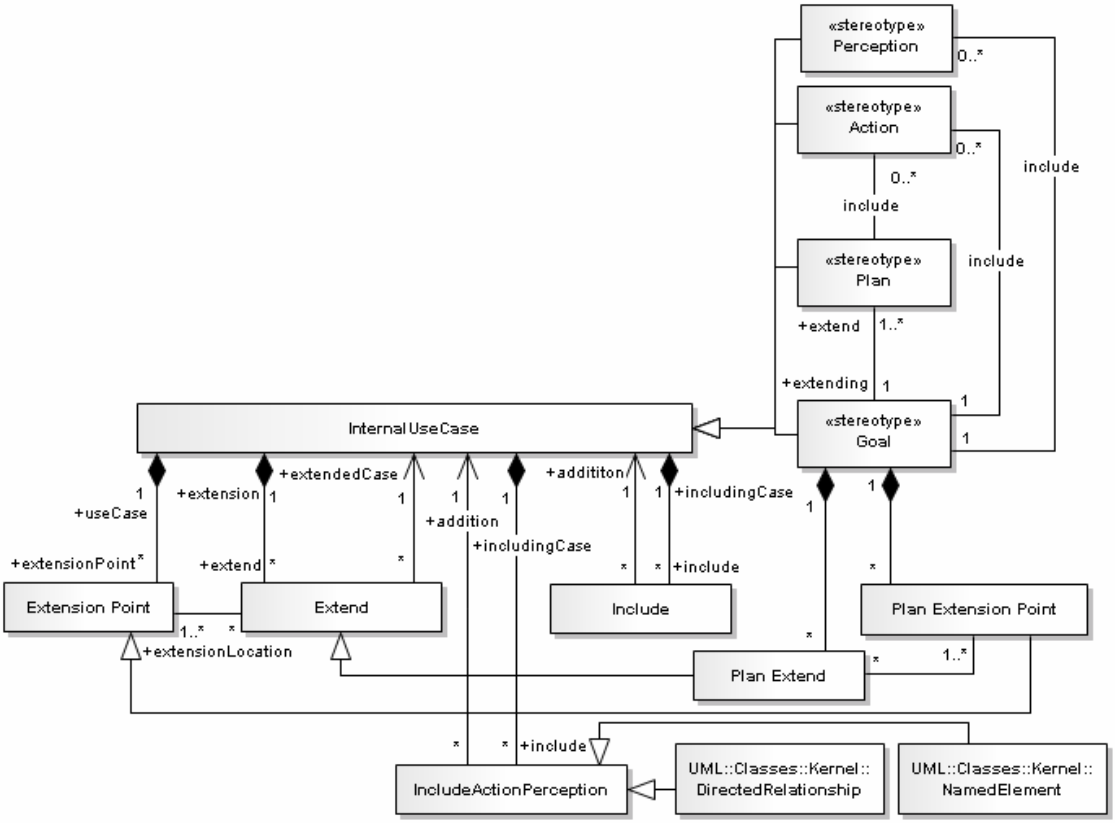


Figura 3.8: Metaclasse IncludeActionPerception

3.5 Estudo de Caso I – O sistema MCOE (Multi-agent CO-operative Environment – Ambiente Co-operativo Multi-agente)

O sistema MCOE (GIRAFFA, 1998) já foi implementado há alguns anos, nós estamos aplicando o metamodelo em sistemas já desenvolvidos anteriormente como base para nosso estudo. Este estudo de caso foi publicado no SEKE 2010 – 22º. Congresso Internacional de Engenharia de Software e Engenharia do Conhecimento (GUEDES, 2010a) em Redwood City, California, EUA.

A arquitetura MCOE é composta por uma sociedade híbrida de agentes que trabalham em conjunto para atingir o objetivo comum de lutar contra a poluição e manter o equilíbrio do sistema de um lago. Este sistema possui agentes reativos como microrganismos, plantas e peixes, e agentes cognitivos, compostos pelo agente tutor e pelos agentes alunos, representados dentro do sistema pelos personagens Mãe Natureza, Prefeito, Pescador e Turista.

O agente tutor recebe informações sobre cada modelo de estudante. Tal informação é composta pelos estados mentais de cada aluno, as ações dos aprendizes e os sensores do ambiente, que retornam informações sobre os níveis de energia. O tutor usa tal informação para selecionar uma estratégia específica com a tática pedagógica associada. Os estados mentais dos estudantes e suas ações são as percepções que o tutor recebe do ambiente.

O agente tutor tem somente uma intenção, que é a de ajudar o aluno a gerenciar um nível de energia adequado ao ambiente. Ele acredita que pode ajudar enviando mensagens aos

aprendizes. Os conteúdos destas mensagens dependerão da estratégia adotada pelo tutor no momento.

Quando há somente um desejo, este desejo é o único candidato para uma intenção. Ele será adotado como uma intenção se o tutor acreditar que há uma seqüência de ações que os alunos podem executar. Neste momento, o tutor depende de seu conhecimento sobre os estados mentais dos alunos de forma a decidir o que fazer. Se ele não possuir este conhecimento ainda, ele então esperará. Quando recebe novas informações, o tutor insere este conhecimento em suas crenças, o que pode fazê-lo reconsiderar suas opções quando as interações ocorrem.

Uma vez que o tutor tenha assumido essa intenção, ele usa suas crenças sobre o ambiente, sobre como lutar contra a poluição no lago e como ajudar os alunos a controlar o ambiente.

Aplicando o metamodelo proposto, os alunos foram modelados como atores normais, já que eles são usuários humanos que jogam o jogo assumindo personagens (Mother Nature, Mayor, Fisherman e Tourist – Mãe Natureza, Prefeito, Pescador e Turista), mas mesmo assim permanecem externos ao sistema. Cada um dos personagens pode executar funções diferentes que podem aumentar ou diminuir o nível de energia do lago, Se um aluno assumir o personagem Mother Nature, ele poderá acelerar a degradação, colocar plantas aquáticas, mudar o fundo do lago ou abrir sol sobre o lago, já se assumir o personagem Mayor, ele poderá colocar latas de lixo, proibir atividades, aplicar multas e remover licenças.

Devido a estas variações de funções, optamos por representar o aluno como um ator normal e atribuir-lhe a funcionalidade “Choose Character” (Escolher Personagem), por meio da qual o estudante escolherá com que personagem jogará o jogo. Uma vez que as demais funcionalidades dependem do personagem escolhido, derivamos sub-atores a partir do ator estudante representando cada um dos personagens disponíveis e atribuindo a estes sub-atores casos de uso representando as funcionalidades que cada personagem pode solicitar ao sistema, conforme pode ser visto na figura a seguir, que apresenta esta parte da modelagem do diagrama de casos de uso.

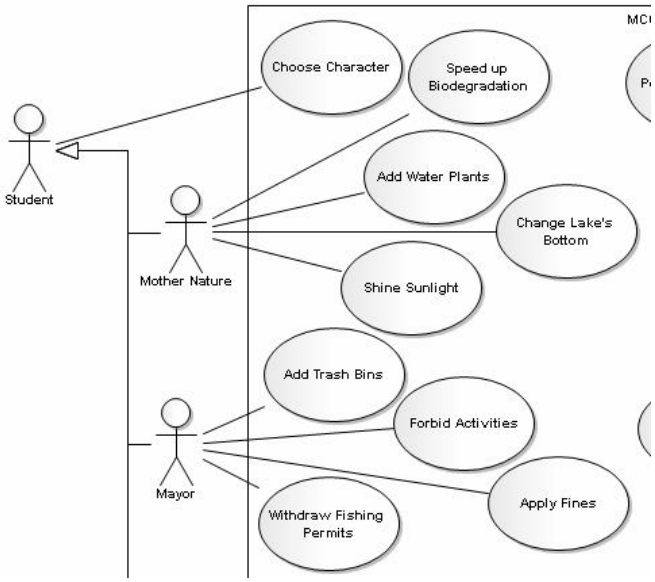


Figura 3.9: Atores e Casos de Uso Normais Representando o Estudante, os Personagens do Jogo e as Funcionalidades Disponíveis.

Os papéis de agente reativos (peixes, plantas e microorganismos) foram modelados como AgentRole_Actors empregando o estereótipo "<<Reactive_AgentRole>>" e estes foram inseridos dentro do sistema, ao contrário dos atores normais, posto que são papéis internos do sistema. Os agentes reativos que interpretarem estes papéis podem executar determinadas ações (representadas por casos de uso internos com o estereótipo Action), de acordo com o nível de energia do lago.

Por exemplo, o papel de agente reativo Peixe pode se reproduzir, alimentar, locomover, definhar ou morrer, se o nível de energia estiver alto (uma pré-condição do caso de uso interno) o peixe poderá executar as 3 primeiras ações. À medida que o nível de energia for baixando o peixe não poderá mais se reproduzir, alimentar ou mesmo locomover e iniciará a definhar mudando de cor e, se o nível de energia baixar mais, ele morrerá. A figura seguinte apresenta um trecho do diagrama de casos de uso do sistema MCOE referente ao papel de agente reativo peixe, representado como um AgentRole_Actor com o estereótipo Reactive_AgentRole, associado a casos de usos internos com o estereótipo Action representando as ações permitidas a este papel.

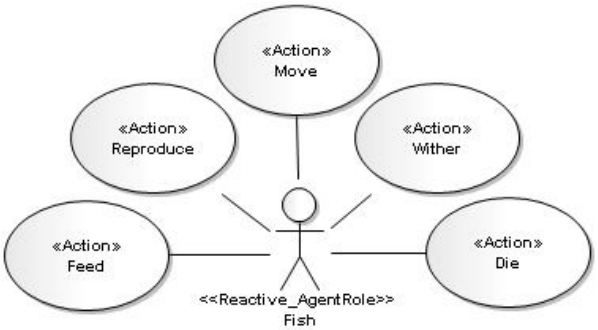


Figura 3.10: Papel de Agente Reativo e Ações modelados por Meio de um AgentRole_Actor e Casos de Uso Internos Estereotipados

Finalmente, passamos à modelagem do papel de agente cognitivo Tutor. Este papel de agente foi representado como um AgentRole_Actor com o estereótipo "<<Cognitive_AgentRole>>" e modelado dentro do sistema. O agente que interpretar o papel de Tutor precisa ser capaz de perceber os estados mentais dos estudantes, perceber as ações dos estudantes e sondar o ambiente. Através destas percepções o agente pode determinar o nível de energia do ambiente (ecômetro) e decidir se o objetivo (desejo) de auxiliar a manter o nível de energia deve se tornar uma intenção e, em caso positivo, disparar o plano a ele associado.

Cada uma das percepções enunciadas anteriormente foi representada como um caso de uso interno com o estereótipo "<<Perception>>", como podemos observar na figura a seguir. O objetivo de auxiliar na manutenção do nível de energia ("aid in the maintanance of energy level") e o plano de ajudar o aluno ("help the student") foram igualmente modelados como casos de uso internos com os estereótipos "Goal" e "Plan" respectivamente.

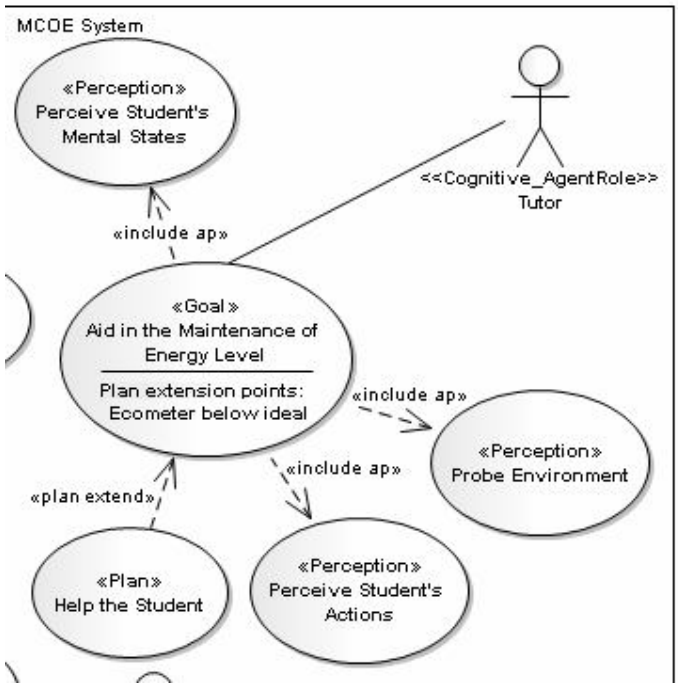


Figura 3.11: Papel de Agente Cognitivo, Percepções, Objetivo e Plano modelados por Meio de um Ator AgentRole e Casos de Uso Internos Estereotipados

Observe que as percepções estão associadas ao objetivo por meio de associações de inclusão ActionPerception (include ap). Como já explicado anteriormente, uma associação de inclusão ActionPerception determina que o comportamento do caso de uso incluído deve ser executado obrigatoriamente pelo caso de uso que inclui. Como pode ser observado no metamodelo proposto, a metaclasse Include ActionPerception está associada com a metaclasse InternalUseCase. O uso de relacionamentos de inclusão ActionPerception, nessa situação, é necessário porque para determinar a real necessidade de disparar o plano "Help the Student" é preciso verificar os estados mentais dos estudantes, verificar as ações dos mesmos e sondar o ambiente para determinar o nível de energia do lago. Somente se o ecômetro estiver abaixo do ideal será executado o plano para auxiliar os alunos.

É importante observar ainda que foi incluído um "Plan extension point" no caso de uso interno com o estereótipo Goal, que determina em que momento e condição poderá ser estendido o comportamento deste objetivo por meio da execução do plano para auxiliar o estudante. Deve-se notar também que a associação entre o caso de uso interno Goal e o caso de uso interno Plan é uma associação plan extend, usada para diferenciar extensões de planos de extensões normais e que obriga a satisfação de uma condição para que seja considerada verdadeira e, por conseguinte, que o plano seja executado.

O comportamento de um caso de uso pode ser detalhado através de outros diagramas, como por exemplo o diagrama de seqüência. Assim, como pode ser visto na figura a seguir, o caso de uso Goal "Aid in the Maintenance of Energy Level" foi detalhado por meio de um destes diagramas.

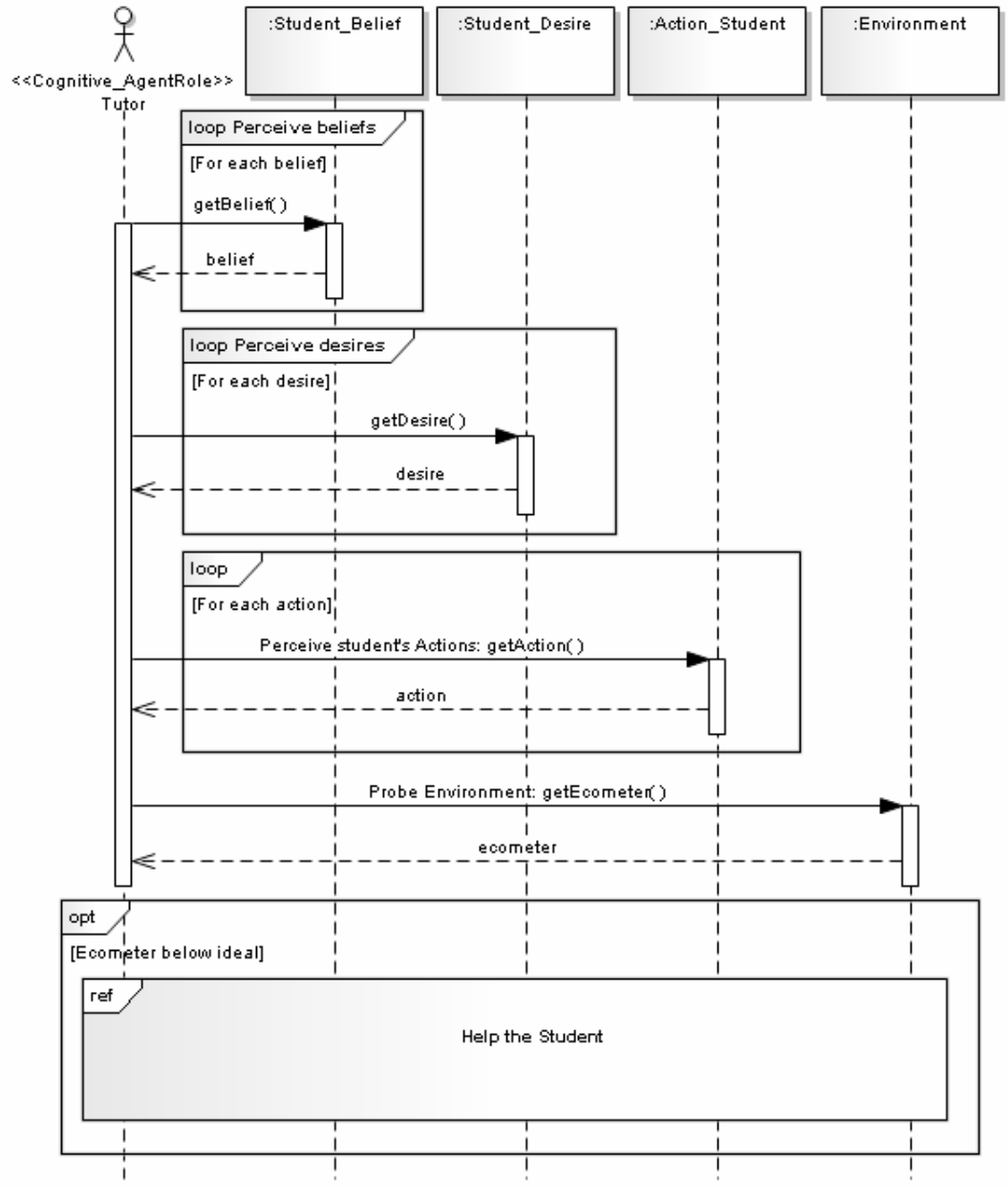


Figura 3.12: Detalhamento do Objetivo "Aid in the Maintenance of Energy Level" (Auxiliar na Manutenção do Nível de Energia)

Neste diagrama a representação do papel de agente Tutor foi mantida como um AgentRole_Actor. Na verdade este ator é uma instância do ator representado no diagrama de casos de uso, assim como os objetos Student_Belief, Student_Desire, Action_Student e Environment são instâncias de classes definidas em um diagrama de classes chamado "Student Knowledge Diagram", onde são armazenadas, entre outras informações, os estados mentais (crenças e desejos) dos alunos, assim como informações sobre os personagens disponíveis, as ações que estes personagens podem realizar e como estas ações modificam o nível de energia do lago, além das ações realizadas por cada aluno e o nível de energia do ambiente, como pode ser observado na figura seguinte.

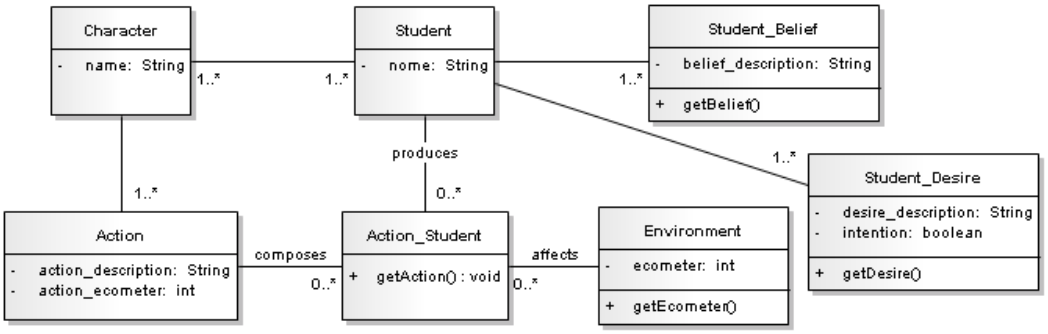


Figura 3.13: Student Knowledge Diagram – Diagrama de Conhecimento do Aluno

Diferente de outras abordagens, até o momento, o metamodelo aqui demonstrado não estendeu as metaclasses usadas pelo diagrama de classes ou de seqüência, dessa forma todas as classes aqui mostradas são classes normais. Neste estudo de caso, por se tratar de um sistema mais simples, para modelar as questões estruturais e comportamentais não cobertas pelo metamodelo utilizamos a linguagem UML propriamente dita.

Voltando à figura 3.12, este diagrama inclui as percepções do agente que interpretar o papel de Tutor, uma vez que estas foram incluídas no caso de uso interno Goal (objetivo); assim, o Tutor primeiramente precisa perceber os estados mentais dos alunos, ou seja, suas crenças e desejos. Dessa forma, o tutor entra em um laço para recuperar todas as crenças dos alunos e, após isso, entra em um segundo laço para recuperar todos os desejos dos alunos. Estes laços são representados por meio de fragmentos combinados do tipo loop.

Em seguida, o tutor passa à segunda percepção, onde é necessário perceber as ações que os alunos realizaram. Assim, é executado um terceiro laço para recuperar cada ação realizada pelos estudantes. A última percepção necessária é sondar o ambiente para recuperar o nível de energia do lago, realizado pelo disparo do método getEcometer que retorna o ecômetro do ambiente.

Finalmente representou-se a possível decisão do agente que interpretar o papel de Tutor em disparar o plano “Help the Student” por meio de um fragmento combinado do tipo Opt, que representa uma opção, cujo conteúdo só será executado se uma condição for satisfeita. Neste caso a condição é determinada pelo nível de energia do lago, se o ecômetro estiver abaixo do ideal então será executado o plano “Help the Student” (Auxiliar o Aluno). Este plano está apenas referenciado, seu detalhamento é feito em outro diagrama de seqüência, conforme demonstrado na próxima figura.

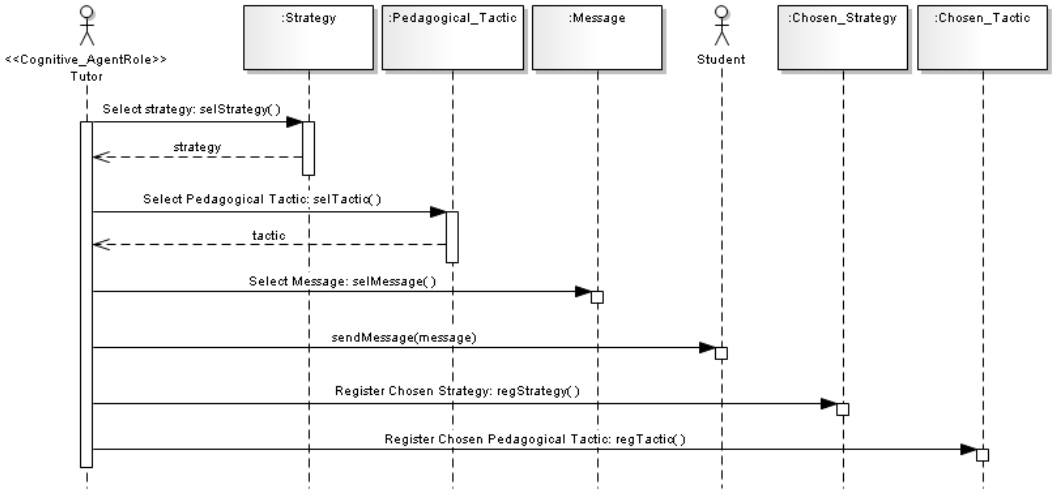


Figura 3.14: Detalhamento do Plano "Help the Student" por meio de um Diagrama de Sequência

Aqui novamente o Tutor foi representado como um AgentRole_Actor e o Aluno como um ator normal, enquanto os objetos envolvidos foram instanciados a partir do Tutor Knowledge Diagram (Diagrama de Conhecimento do Tutor), um diagrama de classes cuja função é armazenar as crenças, desejos e intenções, bem como as estratégias e táticas pedagógicas que podem ser empregadas pelo agente que interpretar o papel de tutor, conforme pode ser visto na figura a seguir.

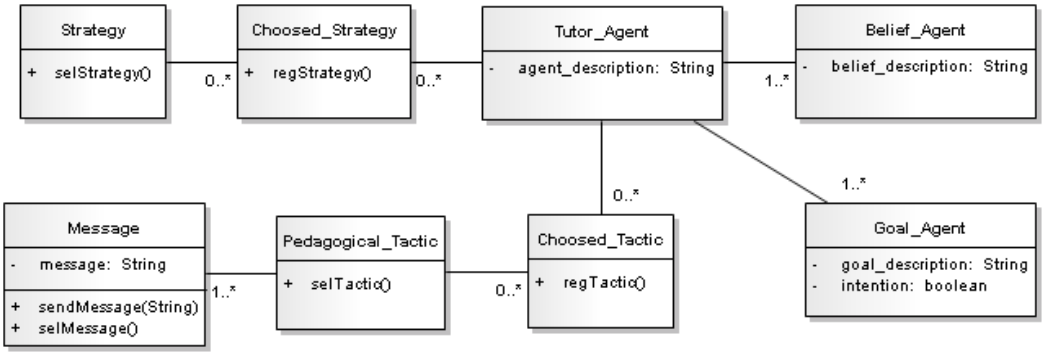


Figura 3.15: Tutor Knowledge Diagram – Diagrama de Conhecimento do Tutor

Voltando à figura 3.14, o plano "Help the Student" consiste primeiramente em selecionar uma estratégia seguida da seleção de uma tática pedagógica. Estas escolhas se baseiam nas percepções anteriores. Com base na estratégia e tática escolhidas e no personagem assumido pelo aluno, o Tutor determina que mensagem enviar ao aluno de maneira a auxiliá-lo a melhorar o nível de energia do lago. Após isso, o tutor registra a estratégia e tática escolhidas por uma questão de registro histórico de forma a saber quais estratégias e táticas já foram tentadas.

As mensagens enviadas pelo agente que interpreta o papel de Tutor ao aluno são mensagens de texto simples pré-definidas enviadas pelo Tutor por meio do sistema aos alunos que, sendo agentes humanos, são externos ao sistema. Se os agentes que assumem o papel de alunos fossem agentes de software, poderia-se empregar uma linguagem de comunicação de agente como FIPA, embora neste caso fosse necessário estender o diagrama de sequência de

forma a modelar mensagens FIPA, ou empregar alguma das linguagens para modelagem de sistemas multi-agentes já descritas, como a MAS-ML.

Para finalizar, apresentamos o diagrama de casos de uso completo desenvolvido para a modelagem dos requisitos do sistema MCOE, de maneira a demonstrar a visão total dos requisitos funcionais deste sistema.

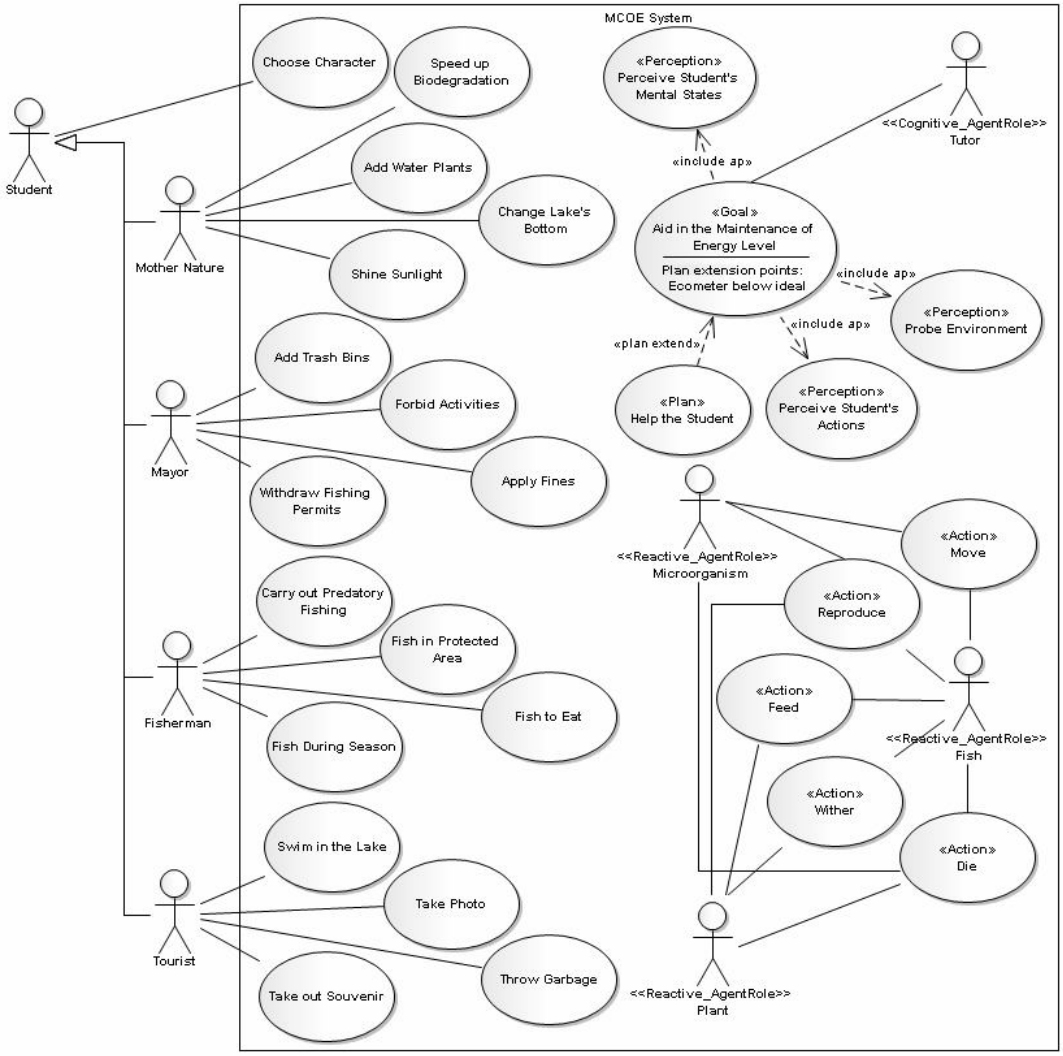


Figura 3.16: Diagrama de Casos de Uso Completo para a Modelagem dos Requisitos Funcionais do sistema MCOE

Neste exemplo particular os casos de uso normais representam funcionalidades que somente podem ser solicitados pelos alunos externos quando assumem determinados personagens. As ações dos alunos podem, no entanto, influir nas percepções dos agentes.

3.6 Estudo de Caso II – O ambiente AME-A - Ambiente Multi-Agente de Ensino-Aprendizagem

O ambiente AME-A – Ambiente Multi-Agente de Ensino-Aprendizagem (D'AMICO, 1998), é composto por uma sociedade híbrida de agentes que cooperam para auxiliar no

aprendizado dos alunos. O ambiente interage com agentes humanos que podem tanto ser o professor como os alunos e possui diversos agentes reativos e cognitivos.

Este estudo de caso foi publicado em (GUEDES, 2010b), na terceira conferência internacional IFIP sobre inteligência artificial em teoria e prática (IFIP AI 2010), sendo parte do Congresso Mundial de Computação IFIP (WCC 2010), em Brisbane na Austrália. A modelagem deste sistema foi também publicada, com maior detalhe, em (GUEDES, 2010c), na 7ª. conferência internacional sobre qualidade de informação e tecnologia de comunicações, na trilha de Engenharia de Requisitos – QUATIC 2010. Finalmente, este estudo de caso foi também apresentado, de uma forma sucinta, em (GUEDES, 2011a) na 23ª. conferência internacional sobre engenharia de software e engenharia do conhecimento – SEKE 2011.

Neste ambiente, o professor pode criar uma atividade de aprendizagem ou avaliar os alunos com o auxílio do agente reativo que assumir o papel de agente Teacher's Tools (Ferramentas para Professor). Já o aluno pode escolher entre executar uma sessão de aprendizagem não monitorada ou uma sessão de aprendizagem monitorada. Na primeira ele só interage com o agente reativo que assumir o papel de Unsupervised Learning (Aprendizagem Não Monitorada), que apenas apresenta o conteúdo a ser aprendido.

A atividade de aprendizagem monitorada é o foco principal do sistema, onde se procura maximizar o aprendizado do aprendiz por meio do auxílio de cinco papéis de agente cognitivos, a saber: Student Modeling (SM) – Modela Aprendiz, Methodology and Teaching Plan (MTP) – Metodologia e Plano de Ensino, Learning Orientation (LO) – Orienta Aprendizagem, Learning Analysis (LA) – Analisa Aprendizagem e Knowledge Application Orienting (KAO) – Orienta Aplicação do Conhecimento. O primeiro modela o perfil do aluno de forma dinâmica, enquanto o segundo escolhe a metodologia e plano de ensino mais adequada ao perfil do aluno sempre que este mudar ou quando o desempenho do aluno for aquém do esperado; o papel de agente Orienta Aprendizagem seleciona o material a ser ensinado e a forma como este será apresentado de acordo com a metodologia escolhida; já o papel de agente Analisa Aprendizagem verifica o desempenho do aluno ao longo da sessão e o papel de agente Orienta Aplicação do Conhecimento aplica uma avaliação geral após o término da sessão.

Uma vez que o professor e o aluno são papéis assumidos por agentes humanos que interagem com o sistema, mas são externos a ele, os mesmos foram representados como atores normais, fora da fronteira do sistema e associamos ao ator que representa o professor as funcionalidades, representadas por casos de uso normais, “Create learning activity” (Criar atividade de ensino) e “Evaluate students” (Avaliar alunos). Observe que nestas funcionalidades há também uma interação com o agente que interpreta o papel de agente Teacher Tools, que, sendo um papel de agente reativo, foi representado como um AgentRole_Actor com o estereótipo “Reactive_AgentRole” e posto dentro da fronteira do sistema, uma vez que ele está inserido no software.

Já o ator aluno foi associado às funcionalidades “Execute an unmonitored learning session” (Executar sessão de atividade não monitorada) e “Execute a monitored learning session” (Executar sessão de aprendizagem monitorada), igualmente representadas como casos de uso normais. Na primeira funcionalidade há também uma interação com o agente reativo Unsupervised Learning (Aprendizagem não Supervisionada), que, da mesma forma que o agente anterior foi representado como um AgentRole_ Actor com o estereótipo “Reactive_AgentRole”.

A funcionalidade “Execute monitored learning session” representa a atividade mais importante do sistema e é a que apresenta maior complexidade, envolvendo os cinco papéis de

agente cognitivos do software, representados como AgentRole_Actors dentro do sistema, contendo o estereótipo “Cognitive_AgentRole”.

Estas funcionalidades foram representadas como casos de uso normais, devido a representarem serviços que os usuários externos podem solicitar ao sistema. A próxima figura ilustra este trecho da modelagem. É importante destacar que esta figura apresenta apenas uma parte do diagrama de casos de uso. Por uma questão de visibilidade, suprimimos a representação dos objetivos, planos, percepções e ações dos papéis de agente cognitivos, detalhando em separado os requisitos funcionais de cada papel de agente. Para manter a ligação com este trecho da modelagem, manteremos a associação dos papéis de agente cognitivos com o caso de uso “Execute monitored learning activity”.

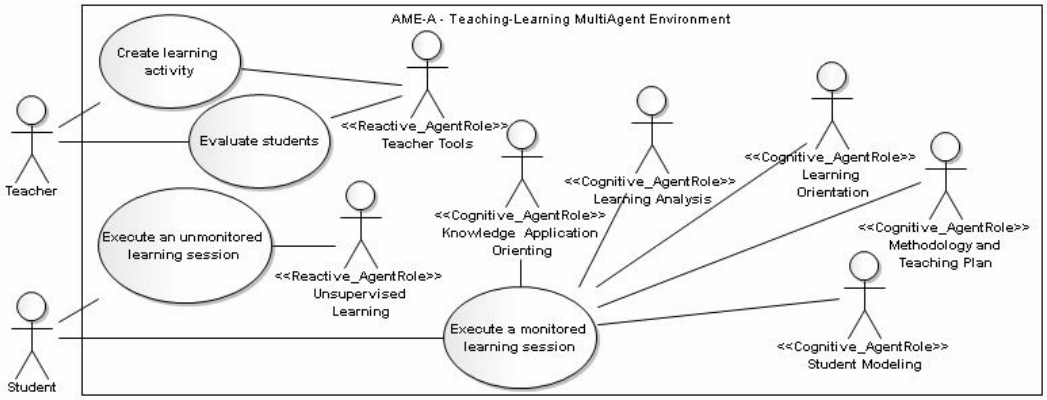


Figura 3.17: Atores Normais Modelando Papéis Externos, Casos de Uso Normais Modelando Funcionalidades e AgentRole_Actors Estereotipados Modelando Papéis de Agente Reativos e Cognitivos

O papel de agente SM tem como objetivo modelar o perfil do aluno de forma dinâmica. Este papel de agente possui duas percepções, o agente que o assumir precisa perceber o início da sessão de aprendizagem e, nesse caso, disparar o plano “Apply questionary” (aplicar questionário) para determinar o perfil do aluno. Além disso, o agente precisa perceber quando o comportamento do aluno muda, caso em que precisa disparar o plano para remodelar o perfil do aluno. A figura a seguir ilustra a modelagem destes requisitos para o papel de agente SM.

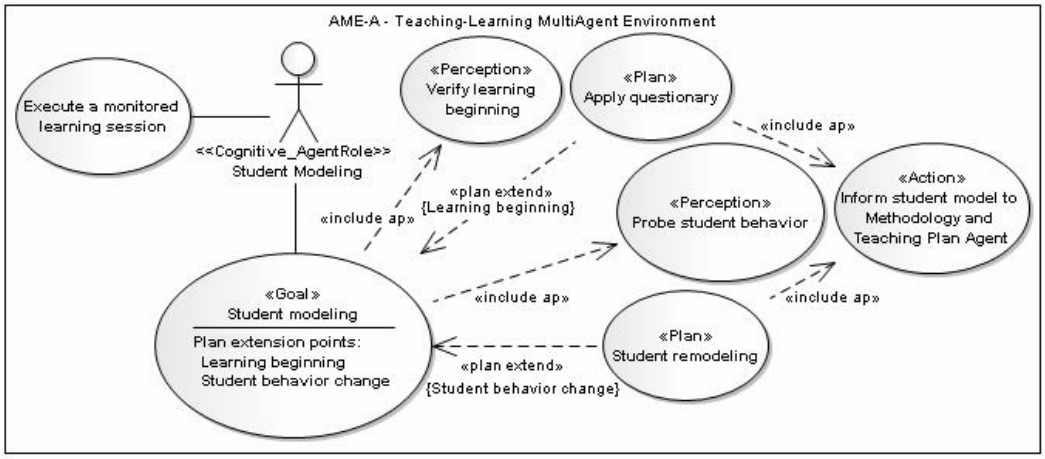


Figura 3.18: Casos de Uso Internos com Estereótipos Goal, Plan, Perception e Action

Nesta figura associamos o papel de agente SM a um caso de uso interno com o estereótipo Goal representando o objetivo que o agente que assume este papel tem, ou seja, o de modelar o perfil do aluno. Observe que este objetivo tem duas associações de inclusão ActionPerception com dois casos de uso internos que representam as percepções que o agente precisa possuir para determinar se é necessário disparar algum dos planos associados ao objetivo. Assim, para atingir seu objetivo, o agente que assumir o papel SM tem obrigatoriamente de executar também o comportamento destas percepções. Dessa forma, foi utilizado um caso de uso interno com o estereótipo Perception para representar a percepção de início da sessão de aprendizagem e outro caso de uso interno com o mesmo estereótipo para representar a percepção do comportamento do aluno.

Observe que, no caso de uso interno que representa o objetivo, existem dois Plan extension points, que representam os pontos no comportamento do objetivo onde podem ser estendidos os planos associados a ele, além de também estabelecerem as condições para que estes sejam executados. Dessa maneira, caso o agente (que interpreta o papel) perceba o início de uma sessão de aprendizagem, será disparado o plano para aplicar um questionário ao aluno, representado por um caso de uso interno com o estereótipo Plan; e, se o agente perceber uma mudança no comportamento do aluno, será disparado o plano para remodelar o perfil do aluno, igualmente representado por um caso de uso interno com o estereótipo Plan. É importante destacar que as condições para disparo dos planos são independentes, ou seja, a condição para disparar o plano para aplicar um questionário ao aluno não necessita que a condição para disparar o plano para remodelar o perfil do aluno seja verdadeira também, dessa maneira a execução de um plano não implica a execução de outro.

Observe ainda que os dois casos de uso internos que representam os planos estão associados ao caso de uso interno Goal por meio de associações plan extend, ou seja, estes casos de uso internos só serão executados mediante a satisfação das condições detalhadas pelos Plan extension points. Finalmente, observe que a execução de ambos os planos implica em enviar o perfil do aluno modelado durante a execução do plano ao agente que interpretar o papel de agente MTP. Isto é representado por meio de um caso de uso interno contendo o estereótipo Action e associados aos casos de uso internos Plan por meio de associações de inclusão ActionPerception, indicando a obrigatoriedade da execução desta ação por estes planos.

O papel de agente seguinte, MTP, tem como objetivo escolher a metodologia e o plano de ensino mais adequado ao aluno, para isso o agente que o interpretar precisa perceber quando o modelo do aluno muda, isso inclui a percepção de quando o aluno é modelado pela primeira vez, quando não há modelo algum. Esse objetivo possui um plano associado, "Change Learning Methodology and Teaching Plan" (Mudar metodologia de aprendizando e plano de ensino), que será disparado quando o modelo do aluno mudar ou quando o desempenho do aluno for baixo. A execução deste plano inclui o envio de uma mensagem ao agente que interpreta o papel de agente LO informando que a metodologia foi alterada.

Conforme pode ser visto na próxima figura, para modelar estes requisitos associou-se um caso de uso interno com o estereótipo Goal ao AgentRole_Actor MTP para representar seu objetivo. Em seguida, foram relacionados a este objetivo, por meio de associações de inclusão ActionPerception, dois casos de uso internos com o estereótipo Perception, para representar as percepções de mudança do modelo do aluno e de desempenho do mesmo.

Após isso, foi criada uma associação plan extend para ligar o caso de uso interno com o estereótipo Plan, que representa o plano de mudança de metodologia e plano de ensino ao caso de uso interno Objetivo. Este plano somente será disparado quando o modelo do

estudante mudar ou quando o desempenho do aluno for baixo, conforme demonstra os Plan extension points do caso de uso interno Goal. Finalmente, caso o plano seja executado, é preciso comunicar ao agente que assume o papel de agente LO a mudança da metodologia, como isso é feito por meio de uma comunicação entre agentes, identificamos isto como uma ação e a associamos ao plano por meio de um caso de uso interno com o estereótipo Action.

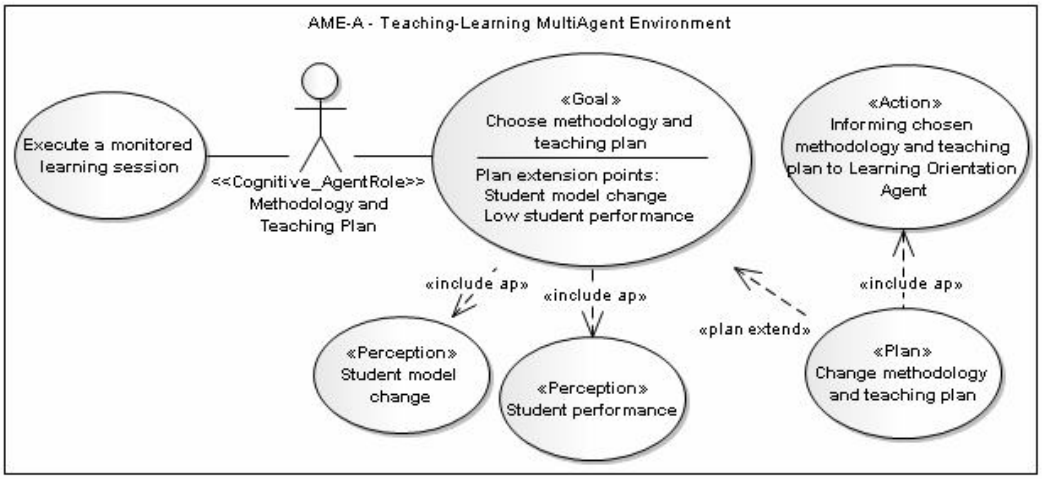


Figura 3.19: Requisitos Relativos ao Papel de Agente Methodology and Teaching Plan

O papel de agente LO tem como objetivo apresentar material de aprendizagem ao aluno e como percepção a escolha da metodologia e plano de ensino. Quando a escolha de uma metodologia e plano de ensino é percebida pelo agente que assume o papel LO, ele executa o plano "Select material for learning" (Selecionar material para aprendizagem). Estes requisitos funcionais são modelados na figura a seguir, onde o objetivo, a percepção e o plano foram representados como casos de uso internos contendo respectivamente os estereótipos Goal, Perception e Plan. Como o plano somente será disparado quando for percebida a escolha de uma metodologia, há uma associação de inclusão ActionPerception entre o objetivo e a percepção, obrigando o agente a verificar se isso ocorre. Há também uma associação plan extend entre o objetivo e o plano, uma vez que o plano só será executado se a condição for satisfeita.

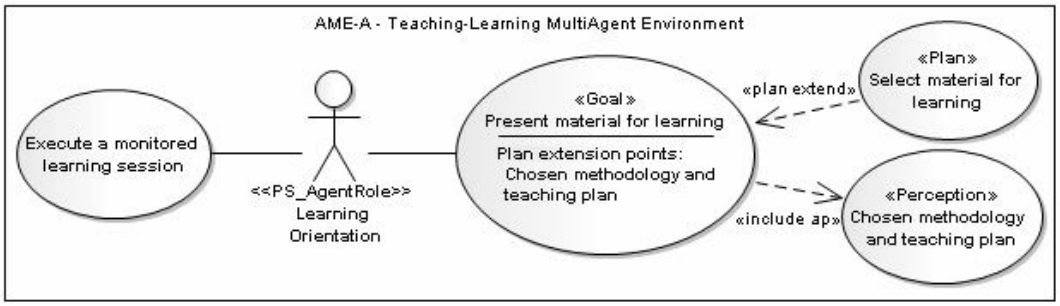


Figura 3.20: Requisitos relativos ao Papel de Agente Learning Orientation

A próxima figura apresenta a modelagem relacionada aos requisitos do papel de agente LA, que tem como objetivo verificar o conhecimento adquirido pelo aluno, representado por um caso de uso interno contendo o estereótipo Goal. Para executar este objetivo, o agente que interpretar este papel precisa perceber o desempenho do aluno, esta percepção é representada por um caso de uso interno contendo o estereótipo Perception. Além disso, o agente que

assumir o papel LA necessita informar ao agente que interpreta o papel MTP este desempenho, isto é representado como um caso de uso interno contendo o estereótipo Action. Se o desempenho do aluno for considerado baixo, então é disparado o plano "Boosting student" (Incentivar aluno), igualmente representado como um caso de uso interno com o estereótipo Plan. Este plano executa a ação de enviar mensagens de incentivo ao aluno, dessa forma, isso foi identificado como um caso de uso interno contendo o estereótipo Action e este foi ligado ao plano por meio de uma associação de inclusão ActionPerception.

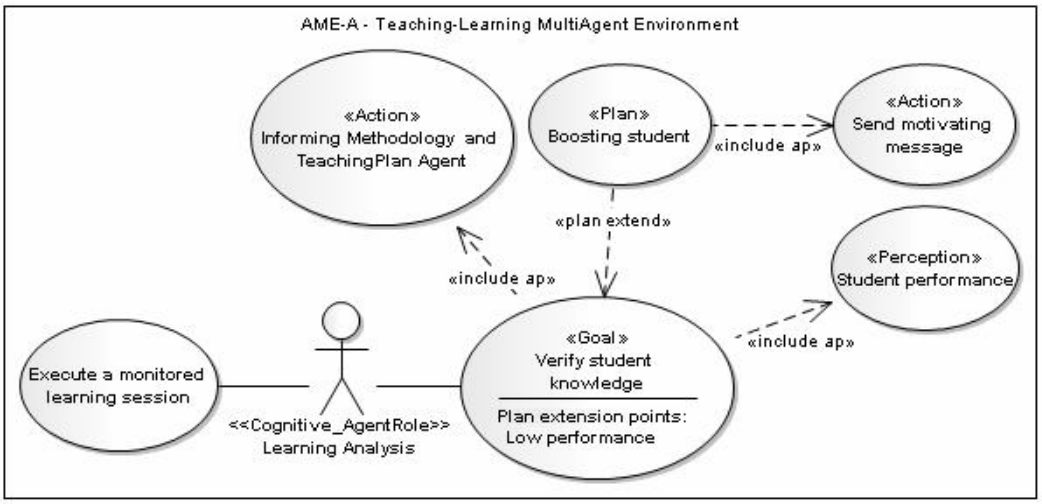


Figura 3.21: Requisitos relativos ao Papel de Agente Learning Analysis

O último papel de agente envolvido no processo, o papel de agente KAO, tem como objetivo avaliar o aluno após o fim da sessão. Dessa forma, o agente que interpretar este papel precisa perceber o fim da sessão de aprendizado para saber quando disparar o plano "Apply evaluation" (Aplicar avaliação). Aqui aplicamos os mesmos estereótipos utilizados nos exemplos anteriores em casos de uso internos, conforme demonstrado na figura a seguir.

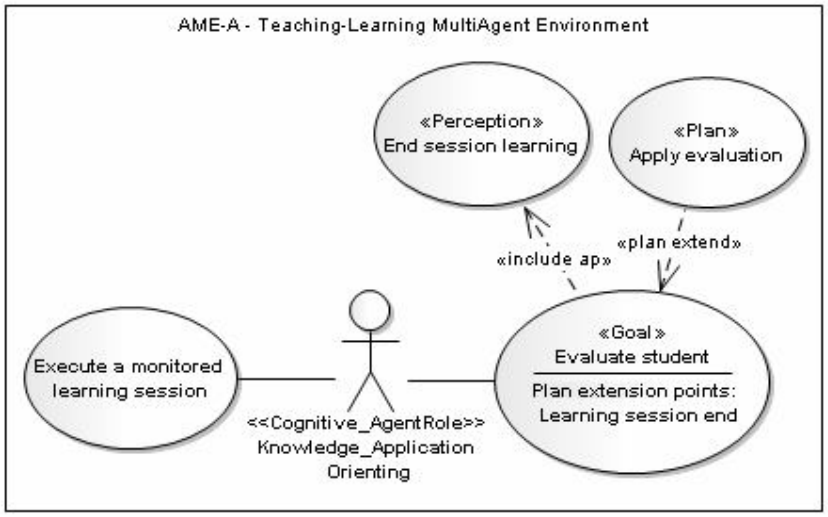


Figura 3.22: Requisitos Relativos ao Papel de Agente Knowledge Application Orienting

Para apresentar uma visão geral dos requisitos do projeto AME-A, a figura seguinte apresenta o diagrama de casos de uso completo para este sistema.

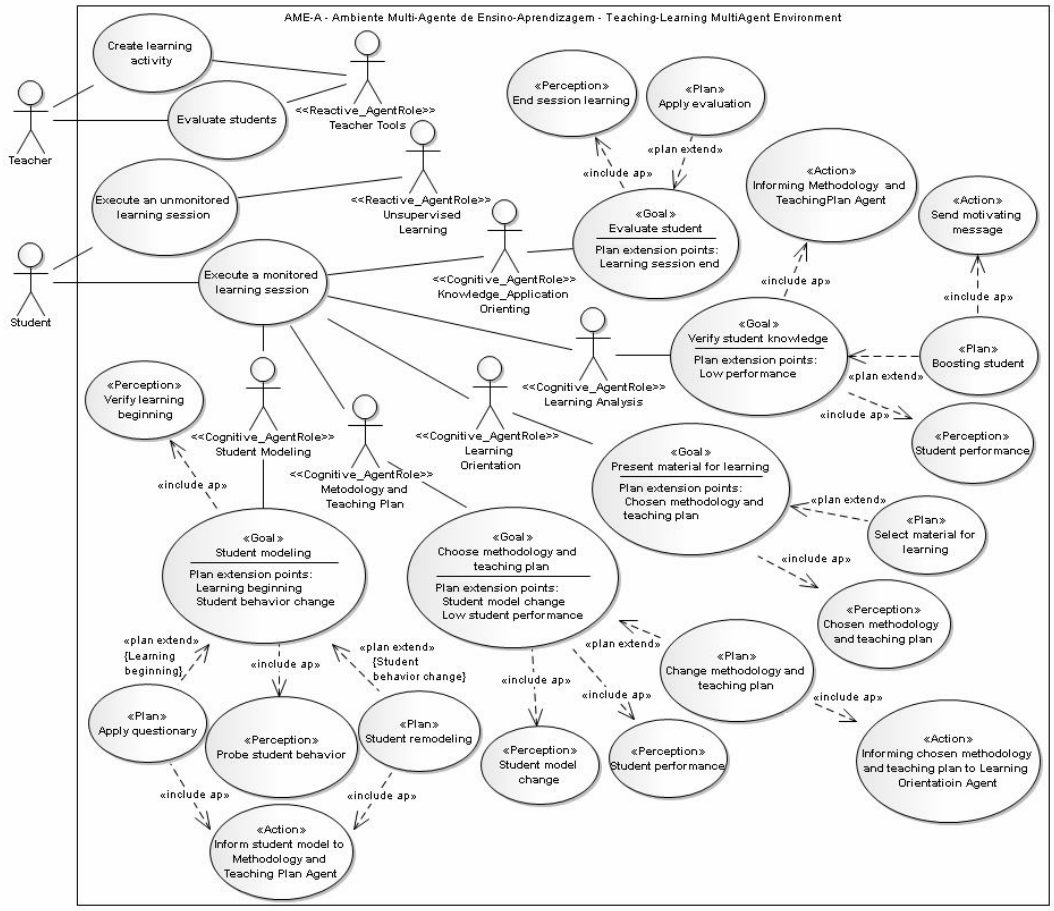


Figura 3.23: Diagrama de Casos de Uso Completo para a Modelagem dos Requisitos do Sistema AME-A

Neste diagrama, como no estudo de caso anterior, foram representados explicitamente as percepções e ações associadas aos objetivos e planos. Poderia-se argumentar que estas percepções e ações poderiam estar inclusas no comportamento dos objetivos ou planos, contudo consideramos ser importante representar graficamente estas percepções e ações de forma a tornar claras e visíveis as percepções e ações necessárias a um determinado papel de agente. As ações aqui modeladas representam atividades realizadas pelos agentes que, de alguma forma, afetam o ambiente e/ou podem ser percebidas pelos outros agentes.

3.7 Adequação do Metamodelo aos Princípios de Projeto Propostos por Vicari (2007)

Levando em consideração que Vicari (2007) sugere o uso dos agentes especialistas Problem Solving (PS), Users and Agents Modelling (UAM) e Social Mediated Interactions (SMI) para o projeto de sistemas multi-agentes, como já foi descrito anteriormente, evoluímos o metamodelo proposto de maneira a ser possível modelar tais tipos de papéis de agentes, como demonstra a figura a seguir, onde pode-se observar que foram derivadas três metaclasses a partir da metaclasse Cognitive_AgentRole, a saber: PS_AgentRole, UAM_AgentRole e SMI_AgentRole, para identificar papéis referentes a cada um dos tipos de agentes propostos por Vicari (2007).

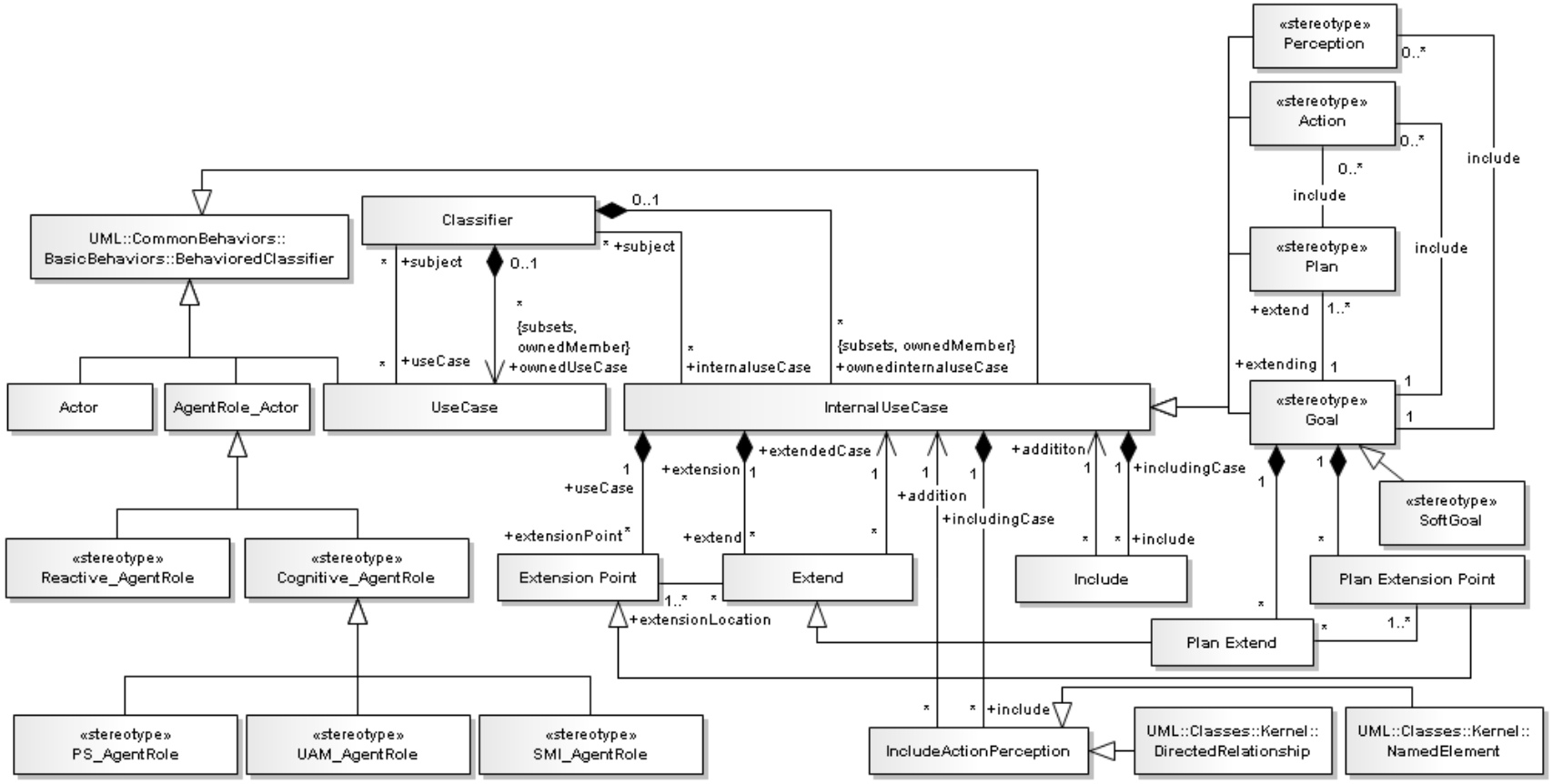


Figura 3.24: O Metamodelo UML Estendido para Suportar os Tipos de Agente Propostos por Vicari (2007).

A partir desta nova versão do metamodelo, modificou-se a modelagem do projeto AME-A de acordo com esses novos estereótipos, como demonstra a figura seguinte.

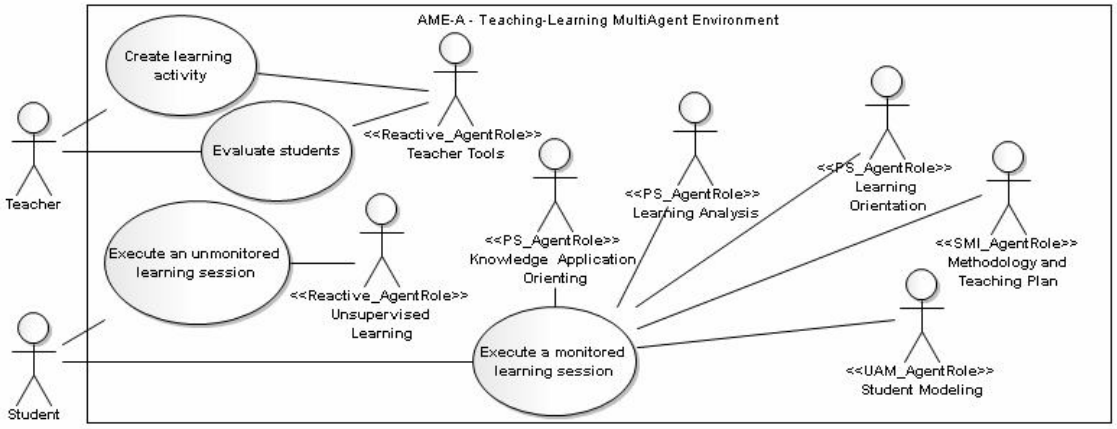


Figura 3.25: Modelagem dos Requisitos do Projeto AME-A de Acordo com os Novos Estereótipos Criados.

Ao observar esta figura percebe-se que todos os papéis de agentes cognitivos foram classificados como papéis de agente PS, UAM ou SMI, como demonstram seus estereótipos. Assim, o papel de agente Student Modeling recebeu o estereótipo UAM_AgentRole, uma vez que, como seu próprio nome diz, ele é responsável pela modelagem dos alunos que interagirão com o sistema.

Por sua vez o papel de agente Methodology and Teaching Plan recebeu o estereótipo SMI_AgentRole, já que este é responsável por estabelecer qual a metodologia deverá ser utilizada com cada aluno, o que implica também em como será a forma de interação entre os agentes de software e o agente humano.

Todos os outros papéis de agente cognitivos receberam o estereótipo PS_AgentRole, pois são papéis de agente que devem possuir conhecimento a respeito do problema que a aplicação pretende resolver.

Vicari (2007) também propõe, como já foi descrito anteriormente, uma arquitetura em camadas para agentes individuais, formada pelas camadas de Julgamento e Decisão (JDL - Judgment and Decicion Layer), de Avaliação e Planejamento (APL - Assessment and Planning Layer) e de Percepção e Ação (PAL - Perception and Action Layer), como pode ser visto na figura a seguir, onde cada camada é representada por um símbolo de pacote UML, enquanto as setas de dependência demonstram que a informação entre as camadas trafega nos dois sentidos.

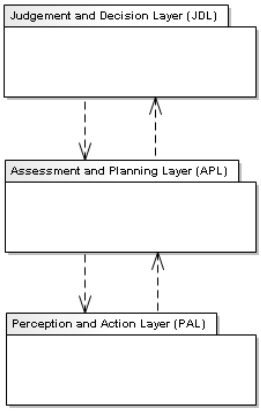


Figura 3.26: Arquitetura de Camadas Proposta por Vicari (2007).

Com o objetivo de usar a arquitetura proposta por Vicari (2007), em um nível de análise de requisitos, representou-se em quais camadas os requisitos modelados para cada papel de agente deveriam ser implementados, como demonstra a figura a seguir.

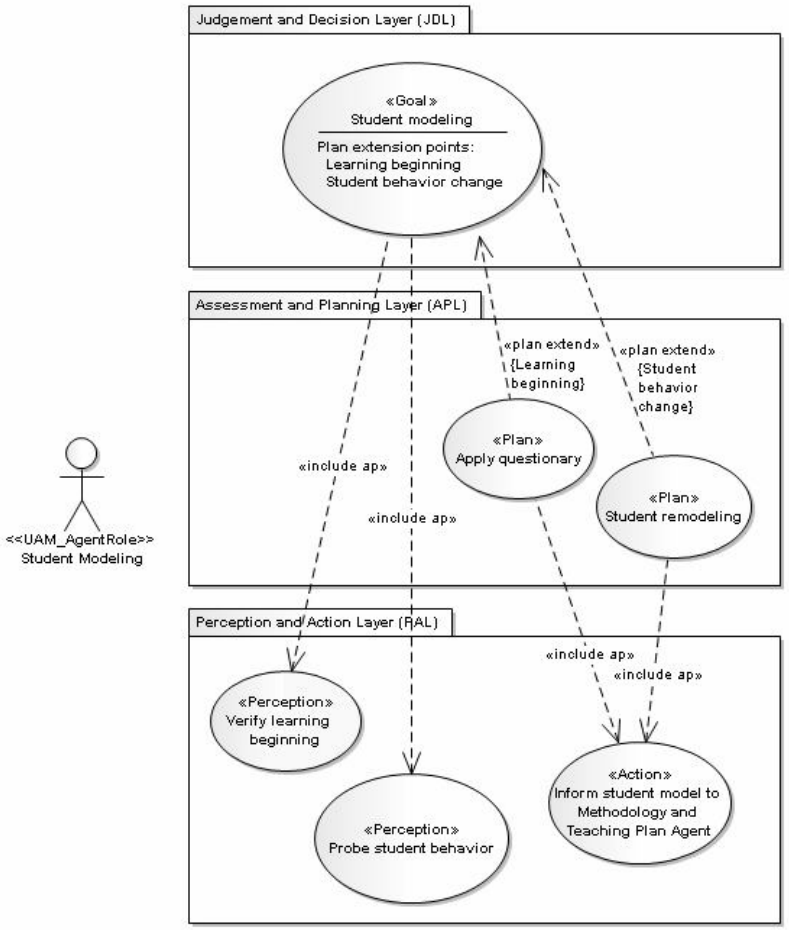


Figura 3.27: Requisitos Modelados dentro da Arquitetura em Camadas Proposta por Vicari (2007) para o Papel de Agente Student Modeling.

Nesta figura os requisitos relacionados às percepções e ações do papel de agente Student Modeling foram colocados na camada de percepção e ação, uma vez que esta camada é

responsável pelas percepções básicas e pelas ações elementares do agente. É nesta camada que o agente percebe os eventos do mundo externo e atua no mundo quando ordenado pelas camadas superiores.

Já os casos de uso internos referentes aos planos que podem ser executados pelo papel de agente foram colocados na camada de avaliação e planejamento, posto que a execução destes planos é responsabilidade desta camada. Esta camada também é responsável por avaliar as percepções identificadas na camada inferior e repassá-las à camada superior, por este motivo as ações de inclusão ActionPerception entre o caso de uso interno Goal e os casos de uso internos que representam suas percepções cruzam esta camada.

Finalmente o caso de uso interno referente ao objetivo de modelar o aluno foi posto na camada de julgamento e decisão. Nesta camada o agente que assumir o papel de agente em questão receberá as percepções detectadas na primeira camada e que foram analisadas e repassadas pela camada intermediária. Baseado nestas percepções o agente decidirá, nesta última camada, se os desejos representados pelos objetivos poderão se tornar ou não intenções, em caso positivo ele executará os planos associados ao objetivo, como demonstram as associações de plan extend entre o caso de uso interno Goal com os casos de uso internos Plan.

As possíveis ações externas que possam vir a ser executadas por um plano devem estar posicionadas na camada inferior, por esta ser responsável pela execução das ações externas.

3.8 Estensibilidade das Metaclasses Definidas no Metamodelo

Com relação à estensibilidade, a maioria das metaclasses são estensíveis, embora neste momento o metamodelo tenha sido dado como concluído, nada impede que sejam derivadas novas metaclasses a partir das metaclasses AgentRole_Actor, InternalUseCase ou das metaclasses derivadas a partir destas ou mesmo que sejam criadas novas metaclasses a partir de metaclasses primitivas como a metaclasse Classifier.

Uma exceção seja talvez com relação às metaclasses Plan Extend, Plan Extension Point e IncludeActionPerception, uma vez que acreditamos que suas funções não necessitam de novas extensões, mas mesmos estas, se for considerado necessário, também são passíveis de ser estendidas.

3.9 Estudo de Caso III – Sistema de Apoio Pedagógico ao Aluno – Sistema APA

Nesta seção será apresentada a modelagem do sistema APA já utilizando os estereótipos criados para suportar os tipos de agentes sugeridos por Vicari (2007).

O sistema APA (Apoio Pedagógico ao Aluno) é um projeto que pretende utilizar o padrão OBAA, um padrão de metadados para objetos de aprendizagem que almeja produzir, editar e distribuir conteúdos digitais interativos na Web, em dispositivos móveis e televisão digital. Neste sistema os usuários poderão procurar por objetos de aprendizagem por meio de consultas (queries) ou receber sugestões sobre objetos de aprendizagem de uma forma pró-ativa dos agentes que irão interpretar os papéis de agente suportados pelo software.

O sistema suporta três papéis de agente que tem como funções receber as perguntas dos usuários e então tentar retornar metadados relacionados a objetos de aprendizagem que estejam de acordo com as pesquisas realizadas. Além disso, os agentes que viessem a interpretar esses papéis deveriam ser pró-ativos, ou seja, deveriam perceber as ações dos

usuários, modelar seus perfis e sugerir metadados sobre os objetos de aprendizagem de acordo com estas ações e perfis sem precisar haver uma solicitação disso por parte do usuário.

A única funcionalidade externa oferecida pelo sistema é a funcionalidade "Performing Query" (Realizar Consulta), por meio da qual o usuário pode solicitar uma pesquisa sobre objetos de aprendizagem. Quando recebem estas consultas, os agentes tentam retornar metadados referentes aos objetos de aprendizagem relacionados às pesquisas. O sistema é composto pelos papéis de agente Intelligent Search (Pesquisa Inteligente), User Model (Modela Usuário) e Pedagogical Support (Apoio Pedagógico).

A função do papel de agente Intelligent Search é a de retornar metadados sobre os objetos de aprendizagem de acordo com as consultas; enquanto estiver envolvido nesse processo, ele deve tentar reduzir a sobrecarga de informação, evitando o retorno de quantidades imensas de informação desnecessária. Já o papel de agente Pedagogical Support tem por objetivo sugerir material pedagógico ao usuário, enquanto o objetivo do papel de agente User Model é o de modelar perfis dos usuários.

A figura seguinte apresenta a modelagem, por meio do diagrama de casos de uso, dos requisitos funcionais deste sistema. Levando em consideração que o usuário é um papel assumido por um agente humano externo, este foi modelado como um ator normal atuando fora da fronteira do sistema e a este ator foi associada a funcionalidade “Performing Query” representada por um caso de uso normal, posto que é por meio desta funcionalidade que o usuário pode submeter consultas ao sistema.

Todos os papéis de agente que compõem este sistema foram modelados como AgentRole_Actors e representados dentro da fronteira do sistema, uma vez que estes papéis são internos e transparentes para os usuários externos e serão assumidos pelos agentes que irão compor o sistema.

O AgentRole_Actor que representa o papel de agente User Model recebeu o estereótipo UAM_AgentRole, devido ao fato deste papel de agente possuir como função modelar os perfis dos usuários do sistema, assim, o agente que assumir este papel precisa ter conhecimento sobre como modelar estes perfis.

Já o papel de agente Intelligent Search foi atribuído o estereótipo PS_AgentRole, porque o agente que assumir este papel deve possuir conhecimento a respeito do problema que o sistema deve auxiliar a solucionar.

Finalmente ao papel Pedagogical Support foi imposto o estereótipo SMI_AgentRole posto que o agente que assumir este papel deverá possuir conhecimento sobre como mediar as interações entre os usuários e os agentes.

O objetivo do papel de agente Intelligent Search foi representado como um caso de uso interno contendo o estereótipo Goal. Há dois meios deste objetivo se tornar um desejo para o agente que interpretar este papel, ou seja, do agente passar a ter a intenção de atingir este objetivo; o primeiro quando o agente percebe que o usuário submeteu uma consulta; e o segundo quando o agente recebe uma solicitação de consulta por parte do agente que interpreta o papel de agente Pedagogical Support.

Estas percepções foram modeladas como casos de uso internos com o estereótipo Perception e denominados "Verifyig User Query" (Verificar Consulta de Usuário) e "Verifying Search Solicitation Pedagogical Support Agent" (Verificar Solicitação de Pesquisa pelo Agente de Apoio Pedagógico). Se quaisquer dessas percepções se provar verdadeira, o plano "Searching Metadata" (Pesquisar Metadados) irá ser disparado. Este plano foi modelado como um caso de uso interno contendo o estereótipo "Plan", a associação de Plan extend entre os casos de uso internos Goal e Plan demonstra que uma condição deverá ser satisfeita para que o comportamento do plano seja acrescentado ao comportamento do objetivo.

Os Plan extension points do caso de uso interno Goal determinam os pontos nos quais o comportamento deste objetivo pode ser estendido pelo caso de uso interno Plan. A execução do plano implica a execução da ação de enviar metadados ao agente que assumir o papel de agente Pedagogical Support. Como esta é uma ação que envia uma mensagem para outro papel de agente, ela foi modelada como um caso de uso interno com o estereótipo Action.

O AgentRole_Actor que representa o papel de agente User Model foi associado a um caso de uso interno chamado "Modeling User Profile" (Modelar Perfil de Usuário). Como pode-se ver pelo estereótipo Goal deste caso de uso interno, ele representa o objetivo do agente que interpretar este papel de modelar os perfis dos usuários que interagem com o sistema.

Este objetivo se torna um desejo quando o agente percebe as ações realizadas pelos usuários do sistema. Portanto, modelou-se esta percepção (Verifying User Action – Verificar Ação do Usuário) como um caso de uso interno e atribuiu-se a ele o estereótipo Perception. Observe que há uma associação Include ActionPerception (include ap) entre o caso de uso interno Goal e o caso de uso interno Perception, o que implica em uma obrigatoriedade, ou seja, quando o primeiro caso de uso interno for executado o segundo irá também ser executado.

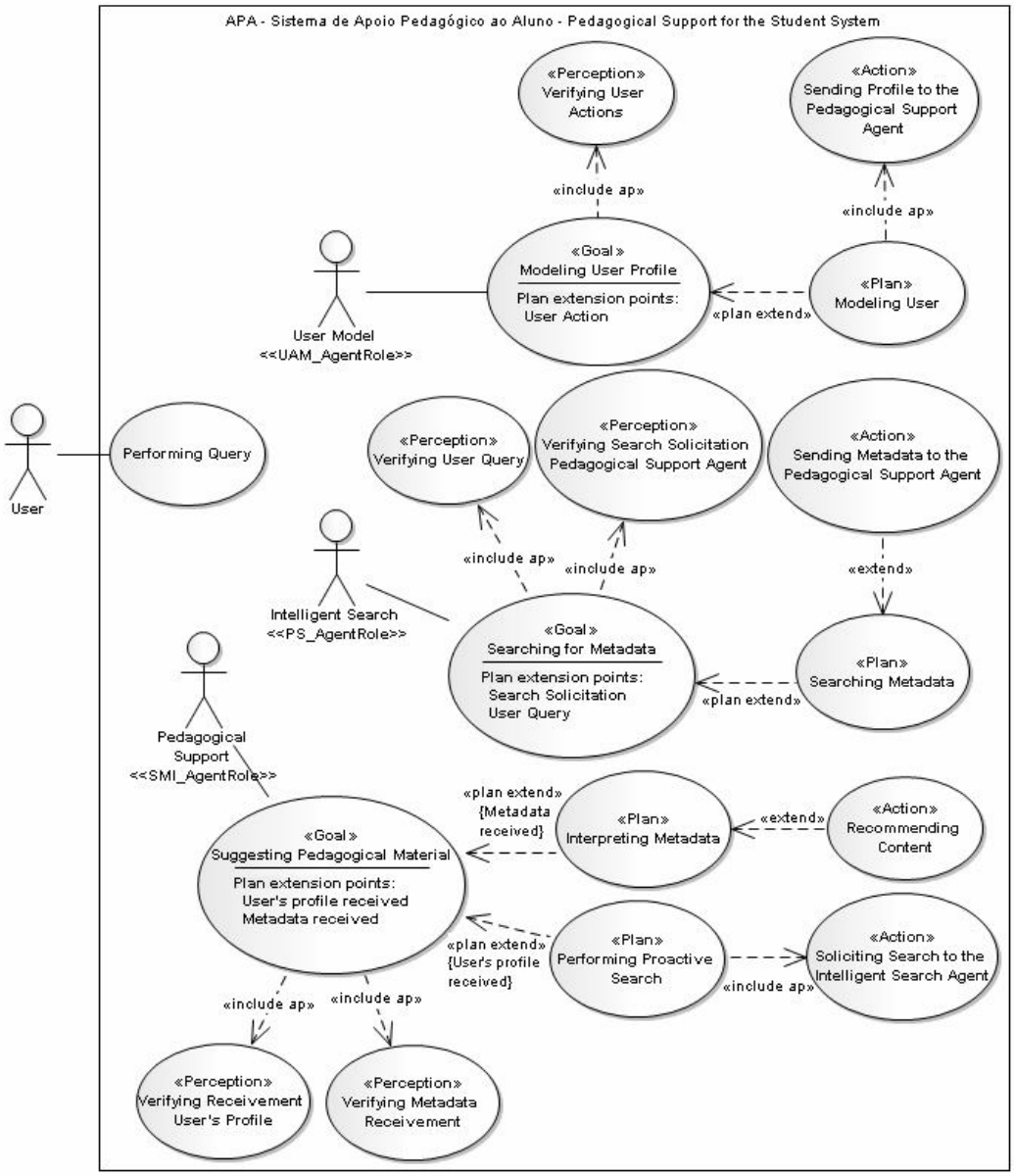


Figura 3.28: Diagrama de Casos de Uso para a Modelagem de Requisitos do Sistema APA

Assim, sempre que o agente perceber uma ação de um usuário ele irá disparar o plano denominado "Modeling User" (Modelar Usuário), representado por um caso de uso interno com o estereótipo Plan, como demonstra a associação de extensão de plano (Plan extend association) entre os casos de uso internos Goal e Plan. Este plano realiza uma ação que comunica o perfil do usuário ao agente que interpreta o papel de Pedagogical Support. Esta ação foi representada por um caso de uso interno contendo o estereótipo Action.

O AgentRole_Actor que representa o papel de agente Pedagogical Support foi associado ao caso de uso interno "Suggesting Pedagogical Material" (Sugerir Material Pedagógico) e este, como pode ser percebido pelo estereótipo Goal, representa o objetivo deste papel de agente. Há duas situações nas quais este objetivo pode se tornar um desejo. Estas situações são representadas por dois plan extension points que representam os pontos no

comportamento do objetivo onde os planos associados a ele podem ser estendidos, além de estabelecer as condições para que estes planos sejam executados.

A primeira situação ocorre quando o agente percebe a recepção de um perfil de usuário, esta percepção foi representada como um caso de uso interno com o estereótipo Perception. Como demonstrado pelo primeiro Plan extension point, este evento torna verdadeiras as condições para que o comportamento do caso de uso interno Goal seja estendido pelo caso de uso interno Plan denominado "Performing Proactive Search" (Realizar Pesquisa Pró-Ativa).

Durante a execução deste plano o agente analizará o perfil recebido e solicitará permissão para realizar uma pesquisa de acordo com este perfil para o agente que interpreta o papel Intelligent Search. Esta requisição é representada por um caso de uso interno contendo o estereótipo Action.

A segunda situação ocorre quando o agente que interpreta o papel Pedagogical Support percebe a recepção de metadados enviados pelo agente que interpreta o papel de Intelligent Search, em resposta a uma requisição ou como uma resposta a uma consulta do usuário. Esta percepção foi igualmente representada por um caso de uso interno com o estereótipo Perception.

Neste caso, o agente dispara o plano "Interpreting Metadata" (Interpretar Metadados), representado pelo caso de uso interno com o estereótipo Plan, onde os metadados serão avaliados de acordo com o perfil do usuário e então apresentados ao usuário. Esta ação é representada por um caso de uso interno com o estereótipo Action (Recommending Content – Recomendar Conteúdo).

3.9.1 Arquitetura de Camadas Proposta por Vicari (2007)

Nesta seção nós modelamos os requisitos do papel de agent Pedagogical Support dentro das camadas de Vicari (2007), conforme demonstra a figura a seguir. Poderia-se ter feito isto para todos os papéis de agente, no entanto, somente este papel já seria suficientemente ilustrativo para este estudo de caso, levando-se em consideração que esta arquitetura foi projetada para ser utilizada para cada agente individualmente.

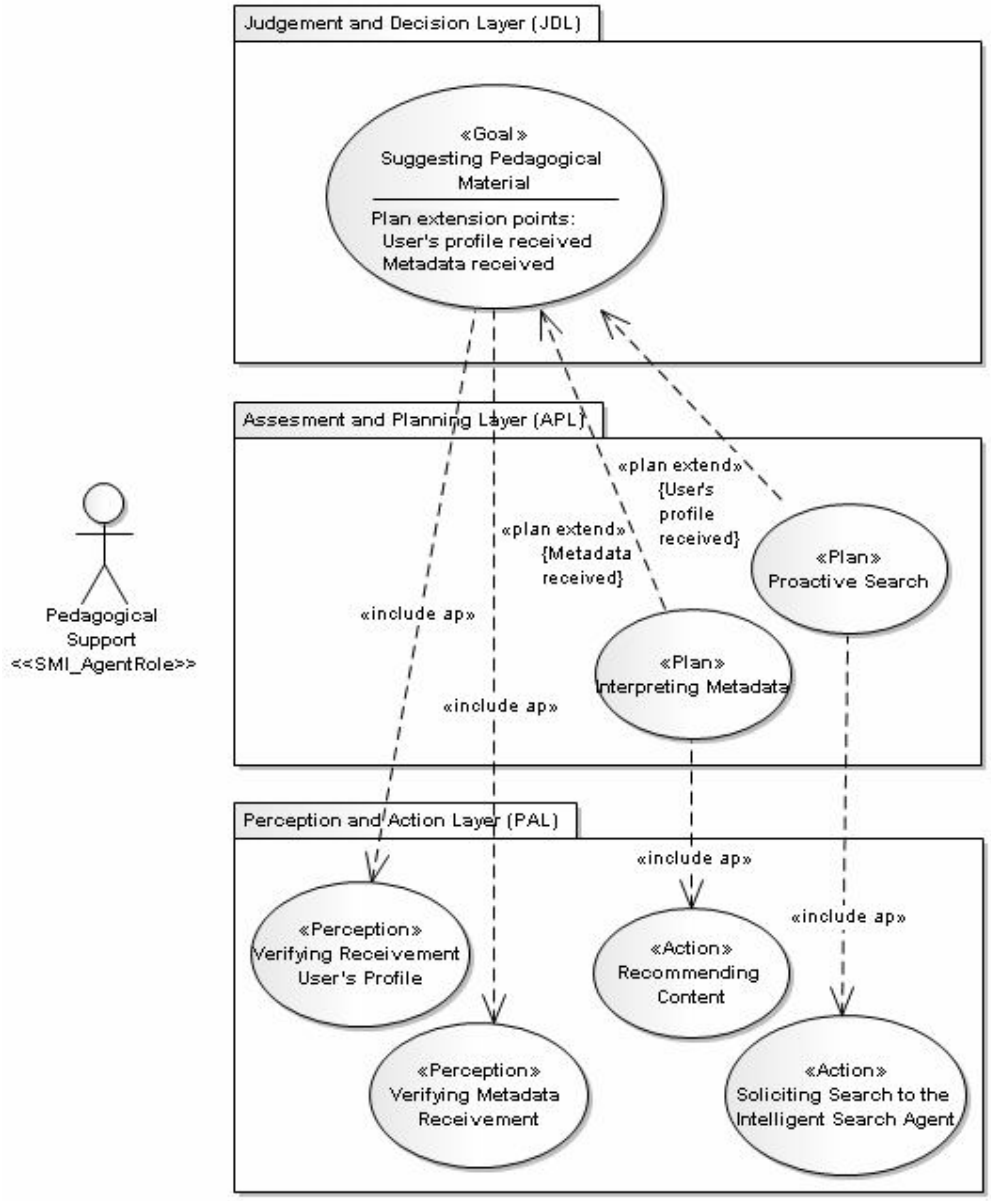


Figura 3.29: Requisitos Modelados dentro da Arquitetura em Camadas Proposta por Vicari (2007) para o Papel de Agente Pedagogical Support

Como pode ser observado, os casos de uso internos relacionados às percepções e ações do papel de agente Pedagogical Support foram colocados na camada de Percepção e Ação, devido a esta camada ser responsável pelas percepções básicas e pelas ações elementares do agente. É nesta camada que o agente percebe os eventos externos e age sobre o ambiente.

Os casos de uso internos relativos aos planos foram colocados na camada de Avaliação e Planejamento, já que a execução dos planos é de responsabilidade desta camada.

Finalmente, o caso de uso interno que se refere ao objetivo foi posto na camada de Julgamento e Decisão. Nesta camada o agente que executa este papel receberá as percepções identificadas na primeira camada e que foram analizadas e repassadas pela camada intermediária. Baseado nestas percepções o agente decidirá se os desejos representados pelos objetivos podem tornar-se intenções ou não; se a resposta for positiva, ele irá ordenar a

execução dos planos associados ao objetivo, como demonstram as associações de extensão de plano.

3.10 Mapeamento de Conceitos do Metamodelo Proposto para a Linguagem MAS-ML

Nesta seção propomos uma forma de mapeamento de conceitos do metamodelo proposto para a linguagem MAS-ML, o que seria útil para complementar tanto os artefatos produzidos por meio do metamodelo como os produzidos pela própria linguagem, caso se queira utilizar ambos.

Acreditamos que o conceito que mais se assemelhe ao AgentRole_Actor na MAS-ML seja a AgentRoleClass. Dessa forma mapeamos o AgentRole_Actor que representa o papel de agente Student Modeling juntamente com os casos de uso internos a ele associados em uma AgentRoleClass, conforme demonstra a figura seguinte.

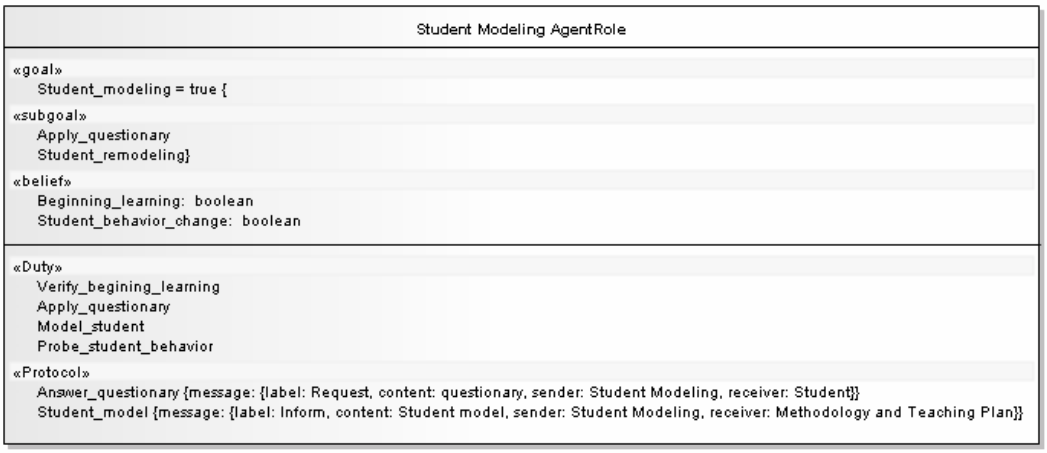


Figura 3.30: Mapeamento do AgentRole_Actor Student Modeling e os Casos de Uso Internos a ele Associados em uma AgentRoleClass

Nesta figura o AgentRole_Actor Student Modeling foi mapeado na AgentRoleClass Student Modeling. É importante observar que essa classe incorpora em suas características os objetivos, crenças, deveres e protocolos do papel de agente que representa.

Dessa forma, mapeou-se o caso de uso interno Goal que representa o objetivo deste papel de agente que é o de modelar o aluno (Student modeling) no atributo estereotipado goal da AgentRoleClass Student Modeling. Os objetivos em MAS-ML são representados dentro de classes Agent ou AgentRole, não possuindo um símbolo próprio.

Uma vez que este objetivo possui dois planos associados a ele, dividiu-se o objetivo em dois sub-goals, Apply_questionary (aplicar questionário) e Student_remodeling (remodelar aluno), que se referem a duas situações diferentes, o primeiro está associado ao evento de início da seção de aprendizado, quando é preciso aplicar um questionário ao aluno para produzir o primeiro modelo do aluno e o segundo refere-se à percepção de que o comportamento do aluno mudou.

Os dois Plan extension points do caso de uso interno Goal foram mapeados em crenças na AgentRoleClass Student Modeling, podendo estas ser verdadeiras ou falsas, conforme demonstra o tipo boolean.

As percepções do AgentRole_Actor Student Modeling foram mapeados na AgentRoleClass Student Modeling em deveres (duties), assim o agente que assumir este papel passará a ter o dever de verificar o início da aprendizagem e sondar o comportamento do aluno. Da mesma forma o agente terá o dever de aplicar um questionário ao aluno e modelar

seu perfil, funções que correspondem aos planos associados ao objetivo de modelar alunos do AgentRole_Actor Student Modeling.

A última informação da AgentRoleClass Student Modeling corresponde aos protocolos de comunicação entre os agentes. Dessa forma existe o protocolo Answer_questionary que é uma mensagem do tipo FIPA Request (FIPA, 2004), que possui como conteúdo o questionário a ser respondido pelo aluno, o remetente é o agente que assume o papel de Student Modeling e o destinatário é o agente humano que assume o papel de aluno. O segundo protocolo refere-se à transmissão do modelo do aluno pelo agente que interpreta o papel de agente SM ao agente que assume o papel de agente MTP, pode-se perceber que esta é uma mensagem do tipo FIPA inform.

A figura seguir apresenta um trecho de um diagrama de organização em MAS-ML produzido baseado no diagrama de casos de uso do projeto AME-A.

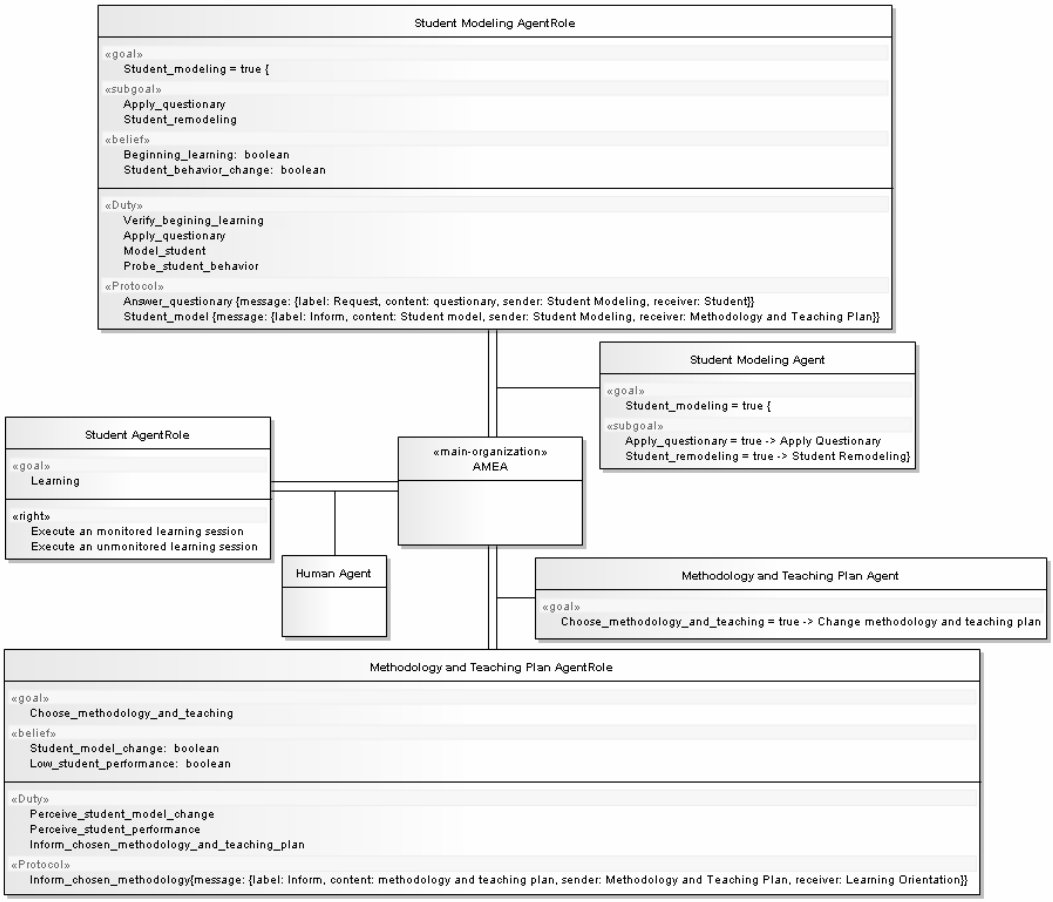


Figura 3.31: Diagrama Parcial de Organização em MAS-ML Mapeado a Partir do Diagrama de Casos de Uso do Projeto AME-A

Neste diagrama, embora não haja realmente uma organização principal no projeto AME-A, resolvemos considerar o próprio ambiente AME-A como a organização principal do projeto para aumentar a compatibilidade com a linguagem MAS-ML.

Existem, neste diagrama, três classes AgentRole e três classes Agent. As classes AgentRole correspondem aos AgentRole_Actors Student Modeling, Methodology and Teaching Plan e Student (este é um papel de agente interpretado por um agente humano, externo ao sistema). Os outros AgentRole_Actors poderiam também ter sido mapeados, mas acreditamos que apenas estes já são suficientes para ilustrar o mapeamento.

Os símbolos aqui empregados não são iguais aos propostos na linguagem MAS-ML, uma vez que não se possui uma ferramenta que os produza, dessa forma informou-se, no nome das classes, que elas representam classes Agent ou AgentRole. No caso da main-organization, o próprio estereótipo já é suficiente para identificá-la como tal.

O AgentRole_Actor Student Modeling foi mapeado na classe Agent Role Student Modeling. O caso de uso interno Goal associado a este AgentRole_Actor foi mapeado para o atributo com estereótipo goal na classe Agent Role. Devido a este objetivo possuir dois planos associados a ele, optou-se por representar dois sub-objetivos na classe AgentRole.

Esta classe AgentRole possui duas crenças que podem ser verdadeiras ou falsas (tipo boolean) e determinam se o agente acredita ou não que a sessão de aprendizado foi iniciada e se o agente acredita ou não que o comportamento do aluno mudou.

Já os deveres desta classe AgentRole são os de verificar o início da sessão de aprendizagem, aplicar questionário ao aluno quando a sessão iniciar, modelar o perfil do aluno e sondar o comportamento do aprendiz. Estes deveres correspondem às percepções do agente que interpreta o papel, modeladas por meio de nosso metamodelo, e também às tarefas que o agente deverá realizar quando suas crenças forem verdadeiras, ou seja, quando os planos forem executados.

Finalmente, os protocolos detalhados na classe AgentRole descrevem a comunicação entre o agente que assume o papel e o aluno, solicitando que este responda um questionário; e a comunicação entre o agente que interpreta o papel Student Modeling e o agente que assume o papel Methodology and Teaching Plan por meio da qual é comunicado o modelo do perfil do aluno.

O AgentRole_Actor Methodology and Teaching Plan foi mapeado na classe AgentRole de mesmo nome. O caso de uso interno que representa o objetivo deste agente foi mapeado no atributo estereotipado goal.

Esta classe de papel de agente possui duas crenças que se referem à mudança do modelo do aluno (Student_model_change), resultado da percepção do envio de um novo modelo de perfil de aluno enviado pelo agente que assumir o papel de agente SM, e baixo desempenho do aluno (Low_student_performance). Estas crenças foram mapeadas a partir dos Plan extension points do caso de uso interno Goal associado ao AgentRole_Actor Methodology and Teaching Plan.

A classe de papel de agente em questão ainda possui os deveres referentes às percepções de mudança do modelo do perfil do aluno (Perceive_student_model_change), e ao desempenho do aluno (Perceive_student_performance), além de possuir o dever de informar a metodologia escolhida ao papel de agente Learning Orientation.

Este último dever também foi mapeado no protocolo “Inform_chosen_methodology”, que representa uma mensagem do tipo FIPA Inform (FIPA, 2004), cujo conteúdo é a metodologia e o plano de ensino escolhido, o remetente é o papel de agente Methodology and Teaching Plan e o destinatário é o papel de agente Learning Orientation.

Existe ainda a classe AgentRole correspondente ao Actor Student, que possui como objetivo aprender (Learning) e o direito de executar uma sessão de aprendizado monitorada (Execute an monitored learning session) ou não monitorada (Execute an unmonitored learning session).

Todas as classes de papel de agente aqui modeladas possuem um relacionamento ownership com a organização principal, ou seja, são papéis suportados pela organização. Existe ainda uma classe Agent que representa o agente humano. Um agente humano pode tanto ser um aluno como um professor. Neste diagrama modelamos a situação em que o agente humano possui uma associação play com o papel de agent Student e a organização AMEA, o que significa que o agente humano interpreta este papel na organização.

Embora seja um pouco redundante, foram criadas as classes de agente Student Modeling e Methodology and Teaching Plan, para tornar este diagrama o mais compatível possível com a linguagem MAS-ML. Assim, haverá uma classe de agente Student Modeling associada por um relacionamento play com o papel de agente Student Modeling, o mesmo ocorrendo entre a classe de agente Methodology and Teaching Plan e a classe de papel de agente Methodology and Teaching Plan.

Observe que estas duas classes de agente possuem os mesmos objetivos que as classes de papel de agente a elas associadas, acrescidos dos planos para atingir estes objetivos. Procedeu-se desta forma porque, de acordo com Silva (2004a), diferente das classes de agente, as classes de papel de agente não representam planos associados a seus objetivos.

Dessa forma, ao se observar a classe de agente Methodology and Teaching Plan, percebe-se que, caso o objetivo "Choose_methodology_and_teaching" seja verdadeiro, ou seja, se o agente acreditar que o desejo representado pelo objetivo possa se tornar uma intenção, então será disparado o plano "Change methodology and teaching plan" para remodelar o perfil do aluno.

Assim, o caso de uso interno que representa o plano associado ao objetivo do AgentRole_Actor Methodology and Teaching Plan foi mapeado para a classe de agente Methodology and Teaching Plan e associado ao atributo estereotipado goal desta classe de agente.

Caso semelhante ocorre com a classe de agente Student Modeling, onde os dois planos, representados pelos casos de uso internos com estereótipo Plan, atribuídos ao AgentRole_Actor Student Modeling foram mapeados para a AgentRoleClas Student Modeling e associados aos atributos estereotipados sub-goals desta classe.

Cada um destes sub-goals está associado a um plano. Assim, se o objetivo Apply_questionary se tornar verdadeiro, ou seja, se tornar uma intenção (Apply_questionary = true), então será executado o plano para aplicação do questionário ao aluno. O mesmo ocorre com o segundo sub-objetivo, se este for verdadeiro será disparado o plano relativo à remodelagem do perfil do aluno.

Optou-se por não identificar as crenças das classes de agente e mantê-las apenas nas classes de papel de agente para evitar maiores redundâncias.

Como já foi dito anteriormente, a MAS-ML define o diagrama de papéis, que representa, entre outras informações, os relacionamentos entre os papéis e as classes que podem ser acessadas pelos papéis de agente. Dessa forma, foi desenvolvido um diagrama de papéis para complementar o diagrama de organização descrito anteriormente, conforme demonstra a próxima figura.

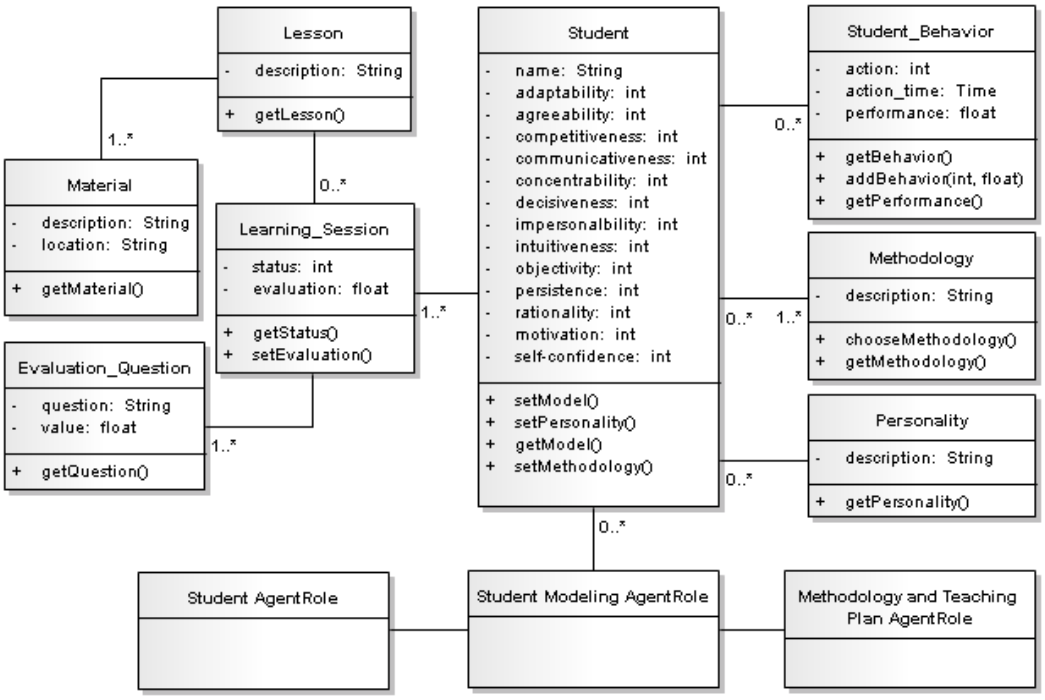


Figura 3.32: Diagrama de Papéis para o Ambiente AME-A.

Neste diagrama há três classes AgentRole que representam os papéis de agente, Aluno (Student), Modela-Aprendiz (Student Modeling) e Metodologia e Plano de Ensino (Methodology and Teaching Plan). Todas as outras classes são classes normais. As associações entre a classe AgentRole Student Modeling e as outras classes AgentRole denotam os relacionamentos entre elas, uma vez que há troca de mensagens entre as mesmas.

A linguagem MAS-ML estende diagramas dinâmicos, como o diagrama de seqüência para modelar as interações entre os agentes. Assim, utilizamos diagramas de seqüência para modelar as etapas percorridas pelo agente interpretando o papel Student Modeling para determinar se o objetivo de modelar o perfil do aluno pode se tornar uma intenção, além das etapas percorridas pelo mesmo agente para executar o plano “Apply questionary”, como pode ser visto nas duas figuras a seguir.

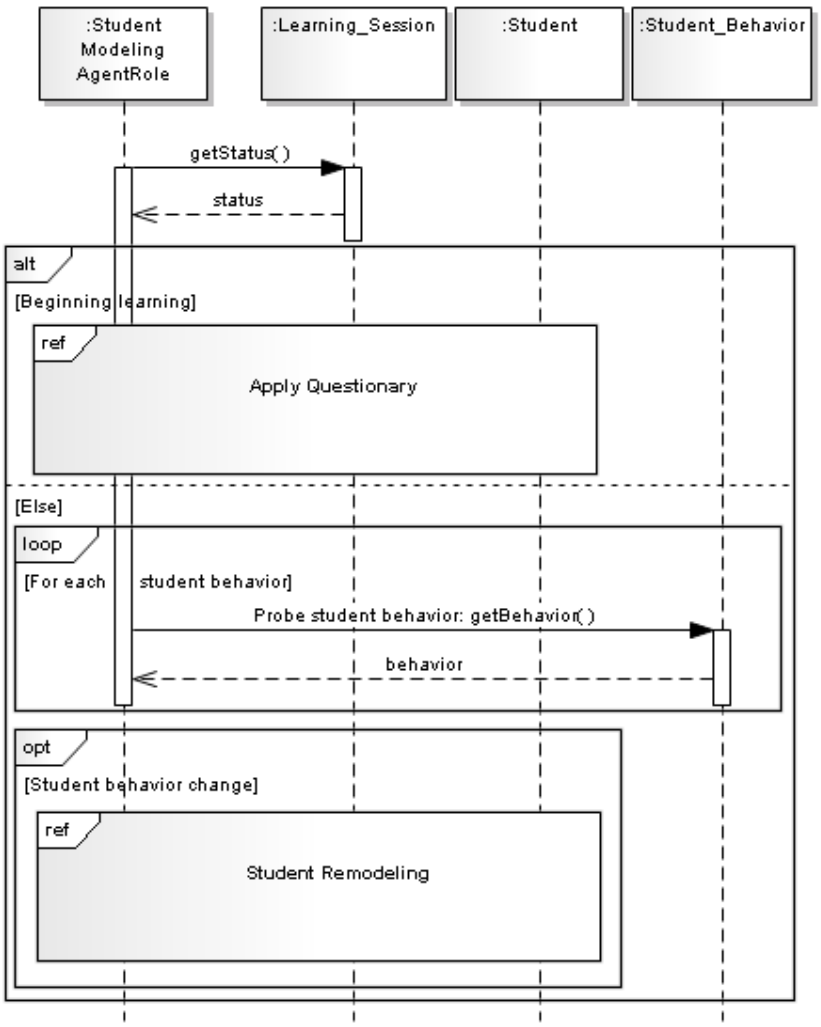


Figura 3.33: Diagrama de Seqüência Referente ao Objetivo de Modelar o Perfil do Aluno pelo Papel de Agent Student Modeling

Nesta figura o papel de agente Student Modeling primeiramente percebe o estado da sessão de aprendizagem por meio do disparo do método getStatus para determinar se a sessão está em seu início.

Em seguida, por meio de um fragmento combinado do tipo Alt realizou-se um teste, se for o início da sessão de aprendizagem então é disparado o plano Apply Questionary, aqui apenas referenciado, uma vez que está detalhado no diagrama sequinte.

Caso não seja o início da sessão de aprendizagem, então, por meio de outro fragmento combinado do tipo loop são recuperados os comportamentos do aluno. Após isso é feito um teste, representado por um fragmento combinado do tipo opt, para determinar se o comportamento do aluno mudou. Em caso positivo será disparado o plano para remodelar o perfil do aluno.

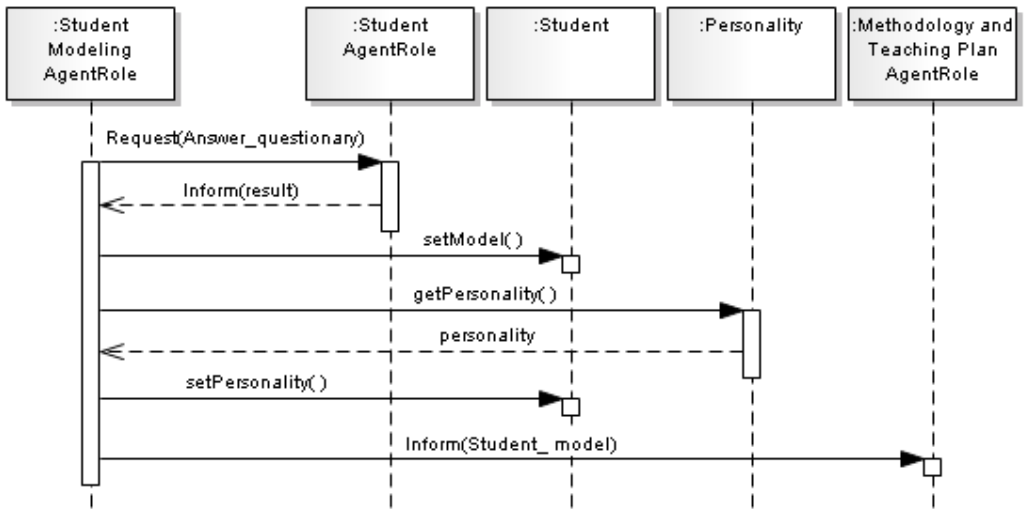


Figura 3.34: Diagrama de Seqüência Referente ao Plano “Apply questionary”

Neste diagrama, o papel de agente Student Modeling (na verdade o agente que o interpretar) solicita ao papel de agente Student que responda ao questionário, por meio de um FIPA Request (FIPA, 2004). Em resposta o papel de agente Student informa o resultado do preenchimento do questionário por meio de uma FIPA Inform. Com base nessas respostas o papel de agente Student Modeling define o modelo e a personalidade do aluno e, informa o modelo selecionado para o papel de agente Methodology and Teaching Plan por meio de uma mensagem FIPA Inform.

3.11 Mapeamento de Conceitos do Metamodelo Proposto para a Linguagem AML

Nesta seção propõe-se uma forma de mapeamento de conceitos do metamodelo para a linguagem AML, o que, da mesma forma que na seção anterior com relação à linguagem MAS-ML, poderia vir a ser útil para complementar tanto o metamodelo como a própria linguagem, caso se queira utilizar ambas. Aqui novamente tomamos como exemplo o ambiente AME-A modelado através de nosso metamodelo.

Concluiu-se que a construção da AML mais semelhante ao conceito AgentRole_Actor, proposto no metamodelo, é o tipo de papel de entidade (entity role type) que representa um papel que pode ser interpretado por um agente. Dessa forma, como mostra a figura a seguir, mapeou-se os AgentRole_Actors utilizados na modelagem do ambiente AME-A – tanto os reativos como os cognitivos – e até mesmo os atores normais (uma vez que um ator representa um tipo de papel executado por uma entidade que interage com o sistema) em tipos de papel de entidade.

É importante observar que criou-se também tipos de agentes, embora o metamodelo proposto não contemple exatamente agentes, mas sim os papéis que eles podem interpretar, a AML faz a ligação entre tipos de agente e tipos de papel de entidade por meio de associações play.

Sendo assim, criou-se um tipo agente Person (pessoa) para representar os agentes externos humanos que podem interpretar o papel de professor ou aluno (é importante observar a restrição “xor” que indica que somente poderá interpretar um dos dois papéis). Também foi criado o tipo agente Reactive Agent que representa um agente reativo que interpretará os tipos de papel de entidade Teacher_Tools ou Unsupervised_Learning.

Finalmente, foram criados tipos de agente para os tipos de agente cognitvos propostos por Vicari (2007), assim um agente Users and Agents Modelling (UAM) interpretará o tipo de papel de entidade Student Modelling, um agent Social Mediated Interactions (SMI) interpretará o tipo de papel de entidade Methodology and Teaching Plan e um agente Problem Solving (PS) interpretará os tipos de papel de entidade Learning Orientation, Learning Analysis e Knowledge Application Orienting.

É importante notar que, ao invés de utilizar os estereótipos gráficos da AML, utilizamos estereótipos de texto equivalentes.

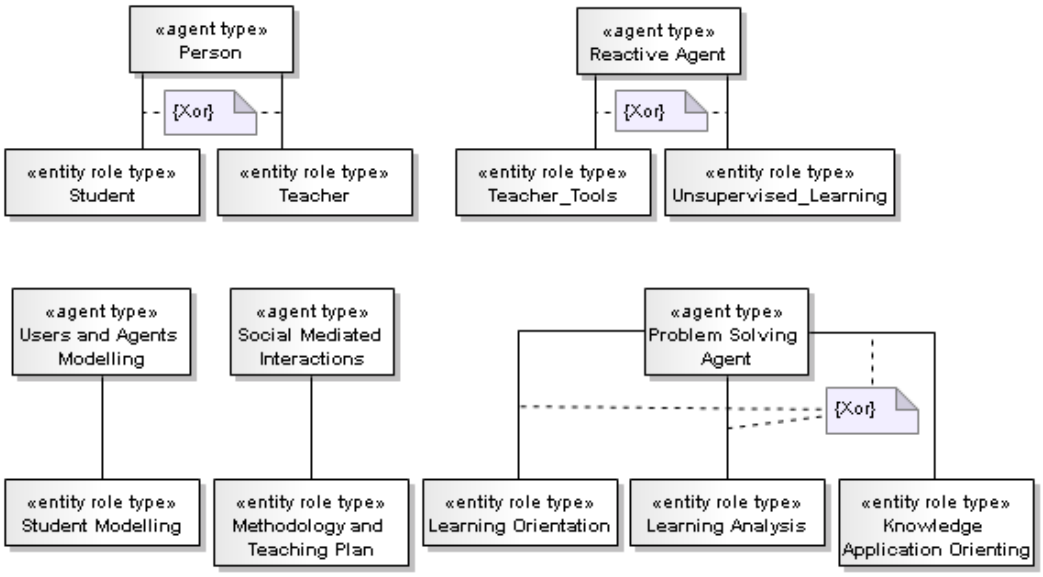


Figura 3.35: Mapeamento de AgentRole_Actors em Tipos de Papel de Entidade AML

A figura a seguir produz um mapeamento das percepções, ações e objetivos do AgentRole_Actor Methodology and Teaching Plan, identificadas por meio do metamodelo proposto, para a linguagem AML, onde os casos de uso internos “Perception” foram mapeados como observações (duas primeiras portas superiores do tipo de papel de entidade Methodology and Teaching Plan), o caso de uso interno “Action” foi mapeado como uma interação de efeito (porta inferior) e o objetivo “Change Methodology and Teaching Plan” foi mapeado em um fragmento de comportamento. Novamente adotaram-se estereótipos de texto ao invés dos estereótipos gráficos da AML.

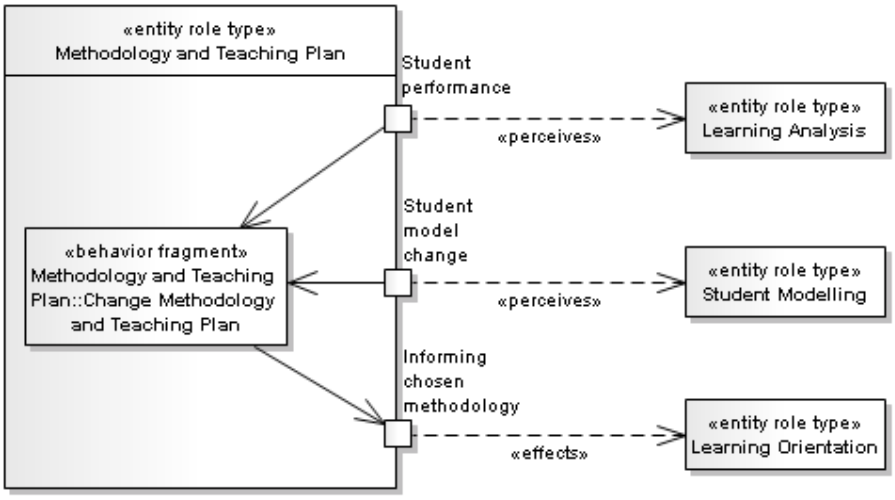


Figura 3.36: Observações e Interações de Efeito do Tipo de Papel de Entidade Methodology and Teaching Plan

Nesta figura, o tipo de papel de entidade Methodology and Teaching Plan percebe o desempenho do aluno por meio do tipo de papel de entidade Learning Analysis; percebe ainda as possíveis mudanças no modelo do aluno por meio do tipo de papel de entidade Student Modelling; e produz uma interação de efeito para o tipo de papel de entidade Learning Orientation, informando a metodologia escolhida. Nesta situação, o objetivo "Change Methodology and Teaching Plan" foi mapeado em um fragmento de comportamento dentro do contexto deste diagrama.

A figura a seguir demonstra um modelo mental onde são identificados em AML os objetivos de cada tipo de papel de entidade e as contribuições de crença associadas a cada objetivo.

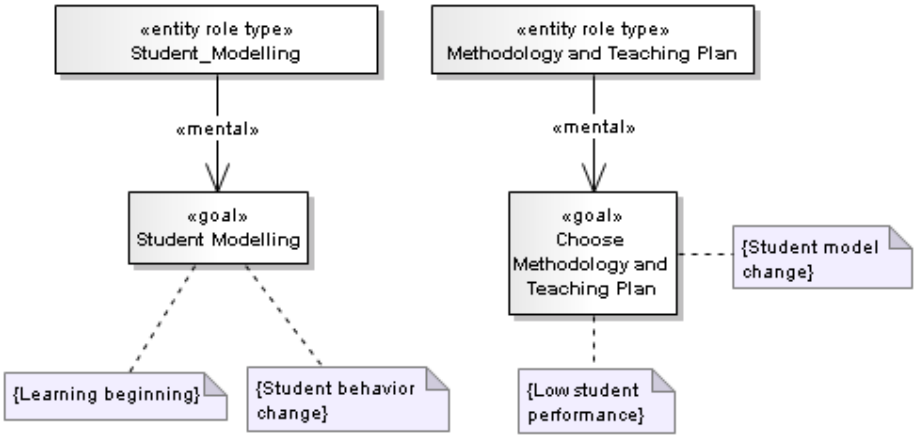


Figura 3.37: Modelo Mental

Nesta figura o objetivo "Student Modelling" precisa que a contribuição de crença "Learning beginning" ou a contribuição de crença "Student behavior change" sejam verdadeiras para que ele se torne uma intenção. Do mesmo modo o objetivo "Choose Methodology and Teaching Plan" necessita que a contribuição de crença "Student model change" ou a contribuição de crença "Low student performance" sejam verdadeiras para que este objetivo se torne uma intenção. Novamente foram utilizados estereótipos de texto neste exemplo.

3.12 Documentação Textual dos Casos de Uso Internos

Diversos autores como Bezerra (2002), sugerem formas de documentação textuais para casos de uso. Assim, com o objetivo de sugerir uma documentação alternativo/complementar da documentação realizada por meio dos diagramas comportamentais da linguagem MAS-ML, propõe-se nesta seção como documentar os comportamentos dos casos de uso internos por meio de documentos textuais. Nesta seção as tabelas apresentadas procuram documentar os casos de uso internos do estudo de caso referente ao ambiente AME-A.

Tabela 3.1: Documentação do Caso de Uso Interno Student Modeling

Nome do Caso de Uso Interno	Student Modeling
Estereótipo	Goal
AgentRole_Actor Principal	UAM_AgentRole Student Modeling
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	Verify learning beginning, Probe student behavior
Ações Associadas	
Planos Associados	Apply questionary, Student remodeling
Plan Extension Points	Learning beginning, Student behavior change
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Student Modeling para verificar se o objetivo de modelar o perfil do aprendiz pode se tornar uma intenção.
Pré-Condições	Para que o objetivo se torne uma intenção, o agente que assumir o papel de Student Modeling deve perceber o início de uma nova sessão de aprendizagem por um aluno ou perceber a mudança de comportamento do aluno.
Pós-Condições	No momento que o objetivo se torne uma intenção deverá ser disparado o plano Apply Questionary quando a primeira percepção for verdadeira ou o plano Student Remodeling quando a segunda percepção for verdadeira.
Fluxo Principal	
1. Executar o processo representado pelo objetivo Student Modeling para determinar se este pode tornar-se uma intenção	
2. Executar o caso de uso interno referente à percepção para determinar se uma sessão de aprendizagem foi iniciada	
3. Em caso negativo executar o caso de uso interno referente à percepção para sondar o comportamento do aluno	
Fluxo Alternativo I – Início de Uma Nova Sessão de Aprendizagem	
Executar o caso de uso interno referente ao plano Apply Questionary	
Fluxo Alternativo II – Mudança no Comportamento do Aluno	
Executar o caso de uso interno referente ao plano Student Remodeling	

Ao observar estas tabelas percebe-se que suas informações contém o nome do Caso de Uso Interno, seu estereótipo, o principal AgentRole_Actor que o utiliza, os possíveis AgentRole_Actors secundários, outros atores que possam estar envolvidos no processo, as possíveis percepções associadas ao caso de uso interno, as possíveis ações associadas, os possíveis planos associados, pontos de extensão de plano (se existirem), um resumo da função do caso de uso interno. As possíveis pré e pós-condições que devem ser satisfeitas antes e depois da execução do caso de uso interno, o fluxo principal e os possíveis fluxos alternativos de cada caso de uso interno.

Tabela 3.2: Documentação do Caso de Uso Interno Apply Questionary

Nome do Caso de Uso Interno	Apply Questionary
Estereótipo	Plan
AgentRole_Actor Principal	UAM_AgentRole Student Modeling
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	
Ações Associadas	Ask the student to answer the questionary, Inform student profile to Methodology and Teaching Plan Agent
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Student Modeling para executar o plano de aplicar um questionário ao aprendiz e com base nas respostas determinar o perfil do aluno.
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Student Modeling perceba o início de uma nova sessão de aprendizagem por um aluno.
Pós-Condições	A execução deste plano implica na ação de informar o perfil modelado ao agente que assumir o papel Methodology and Teaching Plan.
Fluxo Principal	
1. Executar o caso de uso interno Action “Ask the student to answer the questionary” para apresentar um questionário ao aluno solicitando que ele o preencha	
2. Receber o questionário preenchido pelo aluno	
3. Analisar o questionário e modelar o perfil do aluno	
4. Executar o caso de uso interno Action “Inform student model to Methodology and Teaching Plan Agent” para enviar o perfil do aluno ao agente que assumir o papel de Methodology and Teaching Plan	

Tabela 3.3: Documentação do Caso de Uso Interno Student Remodeling

Nome do Caso de Uso Interno	Student remodeling
Estereótipo	Plan
AgentRole_Actor Principal	UAM_AgentRole Student Modeling
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	
Ações Associadas	Inform student profile to Methodology and Teaching Plan Agent
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Student Modeling para executar o plano de remodelar o perfil do aprendiz.
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Student Modeling perceba que o comportamento do aluno mudou.
Pós-Condições	A execução deste plano implica na ação de informar o perfil modelado ao agente que assumir o papel Methodology and Teaching Plan.
Fluxo Principal	
1. Analisar as variações de comportamento do aluno	
2. Definir outro perfil para o aluno	
3. Executar o caso de uso interno Action para enviar o perfil do aluno ao agente que assumir o papel de Methodology and Teaching Plan	

Tabela 3.4: Documentação do Caso de Uso Interno Choose Methodology and Teaching Plan

Nome do Caso de Uso Interno	Choose methodology and teaching plan
Estereótipo	Goal
AgentRole_Actor Principal	SMI_AgentRole Methodology and Teaching Plan
AgentRole_Actors Secundários	
Outros Atores Envolvidos	
Percepções Associadas	Student model change, Student performance
Ações Associadas	
Planos Associados	Change methodology and teaching plan
Plan Extension Points	Student model change, Low student performance
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Methodology and Teaching Plan para escolher uma metodologia e um plano de ensino adequado ao perfil do aluno
Pré-Condições	Para que o objetivo se torne uma intenção,

	o agente que assumir o papel de Methodology and Teaching Plan deve perceber a mudança no perfil do aluno ou perceber o baixo desempenho do aluno
Pós-Condições	No momento que o objetivo se torne uma intenção deverá ser disparado o plano Change methodology and teaching plan
Fluxo Principal	
1. Executar o processo representado pelo objetivo Choose methodology and teaching plan para determinar se este pode tornar-se uma intenção	
2. Executar o caso de uso interno referente à percepção para determinar se o perfil do aprendiz mudou	
3. Em caso negativo executar o caso de uso interno referente à percepção para determinar se o desempenho do aluno foi recebido.	
Fluxo Alternativo I – Perfil do Aluno Mudou	
Executar o caso de uso interno referente ao plano Change methodology and teaching plan	
Fluxo Alternativo II – Mudança no Comportamento do Aluno	
Executar o caso de uso interno referente ao plano Change methodology and teaching plan	

Tabela 3.5: Documentação do Caso de Uso Interno Change Methodology and Teaching Plan

Nome do Caso de Uso Interno	Change methdology and teaching plan
Estereótipo	Plan
AgentRole_Actor Principal	SMI_AgentRole Methodology and Teaching Plan
AgentRole_Actors Secundários	
Outros Atores Envolvidos	
Percepções Associadas	
Ações Associadas	Inform chosen methdology and teaching plan to Learning Orientation agent
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Methodology and Teaching Plan para executar o plano Change methodology and teaching plan
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Methodology and Teaching Plan perceba a mudança do perfil do aprendiz ou que perceba que o desempenho do aprendiz está abaixo do esperado
Pós-Condições	A execução deste plano implica na ação de informar a metodologia e o plano de ensino escolhidos para o agente que assumir o papel de Learning Orientation

Fluxo Alternativo I – Mudança do Perfil do Aluno
1. Analisar o perfil enviado pelo agente que assumir o papel Student Modeling
2. Selecionar a metodologia e o plano de ensino mais adequados ao perfil enviado
3. Executar o caso de uso interno Action para enviar a metodologia e o plano de ensino ao agente que assumir o papel de Learning Orientation
Fluxo Alternativo II – Baixo Desempenho do Aluno
1. Selecionar uma nova metodologia e plano de ensino mais adequados ao perfil enviado
2. Executar o caso de uso interno Action para enviar a metodologia e o plano de ensino ao agente que assumir o papel de Learning Orientation

Tabela 3.6: Documentação do Caso de Uso Interno Present Material for Learning

Nome do Caso de Uso Interno	Present material for learning
Estereótipo	Goal
AgentRole_Actor Principal	PS_AgentRole Learning Orientation
AgentRole_Actors Secundários	
Outros Atores Envolvidos	
Percepções Associadas	Chosen methodology and teaching plan
Ações Associadas	
Planos Associados	Select material for learning
Plan Extension Points	Chosen methodology and teaching plan
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Learning Orientation para apresentar material de ensino de acordo com a metodologia e plano de ensino escolhida.
Pré-Condições	Para que o objetivo se torne uma intenção, o agente que assumir o papel de Learning Orientation deve perceber que uma metodologia e plano de ensino foi escolhida para o aluno
Pós-Condições	No momento que o objetivo se torne uma intenção deverá ser disparado o plano Select material for learning
Fluxo Principal	
1. Executar o processo representado pelo objetivo Present material for learning para determinar se este pode tornar-se uma intenção	
2. Executar o caso de uso interno referente à percepção para determinar se uma metodologia e plano de ensino foram escolhidos	
3. Em caso positivo executar o plano Select material for learning	

Tabela 3.7: Documentação do Caso de Uso Interno Select Material for Learning

Nome do Caso de Uso Interno	Select material for learning
Estereótipo	Plan
AgentRole_Actor Principal	PS_AgentRole Learning Orientation
AgentRole_Actors Secundários	
Outros Atores Envolvidos	

Percepções Associadas	
Ações Associadas	
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Learning Orientation para executar o plano Select material for learning
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Learning Orientation perceba que uma metodologia e um plano de ensino foram escolhidos
Pós-Condições	
Fluxo Principal	
1. Selecionar conteúdo de acordo com a metodologia e o plano de ensino escolhidos	
2. Apresentar o conteúdo ao aprendiz	

Tabela 3.8: Documentação do Caso de Uso Interno Verify Student Knowledge

Nome do Caso de Uso Interno	Verify student knowledge
Estereótipo	Goal
AgentRole_Actor Principal	PS_AgentRole Learning Analysis
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	Student performance
Ações Associadas	Informing Methodology and Teaching Plan AgentRole
Planos Associados	Boosting student
Plan Extension Points	Low performance
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Learning Analysis para determinar se o objetivo de verificar o conhecimento do aprendiz pode se tornar uma intenção.
Pré-Condições	Para que o objetivo se torne uma intenção, o agente que assumir o papel de Learning Analysis deve perceber o baixo desempenho do aluno.
Pós-Condições	No momento que o objetivo se torne uma intenção deverá ser disparado o plano Boosting student.
Fluxo Principal	
1. Executar o caso de uso interno Perception Student performance para recuperar o desempenho do aluno	
2. Executar o caso de uso interno Action para enviar o desempenho do aluno ao agente que assumir o papel de Methodology and Teaching Plan	
3. Verificar o desempenho do aluno	
4. Se o desempenho for baixo, executar o caso de uso interno Plan Boosting student	

Tabela 3.9: Documentação do Caso de Uso Interno Boosting Student

Nome do Caso de Uso Interno	Boosting student
Estereótipo	Plan
AgentRole_Actor Principal	PS_AgentRole Learning Analysis
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	
Ações Associadas	Send motivating message
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Learning Analysis para executar o plano para incentivar o aluno.
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Learning Analysis perceba que o desempenho do aluno seja baixo.
Pós-Condições	A execução deste plano implica na ação de enviar mensagens motivacionais ao aluno.
Fluxo Principal	
1. Selecionar mensagens motivacionais para incentivar o aluno	
2. Executar o caso de uso interno Action Send motivating message para enviar as mensagens motivacionais para o aluno	

Tabela 3.10: Documentação do Caso de Uso Interno Evaluation Student

Nome do Caso de Uso Interno	Evaluation student
Estereótipo	Goal
AgentRole_Actor Principal	PS_AgentRole Knowledge Application Orienting
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	End session learning
Ações Associadas	
Planos Associados	Apply evaluation
Plan Extension Points	Learning session end
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Knowledge Application Orienting para determinar se o objetivo de avaliar o aluno pode se tornar uma intenção.
Pré-Condições	Para que o objetivo se torne uma intenção, o agente que assumir o papel de Knowledge Application Orienting deve perceber o fim da sessão de aprendizagem.
Pós-Condições	No momento que o objetivo se torne uma

	intenção deverá ser disparado o plano Apply evaluation.
Fluxo Principal	
1. Executar o caso de uso interno Perception End session learning para determinar se a sessão de aprendizagem chegou ao fim	
2. Em caso positivo executar o plano Apply evaluation	

Tabela 3.11: Documentação do Caso de Uso Interno Apply Evaluation

Nome do Caso de Uso Interno	Apply evaluation
Estereótipo	Plan
AgentRole_Actor Principal	PS_AgentRole Knowledge Application Orienting
AgentRole_Actors Secundários	
Outros Atores Envolvidos	Aluno
Percepções Associadas	
Ações Associadas	
Planos Associados	
Plan Extension Points	
Resumo	Este caso de uso interno descreve as etapas percorridas pelo agente que assumir o papel Knowledge Application Orienting para executar o plano Apply evaluation.
Pré-Condições	Para que o plano seja executado é necessário que o agente que assumir o papel de Knowledge Application Orienting perceba que a sessão de aprendizagem terminou.
Pós-Condições	
Fluxo Principal	
1. Selecionar a avaliação mais adequada	
2. Apresentar a avaliação ao aluno	

3.13 Como Aplicar o Metamodelo

Embora esta tese proponha uma estensão para a notação UML, de maneira que esta permita a modelagem de requisitos funcionais específicos para o projeto de sistemas multi-agentes, e não uma metodologia para aplicá-la, nesta seção sugerem-se alguns passos para utilizar o metamodelo proposto.

Primeiramente, deve-se identificar os papéis de agentes que irão compor o sistema e modelá-los por meio de AgentRole_Actors. Em seguida deve-se identificar os objetivos relacionados a cada papel, por meio de casos de uso internos com o estereótipo Goal e associar estes objetivos aos ditos papéis. A partir disso identificam-se as possíveis ações que podem estar associadas a um objetivo e as possíveis percepções que podem auxiliar a determinar se um objetivo pode ou não se tornar uma intenção. As ações e percepções associadas a um determinado objetivo devem ser representadas como casos de uso internos contendo os respectivamente os estereótipos Action e Perception e relacionadas ao objetivo por meio de associações do tipo “include ap”.

Em seguida tenta-se identificar os planos que devem ser executados quando um objetivo se tornar uma intenção, por meio de casos de uso internos contendo o estereótipo Plan, e

relacioná-los aos objetivos por meio de associações do tipo “plan extend”, o que destacaria que uma condição deve ser satisfeita para que os planos sejam disparados. Deve-se ainda determinar os pontos no comportamento de cada objetivo a partir dos quais o comportamento do objetivo poderão ser estendidos pelos planos, por meio de Plan extension points.

Finalmente, deve-se determinar se existe alguma ação que afete o ambiente executada por algum plano. Neste caso deve-se identificar cada ação por meio de casos de usos internos com o estereótipo “Action” e associá-las aos planos por meio de associações do tipo “include ap”.

3.14 Validação do Metamodelo Proposto

3.14.1 Comparação dos Diagramas Produzidos por Meio do Metamodelo com Diagramas Produzidos por Meio da UML

Com relação à validação, foi feita uma comparação entre os modelos criados por meio deste metamodelo com modelos criados a partir do diagrama de casos de uso original da UML, tentando com isso demonstrar que o novo metamodelo proposto permite a modelagem de conceitos não suportados pelo diagrama de casos de uso padrão.

Dentro desta idéia, produziu-se um diagrama de casos de uso para o sistema MCOE utilizando somente a UML padrão e este foi comparado com o diagrama produzido por meio do metamodelo, como pode ser visto na figura a seguir.

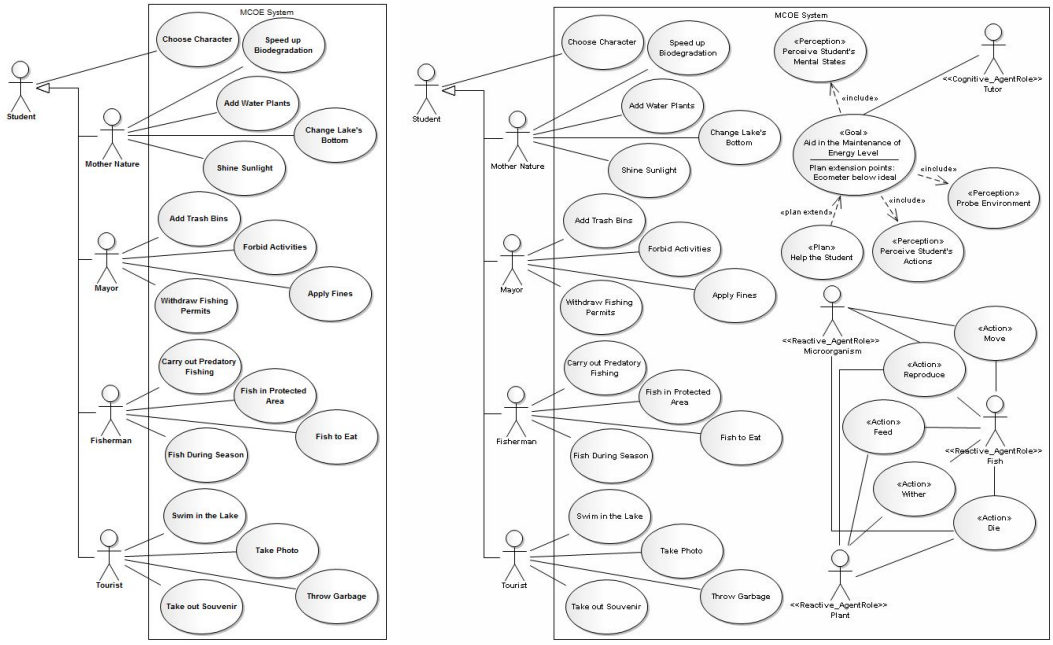


Figura 3.38: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao sistema MCOE.

Ao se comparar as duas figuras, percebe-se que somente os casos de uso referentes às funções permitidas aos personagens interpretados pelo aluno foram modelados no diagrama da esquerda. Procedeu-se dessa maneira porque um caso de uso modela um requisito funcional externo, ou seja, uma função que pode ser solicitada por um ator externo. Assim, não foi considerado possível modelar no diagrama da esquerda as mesmas funcionalidades modeladas no diagrama da direita referentes aos casos de uso associados ao papel de agente cognitivo Tutor e aos papéis de agente reativos Fish, Microorganism e Plant.

Primeiramente não foi nem mesmo possível representar os agentes no modelo da esquerda, uma vez que estes estão inseridos no ambiente e atores só podem ser usados para

representar atores externos ao sistema. Em segundo lugar não foi possível representar as funções solicitadas por estes atores, uma vez que se tratam de funcionalidades internas, algo que os usuários externos não têm acesso. Dessa forma concluiu-se não ser possível representar papéis de agente e suas funções em um sistema por meio do diagrama de casos de uso padrão, todavia é possível fazê-lo por meio do metamodelo proposto.

A figura seguinte faz comparação semelhante entre diagramas de casos de uso criados por meio da UML padrão e por meio do metamodelo proposto, referentes ao projeto AME-A.

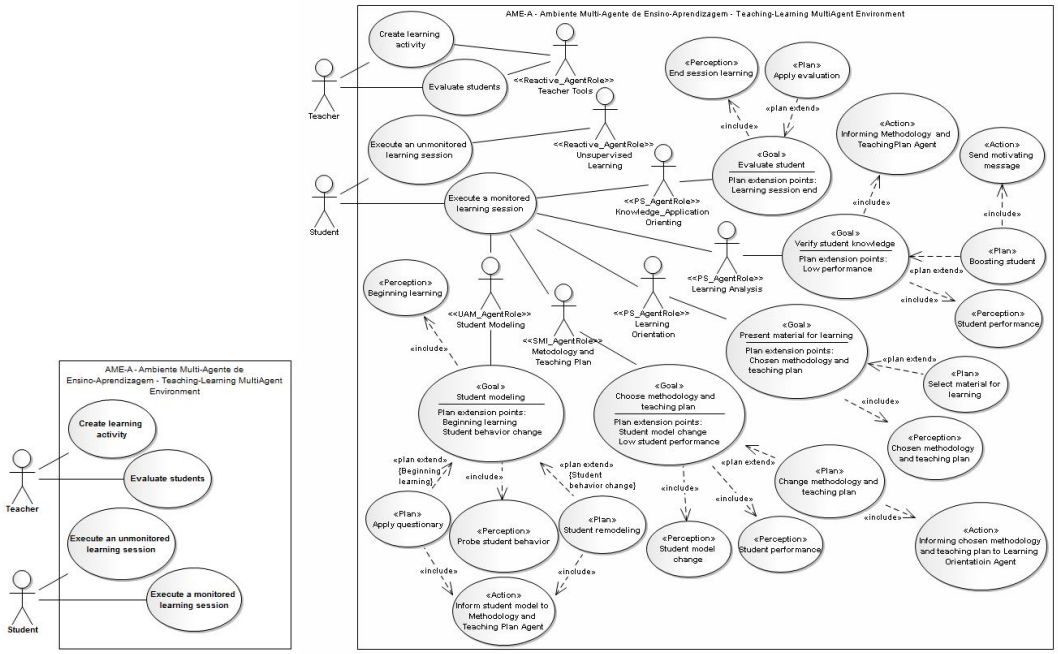


Figura 3.39: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao projeto AME-A.

Nesta figura, pelos mesmos motivos anteriores, apenas modelou-se no diagrama da esquerda, as funcionalidades que podem ser solicitadas pelo aluno ou pelo professor, uma vez que neste diagrama utilizou-se a UML padrão. Os papéis de agente não foram representados por estes serem internos ao sistema e também não foram representados os objetivos, planos, percepções ou ações destes papéis, por estas serem funcionalidades internas, que os usuários externos não teriam acesso. Além disso, a UML padrão simplesmente não possui mecanismos para representar objetivos, planos, percepções ou ações em um diagrama de casos de uso.

Alternativamente, utilizando a UML padrão, poderia-se tentar fazer com que o caso uso “Execute monitored learning session” englobasse os passos dos casos de uso “Student modeling”, “Choose methodology and teaching plan”, “Present material for learning”, “Verify student knowledge” e “Evaluate student”. O que se poderia fazer, nessa situação específica, seria associar os casos de uso que representam os planos ao caso de uso “Execute a monitored learning session” por meio de associações de extensão, estabelecendo as condições para que estes fossem ser executados, conforme demonstra a figura seguinte.

Com esta abordagem tentou-se utilizar a UML padrão para atingir o mesmo objetivo conseguido com o uso do metamodelo proposto. Como pode-se perceber continuou-se a suprimir a representação de agentes, uma vez que estes são internos ao sistema e um ator representa entidades externas ao software.

Contudo, estas funcionalidades não foram concebidas como serviços a serem executados da forma “tradicional” pelo software e sim para serem divididas entre agentes cognitivos, ou seja, são funções separadas da funcionalidade “Execute a monitored learning session”, que

representa um serviço oferecido ao aluno, onde ele escolherá, por exemplo, o tema da sessão de aprendizado, mas a partir daí haverá uma interação com os agentes cognitivos, onde estes assumirão uma série de tarefas para auxiliar o aluno, necessitando para isso perceber eventos, tomar decisões e realizar ações a respeito de que atitudes deverão executar. Não é possível explicitar estes tipos de funcionalidade utilizando a UML padrão.

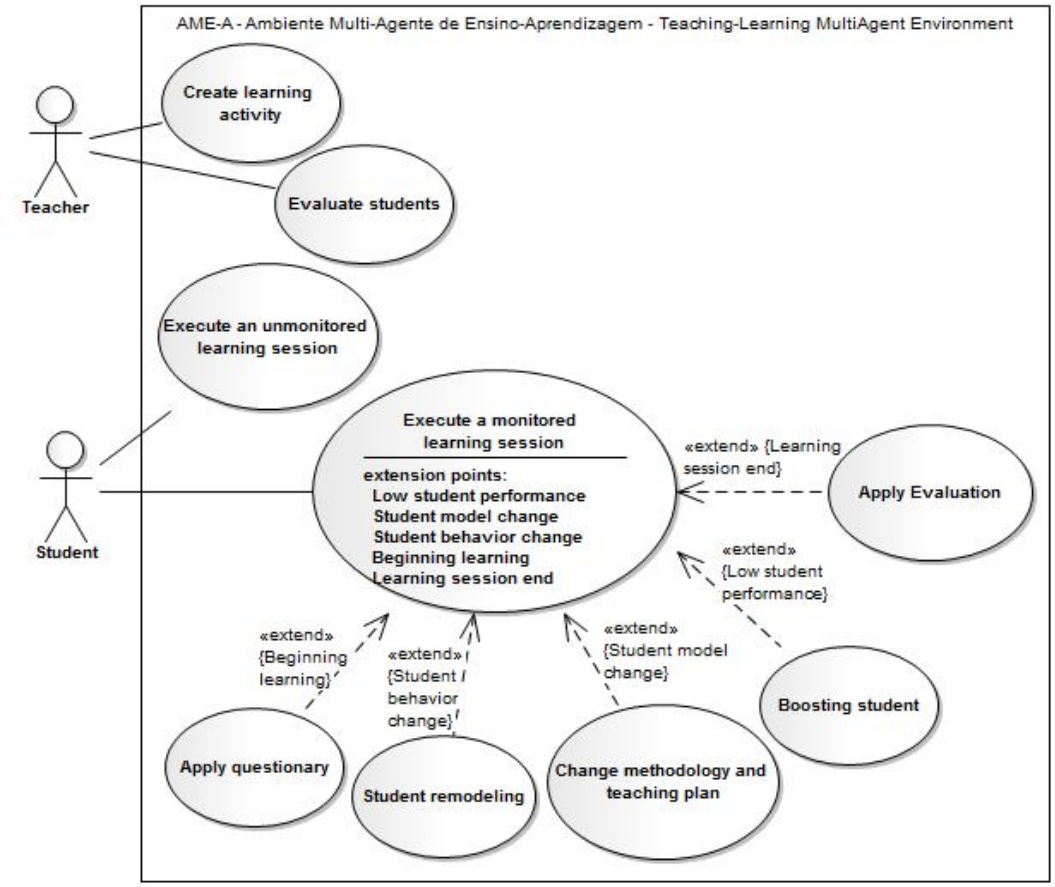


Figura 3.40: Alternativa para Modelagem AME-A por meio da UML Padrão Utilizando Associações de Extensão.

No entanto, tentou-se associar ao caso de uso “Execute a monitored learning session” os casos de uso originalmente usados para representar os planos que deveriam ser executados pelos agentes. Dessa maneira, os pontos de extensão de plano dos casos de uso internos Goal foram atribuídos ao caso de uso “Execute a monitored learning session”, tentou-se dessa maneira demonstrar sob quais condições cada “plano” seria executado.

É importante observar, porém, que na realidade os casos de uso associados por meio de extensões não são funcionalidades que o ator aluno tenha acesso, estas funcionalidades deveriam ser executadas por agentes de acordo com seus objetivos. Deve-se destacar ainda que não é possível representar os objetivos de um agente nem os planos associados a cada objetivo usando a UML padrão. Da mesma forma a UML padrão não suporta a representação de percepções e ações e tampouco é capaz de representar papéis de agentes internos ao sistema como atores.

A figura a seguir faz uma nova comparação entre diagramas de casos de uso criados por meio da UML padrão e por meio do metamodelo proposto, referentes ao projeto do sistema APA.

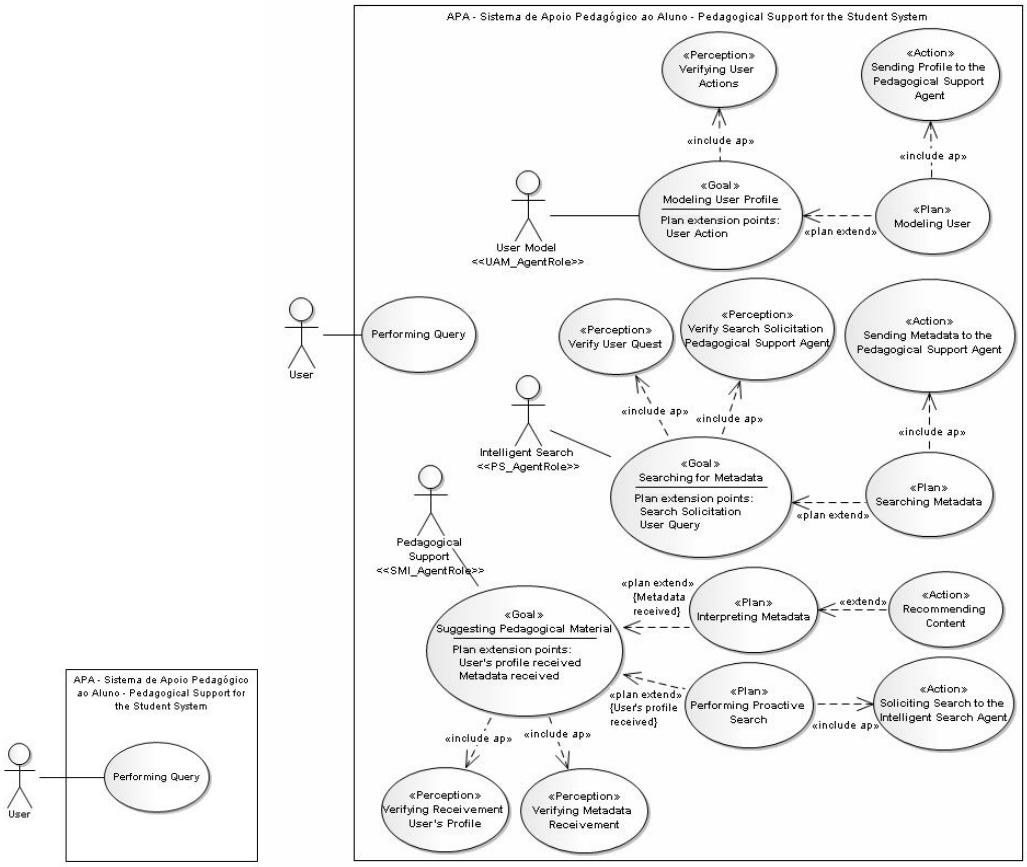


Figura 3.41: Comparação entre os diagramas de casos de uso produzidos usando a UML padrão (esquerda) e utilizando o metamodelo proposto (direita), referentes ao projeto do sistema APA.

Na figura da esquerda somente foi modelado o ator externo, ou seja, o usuário humano que interage com o sistema e a funcionalidade que pode ser solicitada por ele. Como nos exemplos anteriores, não foram representados os papéis de agente porque eles são internos ao sistema e as funcionalidades internas não foram modeladas porque casos de uso normais modelam somente requisitos externos, ou seja, funcionalidades que podem ser solicitadas por atores externos.

Dessa forma nós não consideramos possível modelar os objetivos, planos, percepções ou ações destes papéis, porque, como já havia sido explicado anteriormente, estas são funcionalidades internas para as quais os usuários externos não têm acesso.

Como no exemplo anterior, utilizando a UML padrão, tentou-se representar o caso de uso interno “Performing Query” como um caso de uso normal englobando os passos dos casos de usos internos “Modeling User Profile”, “Searching for Metadata” e “Suggesting Pedagogical Material”, como pode ser visto na próxima figura.

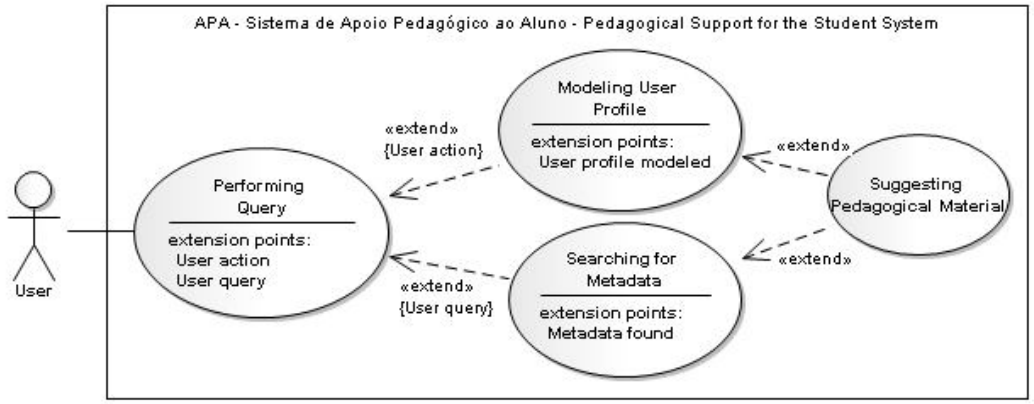


Figura 3.42: Alternativa para Modelagem do sistema APA por meio da UML Padrão Utilizando Associações de Extensão.

Nesta figura tentou-se associar os objetivos associados aos papéis de agente ao caso de uso "Performing Query" por meio de associações de extensão. Assim considerou-se o evento de uma ação do usuário como um ponto de extensão, que causaria a execução do caso de uso "Modeling User Profile". Considerou-se o evento de uma consulta do usuário como um segundo ponto de extensão que iria levar a execução do caso de uso "Searching for Metadata". Já o caso de uso "Suggesting Pedagogical Material", seria executado quando o perfil do usuário fosse modelado ou sempre que metadados fossem encontrados, como demonstram os pontos de extensão dos casos de uso “Modeling User Profile” e “Searching for Metadata”.

Os papéis de agente continuaram a não ser representados, posto que eles são internos ao sistema e um ator representa entidades externas. Contudo, na realidade os casos de uso aqui associados por meio de extensões não são funcionalidades que o ator usuário tenha acesso. Estas funcionalidades deveriam ser executadas por agentes interpretando papéis de acordo com os objetivos associados aos papéis em questão.

Entretanto, da mesma forma que no exemplo anterior sobre o sistema AME-A, estas funcionalidades não foram concebidas como serviços a ser executados na forma “tradicional” pelo software, mas para serem divididas entre agentes cognitivos, ou seja, são funções distintas da funcionalidade "Performing Query" que representa um serviço oferecido ao usuário, através do qual ele submeterá consultas, mas a partir daí haverá uma interação com agentes cognitivos, em que eles tentarão auxiliar o usuário. Para isto, eles precisarão perceber eventos, tomar decisões e realizar ações com respeito a quais atitudes deveriam ser tomadas. Não é possível tornar explícitas todos esses tipos de funcionalidades utilizando o diagrama de casos de uso da UML padrão.

3.14.2 Validação por Justificativa

Nesta seção será apresentada outra abordagem para validar o metamodelo proposto, onde se tentará validar os conceitos propostos por meio da justificativa da necessidade de representá-los para a modelagem de requisitos em sistemas multi-agentes e por estes ditos conceitos não serem suportados pela linguagem UML.

3.14.2.1 Metaclasse AgentRole_Actor e Metaclasses Derivadas a Partir desta Metaclasse

Agentes podem assumir diversos papéis em um sistema, da mesma forma que um usuário humano pode assumir mais de um papel ao interagir com um software. Um usuário humano pode, por exemplo, assumir o papel de gerente em um determinado momento com permissões para utilizar muitos recursos do sistema ou assumir o papel de um usuário com um nível de permissões menor, podendo utilizar recursos mais restritos do software. Isso na UML propriamente dita é representado pelo conceito de Actor, que permite representar os papéis suportados pelo software. No entanto, os atores da UML padrão representam papéis que são assumidos por usuários externos ao sistema enquanto os agentes de software normalmente fazem parte do software e assumem papéis internos. Como os atores UML obrigatoriamente precisam ser externos ao software concluímos ser necessário criar a metaclasse AgentRole_Actor para modelar atores internos ao software que representem os papéis que os agentes poderiam assumir durante a execução do sistema.

Uma vez que papéis de agente podem se referir tanto a papéis reativos como a papéis cognitivos, optou-se por derivar, a partir da metaclasse AgentRole_Actor, as metaclasses Reactive_AgentRole e Cognitive_AgentRole para diferenciar esses dois tipos de papéis, conceitos que também não são suportados pela UML padrão.

Finalmente, com o intuito de adaptar o metamodelo ao trabalho proposto por Vicari (2007), foram derivadas as metaclasses PS_AgentRole, UAM_AgentRole e SMI_AgentRole a partir da metaclasse Cognitive_AgentRole para representar os papéis de agente especialistas Problem Solving (PS), Users and Agents Modelling (UAM) e Social Mediated Interactions (SMI). Vicari (2007) propõe o uso destes tipos de papéis de agentes para representar papéis que contenham conhecimento sobre o problema que o sistema pretende auxiliar a solucionar, sobre como modelar agentes e usuários e como mediar interações sociais entre os agentes e entre os agentes e os usuários. Novamente nenhum destes conceitos é suportado pela UML.

3.14.2.2 Metaclasse InternalUseCase

Casos de uso, como já foi dito anteriormente, são utilizados para a modelagem de requisitos do sistema. Eles representam processos que podem ser solicitados por atores externos ao software. Contudo, sistemas multi-agentes costumam ter processos internos que são transparentes aos usuários externos e que não podem ser solicitados por estes usuários. Estes processos internos costumam ser solicitados por agentes de software, que compõem os sistemas multi-agentes, quando interpretando um determinado papel.

Assim, considerando que a representação destes processos internos é importante na modelagem de requisitos de um sistema multi-agente e considerando que casos de uso normais não podem ser utilizados para modelá-los, concluí-se ser necessário criar uma nova metaclasse, a qual chamamos InternalUseCase, para representar estes processos.

Com isso rompeu-se, de certa forma, com a filosofia do diagrama de casos de uso, uma vez que este diagrama é voltado para a representação de requisitos externos, no entanto, sistemas multi-agentes apresentam diferenças com relação aos outros softwares, caracterizada justamente pela existência de agentes, que são entidades autônomas que interpretam papéis e solicitam serviços internos ao sistema, sem que os usuários externos normalmente tenham idéia de quais processos sejam ou mesmo que eles existam. É importante destacar que o conceito de caso de uso interno não existe na UML padrão.

3.14.2.3 Metaclasses Perception e Action

Dentre os processos que um agente de software, ao interpretar um papel, pode possuir, estão os de perceber eventos que ocorrem no ambiente onde o agente está inserido e o de executar ações sobre este ambiente, como enviar uma mensagem a outro agente, por exemplo.

Assim, levando em consideração que um agente ao assumir um determinado papel normalmente terá de perceber a ocorrência de eventos no ambiente e/ou executar ações sobre esse ambiente, decidiu-se derivar as metaclasses Perception e Action a partir da metaclasse InternalUseCase e aplicou-se a elas o estereótipo “stereotype”, o que significa que elas serão usadas como estereótipos dos casos de uso internos.

Pretende-se assim que estes casos de uso internos estereotipados sejam utilizados para modelar os passos necessários para que um agente perceba um determinado evento ou realize uma determinada ação.

Os conceitos de percepção e ação não são suportados pelo diagrama de casos de uso da UML padrão, porém, dentro do contexto de modelagem de requisitos de sistemas multi-agentes, julgou-se importante modelá-los, uma vez que são requisitos normalmente necessários aos papéis interpretados pelos agentes que fazem parte deste tipo de sistema.

3.14.2.4 Metaclasse Goal

Agentes de software, principalmente agentes cognitivos baseados na arquitetura BDI, costumam possuir objetivos, que representam os desejos que o agente possui, ou seja, situações que este agente almeja atingir. Estes desejos tornam-se intenções no momento em que um agente acredita que eles possam ser atingidos.

Levando em consideração a importância de representar os objetivos associados a cada papel de agente, concluiu-se ser necessário derivar uma terceira metaclasse a partir da metaclasse InternalUseCase, a metaclasse Goal. Da mesma forma que as metaclasses Perception e Action, a metaclasse Goal recebeu o estereótipo chamado stereotype, o que significa que ela será usada em conjunção com a metaclasse InternalUseCase para modelar processos internos que representem os objetivos de um determinado papel de agente.

Assim, um caso de uso interno empregando o estereótipo Goal conterá uma descrição de um desejo a possuído por um papel de agente, as possíveis condições para que o desejo torne-se uma intenção e os passos necessários para que um agente, quando assumir o papel, determine se o objetivo em questão pode realmente se tornar uma intenção. Cumpre destacar que o conceito de objetivo não existe no diagrama de casos de uso padrão da UML.

3.14.2.5 Metaclasse Plan

No momento em que um objetivo se torna uma intenção para um agente, este irá executar um ou mais planos para tentar atingir o objetivo em questão.

Considerando que um objetivo pode eventualmente possuir mais do que um plano e que este ou estes planos somente serão executados mediante certas condições, decidiu-se derivar uma quarta metaclasse a partir da metaclasse InternalUseCase, chamada Plan, para identificar os planos associados aos objetivos de um papel de agente. Novamente aplicou-se o estereótipo stereotype a esta nova metaclasse, o que significa que ela só poderá ser utilizada em conjunto com casos de usos internos.

Assim, um caso de uso interno contendo o estereótipo Plan conterá os passos referentes à execução de um plano para atingir um objetivo. O conceito de plano não é suportado pelo diagrama de casos de uso da UML padrão.

3.14.2.6 Metaclasse Plan Extend

Planos somente são executados quando o agente que interpreta um papel acredita que um objetivo possa se tornar uma intenção. Muitas vezes as crenças de um agente são modificadas quando o agente percebe a ocorrência de determinados eventos. Sendo assim é necessário que uma ou mais condições (que causem uma mudança nas crenças de um agente) sejam satisfeitas para que um objetivo se torne uma intenção e que um plano seja executado.

Uma vez que um plano somente é disparado depois que uma ou mais condições são satisfeitas, decidiu-se estender a metaclasse Plan Extend a partir da metaclasse Extend e associá-la a metaclasse Goal. Este novo tipo de associação, além de destacar que um caso de uso interno Plan poderá estender o comportamento de um caso de uso interno Goal, poderá também conter, por meio de restrições, a condição para que um plano possa estender o comportamento de um objetivo.

Procedeu-se desta forma para diferenciar associações de extensão normais de associações de extensão de plano, procurando assim tornar claro, que esta associação poderá estender o comportamento de um caso de uso interno do tipo Goal quando uma ou mais condições forem satisfeitas, ou seja, esta associação só poderá existir entre casos de uso internos do tipo Goal e do tipo Plan. Contudo, as associações de extensão normais continuam podendo ser utilizadas. O conceito de associação de extensão de plano não existe na UML padrão.

3.14.2.7 Metaclasse Plan Extension Point

Esta metaclasse foi derivada a partir da metaclasse Extension Point pertencente a UML padrão.

Esta metaclasse foi derivada para estabelecer uma diferença entre pontos de extensão de plano e pontos de extensão normais, além de permitir determinar os pontos no comportamento de um caso de uso interno Goal que podem ser estendidos por planos, além de também permitir destacar as condições para que um ou mais planos sejam executados.

3.14.2.8 Metaclasse IncludeActionPerception

Esta metaclasse foi derivada a partir das mesmas metaclasses que a metaclasse padrão Include foi derivada. Como já foi dito antes, um desejo se torna uma intenção no momento em que um agente acredita que existe uma possibilidade real de ele ser atingido, ou seja, quando as crenças do agente o fazem acreditar nisso. Para isso muitas vezes é necessário que o agente perceba a ocorrência de eventos que causem a mudança em suas crenças.

Assim foi considerado necessário incluir, por meio de associações, os comportamentos representados pelos casos de usos internos do tipo Perception, em casos de uso internos do tipo Goal, para auxiliar a se um objetivo pode se tornar uma intenção ou não.

Contudo, não parece ser possível utilizar a associação padrão de Inclusão da UML para essa função, uma vez que recomenda-se que o uso desta associação seja aplicado em situações onde o comportamento de um caso de uso possa ser incluído no comportamento de dois ou mais casos de uso. Embora objetivos possam compartilhar percepções, isto não é uma regra, assim nós consideramos que não é possível utilizar a associação de inclusão padrão para incluir o comportamento de casos de uso internos Perception no comportamento de casos de uso internos Goal.

Além disso, a execução de um plano por um agente pode incluir a execução de ações que causem mudanças no ambiente e/ou nas crenças de outros agentes, como o envio de mensagens a eles, por exemplo. Dessa forma, por se tratar de ações que causam eventos no

ambiente e/ou mudanças nas crenças de outros agentes, considerou-se válido representar estas ações como casos de uso internos com o estereótipo Action e incluí-las no comportamento dos casos de uso internos com o estereótipo Plan (ou eventualmente Goal), quando estes necessitarem realizar esse tipo de ação, além do que uma ação pode ser compartilhada por mais de um plano. Todavia, como não é uma regra que planos diferentes compartilhem ações, da mesma forma que com relação aos objetivos e percepções, não foi considerado possível utilizar a associação de inclusão padrão da UML para isso.

Assim, optou-se por criar uma nova metaclasse, chamada IncludeActionPerception, na qual foram copiadas as mesmas características da metaclasse Include, porém retirando a restrição de que ela seja usada quando há comportamentos comuns a mais de um caso de uso, dessa forma aplicaría-se essa nova associação para incluir o comportamento de percepções no comportamento de objetivos ou o comportamento de ações no comportamento de planos.

3.15 Ferramenta CASE Utilizada

A ferramenta utilizada para produzir os diagramas aqui apresentados foi a Enterprise Architect. Praticamente todas as inovações propostas pelo metamodelo são suportadas por esta ferramenta CASE, uma vez que ela permite a criação de estereótipos de texto de maneira bastante fácil. A ferramenta também permite editar os estereótipos das associações de “include” e “extend”, o que permite modificá-los para “include ap” ou “plan extend”. A única estensão não suportada por esta ferramenta é a Plan extension point, sendo necessário editar os diagramas produzidos pela ferramenta para realizar esta modificação.

4 DISCUSSÃO E CONSIDERAÇÕES FINAIS

O metamodelo apresentado é exclusivamente voltado para a modelagem de requisitos funcionais específicos para sistemas multi-agentes. Este metamodelo permite representar papéis de agente reativos e cognitivos, além dos tipos de papel de agente propostos por Vicari (2007), bem como permite também modelar as percepções e ações que os agentes deveriam possuir quando interpretando um papel, além dos objetivos que estes agentes deveriam atingir, os planos para satisfazer estes objetivos e as condições para que estes planos possam ser executados.

O metamodelo em questão preenche uma lacuna não enfocada por outras abordagens que estenderam a linguagem UML para o projeto de sistemas multi-agentes, uma vez que, nenhuma das linguagens estudadas preocupou-se em abordar a questão de análise de requisitos, uma fase de suma importância para o sucesso de um bom projeto de software.

Como forma de demonstrar a aplicabilidade desta proposta, foram desenvolvidos três estudos de caso, onde foram modelados os requisitos funcionais dos sistemas MCOE, AME-A e APA, além disso, adaptou-se a proposta original de forma a torná-la compatível com os princípios de projeto propostos por Vicari (2007) e demonstrou-se a aplicabilidade deste metamodelo dentro da arquitetura de camadas de agente, igualmente proposta por Vicari.

Há questões estruturais e comportamentais que precisam ser melhor detalhadas, uma vez que o metamodelo apresentado foca somente na modelagem de requisitos. Em fases posteriores à análise de requisitos, seria necessário representar informações como crenças, desejos e intenções por meio de diagramas de classe e os processos representados pelos casos de uso internos precisariam ser melhor detalhados por meio de diagramas comportamentais.

Nesta proposta foram apresentadas duas abordagens para contornar estas questões, na primeira, com relação ao sistema MCOE, utilizou-se diretamente os diagramas UML sem adaptação, considerando que este primeiro sistema era suficientemente simples para isso.

Na segunda abordagem foi feito um mapeamento de conceitos do metamodelo proposto para a linguagem MAS-ML, onde os AgentRole_Actors foram mapeados em classes AgentRole da MAS-ML e aplicou-se a linguagem para representar os processos comportamentais necessários. Foi feito ainda um mapeamento semelhante com relação à linguagem AML, onde se procurou identificar as semelhanças entre as duas abordagens e que conceitos do metamodelo poderiam ser mapeados para esta linguagem.

Com relação à validação do metamodelo, foi feita uma comporação entre os modelos criados por meio deste metamodelo com modelos criados a partir do diagrama de casos de uso original da UML, tentando com isso demonstrar que o novo metamodelo proposto permite a modelagem de conceitos não suportados pelo diagrama de casos de uso padrão. Foi feita ainda uma validação por justificativa, onde procuramos justificar a criação das metaclasses propostas em nosso metamodelo, demonstrando sua importância para a modelagem de requisitos para sistemas multi-agentes.

É importante considerar ainda que, uma vez que o metamodelo proposto enfoca uma área não coberta pelas linguagens de modelagem estudadas, ele pode ser acrescido a elas, podendo dessa forma torná-las mais completas. Os mapeamentos aqui apresentados entre os conceitos representados pelo metamodelo e os conceitos das linguagens AML e MAS-ML tentam demonstrar as compatibilidades existentes entre o metamodelo e estas linguagens e, por conseguinte, que o metamodelo pode ser complementado por elas e, da mesma forma, que estas linguagens podem ser complementadas pelo metamodelo.

A escolha da UML como a linguagem base a partir da qual este metamodelo foi estendido justifica-se por esta ser uma linguagem amplamente aceita e compreendida nos meios acadêmicos e na indústria de engenharia de software, o que facilitaria a compreensão das inovações fornecidas pelo metamodelo. Além disso, a UML é uma linguagem genérica projetada com o objetivo de ser aplicada a todos os domínios e quando isto não for possível, para ser facilmente estendida e adaptada a eles.

Optou-se por estender diretamente a UML e não alguma das linguagens já derivadas a partir desta com o objetivo de que a extensão proposta pudesse ser utilizada por quaisquer das linguagens que estenderam a UML para o projeto de sistemas multi-agentes e que não produziram mecanismos para a modelagem de requisitos para este tipo de software.

Com relação ao conceito proposto de Internal Use Case ou Caso de Uso Interno, como já foi falado anteriormente, se rompeu, de certa maneira, com a filosofia do diagrama de casos de uso da UML, posto que este diagrama é voltado para a representação de requisitos externos do sistema, enquanto casos de usos internos foram concebidos para representar os requisitos internos de sistemas multi-agentes.

Contudo, enquanto em muitos softwares os requisitos funcionais podem ser identificados como as funcionalidades que deverão ser oferecidas aos usuários externos pelo sistema, representadas por casos de uso, isso não é tão verdadeiro quando se trata de sistemas multi-agentes, já que este tipo de software apresenta diferenças com relação aos outros sistemas, caracterizada justamente pela existência de agentes de software, que são entidades autônomas e pró-ativas que, ao assumir papéis, solicitam funcionalidades internas ao sistema, sem que os usuários externos normalmente tenham idéia de quais funcionalidades sejam essas ou mesmo que elas existam.

Dessa forma, foi considerado importante identificar requisitos que representassem as funcionalidades internas necessárias aos papéis assumidos pelos agentes de software que viessem a compor um sistema, posto que concluiu-se ser preciso representar as percepções, ações, objetivos e planos associados aos já citados papéis durante a fase de levantamento e análise de requisitos.

A simples identificação dos requisitos externos em um sistema multi-agente foi considerada muito vaga, se for levado em consideração que as funcionalidades internas atribuídas aos papéis de agente muitas vezes consistiriam em uma parte importante do processamento do software a ser modelado, sendo a representação destes requisitos internos tão ou mais importante que os requisitos externos normalmente identificados pelo diagrama de casos de uso padrão da UML.

Com relação aos símbolos empregados para os AgentRole_Actors e InternalUseCases, seria possível criar símbolos particulares para eles, para diferenciá-los dos atores e casos de uso normais, no entanto o objetivo de se utilizar a mesma simbologia foi a de manter a maior compatibilidade possível com o diagrama de casos padrão da UML, destacando mais as semelhanças entre atores e casos de uso normais e AgentRole_Actors e casos de usos internos, tentando assim facilitar a compreensão de sua aplicação por parte de quem já domina o diagrama de casos de uso.

Quanto aos estereótipos utilizados serem estereótipos de texto que não modificam o desenho padrão dos símbolos empregados, optou-se por esta alternativa por se acreditar que isto facilitaria a compreensão de quem já é familiarizado com a aplicação do diagrama de casos de uso. Poderia-se ter criado estereótipos gráficos, porém a criação de símbolos diferentes poderiam causar confusão em quem fosse utilizar o metamodelo, além do que o uso de estereótipos de texto permite a sua utilização pela maioria das ferramentas CASE, posto que elas suportam a utilização de novos estereótipos de texto. Contudo, nada impede que se adicionem estereótipos gráficos ao metamodelo, modificando os símbolos utilizados atualmente.

Entre os comentários recebidos sobre os artigos que foram submetidos a respeito do metamodelo proposto estão afirmações de que este metamodelo poderia causar certa rigidez, devido a ser necessário identificar todos os requisitos de um sistema multi-agente antes de construir todo o projeto sobre os requisitos funcionais identificados. Afirmou-se que alguns sistemas multi-agentes são muito complexos e determinados objetivos, percepções, ações e principalmente planos poderiam não ser identificados completamente durante a análise de requisitos e que durante as fases seguintes poderia-se perceber que alguns requisitos não foram identificados. Com relação a planos, em uma revisão afirmou-se que planos não devem ser identificados na fase de análise de requisitos, porque isto poderia ser muito precipitado nesta fase.

Nós, por outro lado, acreditamos que os requisitos de um sistema devem ser identificados da forma mais completa possível e que deve-se evitar passar para novas fases sem ter certeza realmente do que deve ser projetado e desenvolvido. Com relação a planos, uma vez que identificamos os objetivos associados a um papel de agente e as condições para que estes se tornem intenções, acreditamos ser lógico identificar a necessidade da existência de um ou mais planos para atingir os objetivos em questão, estes planos, no entanto, podem e devem ser esmiuçados por meio de outros diagramas mais detalhados em fases posteriores. Na fase de levantamento e análise de requisitos a necessidade destes planos deve ser apenas identificada, havendo pouco detalhamento de seu comportamento, o detalhamento mais profundo deste comportamento é função de outros diagramas.

Todavia, é importante destacar novamente que a UML não é um método de desenvolvimento de software, ela é uma linguagem de modelagem que pode ser aplicada por qualquer método (inclusive da forma que o usuário desejar, o que não deixa de ser também um método) e cabe a estes métodos determinar como ela será aplicada.

Dessa forma, seria possível desenvolver um método cíclico, se assim fosse considerado necessário, onde somente os requisitos mais importantes seriam identificados e posteriormente detalhados em outras fases. Ao concluir um ciclo, novos requisitos poderiam ser identificados em um novo ciclo, repentindo-se este processo até considerar-se que o projeto estivesse terminado. Dessa forma, a identificação de planos não precisaria ser feita necessariamente na execução do primeiro ciclo, mas em ciclos posteriores, porém ainda assim os planos deveriam ser identificados como requisitos. Isto permitiria, ao olhar um diagrama de casos de uso, ter uma visão geral das funcionalidades do software que incluiria quais planos estariam associados a quais objetivos e quais as condições básicas para que estes pudessem ser disparados.

Poderia-se argumentar que em alguns sistemas multi-agentes pretende-se que os agentes elaborarem seus planos em tempo de execução, o que causaria uma complexidade maior na representação gráfica deste tipo de comportamento. Obviamente, se um plano precisa ser criado em tempo de execução não é possível representá-lo graficamente, poderia-se, no entanto, representar que seria necessário criar um ou mais planos associados a um

determinado objetivo em tempo de execução. Assim, mesmo nessas situações a representação dos planos poderia ser feita.

Com relação aos requisitos não-funcionais, a representação deste tipo de requisito não é suportada pelo metamodelo aqui proposto, posto que ele se atém à representação de requisitos funcionais. A representação de requisitos não-funcionais poderia ser feita – em alguns casos – por meio da utilização de restrições UML normais, isto poderia ser possível, em algumas situações, quando estes requisitos não-funcionais tratassem de restrições, validações e consistências referentes a um determinado requisito funcional, no entanto, existe um grande número de diferetes tipos de requisitos não-funcionais, alguns nem mesmo se referem a um requisito funcional específico mas ao sistema como um todo, dessa maneira a representação de requisitos não-funcioanis pode vir a ser bastante complexa.

5 TRABALHOS FUTUROS

Com relação aos trabalhos futuros, consideramos que poderia ainda ser necessário:

- Definir uma forma de representação de requisitos não-funcionais, característica não coberta pelo metamodelo aqui proposto. Requisitos não-funcionais, em sua forma mais básica, correspondem às restrições, condições, consistências, validações que devem ser levadas a efeito sobre os requisitos funcionais. Assim, a definição deste tipo de requisitos não-funcionais poderia ser útil para complementar o metamodelo proposto neste trabalho e expressar informações extras, úteis para fases posteriores da modelagem e para a própria implementação do software. No entanto, cabe aqui uma ressalva, no que tange à essa definição, posto que existem outros tipos de requisitos não-funcionais que, por vezes, não estão associados diretamente a um requisito funcional específico, estando estes mais ligados ao sistema como um todo. Por esse motivo a representação de requisitos não-funcionais, devido à sua diversidade, pode ser bastante complexa.

- Estudar a possibilidade de desenvolver uma metodologia para a aplicação do metamodelo. Como já foi dito antes, a UML é apenas uma linguagem, uma notação, assim uma metodologia serviria para definir os passos de como aplicar o metamodelo proposto na modelagem de um sistema multi-agente.

- Elaborar uma ontologia, estabelecendo um vocabulário formal para o metamodelo. “Uma ontologia é uma especificação formal explicita de uma conceitualização compartilhada” (Fensel, 2001), assim uma ontologia serviria para estabelecer formalmente a definição correta dos termos utilizados no metamodelo.

REFERÊNCIAS

ARLOW, J., NEUSTADT, I. **UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition).** (2005). Addison-Wesley.

AUML Official Website. (2007). Disponível em: www.auml.org. Acesso em: mar. 2009.

BAUER, B., ODELL, J.: **UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard.** in Journal of Engineering Applications of AI, Volume 18, Issue 2, pp 141-157. (2005).

BEER, M., et al. **Designing Community Care Systems with AUML.** in Proceedings of the International Conference E.U-LAT e-Health, Cuernavaca, Mexico, 2003.

BERENBACH, B. **Software & Systems Requirements Engineering In Practice.** (2009). Livro, McGraw-Hill, New York.

BERGAMASCHI, S., et alii. **Experiencing AUML for the WINK Multi-Agent System.** in Proceedings of AIIA and TABOO Workshop: From Object to Agents (WOA03), Villassimius, CA, 2003.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML.** (2002). 286 p. Livro, Campus, Rio de Janeiro.

BOOCH, G., RUMBAUGH, J. e JACOBSON, I. **UML Guia do Usuário.** (2000). 472 p. Livro, Campus, Rio de Janeiro.

BRATMAN, M.E. **Intention, Plans, and Practical Reason.** CSLI Publications. ISBN 1-57586-192-5, 1999.

BRESCIANI P. et alii. **Tropos: An Agent-Oriented Software Development Methodology.** Autonomous Agents and Multi-Agent Systems. v.8, n.3, 203-236 (2004).

BURRAFATO, P., COSSENTINO M. **Designing a multi-agent solution for a bookstore with the PASSI methodology.** Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) - 27-28 May 2002, Toronto (Ontario, Canada) at CAiSE'02. www.csai.unipa.it/cossentino/paper/AOIS02.pdf

CAIRE, Giovanni et alii. **Agent Oriented Analysis Using Message/UML.** Agent-Oriented Software Engineering II. Springer Berlin / Heidelberg. ISBN 978-3-540-43282-1, 2002.

CERNUZZI, L.; ZAMBONELLI, F. **Experiencing AUML in the GAIA Methodology.** ICEIS 2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004a.

CERNUZZI, L.; JUAN, T.; STERLING, L.; ZAMBONELLI, F. **The Gaia Methodology: Basic Concepts and Extensions.** In Methodologies and Software Engineering for Agent Systems, Kluwer, 2004b.

CERVENKA, R. **Modeling Notation Source Prometheus**. Foundation for Intelligent Physical Agents Modeling Notation Source – Prometheus. Version: 03-04-02. 2003.

CERVENKA, R. **Class Diagram Superstructure Metamodel – Response from Radovan Cervenka. Working Documents.** AUML Official Website, 2004. Disponível em: <http://www.auml.org/>. Acesso em: set. 2007.

CERVENKA, R.; TRENCANSKY, I.; CALISTI, M.; GREENWOOD, D. **AML: Agent Modeling Language. Toward Industry-Grade Agent-Based Modeling.** In ODELL, J.; GIORGINI, P.; MULLER, J., eds.: Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004, Springer-Verlag (2005a).

CERVENKA, R., TRENCANSKY, I., CALISTI, M. **Modeling Social Aspects of Multi-Agent Systems: The AML Approach.** Proceedings of AAMAS05, AOSE, Utrecht, The Netherlands, p 85-96, 2005b.

CHEONG, C.; WINIKOFF, M. Improving Flexibility and Robustness in Agent Interactions: Extending Prometheus with Hermes. **In Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications (SELMAS post-proceedings, published by Springer)**, 2005.

COHEN, P. & LEVESQUE, H. **Intention Is Choice with Commitment. Artificial Intelligence,** n. 42, p. 213-261, 1990.

D'AMICO, C. B., PEREIRA, A. S,, GEYER, C. F. R., VICARI, R. **Adapting Teaching Strategies in a Learning Environment on WWW.** In: Proceedings of the WebNet World Conference of the WWW, Internet & Intranet. Florida, USA. 1998.

DELOACH, S. A. Analysis and Design using MaSE and agentTool. **12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)**, Miami University, Oxford, Ohio, March 31 - April 1, 2001.

DELOACH, S. Multiagent Systems Engineering of Organization-based Multiagent Systems. **4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05).** Springer LNCS Vol 3914, (2006).

DILLENBOURG, P. e SELF, J.A. A framework for learner modelling. **Interactive Learning Environments,** n. 2, p.111–137, (1992).

FENSEL, D., DING, Y., SCHULTEN, E., OMELAYENCO, B., BOTQUIN, G., BROWN, M., FLETT, A. **Product Data Integration in B2B E-Commerce.** IEEE Intelligent System. Volume 16, 2001.

FIPA - The Foundation for Intelligent Physical Agents. FIPA Specifications Repository. (2004). <http://www.fipa.org/repository/index.html>.

FLAKE, S., Geiger C., e Küster J. **Towards UML-based Analysis and Design of Multi-Agent Systems.** Proceedings of ENAIS'2001, Dubai, March 2001.

FRANKLIN, S. & GRAESSER, A. **Is it an agent, or just a program?: A taxonomy for autonomous agents.** In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, pages 21–35, Berlin, Germany. Springer Verlag, 1996.

GIRAFFA, L., Mora, M. e Viccari, R. **Modelling the MCOE Tutor using a Computational Model**, Lectures Notes on Artificial Intelligence - SBIA'98, 1998.

GUEDES, G. T. A., VICARI, R. **A UML Profile Oriented to the Requirements Collecting and Analyzing for the MultiAgent Systems Project.** Proceedings of the 22nd International

Conference on Software Engineering and Knowledge Engineering - SEKE2010, Redwood, California, USA (2010a).

GUEDES, G. T. A., VICARI, R. **A UML Profile Oriented to the Requirements Modeling in Intelligent Tutoring Systems Projects.** Proceedings of The Third IFIP tc 12 International Conference on Artificial Intelligence, IFIP AI 2010, Held as Part of WCC 2010 – IFIP AI, Brisbane, Australia (2010b).

GUEDES, G. T. A., VICARI, R. **Applying a UML Profile in the Requirements Modeling to Multi-Agents Systems.** Proceedings of 7th International Conference on the Quality of Information and Communications Technology – Quality in ICT Requirements Engineering – Quatic 2010, Porto, Portugal (2010c).

GUEDES, Gilleanes T. A.; VICARI, R. M. **Applying and Validating a UML Metamodel for the Requirements Analysis in Multi-Agent Systems: The AME-A Case Study.** Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering, Volume 1, Páginas 746 a 751, ISBN 1891706292, Miami Beach, USA (2011a).

GUEDES, Gilleanes T. A.; VICARI, R. M. **Applying a UML Metamodel to the Requirements Modeling in Multi-Agents Systems Projects - The APA Case Study.** Proceedings of the XV Portuguese Conference on Artificial Intelligence - EPIA 2011 – MASTA – Thematic Track on Multi-Agent Systems: Theory and Applications. Volume 1, Páginas 609 a 623, ISBN 9783642247699, Lisboa, Portugal (2011b).

HUGET, M. P. et alii. Class Diagram. Working Documents. **AUML Official Website**, 2003a. Disponível em <http://www.auml.org/>. Acesso em: set. 2007.

HUGET, M. P. et alii. Interaction Diagrams. Working Documents. **AUML Official Website**, 2003b. Disponível em: <http://www.auml.org/>. Acesso em: set. 2007.

HUGET, M. P.; ODELL, J. **Representing Agent Interaction Protocols with Agent UML.** The Third International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2004). New York City, New York, 2004.

HULL, E., JACKSON, K.; DICK, J. **Requirements Engineering – Second Edition.** (2005). Livro, Springer, London Berlin Heidelberg, ISBN 1-85233-879-2.

JAYATILLEKE, G. B.; PADGHAM, L.; WINIKOFF, M. **A Model Driven Development Toolkit for Domain Experts to Modify Agent Based Systems.** Agent-Oriented Software Engineering VII – Workshop Notes for the 7th International Workshop on Agent-Oriented Software Engineering – AOSE 2006. In: 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), ACM 2006, ISBN 1-59593-303-4, Hakodate, Japan, May 8-12, 2006.

JENNINGS, N. R.; WOOLDRIDGE, M. J. **Intelligent Agents: Theory and Practice. Knowledge Engineering Review**, v; 10 n. 2, 1995.

JUAN, T. et alii. **ROADMAP: Extending the Gaia methodology for Complex Open Systems,** Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, July 2002.

KASANEN, F.; KARI, L.; ARTO, S. **The constructive approach in management accounting research.** Journal of Management Accounting Research, [S.l.], v.5, p.243– 259, 1993.

KHALLOUF, J. e WINIKOFF, M. **Goal-Oriented Design of Agent Systems: A Refinement of Prometheus and its Evaluation.** International Journal Agent-Oriented Software Engineering, Volume 3, Number 1, (2009).

LUKKA, K. **The constructive research approach.** In: CASE STUDY RESEARCH IN LOGISTICS, 2003. Anais. . . [S.l.: s.n.], 2003. v.B1, p.83–101.

MARCH, S. T.; SMITH, G. F**. Design and natural science research on information technology.** Decision Support Systems, [S.l.], v.15, n.4, p.251 – 266, 1995.

MELO, A. C. **Desenvolvendo Aplicações com UML – Do Conceitual à Implementação.** (2002). 262 p. Livro, Brasport, Rio de Janeiro.

MYLOPOULOS, J.; KOLP, M.; CASTRO, J. **UML for Agent-Oriented Software Development: The Tropos Proposal,** Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools**,** Lecture Notes in Computer Science, Springer-Verlag London UK, V. 2185. ISBN 3-540-42667-1, 2001.

ODELL, J. et alii. Class Diagram Superstructure Metamodel - The result of London FIPA meeting and interim work. Working Documents. **AUML Official Website**, 2004. Disponível em: <http://www.auml.org/>. Acesso em: set. 2007.

OMG - Object Management Group. **Unified Modeling Language: Infrastructure Specification - Version 2.1.1.** OMG (2007a), http://www.omg.org.

OMG - Object Management Group. **Unified Modeling Language: Superstructure Specification - Version 2.1.1.** OMG (2007b). <http://www.omg.org>.

PADGHAM, L.; WINIKOFF, M. Prometheus: **A Methodology for Developing Intelligent Agents.** In: Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02. (2002).

PADGHAN, L. e WINIKOFF, M. **Developing Intelligent Software Agents: A Practical Guide.** John Wiley & Sons (2004).

PADGHAM, L.; WINIKOFF, M. Prometheus: A Methodology for Developing Intelligent Agents. **In: Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02.** (2002).

PADGHAN, L.; THANGARAJAH, J.; PAUL, P. Prometheus Design Tool - Version 2.5 - User Manual. RMIT Agents Group, 2006. <http://www.cs.rmit.edu.au/agents/pdt/docs/PDT-Manual.pdf>. Acesso em: mar. 2006.

PAPASIMEON, M.; HEINZE, C. **Specifying Requirements in a Multi-Agent System with Use Cases.** Technical Report February, Defence Science and Technology Organisation, 2003.

PAULA FILHO, W. P. **Engenharia de Software – Fundamentos, Métodos e Padrões.** (2003), 602 p. Livro, LTC, Rio de Janeiro.

PERES, J.; BERGMANN, U. **Experiencing AUML for MAS Modeling: A Critical View.** In Proceedings of Software Engineering for Agent-Oriented Systems (SEAS), p. 11-20, (2005).

PRESSMAN, R. **Engenharia de Software.** 1056 p. (1995). Livro, Makron Books. São Paulo.

REGNELL, B. **Requirements Engineering with Use Cases – A Basis for Software Development.** (1999). 219 p. Livro, Lund University, Sweden.

RMIT – Computer Science and Information Technology. Tutorial Prometheus. 2004. Disponível em: < http://www.cs.rmit.edu.au/agents/pdt/docs/Tutorial.pdf >. Acesso em: set. 2007.

RODRIGUEZ, L.; INSFRAN, E.; CERNUZZI, L. **Requirements Modeling for Multi-Agent Systems.** Multi-Agent Systems Modeling, Control, Programming, Simulations and Applications – Capítulo 1, ISBN 978-953-307-174-9, 2011.

SILVA, V.; GARCIA, A.; BRANDÃO, A.; CHAVEZ, C.; LUCENA, C.; ALENCAR., P. **Taming Agents and Objects in Software Engineering.** Software Engineering for Large-Scale Multi-Agent System, LNCS 2603, Springer-Verlag, pp. 1-26, (2003).

SILVA, V. T. **From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language.** (2004a). 259 p. Tese de Doutorado, Pontífica Universidade Católica do Rio de Janeiro.

SILVA, V. T., CHOREN, R., LUCENA, C. **Using the MAS-ML to Model a Multi-Agent System.** (2004b). In: C. Lucena; A. Garica; A. Romanovsky; J. Castro; P. Alencar. (Org.). Software Engineering for Multi-Agent SystemsII. : Springer-Verlag, v. LNCS, p. 129-148.

SILVA, V.; CHOREN, R.; LUCENA, C. **A UML Based Approach for Modeling and Implementing Multi-Agent Systems.** In Proceeding of the third International Conference on Autonomous Agents and Multi-Agents Systems, USA. (2004c).

SILVA, C. T. L. L. **Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case.** (2007). 172 f. Tese (Doutorado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco – UFPE, Recife.

SILVA, V. T. da, CHOREN, R., LUCENA, C. J. P.de (2008). MAS-ML: A Multiagent System Modelling Language. **International Journal of Agent Oriented Software Engineering.** Volume 2 Número 4, 2008.

SOMMERVILLE, I. **Engenharia de Software – 6ª. Edição.** (2003). 592 p. Livro. Addison Wesley, São Paulo.

SYCARA, K. P. Multiagent Systems (1998). AI Magazine, vol. 19(2), pp. 79-92.

TAVETER, K., WAGNER, G. Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modelling. In: Idea Group(Hershey). **Agent-Oriented Methodologies.** ISBN 1-59140-581-5, USA, 2005.

TRENCANSKY, I., CERVENKA R.. (2005). Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS. **Informatica.** Volume 29, Número 4, November 2005.

TIRYAKI, A. M.; ÖZTUNA, S.; DIKENELLI, O.; ERDUR, R. C. **SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems.** Agent-Oriented Software Engineering VII – Workshop Notes for the 7th International Workshop on Agent-Oriented Software Engineering – AOSE 2006. In: 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), ACM 2006, ISBN 1-59593-303-4, Hakodate, Japan, May 8-12, 2006.

VICARI, R. M.; GLUZ, J. C., An Intelligent Tutoring System (ITS) View on AOSE. **International Journal of Agent-Oriented Software Engineering.** V. 1, n 3-4, p.295-333, 2007.

WAGNER, G. **The agent-object-relationship meta-model: Towards a unified view of state and behavior.** Information Systems, 28(5), 475-504. (2003a). http://aor.rezearch.info/

WAGNER, G. **A UML Profile for External AOR Models.** In F.Giunchiglia, J. Odell, & G. Weiss (Eds.), Agent-Oriented Software Engineering III, Third International Workshop on

Agent-Oriented Software Engineering. Bologna, Italy, July 15, Revised Papers and Invited Contributions. LNCS, Vol. 2585. Berlin: Springer-Verlag, ISBN 3-540-00713-X. (2003b).

WAZLAWICK, R. S. **Análise e Projetos de Sistemas de Informação Orientados a Objetos.** (2004). 253 p. Livro. Campus, São Paulo.

WERNECK, V. M. B.; KANO, A. Y.; CYSNEIROS, L. M. **Evaluating ADELFE Methodology in the Requirements Identification.** 10th Workshop on Requirements Engineering, p. 13-24, Toronto, Canada (2007).

WINIKOFF, M. **The Prometheus Methodology.** 433-682 (Intelligent) Software Agents 2004. Department of Computer Science and Software Engineering, The University of Melbourne, 2nd April 2004.

XIAO, L. **Agent-oriented Requirements Modeling.** Dissertação de Mestrado. School of Informatics, University of Edinburgh, September 6, 2003

YU, E. **Why Agent-Oriented Requirements Engineering.** Proceedings of 3rd International Workshop on Requirements Engineering: Foundations for Software Quality (June 16-17, 1997, Barcelona, Catalonia). E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, 1997.

ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE M. J. **Developing Multiagent Systems: The Gaia Methodology.** ACM Transactions on Software Engineering and Methodology, V. 12, No. 3, July, p. 317–370, 2003.

ZAVE, P. **Classification of Research Efforts in Requirements Engineering.** ACM Computing Surveys, 29(4): 315-321, 1997.

ANEXO I

Sistema Multiagente para Criação/Edição/Alteração de OA

Neste anexo foi incluída uma descrição de um projeto para um sistema multiagente para edição de objetos de aprendizagem. Este projeto encontra-se em uma fase inicial e está sendo desenvolvido pelo aluno de doutorado Ederson Silveira na Unisinos. O metamodelo proposto nesta tese foi aplicado no projeto deste aluno como será demonstrar a seguir.

O objetivo da aplicação do metamodelo por outra pessoa foi o de verificar a facilidade com que outras pessoas conseguem aplicá-lo, bem como apresentar uma outra forma de validar o metamodelo proposto.

1 APRESENTAÇÃO DO SISTEMA

O principal objetivo do trabalho é desenvolver uma solução de software que de apoio à edição e criação de objetos de aprendizagem. Conforme descrito no referencial técnico um Objeto de Aprendizagem (OA) pode ser visto como um artefato composto de duas camadas (ou níveis):

- Camada dos Metadados: que engloba as informações de catalogo do objeto de aprendizagem, dizendo como este é composto e informações diversas do mesmo, dentre estas informações salientamos dados descritivos utilizados em busca, localização, recuperação e apresentação do conteúdo;
- Camada de Conteúdo: que contém o material de aprendizado em si, descrito pelos metadados, mas que deve ser visualizado pelo usuário para atingir os objetivos de determinada lição.

Sabendo que os metadados são informações que descrevem os objetos de aprendizagem, sempre que houver uma alteração no formato do conteúdo ou mesmo a existência de outros formatos para o mesmo conteúdo, é importante que isso seja descrito nos metadados para manter unificado o catálogo a respeito de um OA, mesmo que este OA possa oferecer visualizações distintas.

Desta forma, para que seja possível reutilizar um objeto de aprendizagem em múltiplas plataformas, é necessário que este seja adaptado tanto em relação aos seus metadados, quanto em relação ao seu conteúdo.

A edição/alteração de metadados requer um alto nível de conhecimento dos padrões existentes no mercado. Apenas esta etapa de alteração dos OAs por si só oferece um sério problema de projeto e desenvolvimento. Muito deste trabalho de estudo e alteração de metadados estão contidos nas atividades conduzidas pelo projeto OBAA. A utilização deste padrão permite a aderência aos padrões de mercado, ao mesmo tempo em que beneficia novos projetos com características importantes como a edição/alteração de OAs para múltiplas plataformas.

Com o objetivo de maximizar a aderência aos padrões e garantir a compatibilidade dos metadados descritos pelo projeto, este trabalho irá utilizar somente objetos de aprendizagem compatíveis com o padrão OBAA.

A proposta de metadados OBAA é uma proposta aberta compatível com os padrões educacionais e também adequada às tecnologias multimídias atuais. Este projeto tem por finalidade oferecer um repositório para objetos de aprendizagem em plataformas web e televisão digital.

Mesmo utilizando como base os metadados do padrão OBAA, algumas tarefas de atualização ou adição de metadados podem ser necessárias. Para realizar esta tarefa é de grande importância o conhecimento do conjunto de metadados disponibilizado pelo padrão escolhido e o entendimento de como eles devem ser configurados a partir do surgimento de novas versões adaptadas de um mesmo conteúdo.

A partir da definição do OBAA como padrão de metadados do trabalho aqui proposto, o outro artefato que compõe um objeto de aprendizagem, o conteúdo em si, torna-se o foco de alteração, edição ou criação (autoria) de novos conteúdos.

O objetivo do presente trabalho é projetar e desenvolver um protótipo de aplicação capaz de manipular os metadados do padrão OBAA e realizar as alterações do conteúdo correspondente. Para tanto será proposta uma arquitetura de sistemas multiagente que possa ser integrada à infraestrutura MILOS, atualmente sendo projetada e desenvolvida no projeto OBAA-MILOS.

O projeto de pesquisa OBAA-MILOS tem por objetivo a busca da convergência das tecnologias de objetos de aprendizagem (OA) e sistemas multiagente, para auxiliar no desenvolvimento de ambientes de aprendizagem. Tais objetos de aprendizagem são baseados na tecnologia de agentes inteligentes para seu desenvolvimento.

A infraestrutura MILOS, que está sendo desenvolvida nesse projeto, oferecerá suporte a objetos de aprendizagem compatíveis com o padrão OBAA, fornecendo suporte ao ciclo de vida inteiro de um OA e possibilitando que estes objetos sejam interoperáveis na WEB, televisão digitais e dispositivos móveis. A infraestrutura MILOS faz uso extensivo das tecnologias de ontologias e de agentes para desenvolver aplicações inteligentes de manipulação de objetos de aprendizagem. Agentes inteligentes de software, com suporte de mecanismos de inferência capazes de manipular os tipos de representações de conhecimento usados em ontologias (OWL), serão os principais componentes ativos da infraestrutura MILOS.

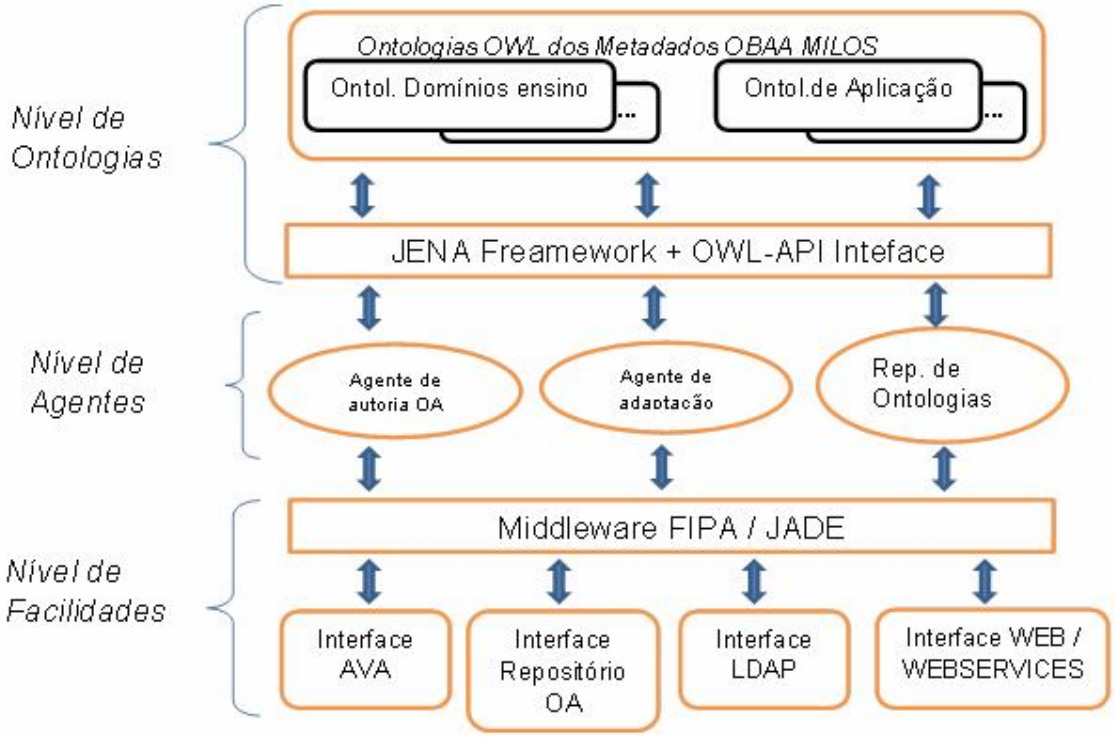


Figura1 – Organização geral MILOS

A arquitetura da infraestrutura MILOS é dividida em três grandes níveis de abstração conforme mostrado na Figura1: (a) Nível das Ontologias: responsável pela especificação do conhecimento que será compartilhado entre os agentes da infraestrutura; (b) Nível de Agentes: responsável pela implementação do suporte aos requisitos de adaptabilidade, interoperabilidade e acessibilidade previstos na proposta OBAA; (c) Nível das Facilidades de Interface: responsável pela comunicação dos agentes da MILOS com servidores Web, ambientes virtuais, repositórios de objetos de aprendizagem, bancos de dados, serviços de diretórios e demais tipos de aplicações educacionais. As principais entidades de cada nível da infraestrutura MILOS, respectivamente ontologias, agentes e facilidades, desempenham papéis organizacionais específicos para que a arquitetura da MILOS possa operar.

1.1 Sistema Multiagente de Autoria de OA

A principal funcionalidade prevista para o Sistema Multiagente de Autoria de OA é a criação e alteração de conteúdos destes objetos. A edição de conteúdo permite transformar o objeto um formato descrito e aceito pela infraestrutura MILOS para um novo objeto de aprendizagem de acordo as instruções do desenvolvedor de OA ou criados pelo agente wizard. Para conseguir o reconhecimento dos diversos tipos de conteúdos disponíveis nos objetos de aprendizagem o sistema multiagente desenvolvido possui um conjunto de agentes e planos de validação e reconhecimento de extensões de arquivos que interagem entre si, para chegar ao objetivo de atender as premissas de padrões de objetos de aprendizagem suportados pela infraestrutura MILOS.

Para representar estas interações entre os componentes de software que compõem o sistema multiagente, a presente dissertação apresenta a seguir diagrama explicativo sobre o processo de seleção dos planos de alteração/criação de objetos de aprendizagem. Este diagrama fora desenvolvido na fase de análise da solução de software para o desenvolvimento da solução. “Nos processos de desenvolvimento de software, encontramos três fases principais: a análise, o projeto e o desenvolvimento” (BAUER; ODELL, 2002). Nas metodologias de desenvolvimento mais recentes, chamadas metodologias ágeis, estas fases também estão presentes, só que em ciclos de menor granularidade onde os artefatos de software são refinados a cada novo ciclo.

Atualmente a UML pode ser utilizada com qualquer metodologia de projeto e a qualquer paradigma, atual, de desenvolvimento de software. Com a Engenharia de Software orientada a agentes não é diferente. Desta forma o sistema multiagente proposto neste trabalho será desenvolvido utilizando ciclos rápidos de análise, projeto e desenvolvimento aliados a diagramas da UML 2.0, conforme proposto em (GUEDES; GILLEANES, 2011).

A fase inicial de um projeto de software é a análise onde é possível definir o que o sistema deve fazer em um modelo chamado diagrama de casos de uso. Por padrão este diagrama apresenta os usuários externos ao sistema, atores, e as funcionalidades que estes podem executar dentro do sistema, os casos de uso. Ao modelar sistemas orientados a agente este conceito sofre uma adaptação para correto entendimento do diagrama. Segundo Bauer e Odell (2002), “nos modelos orientados a agentes encontramos atores internos ao sistema, que são os agentes de software”. Os serviços que cada agente de software disponibiliza são representados pelos casos de uso e ligados diretamente a ele. Os serviços podem ser requisitados por atores externos ao sistema, podendo estes representar agentes de software ou pessoas normais. Estas requisições são modeladas como associações direcionadas aos casos de que representam os eventos que eles devem responder. Na literatura de Engenharia de Software orientada a agentes estes eventos representam as percepções de cada agente, agentes de software (GUEDES; GILLEANES, 2011). As reações dos agentes de software ou a resposta do serviço é modelada como uma associação direcionada ao lado oposto das percepções. O retângulo, que envolve os serviços e os agentes que os disponibilizam, determina os limites do sistema multiagente.

A figura 1.1 apresenta um diagrama de casos de uso do Sistema Multiagente para Autoria de OAs sendo proposto neste trabalho.

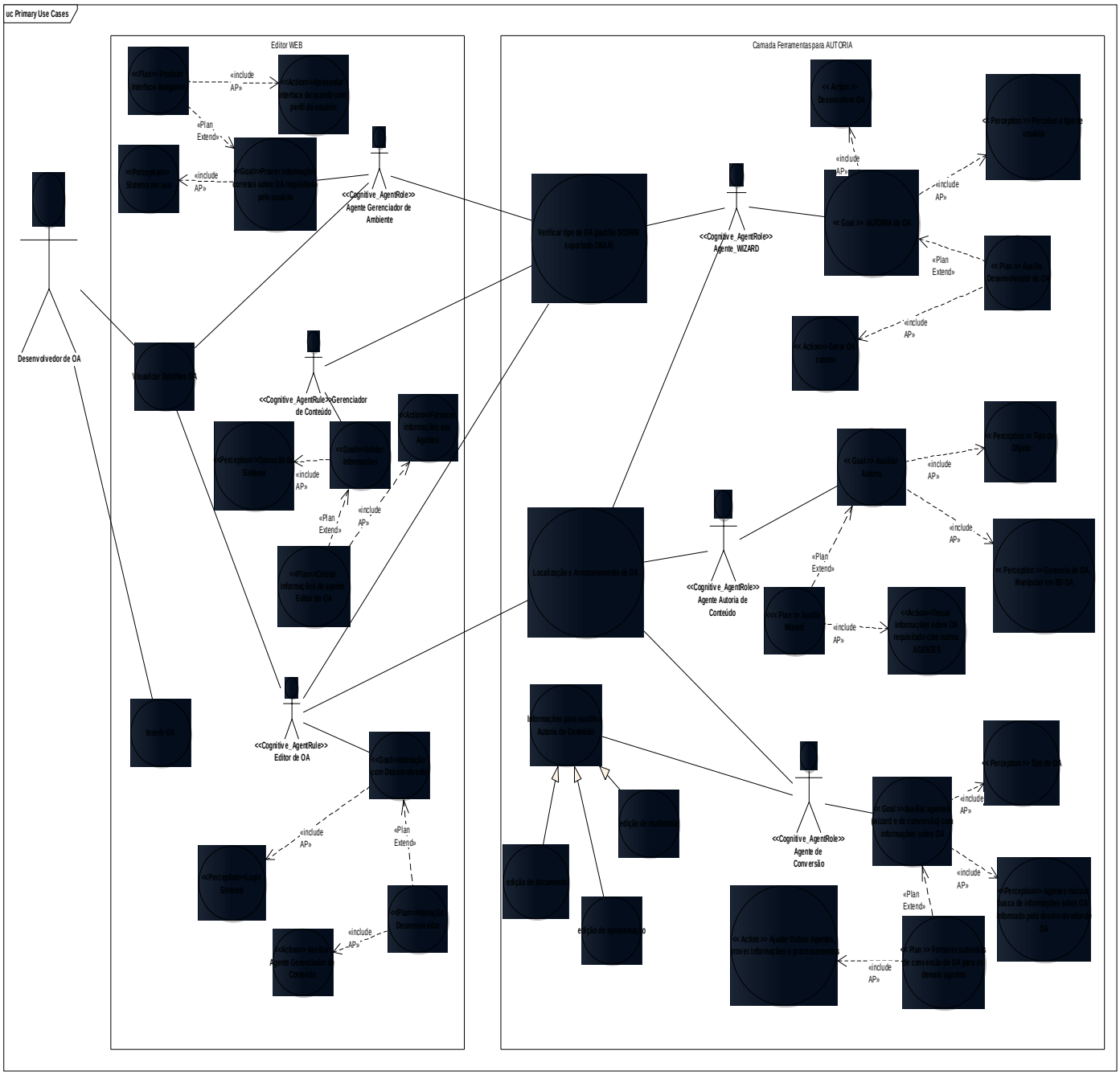


Figura 1.1 – Diagrama de Caso de Uso do Sistema de Autoria de OA

O diagrama da Figura 1.1 ilustra no primeiro retângulo o Sistema Multiagente de Autoria de OAs. Este sistema está sendo inicialmente concebido com sies agentes, capazes de executar diversas atividade e planos para atingir seu principal objetivo a Autoria de Objetos de Aprendizagem. Os agentes de software que compõem o sistema e os serviços que estes devem prover são os seguintes:

- Agente WIZARD: ele será responsável pelo serviço de automatização do processo de autoria de OA, onde este faz chamadas aos demais agentes e define a estratégia de adaptação a partir do tipo de objeto de aprendizagem a ser criado.

- Agente de Autoria de Conteúdo: o agente de edição é um dos agentes que compõem a estratégia de alteração/criação. Este agente será responsável por criar um novo formato de conteúdo do objeto de aprendizagem para um formato que atenda as necessidades do desenvolvedor de OAs atendendo as especificações da infraestrutura MILOS. Além disso, este agente deve fazer adaptações de layout necessárias, assim como definir os melhores tamanhos, cores e tipos de fontes e fundo para o objeto de aprendizagem final. Este agente deverá possuir acesso a um banco de dados local onde serão armazenados os objetos originais recebidos através do desenvolvedor de OA, ou seja, deverá armazenar uma cópia do objeto de aprendizagem. Este agente é responsável por ativar ou informar o Agente de Conversão de Formatos, para formato deverá ser criado o OA.
- Agente de Conversão de Formatos: agente responsável por gerenciar os diversos formatos do AO na camada de "ferramenta para autoria", através deste agente os demais sistemas terão a flexibilidade para aceitar os diversos tipos de objetos e também suporte multiplataforma em nível de sistema operacional. Este agente terá como característica o comportamento de wizard especifico para o objeto alvo selecionado pelo desenvolvedor de objetos de aprendizagem, ou seja, para cada extensão final escolhida, este agente terá o comportamento exclusivo para atender a necessidade de autoria do objeto.
- Agente Gerenciador de Ambiente: este agente é responsável pela organização e gerenciamento de informações apresentadas no ambiente "Editor WEB", sua característica se apresenta como tornar "amigável" a apresentação do sistema para o desenvolvedor em seus níveis de conhecimento (iniciante, intermediário e expert).
- Agente Gerenciador de Conteúdo: este agente é responsável pelo controle e organização das informações relacionadas aos OA's que o usuário deseja fazer a autoria pelo sistema, sempre obedecendo ao escopo do projeto OBAA MILOS em relação aos diversos tipos de objetos de aprendizagem suportados por esta arquitetura.
- Agente Editor de OA: este agente é responsável pelo intercambio de informações do sistema de autoria com o ambiente visível pelo desenvolvedor de objetos de aprendizagem. O agente recebe as informações do desenvolvedor e as transmite para que os demais agentes do sistema "ferramenta para AUTORIA", para que estes possam iniciar suas tarefas.

No segundo retângulo, com o rótulo "Editor WEB", é mostrada uma prévia do sistema web que será utilizado para gerenciar os perfis de alimentação, a autoria do conteúdo dos OAs. O usuário do sistema, o desenvolvedor de OA, interage com o a interface web que será planejada para executar as funcionalidades descritas nos atores. A ferramenta base para auxilio à autoria de objetos de aprendizagem via web, deverá atender aos requisitos do padrão OBAA, além de poder ser integrada a infraestrutura MILOS.

Este diagrama facilita a definição dos papéis dos agentes e a visualização das funcionalidades que o sistema irá disponibilizar. Outra característica deste diagrama apresentada é "mostrar a interação dos agentes que compõem o sistema que será desenvolvido com outros sistemas multiagente". No caso do diagrama apresentado na Figura 1.1 ainda podemos ver a interação do sistema multiagente desenvolvido com sistemas legados, não orientados a agentes, como o caso do "Editor WEB" que é um sistema orientado à web. Outro ator importante do sistema é o Desenvolvedor de OA, mas este não representa um agente de software, ele representar um usuário do sistema.

Após a fase de análise ser concluída com a especificação detalhada dos requisitos do sistema através da descrição dos casos de uso, e verificada a viabilidade do sistema e seu emprego dentro do projeto OBAA. A identificação de todos os atores, todos requisitos do sistema e suas funcionalidades será possível iniciar a fase de projeto, onde serão criados diagramas que definem aspectos dinâmicos e estáticos do sistema. Esta fase será apresenta futuramente após a concretização das informações contidas no presente diagrama de caso de uso.

1.2 Agentes do Sistema

A palavra “Agentes” é um termo utilizado de forma coerente em praticamente todas as áreas do conhecimento onde existam as noções de individuo e comunidade, como no socialismo, na economia, no comportamento animal, na robótica, no software e etc...(ALLY, M 2005). Uma característica que unifica os exemplos citados é a existência de um ambiente onde elementos autônomos representam, manipulam, trocam informações e conhecimento, demonstrando uma capacidade cognitiva em relativamente elevada.

Um agente é uma entidade que executa uma diversidade de tarefas que lhe são incumbidas com algum grau de independência ou autonomia. A necessidade de execução dessas tarefas se tornam os desejos (ou mais propriamente, os objetivos) do agente. O agente executa estas tarefas através do emprego de conhecimentos e das percepções sobre o que está ocorrendo no ambiente.

A seguir são definidos os objetivos, percepções, intenções e capacidades que cada agente deve desempenhar no Sistema de Autoria de OAs.

Agente Gerenciador de Ambiente:

O papel deste agente é o de organizar e manter as informações do ambiente web coerentes com as necessidades do desenvolvedor de objetos de aprendizagem. Este agente tem sua percepção referenciada ao usuário do sistema, quando o sistema é acessado pelo desenvolvedor o agente percebe que o sistema esta sendo utilizado e tem inicio a sua ação de montar a interface com as informações e campos de preenchimento necessário para tipo de usuário do sistema. O agente tem seu plano fixado em tornar a interação “homem/máquina” simples e amigável para qualquer tipo de usuário que possa operar o sistema.

Agente Gerenciador de Conteúdo:

Este agente tem seu papel em recolher e validar o maior número de informações, fornecidas pelo agente “Editor de AO" para que seja possível a geração do OA requisitado e também para manter os requisitos de objetos de aprendizagem suportados pelo OBAA MILOS. Este agente tem sua percepção no inicio das operações do sistema pelo desenvolvedor de OA, e suas ações estão em coletar o maior número de informações e as distribuir para o restante dos agentes do sistema para que todos estes possam concluir seus desejos.

Agente Editor de OA:

Seu papel no sistema é o de alimentar as informações necessárias para o agente “Gerenciador de Conteúdo”, e coletar tais informações do desenvolvedor para atender as demandas provenientes do agente. Sua percepção é em relação à operação inicial do sistema, quando o desenvolvedor executar o sistema este agente inicia sua atividade. O plano deste

agente é somente manter as informações provenientes do desenvolvedor em sincronismo com o agente de “Gerenciamento de Conteúdo”.

Agente Wizard:

O papel do agente Wizard é de auxiliar o desenvolvedor de OA a executar a autoria de um objeto de aprendizagem, podendo ou não o usuário ter o conhecimento da estrutura de um OA. Este agente possui três percepções distintas, onde estas percepções são referentes ao tipo de desenvolvedor de OA que esta utilizando o sistema no momento. As percepções são referentes ao grau de conhecimento do usuário em relação aos OA, estas estão classificadas em três tipos, Iniciante, Intermediário e Expert. Quando este agente possui sua ação proveniente da percepção de que o desenvolvedor de OA é do nível Iniciante, o seu desejo principal se torna o de disponibilizar todos os auxílios e informações que conhece para que o desenvolvedor consiga elaborar um novo OA, o objetivo neste momento é de que o resultado final seja a criação de um OA pelo desenvolvedor Iniciante.

No momento em que a ação deste agente é proveniente da percepção de que o desenvolvedor possui um conhecimento Intermediário de OA, seu desejo principal continua sendo o de ter como resultado final um novo objeto de aprendizagem, porém a interação do agente com o desenvolvedor terá uma incidência maior. Já na ação gerada pela percepção de que o usuário é de nível Expert o agente possuirá o desejo de receber o maior número de informações a serem respondidas pelo desenvolvedor, para que seu objetivo principal seja atingido, a autoria de OA.

Este agente dispara a ação do agente de Autoria de Conteúdo, e troca de informações com agente de autoria para que se atinja o objeto desejado pelo desenvolvedor de OA. O agente Wizard deve ser capaz de trocar informações com os demais sistemas de agentes da infraestrutura MILOS. Após atingir seu objetivo este agente entrega o conteúdo ao usuário, e aguarda novas instruções.

Agente de Autoria de Conteúdo:

O papel deste agente é auxiliar o agente Wizard e gerenciar os OA gerados. Sua percepção esta dividida em duas percepções distintas no sistema. Onde a primeira percepção é em relação ao tipo de objeto de aprendizagem que o desenvolvedor deseja realizar a autoria e alimenta esta informação no sistema, a ação proveniente da percepção do tipo de OA, é de preparar as informações necessárias do sistema para o agente Wizard utilizar em seu processo. O desejo do agente neste momento é de prover as informações necessárias para auxiliar o agente de autoria a completar seu desejo.

A segunda percepção diz respeito ao gerenciamento do OA, de posse da informação do tipo de objeto de aprendizagem, sua ação será a de organizar o armazenamento, a distribuição do endereçamento correto do novo OA criado. Seu desejo é de preencher as informações necessárias para que se tenha um mapeamento correto do armazenamento do OA.

Seu objetivo principal é dar suporte ao agente Wizard na autoria de objetos, para a realização deste objetivo este agente deve converter ou empacotar para formato SCORM e também desempacotar este formato. Este agente recebe uma mensagem do desenvolvedor de objetos de aprendizagem informado as necessidades para o objeto que deve ser criado, de posse destas informações sua percepção esta completa e este inicia processos internos para alcançar seu objetivo. Desta maneira o agente de autoria inicia seu processo de construção das informações do OA, trocando informações com o agente de conversão de formatos, para que se tenha um OA dentro dos padrões do projeto. Quando este agente atinge seu objetivo, as informações do objeto resultante são guardadas em uma base dados local para futuras

consultas. As informações do objeto são entregues ao agente Wizard para que este publice o objeto.

Agente de Conversão de Formatos:

A percepção deste agente é em relação ao tipo de OA que será gerado. Sua ação é de fornecer subsídios aos outros agentes para que estes possam realizar seus desejos. Uma vez tendo a sua ação concreta o agente inicia o processo para alcançar seu objetivo principal de prover informações corretas e concretas para os outros agentes.

Este agente possui seus objetivos fixados em ajudar o agente de autoria a atingir seus objetivos.

1.3 Casos de Uso do Sistema de Autoria

As elipses apresentadas no diagrama de casos de uso da Figura 1.1 representam os casos de uso do sistema de onde são especificamente dois casos de uso, em um primeiro momento com maior relevância para o projeto.

Caso de uso "*Verificar tipo de OA (padrão SCORM suportado OBAA)"*:

Informa aos agentes wizard e o agente de autoria de conteúdo quais os tipos de objetos de aprendizagem suportados pelo projeto. Disponibilizará aos agentes informações relativas à conversão entre os diferentes tipos de objetos de aprendizagem, ou seja, a possibilidade do agente converter um objeto de aprendizagem com o formato original (plataforma PC) para um dispositivo móvel, por exemplo, ou simplesmente utilizar seu conteúdo em um novo OA com características e extensão diferentes do objeto "original".

Caso de uso "*Localização e Armazenamento de OA":*

Sua finalidade será de auxilio ao agente de autoria com informações sobre o armazenamento dos OA, a localização do objeto dentro da base de dados. Este caso de uso possuirá vínculos com os OA originais e os OA novos, onde estes podem ser criados utilizando o conteúdo de outros objetos ou simplesmente alterando o seu conteúdo. Este caso também terá contato com o ator Editor de OA recebendo as informações fornecidas pelo desenvolvedor de OA referentes às características dos objetos de aprendizagem que serão criados. Disponibilizará ao Editor de OA a localização do novo objeto de aprendizagem dentro da base de dados do sistema de autoria. Também em uma eventual recuperação de objetos de aprendizagem por parte do desenvolvedor, a informação sobre o objeto desejado será fornecida por este elemento no sistema.

Caso de uso "*Informações para auxílio a Autoria de Conteúdo"*:

Responsável por fornecer subsídios ao agente de conversão de formatos a atingir seu principal objetivo. Este caso de uso herda funções e características de sua hierarquia de usos de casos, onde estas ajudam o agente de conversão de formatos a manipular e criar novos OA.

Caso de uso "*Edição de Documentos*":

Sua finalidade será de prestar assistência ao uso de caso "Edição de Conteúdo" com informações sobre os OA em formatos de documentos do tipo arquivos de texto (suporte a extensões doc, docx, pdf, odt). Também auxiliar o agente de conversão de formatos em

eventuais buscas de informações ou análises de conteúdo do OA corrente utilizado pelo sistema de autoria.

Caso de uso “Edição de Apresentação”:

Sua característica é de fornecer ao uso de caso “Edição de Conteúdo” herança de funções que tratem e entendam OA em formatos de documentos do tipo arquivos de apresentação com suporte a extensões do tipo ppt, pptx para documentos criados com Microsoft e odp para documentos criados com a ferramenta OpenOffice. O agente de conversão de formatos utilizará este caso de uso para pesquisas e busca de informações sobre os OA que esta sendo utilizado pelo sistema de autoria no momento.

Caso de uso *“Edição Multimídia”:*

Sua finalidade é de fornecer informações e características dos OA do tipo imagem, vídeo ou áudio para o Uso de caso “Edição de conteúdo” e também o agente de conversão de formatos poderá fazer uso de suas informações ou funções primarias. As extensões suportadas por este caso de uso para OA do tipo imagem serão BMP, JPG e PNG, para os OA do tipo vídeo serão suportadas as extensões FLV e MPG4, já os OA do tipo áudio suportados por este uso de caso serão, MP3, WAV e MP4.

2 Aplicação METAMODELO no Sistema Multiagente para Autoria de AO

Seguindo o proposito deste artigo, a aplicação do METAMODELO UML para sistemas multiagentes descrito e proposto por GUEDES; GILLEANES, 2011. Os Agent Role Actors possuem semelhanças aos atores UML, porém estes ao invés de representar entidades externas representam os papéis dos agentes que irão compor o sistema. O papel dos agentes no sistema são papéis que estes assumem no sistema, um agente pode assumir mais de um papel no sistema ou possuir um único papel.

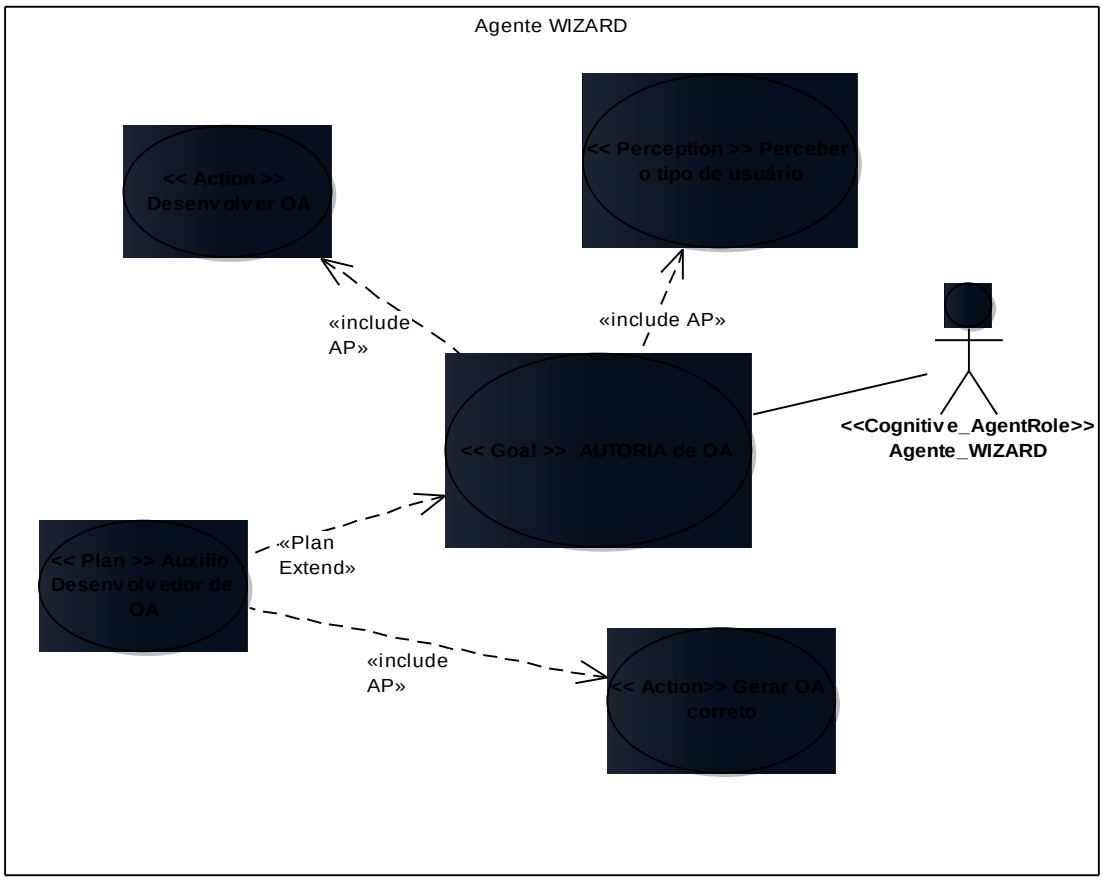


Figura 2.1 – Papel Agente *Wizard*

Na figura 2.1 é apresentado o METAMODELO Agent Role Actors para o agente Wizard do sistema de autoria de OA. Este agente exerce uma função de grande importância para o funcionamento do sistema, este é responsável pela interpretação das necessidades do desenvolvedor de OA e pelo correto funcionamento do sistema de autoria.

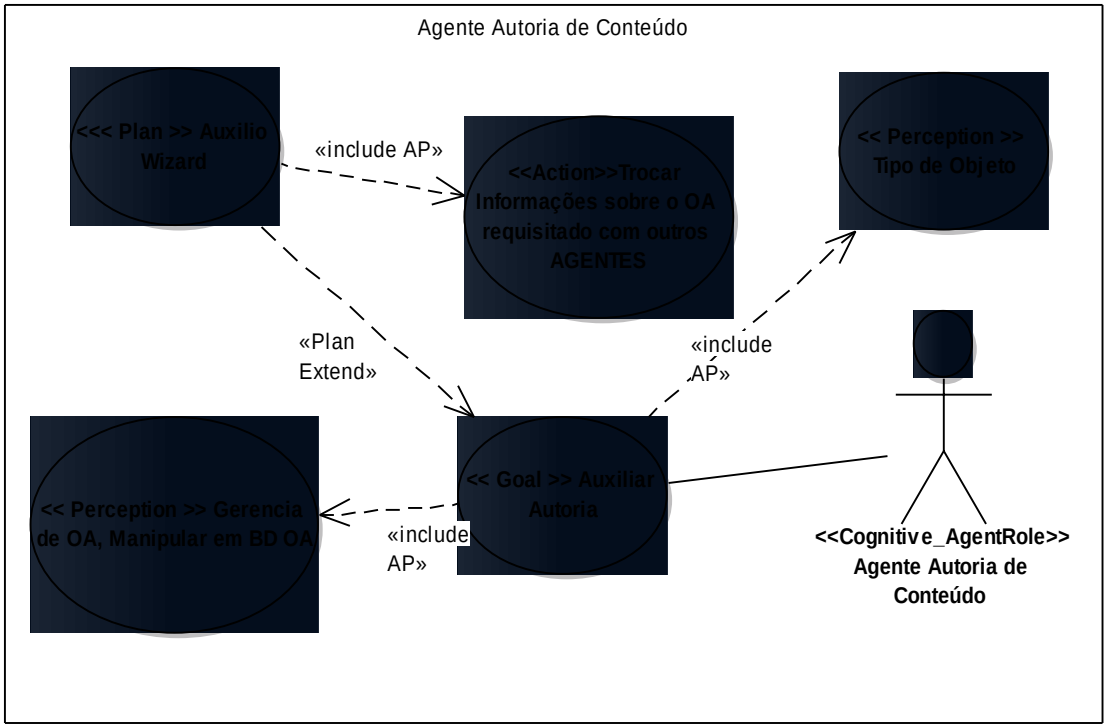


Figura 2.2 – Papel Agente de Autoria de Conteúdo

Na figura 2.2 é mostrada a estrutura de características do agente de Autoria de Objetos de Aprendizagem. Este agente possui interação com a base de dados para o armazenamento correto dos Objetos de Aprendizagem gerados.

Seguindo as necessidades e premissas do projeto MILOS, como extensões e tipos de Objetos de Aprendizagem suportados, o agente de Autoria de Conteúdo deve possuir seu papel no sistema unido e sincronizado com as premissas do OBAA MILOS.

O agente de Conversão de Formatos possui seu papel no sistema, de provedor de dados e informações aos demais agentes referentes às características necessárias para cada Objeto de Aprendizagem que esta sendo ou será manipulado pelo desenvolvedor de OA. Mantendo a operação do sistema de acordo com os tipos de Objetos de Aprendizagem suportados pelo projeto OBAA MILOS.

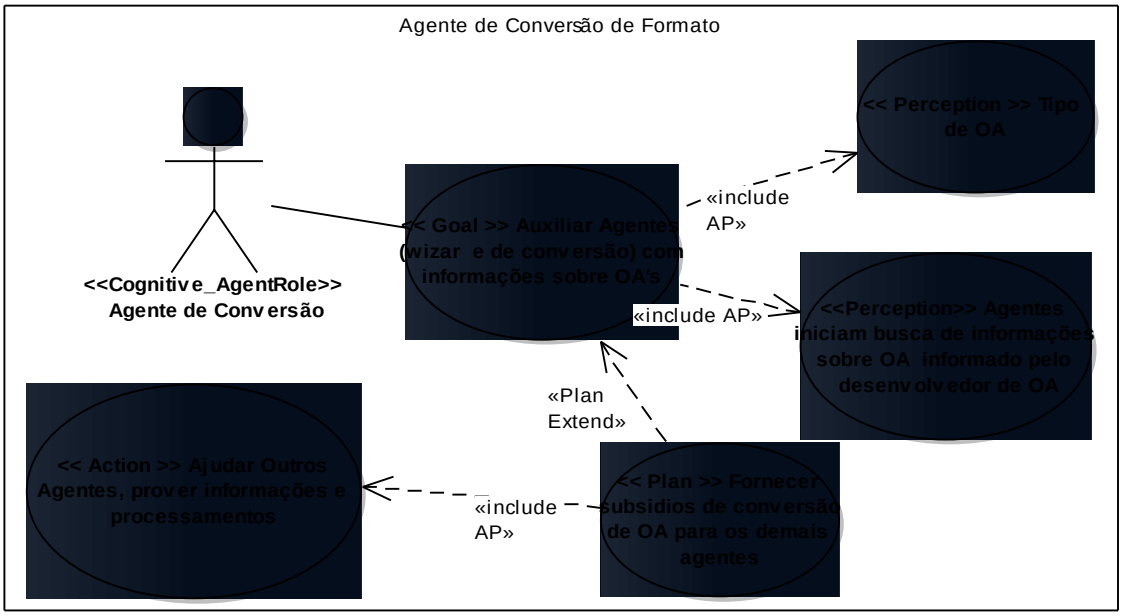


Figura 2.3 – Papel Agente de Conversão de OAs

Na figura 2.3 é mostrada a estrutura de características do agente de Conversão de tipos de Objetos de Aprendizagem. Este agente é responsável por garantir que os tipos e extinções de OA utilizados pelo sistema obedeçam às premissas do projeto MILOS.

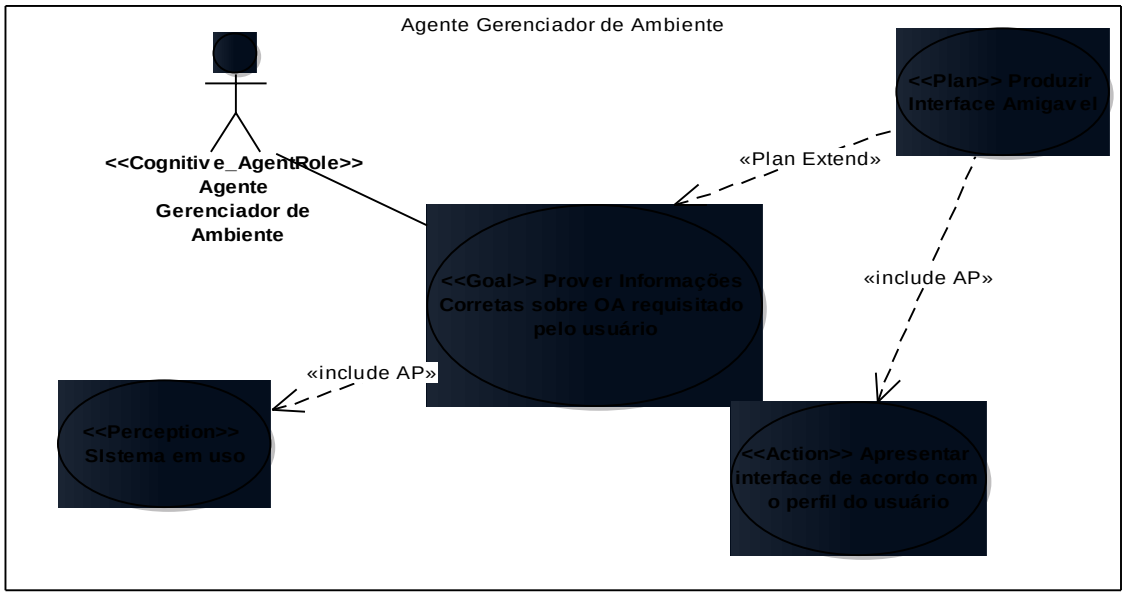


Figura 2.4 – Papel Agente Gerenciador de Ambiente

Na figura 2.4 é mostrada a estrutura de características do agente Gerenciador de Ambiente, onde este agente tem sua tarefa principal de criar uma interface amigável para o desenvolvedor, respeitando a característica deste (inicial, intermediário, expert).

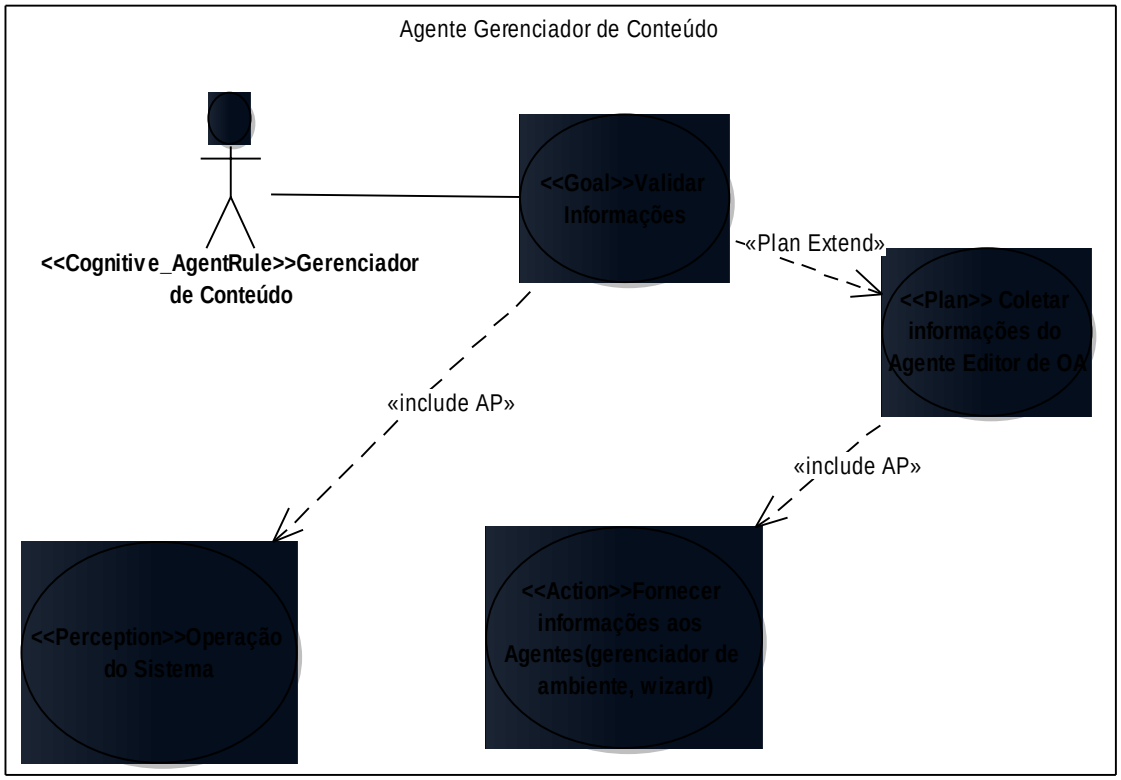


Figura 2.5 – Papel Agente Gerenciador de Conteúdo

A figura 2.5 mostra as características do agente Gerenciador de Conteúdo, onde este agente tem sua principal função em fornecer informações coerentes e de quantidade suficiente para os agentes da “retaguarda” do sistema, onde estes agentes é que irão gerar o objeto de aprendizagem definitivo.

Na figura 2.6 são representadas as características do agente Editor de OA, este agente tem sua principal tarefa a interação e coleta de informações do desenvolvedor de OA's para que estas informações e características sejam passadas para o próximo agente que irá alimentar todos os quesitos e necessidades do sistema para gerar o Objeto de Aprendizagem final.

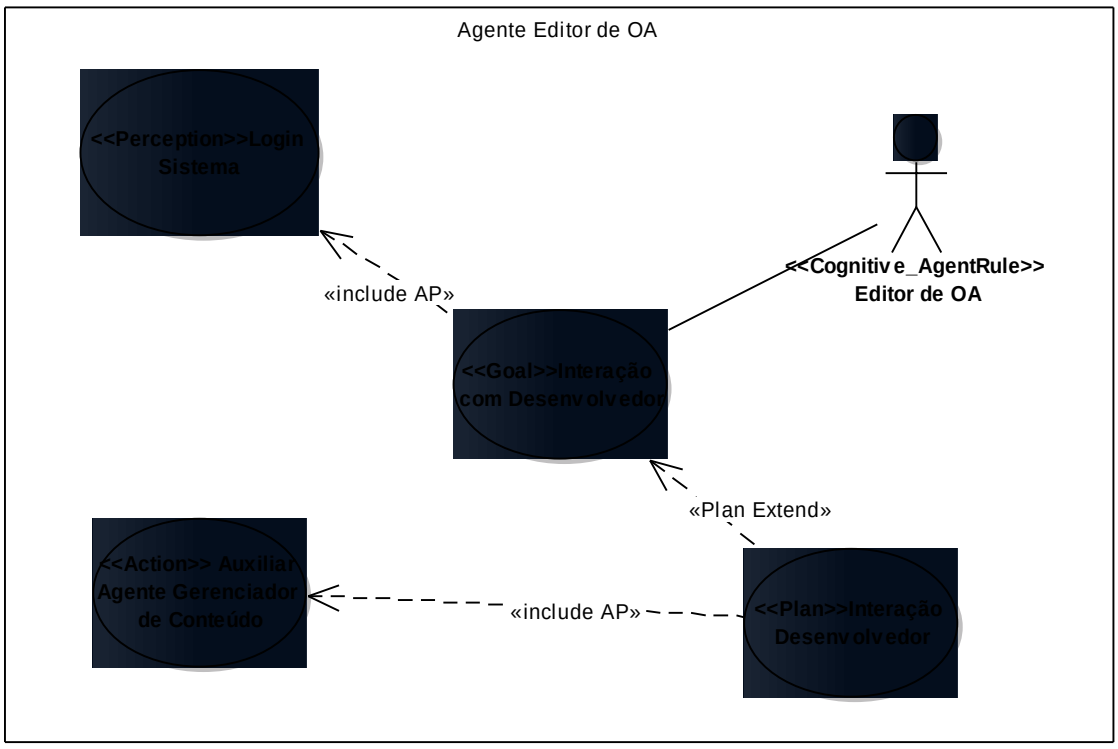


Figura 2.6 – Papel Agente Editor de AO

3 COMENTÁRIOS E OPNIÃO SOBRE APLICAÇÃO DO METAMODELO SUGERIDO

O presente trabalho expos a aplicação do “Metamodelo UML para a Modelagem de Requisitos em Projetos de Sistemas MultiAgentes”, apresentado para tese de Doutorado por Gilleanes Thorwald Araujo Guedes. A modelagem dos agentes para uma apresentação inicial do “Sistema para Autorita de Objetos de Aprendizagem” foi como AgentRole_Actors empregando estereótipos para cada agente do sistema.

O metamodelo sugerido é de fácil compreensão para com os tipos de agentes a serem utilizado no sistema, tendo a rápida identificação do tipo do agente, onde este será cognitivo ou reativo. Para a descrição das caracterizações dos agentes como plano, objetivos, intenções, desejos, percepções, etc... o metamodelo se mostra satisfatório para as necessidades de descrição de tais características por sistemas UML. Onde após uma análise mais aprofundada dos agentes do sistema modelado com AgentRole_Actors é possível compreender características como; qual o plano adotado pelo agente juntamente com sua ação para atingir o seu objetivo maior. A possibilidade de demonstrar todas as percepções da entidade, para que se possa ter certeza de que esta entidade é um agente dentro do sistema com a capacidade de tomar decisões e agir por conta própria.

Em relação a dificuldades de utilização do metamodelo, a nova terminologia para descrever as características dos agentes e suas capacidades, exigindo maior atenção para a construção de sistema multiagentes utilizando AgentRole_Actors, pois o conhecimento em UML 2.1 é mais trivial para descrição de sistemas.

Acredito que este metamodelo possa ser utilizado por outros desenvolvedores com um maior conhecimento na área de agentes para descrever sistemas multiagente complexos. Meu conhecimento é de nível intermediário em relação a sistemas multiagente e diagramas UML 2.1, e tive facilidade em descrever o sistema descrito neste trabalho aplicando o metamodelo proposto por Gilleanes T. A. Guedes em sua tese, com ressalva a terminologia aplicada que não era de meu domínio.

REFERÊNCIAS BIBLIOGRAFICAS

ADL. SCORM Overview. 2001. Acessado em 02/2010, disponível em, http://www.doncio.navy.mil/elearning/downloads/231_SCORM_1.2_Overview.doc.

ALLY, M. et. al. An Intelligent Agent for Adapting and Delivering Eletronic Course Materials to Mobile Learners. 2005.

BAUER, B.; MULLER, J.; ODELL, J. Agent UML: A formalism for specifying multiagent software systems. Int. J. Softw. Eng. Knowl, 2001.207--230.

GUEDES, T. A.; Gilleanes, UML 2.0 Uma Abordagem Prática. 2011.2ª edição NOVATEC.

BERGENTI, F.; POGGI, A. Exploiting UML in the Design of Multi-agent Systems. 2000. Proceedings of the First International Workshop on Engineering Societies in the Agent World: Revised Papers, p.106-113.

CETIC. TIC DOMICÍLIOS e USUÁRIOS 2008 - TOTAL BRASIL. 2008. Acessado em 11/2009, disponível em http://www.cetic.br/usuarios/tic/2008-total-brasil/index.htm.

CHANG, E. et. al. Use of UML 2.1 to Model Multi-Agent Systems based on a Goal-driven Software Engineering Ontology. 2008. Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid. p. 428 – 432.

IEEE, LTSC. Draft Standard for Learning Object Metadata, IEEE Standard 1484.12.1. 2002. Nova York, Institute of Electrical and Electronics Engineers. Acessado em 09/2009, disponível em: http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf.

IM, S.; et. al. Design and Implementation of SCORM Content Conversion for DiTV. 2006. Em Proceedings of the 2006 Internetional Conference on Hybrid Information Technology.

ISO, IEC. Norma ISO/IEC 15938-9:2005, 2005, Information technology -- Multimedia content description interface -- Part 9: Profiles and levels. 2005. Acessado em 03/2010, disponível em http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40392.

JENNINGS, N.; SYCARA, K.; WOOLDRIDGE, M. A Roadmap of Agent Research and Development. 1998. Em Journal Autonomous Agents and Multi-Agent Systems, Holanda.

KNIBERG, H. Scrum e XP Direto das Trincheiras: como nós fazemos Scrum. 2008. InfoQ.

LAAKO, T.; HILTUNEN, T. Adapting Web Content to Mobile User Agents. 2005. IEEE Internet Computing, Volume 9. Número 2. p.46-53.

LIND, J. Agent Oriented Software Engineering. 2006. Acessado em 06/2009, disponível em http://www.agentlab.de/aose.html.

MCCLELLAND, M. Metadata Standards for Educational Resources. 2003. IEEE Computer.

MILOS OBAA. Infraestrutura de Agentes. 2010. Acessado em 05/2010, disponível em http://obaa.unisinos.br/?pg=infraestrutura_de_agentes.html.

OBAA. Recomendações de Interoperabilidade. 2010. Acessado em 04/2010, disponível em http://www.portalobaa.org/obaac/padrao-obaa/recomendacoes-de-interoperabilidade.

ODELL, J. et al. Extending UML for Agents, em Proceedings of the Agent-Oriented Information Systems Workshop. 2000. Austin. Em 17th National conference on Artificial Intelligence.

POLSANI, P. Use and Abuse of Reusable Learning Objects. 2003. Em Journal of Digital Information, Volume 3. Número 4.

RAO, A.; GEORGEFF, M. BDI Agents: From Theory to Practice. 2003. Acessado em 05/06/2009, disponível em http://www.agent.ai/doc/upload/200302/rao95.pdf.

SACRINI, M. O uso da televisão digital no contexto educativo. 2008. Acessado em 10/2009, disponível em http://www.fae.unicamp.br/etd/include/getdoc.php?id=859&article=299&mode=pdf.

VICCARI, R. et al. Projeto OBAA - Relatório Técnico RT-OBAA-01 - Proposta de Padrão para Metadados de Objetos de Aprendizagem Multiplataforma. 2009a. UFRGS/CINTED. Acessado em 12/2009, disponível em: http://www.portalobaa.org/obaac/padrao-obaa/relatorios-tecnicos/RT-OBAA-01.pdf/view.

VICCARI, R. et al. Projeto OBAA - Relatório Técnico RT-OBAA-02 – Grupo Video e Áudio – Mecanismos para criação de conteúdo interoperável entre Web, TV Digital e Móveis, Impacto de diferentes modelos de metadados na integração de tecnologias para Web, TV Digital e Móvel. 2009b. UFRGS/CINTED. Acessado em 12/2009, disponível em http://www.portalobaa.org/obaac/padrao-obaa/relatorios-tecnicos/013_RT-OBAA-02.pdf/view.

VICCARI, R. et al. The OBAA Proposal for Learning Objects Supported by Agents. 2010. Acessado em 06/2010, disponível em http://www.portalobaa.org/obaac/padrao-obaa/artigos-publicados/aamas2010_FINAL.pdf/at_download/file

WILEY, David A. Connecting Learning Objects to Instructional Design Theory: A Definition, A Metaphor, and A Taxonomy. 2001. Em Wiley, David A. (DOC), The Instructional Use of Learning Objects: Online Version. Acessado em 06/2009, disponível em http://reusability.org/read/chapters/wiley.doc.

WOOLDRIDGE, M.; JENNINGS, N. Agent Theories, Architectures, and Languages: A Survey. 1994. Amsterdam, Holanda. Em Workshop on Agent Theories, Architectures and Languages. p. 1-32.

WOOLDRIDGE, M. An Introduction to MultiAgent Systems. 2002. John Wiley & Sons Ltd, paperback.

KLINE, Paul (1986). A handbook of test construction: introduction to psychometric design. New York: Methuen & Co.