UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

Uma Ferramenta Gráfica de Apoio à Metodologia Prometheus AEOlus e à Geração Automática de Código para o Ambiente de Desenvolvimento Jason

Rafhael Rodrigues Cunha

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Computação

Orientador: Profª. Drª. Diana Francisca Adamatti
Co-orientador: Prof. Dr. Cléo Zanella Billa

Rio Grande, 2016

Ficha catalográfica

C972f Cunha, Rafhael Rodrigues.
Uma ferramenta gráfica de apoio à metodologia Prometheus AEOlus e à geração automática de código para o ambiente de desenvolvimento de Jason / Rafhael Rodrigues Cunha. – 2016.
146 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-graduação em Engenharia de Computação, Rio Grande/RS, 2016.
Orientadora: Drª. Diana Francisca Adamatti.
Coorientador: Dr. Cléo Zanella Billa.

1. Engenharia de software 2. Orientação a agentes 3. Sistema multiagente 4. Arquitetura BDI 5. Framework I. Adamatti, Diana Francisca II. Billa, Cléo Zanella III. Título.

CDU 004.41

Catalogação na Fonte: Bibliotecário Me. João Paulo Borges da Silveira CRB 10/2130

UNIVERSIDADE FEDERAL DO RIO GRANDE
Centro de Ciências Computacionais
Programa da Pós-Graduação em Computação
Curso de Mestrado em Engenharia de Computação

ATA DE SESSÃO DE DEFESA DE DISSERTAÇÃO DE MESTRADO

Ata No. ___/2016

Na data de 07 de abril de 2016, às 14:00 horas, ocorreu a Sessão de Defesa de Dissertação de Mestrado de Rafhael Rodrigues Cunha, que apresentou a dissertação intitulada "Uma Ferramenta Gráfica de Apoio a Metodologia à Prometheus AEOlus e Geração Automática de Código para o Ambiente de Desenvolvimento Jason", realizada sob a orientação do Profa. Diana Francisca Adamatti e coorientação do Prof. Cleo Zanella Billa. A banca examinadora foi constituída pelos Profs. Viviane Torres da Silva (Universidade Federal Fluminense), Antonio Carlos da Rocha Costa (FURG) e Eduardo Nunes Borges (FURG), sob a presidência do orientador. Após a apresentação do trabalho, a banca arguiu o candidato e, a seguir, deliberou pela

(x) aprovação da Dissertação
( ) aprovação da Dissertação, sugerindo modificações no texto
( ) reprovação da Dissertação

Rio Grande, 07 de Abril de 2016.

Profa. Dra. Viviane Torres da Silva

Prof. Dr. Antonio Carlos da Rocha Costa

Prof. Dr. Eduardo Nunes Borges

Profa. Dra. Diana Francisca Adamatti
Orientadora

Prof. Dr. Cleo Zanella Billa
Coorientador

Dedico este trabalho a Maria Aparecida Rodrigues Berriel, por todo o apoio e carinho prestado a mim durante toda a minha vida.

AGRADECIMENTOS

Primeiramente agradeço a Deus pela oportunidade e por conduzir-me pelo caminho certo da vida.

A minha orientadora, profª Diana F. Adamatti e meu orientador, profº Cléo Z. Billa, pela confiança depositada, pela oportunidade de trabalhar em conjunto com vocês, e por todo o apoio e incentivo prestado desde o início dessa jornada.

A Daniela Mª. Uez, por autorizar a extensão de sua dissertação de mestrado, compartilhar diversos materiais e sempre estar disposta a sanar dúvidas decorrentes do desenvolvimento do trabalho.

A minha mãe, Maria Aª R. Berriel, pelo carinho, entendimento, apoio e cuidado prestado no decorrer desse trabalho. Sempre será meu alicerce e o meu melhor exemplo de superação e perseverança.

A minha irmã, Andressa R. Cunha, por sempre estar presente e se fazer presente. Juntos seremos mais fortes.

Ao meu pai, José Gustavo L. Cunha, pelo apoio, carinho e conselhos. Nossas conversas no decorrer deste trabalho me motivaram a nunca desistir dos meus objetivos.

Ao meu tio, Luis A. R. Rodrigues, pelo incentivo a ingressar na área, pelo carinho depositado desde meu nascimento e por ser o meu melhor exemplo de humildade. Sempre vou me espelhar em ti.

Ao meu amigo, Gabriel B. Moro, por todo o apoio prestado durante o desenvolvimento deste trabalho. Muitas madrugadas de aprendizado, muita persistência e principalmente muito companheirismo. Dedico a ti também à conclusão desta pesquisa.

Aos meus irmãos da vida, Marcos Fogaça, Claudiane Campagner e Eduardo Moreira. Sei que poderei contar com vocês independente da circunstância e ter vocês junto a mim sempre será um grande prazer. Obrigado também por entenderem os vários momentos de ausência e compartilharem comigo diversos outros de alegria.

Aos meus avós, Leda M. Cunha e José J. Cunha, por me acolherem nas últimas férias, me tratarem com todo o carinho do mundo e me incentivarem a nunca desistir. Saibam que vocês são grandes exemplos em minha vida.

Aos meus colegas do Instituto Federal Farroupilha - Campus Alegrete, pelo apoio e aprendizado durante a realização deste trabalho.

A uma pessoa muito especial, J.M.K, por não medir esforços em me roubar sorrisos e sempre estar preocupada comigo. Desde que tu reapareceste em minha vida, tornou-a mais feliz.

Aos amigos João Victor Guinelli e Carlos Eduardo Pantoja pelo apoio, esclarecimento de dúvidas e incentivo durante a realização deste trabalho.

A todos os meus amigos e familiares que me apoiaram e contribuíram direta ou indiretamente para a conclusão deste trabalho.

A Universidade Federal do Rio Grande (FURG) e a Fundação de amparo e Pesquisa do Rio Grande do Sul (FAPERGS) pelo apoio financeiro para a realização desta pesquisa.

No que diz respeito ao desempenho, ao comprometimento,
ao esforço, à dedicação, não existe meio termo.
Ou você faz uma coisa bem-feita ou não faz.
— AYRTON SENNA

RESUMO

CUNHA, Rafhael Rodrigues. **Uma Ferramenta Gráfica de Apoio à Metodologia Prometheus AEOlus e à Geração Automática de Código para o Ambiente de Desenvolvimento Jason**. 2016. 146 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande.

A Engenharia de Software (ES) é uma área de engenharia que busca a construção de softwares com qualidade, utilizando métodos e respeitando prazos. Contudo, com a ascensão do mercado, suas técnicas tradicionais não supriram a demanda de novas tecnologias provenientes de um mercado em constante atualização, originando uma subárea, denominada *Agent Oriented Software Engineering* (AOSE). A AOSE busca encontrar soluções para aspectos relacionados ao desenvolvimento de sistemas orientados a agentes. Entretanto, ainda não existe uma padronização para a subárea, resultando em diversas metodologias desenvolvidas por motivos distintos. Além disso, outro fator predominante para a instabilidade dessa subárea, consiste em suas ferramentas de apoio serem limitadas no processo de geração automática de códigos para plataformas específicas de desenvolvimento multiagente. O intuito principal deste trabalho é desenvolver um *plug-in* para apoiar a metodologia Prometheus AEOLus, permitindo que o usuário desenvolva quaisquer diagramas presentes na especificação da metodologia. Adicionalmente, como objetivo secundário, buscou-se a criação de um mecanismo capaz de percorrer todas as informações definidas pelo usuário e realizar a geração automática de códigos para a linguagem *AgentSpeak*, que é aderente a plataforma de desenvolvimento Jason. A ferramenta proposta apresentou 75% de cobertura para testes unitários e de usuários, o que a valida como uma nova alternativa para desenvolvimento de sistemas multiagente.

Palavras-chave: Engenharia de Software Orientada a Agentes, Sistema Multiagente, Arquitetura BDI, Framework JaCaMo.

ABSTRACT

CUNHA, Rafhael Rodrigues. **A Graphic Tool to Support the Prometheus AEOlus Methodology and Automatic Code Generation for Jason Development Environment**. 2016. 146 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande.

Software Engineering (SE) is an engineering area in order to develop high quality software, using well-defined methods and according to deadlines. However, based on the rise of the software market, its traditional techniques not supplied the demand for new technologies arising from the constant updating of such market, resulting in a subarea called Agent-Oriented Software Engineering (AOSE). On the one hand, AOSE aims to find solutions to issues related to the agent-oriented systems development. On the other hand, there is still no standardization in this subarea, boosting the design of several methodologies, for distinct reasons. Furthermore, other predominant factor for the instability in this subarea consists of that its support tools are limited in the automatic code generation process for specific multi-agent development platforms. The main goal of this work is to develop a plugin to support the Prometheus AEOLus methodology, allowing the user to develop any diagrams presented in the methodology specification. In addition, a secondary goal is the development of a mechanism able to browse through all the user-defined information and perform automatic code generation for AgentSpeak language, which is an adherent language for Jason development platform. The proposed tool reached 75% accuracy for unit and user testing, which validates it as a new alternative for the multi-agent systems development.

Keywords: Agent-Oriented Software Engineering, BDI Architecture, JaCaMo Framework, Multi-Agent System.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A&A	Agentes e Artefatos
AOSE	*Agent Oriented Software Engeering*
API	*Application Pogramming Interface*
BDI	*Beliefs, Desires e Intentions*
CMS	*Conference Management System*
DSL	*Domain Specific Language*
EMF	*Eclipse Modelling Framework*
IA	Inteligência Artificial
IDK	*Ingenias Development Kit*
GEF	*Graphical Editing Framework*
GMF	*Graphical Modelling Framework*
JADE	*Java Agent Development Framework*
MaSE	*Multiagent Systems Engineering*
MDA	*Model Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model-Driven-Engineering*
MDI	*Model-Driven Integration*
MOF	*Meta-Object Facility*
KQML	*Knowledge Query and Manipulation Language*
OA	Orientada a Agentes
PASSI	*Process for Agent Societies Specification and Implementation*
PDE	*Plug-in Development Environment*
PDT	*Prometheus Design Tools*
PSM	*Plataform Specific Model*
QVT	*Query View Transformation*
SGML	Subconjunto da Norma Generalizadas *Markup Language*

SMA Sistema Multiagente

SPE *Software Process Engineering*

SWT *Standard Widget Toolkit*

TuCSoN *Tuple Centre Spread over the Network*

UML Linguagem de Modelagem Unificada

XMI *XML Metadata Interchange*

XML *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

No decorrer dos anos, a área de engenharia de software tem procurado definir processos de desenvolvimento de software e linguagens de modelagem que visam estabelecer etapas bem definidas para a construção de um software. Este fato tem como propósito tornar a produção de um software mais robusta, rápida, organizada, confiável e de fácil manutenção e reutilização (GUEDES, 2012).

Na área de inteligência artificial, o paradigma orientado a agentes tem sido pesquisado e utilizado para minimizar a complexidade e aumentar a eficiência de softwares distribuídos. Esta prática tem-se mostrado eficiente para a construção de softwares com essas características, incentivando um aumento no desenvolvimento de Sistemas Multiagente (SMA) (GUEDES, 2012).

Entretanto, devido à complexidade e distribuição desse tipo de sistema, novos desafios para a área de engenharia de software tradicional foram encontrados. Esses desafios ocorreram em virtude de metodologias conceituadas para se fazer modelagem na área, como *Unified Modelling Language* (UML), não suprirem as características presentes neste grupo de sistemas. Essa ausência levou ao surgimento de uma sub-divisão da área que mescla conceitos de engenharia de software e inteligência artificial, chamada de *Agent Oriented Software Engineering* (AOSE) ou Engenharia de Software Orientada a Agentes. Seus objetivos principais são propor métodos e linguagens/notações para projetar e modelar softwares orientados a agentes (GUEDES, 2012).

Dentro desse contexto, diversas metodologias foram propostas objetivando suprir a demanda de softwares orientados a agentes (PADGHAM; WINIKOFF, 2002)(BRESCIANI et al., 2004)(DELOACH, 1999)(HENDERSON-SELLERS; GIORGINI, 2005)(CAIRE et al., 2002)(COSSENTINO; POTTS, 2002)(WOOLDRIDGE; JENNINGS; KINNY, 2000)(UEZ, 2013). Essas metodologias foram sendo criadas pelos mais variados motivos. Algumas delas basearam-se em melhorias nos diagramas presentes na UML. Outras criaram seus próprios metamodelos e notações para seu uso. Entretanto, até o momento, não existe uma padronização de técnicas/metodologias para serem utilizadas sob quaisquer circunstâncias que possibilitem a utilização de um sistema multiagente.

Outro ponto que se leva em conta na utilização das metodologias AOSE é a possibili-

dade de geração automática de código após a realização da especificação do SMA. Sabe-se que atualmente existem diversas plataformas para desenvolvimento de SMA, como Jade[1], Jason[2] ou Jack[3]. Entretanto, nenhuma das ferramentas disponibilizadas pelas metodologias possibilita que seja desenvolvido *work products*[4] para as diferentes dimensões presentes em um SMA, bem como sua interligação com ferramentas que apoiem essas dimensões.

Dentro desse contexto, surgiu a metodologia Prometheus AEOlus (UEZ, 2013). Esta metodologia foi desenvolvida baseando-se em duas tecnologias: a metodologia Prometheus e o *framework* JaCaMo. A autora complementa afirmando que o propósito da metodologia é fazer uma extensão da metodologia Prometheus de modo a incluir a especificação de Ambiente e Organização na modelagem de sistemas.

1.1 Justificativa

Segundo BRANDÃO (2014), as metodologias existentes até o presente momento para modelar SMA sobre a perspectiva da engenharia de software são as ilustradas pela Figura 1. Nesta figura, os retângulos azuis indicam metodologias e processos OO; magentos com traços cheios indicam metodologias AO que estendem OO; magentas pontilhadas indicam metodologias AO que estendem outras AO; magentas tracejadas usam princípios OO, mas não estendem metodologias existente; violetas indicam que estende OO mas usam também técnicas de IA/ eng. de conhecimento; e Tropos que tem como base a analise de requisitos usando a*.

A escolha das metodologias a serem estudadas neste trabalho, baseou-se na tentativa de explorar ao menos uma metodologia derivada de cada uma das metodologias clássicas, como RUP, OMT, Fusion e Open. Entretanto, Open foi excluída devido a inexistência de documentações que pudessem levar ao seu entendimento. Além disso, procurou-se selecionar uma metodologia que não apresentava nenhuma derivação, foi o caso de Prometheus; também buscou-se uma metodologia que não herdasse características do paradigma OO, como Tropos. Selecionou-se também uma metodologia derivada da metodologia OMT, sendo ela, o MaSE. Por fim, escolheu-se também para estudo neste trabalho a metodologia Ingenias em virtude da mesma ser derivada de uma outra metodologia OO, chamada RUP.

Ao analisar as ferramentas existentes que apoiam as metodologias orientadas a Agentes selecionadas, (PANTOJA; CHOREN, 2013) mencionam que a geração automática de códigos a partir das ferramentas de suporte as metodologias ainda são muito limitadas,

[1] http://jade.tilab.com/

[2] http://jason.sourceforge.net/

[3] http://aosgrp.com/products/jack/

[4] Work Product é o resultado da conclusão de um trabalho que serve como subsídio para alcançar algum objetivo

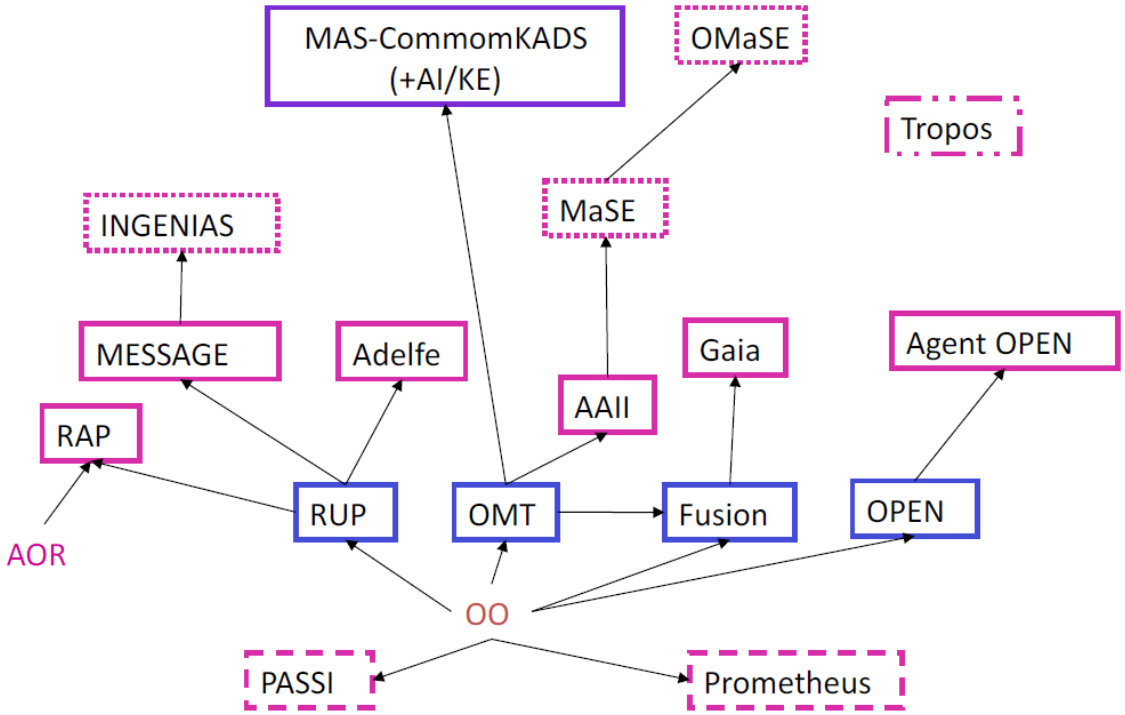


Figura 1: Metodologias AOSE (BRANDÃO, 2014)

muitas vezes ficando totalmente dependentes de conceitos e características das próprias metodologias. Por exemplo, a metodologia Ingenias apresenta uma ferramenta de análise, modelagem e codificação para um SMA que gera código automático a partir da modelagem apenas para a plataforma de desenvolvimento *Java Agent Development Framework* (Jade) (BELLIFEMINE et al., 2005). A metodologia Prometheus, possui um *plug-in* para o Eclipse, chamado PDT Tools, que é utilizado para fazer a modelagem de sistemas especificados através da metodologia Prometheus, entretanto, este *plug-in* limita-se a fazer a geração automática de código para plataforma de desenvolvimento Jack (HOWDEN et al., 2001). A metodologia MaSE, apresenta também um *plug-in* chamado agentTools, entretanto, a documentação da metodologia esclarece que a possibilidade de geração automática de código para plataformas de desenvolvimento será implementada em versões posteriores desse *plug-in*. Por fim, a metodologia Tropos, apresenta também um *plug-in* chamado TAOM4E, mas não é referenciado em sua documentação de que o mesmo tem ligação com alguma plataforma de desenvolvimento.

Com base nesse contexto, a justificativa deste trabalho decorre da inexistência de uma ferramenta gráfica que apoie a metodologia Prometheus AEOlus e, consequentemente, da ausência de uma transformação automática dos *work products* produzidos pela utilização da metodologia em códigos aderentes ao *framework* JaCaMo. Portanto, pretende-se com este trabalho viabilizar uma solução que integre a metodologia a dimensão de agentes do *framework* JaCaMo, assim como o desenvolvimento de uma ferramenta gráfica que a apoie.

Além disso, espera-se que por intermédio de uma ferramenta gráfica de apoio a metodologia Prometheus AEOlus, aumente seu potencial de utilização, difundindo-a de modo que futuramente possa ser uma solução viável de modelagem de um SMA, respeitando todas as dimensões presentes no mesmo e fazendo a integração de maneira automática ao *framework* JaCaMo.

1.2 Objetivos

Este trabalho tem como objetivo desenvolver uma ferramenta de modelagem gráfica que apoie os diversos diagramas presentes na especificação da metodologia Prometheus AEOLus. Adicionalmente, a ferramenta contêm um mecanismo que possibilita a geração automática de códigos para a linguagem *AgentSpeak*, utilizando como referência de informação os *work products* produzidos na etapa de modelagem. Complementarmente, definiu-se os seguintes objetivos específicos:

- Estudar a metodologia Prometheus AEOlus;
- Escolher uma tecnologia para a criação de um editor gráfico que apoie a metodologia Prometheus AEOlus;
- Desenvolver o editor gráfico com base na tecnologia escolhida;
- Desenvolver uma ferramenta geradora de código para a plataforma Jason (linguagem *AgentSpeak*), baseando-se nos diagramas modelados a partir da ferramenta gráfica desenvolvida;
- Realizar testes que provem a eficiência da ferramenta gráfica e o gerador de códigos.

1.3 Organização do Texto

Esta dissertação está dividida em sete Capítulos: no Capítulo 2 são apresentados os principais conceitos referentes à área de Engenharia de Software e de agentes; no Capítulo 3 são apresentados trabalhos que fazem a interligação com o *framework* JaCaMo bem como a metodologia que serviu de base para criação deste trabalho; o Capítulo 4 apresenta as tecnologias utilizadas para o desenvolvimento deste trabalho; no Capítulo 5 são descritos os procedimentos realizados para o desenvolvimento da ferramenta desta dissertação; o Capítulo 6 relata os testes realizados para validar a ferramenta desenvolvida.

Além disso, o Capítulo 7 apresenta as conclusões sobre o trabalho realizado, destacando as suas contribuições, críticas a metodologia Prometheus AEOlus e algumas direções para trabalhos futuros. No Apêndice A são apresentados *work products* gerados durante a modelagem do estudo de caso *Conference Management System*. Já no Apêndice B, são demonstrados os testes realizados nos arquivos com extensão “asl”.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta conceitos base para a compreensão dos tópicos abordados ao longo deste trabalho. Como o objetivo dessa dissertação é o desenvolvimento de uma ferramenta gráfica para apoiar a metodologia Prometheus AEOlus, é necessário que o leitor compreenda conceitos que envolvem a área de SMA para utilizar a ferramenta desenvolvida.

Este Capítulo está dividido como segue: A Seção 2.1 define agentes e apresenta suas principais características; a Seção 2.2 apresenta os sistemas multiagente; a Seção 2.3 descreve o paradigma de programação orientado a agentes; a Seção 2.4 apresenta brevemente a origem e progresso da área de engenharia de software orientada a agentes, bem como algumas metodologias que apoiam a área.

2.1 Agentes

O domínio da Inteligência Artificial (IA) passou por grandes avanços desde a década de 1970. Diversos métodos para resoluções de problemas foram projetados, testados, analisados e utilizados (GARCIA; SICHMAN, 2005). Por trataram-se de métodos distintos, cada um deles gerou sistemas que foram implementados e solucionaram problemas reais significantes. Uma característica comum a esse aglomerado de sistemas é a utilização de uma metáfora da inteligência baseada no comportamento individual do ser humano, levando em consideração como ele pensa, raciocina, toma decisões, planeja, percebe, se comunica ou aprende.

Para GARCIA; SICHMAN (2005), a inteligência não se resume a somente essas características. É importante, ao desenvolver um sistema com essa particularidade, levar em consideração a integração desses conceitos em uma única entidade. Assim, GARCIA; SICHMAN (2005) explica que um sistema integrado com um bom processo de decisão, mas que ativado no momento errado, ou que não consegue integrar-se os resultados de um mecanismo de aprendizagem a um possível planejamento futuro, não pode ser considerado um sistema inteligente. Surgem então os modelos de agentes, procurando suprir as carências encontradas nesses sistemas chamados ”inteligentes”.

Nas palavras de FERBER (1995), um agente é:

> Um agente é uma entidade real ou virtual, capaz de agir num ambiente, de se comunicar com outros agentes, que é movida por um conjunto de inclinações (sejam objetivos individuais a atingir ou uma função de satisfação a otimizar); que possui recursos próprios; que é capaz de perceber seu ambiente (de modo limitado); que dispõe (eventualmente) de uma representação parcial deste ambiente; que possui competência e oferece serviços; que pode eventualmente se reproduzir e cujo o comportamento tende a atingir seus objetivos utilizando as competências e os recursos que dispõe e levando em conta os resultados de suas funções de percepção e comunicação, bem como suas representações internas.

Para WEISS (1999), um agente é comumente descrito como um sistema computacional que está situado em um ambiente, podendo ser virtual ou não, interagindo com este através de sensores e atuadores. Além disso, esse agente é capaz de tomar decisões e executar ações de forma autônoma visando alcançar seus objetivos. Em RUSSELL; NORVIG (2009) é explicado que um agente é qualquer objeto que consegue perceber o ambiente por meio de sensores e atuar sobre ele através de atuadores. Segundo WOOLDRIDGE; JENNINGS (1995), um agente é um sistema de computador situado em um ambiente, e é capaz de realizar ações autônomas em seu ambiente a fim de atingir seus objetivos estabelecidos.

Ao analisar as definições sobre agentes citadas, percebe-se que são complementares e não excludentes. Todos os autores afirmam que o agente está situado em um ambiente; interage com este através de algum meio de comunicação; (WEISS, 1999) e (RUSSELL; NORVIG, 2009) citaram sensores e atuadores, os demais não referenciaram tecnologias; além disso, é consenso entre os autores que os agentes conseguem, a partir de percepções, tomar decisões que visam atingir seus objetivos.

Especificamente, a definição proposta por (FERBER, 1995) deixa clara a utilização de todas as características anteriormente citadas que envolvem sistemas inteligentes. Além disso, esta definição possui um caráter inovador, pois utiliza a noção de agente como um paradigma integrador das técnicas de IA. Entretanto, embora inovadora, essa definição é genérica, pois pode referir-se tanto a um robô móvel como a um robô virtual. O fato de ser um sistema computacional contribui para, de certa forma, restringir o comportamento do agente.

Existem duas categorias básicas para classificar agentes de software. Sendo a primeira, agentes reativos, que a partir de percepções reagem realizando alguma ação; a segunda, os agentes cognitivos, os quais possuem mecanismos que possibilitam racionar, tomar decisões, entre outras características. Sobre os agentes cognitivos, existem algumas arquiteturas que facilitam a modelagem desse tipo de agentes. Uma delas chama-se *Beliefs, Desires e Intentions* (BDI).

A arquitetura BDI, ou arquitetura de crenças, desejos e intenções, é uma forma de modelar o agente pensando em seu estado mental. A fundamentação filosófica para esta concepção de agentes vem dos trabalhos de (DENNETT, 1989) sobre sistemas intencionais e (BRATMAN, 1987) sobre raciocínio prático (BORDINI; VIEIRA, 2003).

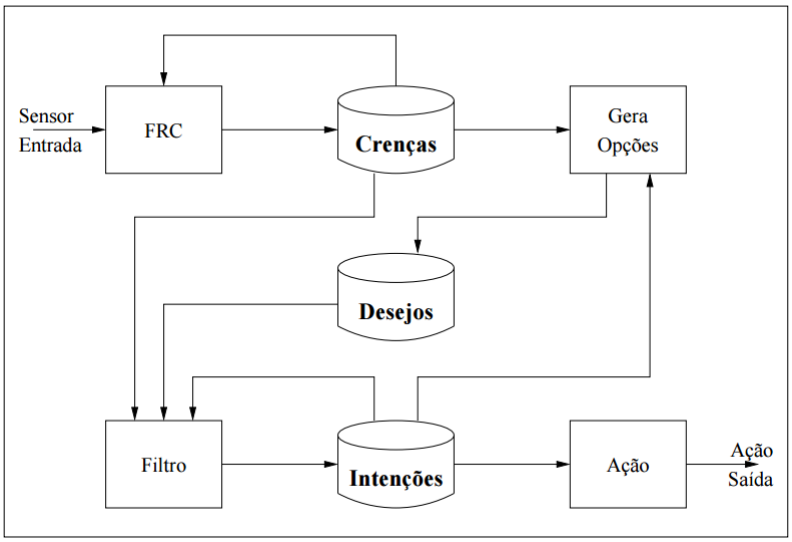


Figura 2: Arquitetura BDI genérica, adaptado de (WEISS, 1999)

A Figura 2 apresenta a arquitetura BDI de uma forma genérica. As *crenças* representam aquilo que o agente sabe sobre o estado do ambiente e dos agentes naquele ambiente (inclusive sobre si próprio). Os *desejos* representam estados do mundo que o agente quer atingir (pensando de outra maneira, são representações daquilo que o agente quer que passe a ser verdadeiro no ambiente no qual ele está atuando) (NUNES, 2007). Segundo BORDINI; VIEIRA (2003), os desejos podem ser contraditórios, pois pode-se desejar coisas que são mutuamente exclusivas no ponto de vista de ação prática. Os *objetivos* são um subconjunto dos desejos que devem ser compatíveis entre si. Já as *intenções* representam uma sequência de ações específicas que um gente se compromete a executar para atingir determinados objetivos.

Conforme BORDINI; VIEIRA (2003), a função de *revisão de crenças* (referenciada na figura por FRC), recebe a informação sensória (percebe propriedades do ambiente) e, consultando as crenças anteriores do agente, atualiza essas crenças para que elas reflitam o novo estado do ambiente. Através dessa nova representação de estado do ambiente, é possível que novas opções de estados a serem atingidos fiquem disponíveis. A função *Gera Opções*, verifica quais as novas alternativas de estados a serem atingidos, que são relevantes para os interesses particulares daqueles agentes (NUNES, 2007). Essa verificação deve ser feita com base no estado atual do mundo, conforme as crenças dos agentes, e nas intenções com que o agente já está comprometido. A atualização dos objetivos então ocorre de duas maneiras: as observações do ambiente possivelmente determinam novos objetivos do agente, e a execução de intenções de mais alto nível pode gerar

a necessidade de que objetivos mais específicos sejam atingidos (BORDINI; VIEIRA, 2003).

Após ser atualizado o conhecimento e a motivação do agente, é preciso decidir que curso de ações específico será usado para alcançar os objetivos atuais do agente. Esse é o papel da função *Filtro* da Figura 2, que atualiza o conjunto de intenções do agente, com base nas crenças e desejos atualizados e nas intenções já existentes (BORDINI; VIEIRA, 2003). Esse processo da função Filtro serve para determinar como atualizar o conjunto de intenções do agente e é normalmente chamado de deliberação.

Segundo BORDINI; VIEIRA (2003), posterior a atualização do conjunto de intenções, a escolha de qual ação específica, entre aquelas pretendidas, será a próxima a ser realizada pelo ambiente, é feita pela função *Ação*. Em casos específicos que não é necessário a priorização entre múltiplas intenções, a escolha pode ser simples, ou seja, basta escolher qualquer uma entre as intenções ativas, desde que seja garantido que todas as intenções terão, em algum momento, a oportunidade de serem escolhidas para serem executadas. Entretanto, há alguns agentes que podem precisar usar escolha de intenções baseadas em critérios mais sofisticados para garantir que determinadas intenções sejam priorizadas em relação a outras em certas circunstâncias.

2.2 Sistemas Multiagente

Para SICHMAN (2003), a definição semântica de Sistema Multiagente (SMA) ainda não existe. O autor explica essa afirmação, através da observação do domínio que este tipo de sistema está incluído. Neste domínio, utiliza-se de resultados, conceitos e práticas de disciplinas tão dispares quanto IA, sociologia, economia, ciências da organização, administração ou filosofia.

Para (BRIOT; DEMAZEAU et al., 2001), um SMA pode ser definido como um conjunto organizado de agentes. Esta organização encarrega-se de definir regras para que os agentes possam conviver em um ambiente comum, dividindo recursos e trabalhando coletivamente. Entretanto, um SMA não está restrito a somente uma organização. Além disso, um agente pode interagir em mais de uma organização, fazendo com que este equilíbrio entre organização e a autonomia dos agente seja um ponto importante no desenvolvimento de um SMA. O ambiente também é parte importante de um SMA, pois além dos agentes, nos ambientes encontram-se também entidades passivas, que podem ser manipuladas pelos agentes.

DEMAZEAU (1995) propõe a divisão de um SMA em quatro componentes principais: o agente, o ambiente, a interação e a organização. São utilizadas as vogais A, E, I e O, respectivamente para representar essas dimensões, fazendo com que essa divisão seja conhecida como paradigma das vogais. Segundo o autor, para que um problema computacional possa ser resolvido, o time de desenvolvimento deve selecionar para cada

dimensão os modelos que são instanciados ou especializados.

No paradigma das vogais, os agentes podem variar de simples autômatos até sistemas complexos baseados em conhecimento. O ambiente é definido pelo domínio da aplicação, sendo geralmente topológico (UEZ, 2013). Sobre as interações, estas podem ser tanto físicas entre os agentes quanto baseadas em atos de fala; por fim, a organização, pode variar desde estudos biológicos até em estudos sociais, que envolvem leis e hierarquias.

Três dessas quatro dimensões são analisadas com mais detalhes neste trabalho. A seção 2.1 descreveu a dimensão de agentes, as dimensões ambiental e organização são descritas respectivamente nas seções 2.2.1, 2.2.2. A dimensão I, correspondente as interações, está implícita na dimensão dos agentes, pois representa a troca de mensagens entre os mesmos dentro do sistema, sendo assim, não é descrita neste trabalho.

2.2.1 Dimensão Ambiental em Sistemas Multiagentes

Segundo UEZ (2013), diversas visões do ambiente têm sido adotadas pela comunidade de agentes. Desde o início, conforme RUSSELL; NORVIG (2009), tudo que é externo ao SMA costuma ser visto como sendo parte do ambiente.

Para WEYNS; PARUNAK; SHEHORY (2009), o ambiente é visto como um repositório de agentes e recursos, ou simplesmente como um meio para a comunicação entre os agentes. Já RICCI; PIUNTI; VIROLI (2011) define o ambiente como uma abstração de primeira classe, sendo portanto parte integrante do SMA, que deve ser pensada e programada explicitamente. Olhando sobre esse ponto de vista, o ambiente provê funcionalidades e serviços que podem ser utilizados pelos agentes. Essa visão possibilita a separação conceitual entre agentes e o ambiente, proporcionando uma forma alternativa de modelar/desenvolver o SMA.

A abordagem de ambiente como abstração de primeira classe foi utilizada no modelo de Agentes e Artefatos (A&A) (RICCI; VIROLI; OMICINI, 2008). No modelo citado, o ambiente é formado por um conjunto dinâmico de entidades computacionais chamadas de artefatos.

No modelo proposto por RICCI; VIROLI; OMICINI (2008), os artefatos representam recursos ou ferramentas que os agentes podem utilizar, sendo organizados em uma ou mais *workspaces*, que podem ser distribuídos em nodos diferentes da rede. Para ter acesso a um artefato do SMA, o agente precisa entrar no *workspace* ao qual este artefato pertence. Além disso, cada artefato fornece um conjunto de operações, que são processos computacionais executados interiormente pelo artefato, e um conjunto de propriedades observáveis.

O *framework* CartAgO (RICCI; PIUNTI; VIROLI, 2011) tem como base o modelo A&A. O CartAgO permite que seja implementado e executado ambientes que utilizam o conceito de artefatos, além de possibilitar a integração com os agentes.

Essa separação do SMA em diversas camadas apresenta algumas vantagens: primeira-

mente, é preservada a abstração do agente, tornando desnecessário o uso de agentes para representar funcionalidades ou ferramentas do ambiente. A integração do agente com as entidades do ambiente é realizada por intermédio de conceitos utilizados nos agentes: ação e percepção. Segundo, o uso de artefatos e *workspaces* permite a modularização e reusabilidade de todo o ambiente ou de alguns artefatos específicos em diferentes aplicações. Terceiro, artefatos podem ser construídos de forma dinâmica, utilizados, substituídos ou adaptados pelos agentes, facilitando a extensão e a adaptação do ambiente.

2.2.2 Dimensão Organizacional em Sistemas Multiagente

Para HUBNER; SICHMAN (2003), a noção de organização é facilmente percebida no nosso cotidiano. Os autores exemplificam tal afirmação através da percepção da mesa de trabalho de um usuário, pois ao visualizá-la, é notório a sua organização ou não. HUBNER; SICHMAN (2003) afirmam que é possível inferir o propósito principal que a organização tem nos sistemas: fazer com que a finalidade do sistema seja facilmente alcançada.

Embora pareça simples caracterizar um sistema sob o ponto de vista organizacional, definir uma organização, o que a constitui, quais são suas formas e estruturas, não é uma tarefa trivial. Entretanto, há um consenso na seguinte definição geral de organização, conforme DIGNUM; DIGNUM (2001):

> A organização de um SMA é um conjunto de restrições ao comportamento dos agentes a fim de conduzí-los a uma finalidade comum.

Essas restrições mencionadas por DIGNUM; DIGNUM (2001), são impostas aos agentes quando entram na sociedade ou segundo as quais são projetados. HUBNER; SICHMAN (2003) explicam que essas restrições tem como objetivo controlar a autonomia dos agentes buscando produzir um comportamento global direcionado a uma finalidade. Essas restrições são descritas estruturalmente, na forma de papéis, ou funcionalmente, na forma de planos.

A literatura de SMA apresenta várias definições para organização, as quais são agrupadas em dois pontos de vista conforme LEMAÎTRE; EXCELENTE (1998): centrado nos agentes e centrado na organização, de acordo com a Figura 3.

Na abordagem centrada nos agentes, o SMA não possui uma representação explícita de sua organização. A representação está distribuída nos seus agentes. HUBNER; SICHMAN (2003) explicam que um observador ou um agente da sociedade somente pode inferir uma descrição subjetiva da organização, ou seja, uma descrição construída por ele mesmo a partir da observação do comportamento dos demais agentes que fazem parte da sociedade. Esta descrição subjetiva da organização é chamada de organização observada (HUBNER; SICHMAN, 2003).

No segundo ponto de vista, a abordagem centrada na organização, a organização existe objetivamente, ou seja, o observador pode obter uma descrição da organização que a

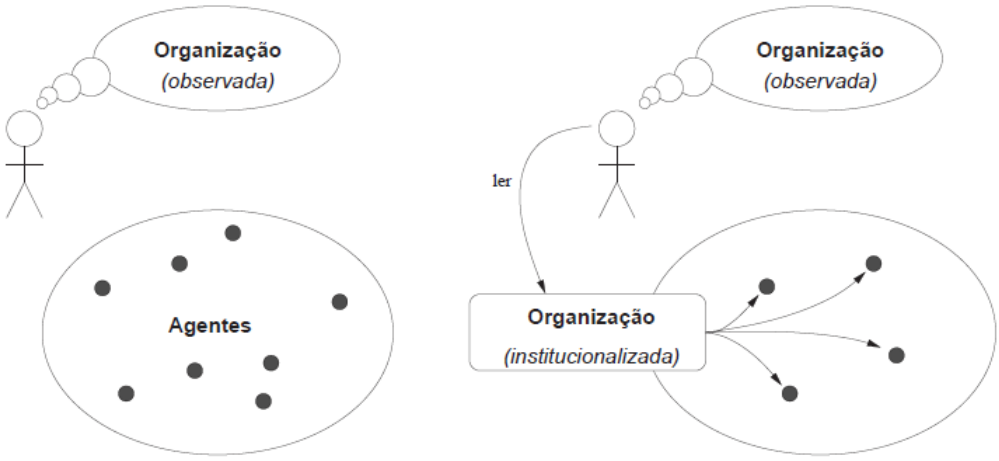


Figura 3: Pontos de vista para organização de um SMA (HUBNER; SICHMAN, 2003).

sociedade está adotando sem precisar observar seu comportamento ou mesmo considerar os agentes que a compõem (HUBNER; SICHMAN, 2003). Para HUBNER; SICHMAN (2003), esta descrição será chamada de organização institucionalizada.

Se do ponto de vista de um observador externo ao SMA podem existir duas situações, na visão dos agentes que formam a sociedade também podem ser concebidas duas situações adversas: Uma onde os agentes são capazes de explicitamente representar uma organização e outra que os agentes não são capazes.

Para permitir a representação da organização utilizada em um SMA, são utilizados modelos organizacionais, como o Moise+ (HÜBNER; SICHMAN; BOISSIER, 2002). Este modelo descreve a organização se baseando em três dimensões: estrutural, funcional e normativa. Na primeira dimensão é definido como a organização está estruturada. Essa dimensão é desenvolvida em três níveis: 1) individual, que define os papéis que podem ser assumidos pelos agentes; 2) social, que define as ligações de conhecimento, comunicação, autoridade e compatibilidade entre os papéis; 3) coletivo, que define os grupos e que papéis farão parte de cada grupo. A dimensão funcional tem como finalidade definir como os objetivos globais são atingidos. Esses objetivos são decompostos em subobjetivos que são agrupados em missões para serem atingidas pelos agentes. Por último, a dimensão normativa estabelece as permissões e obrigações entre os papéis e as missões. Sempre que um agente escolher exercer um papel em uma organização, este aceita as permissões e obrigações relacionadas aquele papel (HÜBNER et al., 2010).

A principal vantagem na utilização do modelo organizacional Moise+ é que ele permite a reorganização do sistema (UEZ, 2013). Isto ocorre em virtude das dimensões estrutural e funcional serem independentes. Assim o sistema pode alterar a estrutura de sua organização e alterar suas funcionalidades e vice-versa (HÜBNER et al., 2010).

2.3 Programação Orientada a Agentes

No trabalho de BORDINI et al. (2006) é feita uma pesquisa a respeito de linguagens de programação e ferramentas de desenvolvimento para SMA. Na parte inicial desta pesquisa, os autores abordam linguagens de programação mais apropriadas para esta categoria de desenvolvimento, classificando-as em: declarativa, imperativa e híbridas. Posteriormente, os autores mencionam os ambientes integrados de desenvolvimento aderentes a essa categoria de sistemas. Para finalizar, é feito um relato a respeito de plataformas e *frameworks* para SMA.

A respeito dos *frameworks* de desenvolvimento, dentre os mais conhecidos, pode-se citar o *Java Agent Development Framework* (JADE) (BELLIFEMINE et al., 2005). JADE é um *framework* desenvolvido na linguagem Java para o desenvolvimento de aplicações multiagente. Ele representa um *middleware* de agente, fornecendo um conjunto de serviços e várias ferramentas gráficas para depuração e testes de SMA. Um dos principais objetivos da plataforma é apoiar a interoperabilidade, a cumprir rigorosamente as especificações FIPA relativas à arquitetura da plataforma, bem como a infraestrutura de comunicação (BORDINI et al., 2006). Além disso, segundo BORDINI et al. (2006), JADE é muito flexível e pode ser adaptado para ser usado em dispositivos com recursos limitados, tais como PDAs e telefones celulares.

O Jadex (BRAUBACH; LAMERSDORF; POKAHR, 2003) é uma estrutura de software para a criação de agentes BDI. O *framework* é fornecido como uma camada de agente racional que fica no topo de uma infraestrutura de agente de middleware, como JADE, apoiando o desenvolvimento do agente com tecnologias bem estabelecidas, como Java e *eXtensible Markup Language* (XML) (BORDINI et al., 2006). JADEX tem sido usada para construir aplicações em diferentes domínios como a simulação, programação e computação móvel.

Tuple Centre Spread over the Network (TuCSoN) (OMICINI; ZAMBONELLI, 1999) é um *framework* para a coordenação de SMA, baseado em um modelo e uma infraestrutura relacionada ao modelo, fornecendo uso geral, serviços programáveis para apoiar a comunicação entre agentes e coordenação (OMICINI; ZAMBONELLI, 1999). O modelo respeita a linguagem ReSpec.

DEsign and Specification of Interacting REasoning components (DESIRE) (BRAZIER et al., 1998) é um método de desenvolvimento composicional para sistemas multiagente, baseado em um conceito de arquitetura de composição, e desenvolvido por (BRAZIER et al., 1998) na Vrije Universiteit Amsterdam. A abordagem DESIRE foi usado para aplicações tais como balanceamento de carga de distribuição de energia elétrica e sistemas de diagnóstico (BORDINI et al., 2006).

Sobre linguagens, uma das mais conhecidas nesse âmbito se chama Jack (BUSETTA et al., 1999). Em linhas gerais, Jack é baseada em ideias de sistemas de planejamento

resultantes do estudo da arquitetura BDI (BORDINI et al., 2006). Para UEZ (2013), Jack é é uma extensão da linguagem de programação Java que permite o desenvolvimento de agentes baseados na arquitetura BDI e também time de agentes.

Segundo BORDINI et al. (2006), Jason (BORDINI; HÜBNER; WOOLDRIDGE, 2007) é um interpretador para uma extensão da linguagem AgentSpeak(L) (RAO, 1996), que permite o desenvolvimento de agentes utilizando a arquitetura BDI. O Jason fornece algumas funcionalidades a seus usuários, como o uso de atos de fala na comunicação entre os agentes; possibilidade de utilizar anotações nos planos que permite uma elaborada seleção de funções; por intermédio da linguagem Java, pode-se realizar a customização de diversos aspectos da plataforma. Jason possui um *plug-in* que possibilita que ele seja integrado ao Eclipse, tornando-o uma IDE que permite a execução e inspeção dos agentes codificados. Outra peculiaridade de Jason é que ele pode ser utilizado em conjunto com o *framework* CArtAgO e o modelo organizacional Moise+, compondo o *framework* JaCaMo (BOISSIER et al., 2013).

O JaCaMo é o resultado da integração de três diferentes componentes de um SMA. Ele possibilita que seja integrado os conceitos das dimensões de agente, ambiente e organização, permitindo que ocorra a programação simplificada de um SMA. Sua lógica de funcionalidade é a seguinte: no Jason, são programados os agentes utilizando a arquitetura BDI; CArtAgO fica encarregado de representa um ou mais ambientes; por último, Moise+ é o modelo organizacional responsável por refinar a execução do agente dentro do ambiente.

De acordo com UEZ (2013), a principal vantagem da utilização do JaCaMo é a possibilidade da separação dos conceitos referentes a cada dimensão que compõem um SMA. Essa separação propicia que aspectos referentes a cada dimensão possam ser desenvolvidos utilizando abstrações e linguagens específicas, contribuindo com uma maior modularidade, reusabilidade, legibilidade, facilidade de extensão e manutenção do código. Complementarmente, pode-se dizer que essa ferramenta foi a primeira no âmbito de agentes que conseguiu separar as dimensões de um SMA, potencializando o desenvolvimento da área e incentivando novos usuários a desenvolver software seguindo esse paradigma.

2.4 Engenharia de Software Orientada a Agentes

Segundo IGLESIAS; GARIJO; GONZÁLEZ (1999), a tecnologia de agentes recebeu uma grande atenção nos últimos anos e, como resultado, a indústria está começando a ficar interessada em usar esta tecnologia para desenvolver seus produtos e automatizar ainda mais as suas soluções.

Para LIND (2001), agentes e SMA são atualmente um dos campos de pesquisa mais interessantes na comunidade da ciências da computação. Especialmente, a forma natural de capturar a estrutura e o comportamento de sistemas complexos tem estimulado as

pesquisas nessa área. Conforme ZAMBONELLI; OMICINI (2004), a computação baseada em agentes é uma abordagem promissora para o desenvolvimento de aplicações em domínios complexos.

Embora ambos os autores citados acreditem que desenvolver SMA seja uma inovação nos paradigmas de desenvolvimento, é consenso entre os mesmos que existe uma série de desafios que precisam ser enfrentados para fazer o paradigma baseado em agentes um modelo amplamente aceito na área de engenharia de software. Surge então uma subdivisão da área, a qual mescla conceitos relacionados a Engenhaira de Software, chamada de *Agent Oriented Software Engineering* (AOSE) ou Engenharia de Software Orientada a Agentes.

A origem da AOSE é explicada através em (LIND, 2001). Para o autor, a área surgiu naturalmente através do desenvolvimento dos paradigmas de programação. Em seu trabalho, LIND (2001) explica que nos primórdios da programação, um programa aparentava ser um bloco monolítico, sem qualquer estrutura inerente. Entretanto, ao passar do tempo, esse ponto de vista foi alterado, admitindo-se que o programa é composto na verdade por várias unidades estruturais menores, chamados de sub-rotinas.

O conceito de sub-rotinas de programas de computador dominou a computação por um tempo considerável. Entretanto, percebeu-se que esta definição não era adequada após o acréscimo do aspecto de controle de fluxo de programação. Consequentemente, LIND (2001) explica que a visão acerca dos paradigmas mudou novamente, desta vez adicionando o agrupamento de dados e os atributos em uma única unidade estrutural, chamada de objetos.

Nos últimos anos, está ocorrendo a terceira mudança na perspectiva da concepção de sistemas (LIND, 2001). Neste momento, LIND (2001) explica que está sendo deixado de lado os objetivos passivos, a fim de evoluir para unidades estruturais chamadas de Agentes.

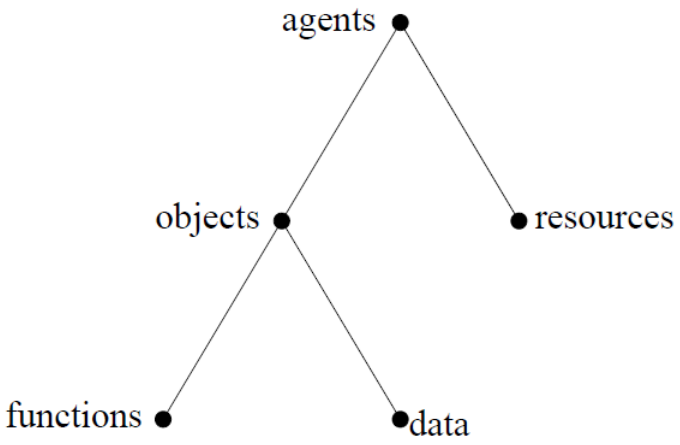


Figura 4: Histórico de desenvolvimento (LIND, 2001).

Para uma melhor compreensão acerca dessa evolução, LIND (2001) confeccionou a

Figura 4. No primeiro passo, os programas são vistos como um conjunto de funções que estabelecem um objetivo bem definido. Essas funções podem ser descritas como uma sequência imperativa de declarações, no caso da programação imperativa, ou uma coleção de expressões matemáticas que estão interligadas entre si, no caso da programação funcional, ou ainda como um conjunto de metas, sem impor uma forma particular de atingir essa meta, caso da programação declarativa.

Na próxima etapa dessa evolução, os programas são interpretados em termos de dados que são manipulados e as funções que operam esses dados. Para LIND (2001), isto leva a programação estruturada, onde os aspectos relacionados semanticamente dos programas são espacialmente relacionados. Uma relação mais concisa entre dados e funções é introduzida por tipos abstratos de dados, resultando na programação orientada a objetos.

No final dessa evolução, os objetos são complementados com recursos, tais como o tempo de computação, que podem ser utilizados livremente. LIND (2001) explica que essa liberdade no processo de alocação de recursos levou a um conceito que ele julga mais adequado na programação Orientada a Agentes (OA), a autonomia.

Para finalizar, LIND (2001) resume a origem da AOSE através da necessidade de um novo quadro de trabalho, construído sobre a propriedade básica da autonomia. Esta propriedade, permite a modelagem e compreensão de sistemas orientados a agentes. Complementarmente, o autor explica que a visão OA é um pré requisito necessário para que ocorra a aceitação integral das técnicas de IA disponíveis atualmente, pois segundo ele, todos devem se acostumar em um futuro próximo com a modelagem de sistemas sobre a perspectiva de objetivos, crenças e desejos, para que de fato a inteligência seja realmente acrescentada em uma máquina com capacidade de processamento.

2.4.1 Metodologias de Engenharia de Software Orientada a Agentes

Nesta seção são apresentados algumas metodologias para se realizar engenharia de software em sistemas baseados em agentes.

2.4.1.1 Prometheus

Segundo PADGHAM; WINIKOFF (2002), Prometheus é uma metodologia para desenvolvimento de agentes inteligentes que se diferencia das demais por ser detalhada e completa, abrangendo todas as atividades necessárias para o desenvolvimento de sistemas de agentes inteligentes. A Figura 5 apresenta as fases de desenvolvimento de um SMA seguindo a metodologia.

Segundo PADGHAM; WINIKOFF (2005) e KHALLOUF; WINIKOFF (2009), Prometheus é uma metodologia que consiste em três fases:

- Especificação do sistema: É composta por duas etapas: Determinar o ambiente do sistema (percepções e ações) e determinar os objetivos e funcionalidades do sistema (objetivos e cenários de casos de uso).

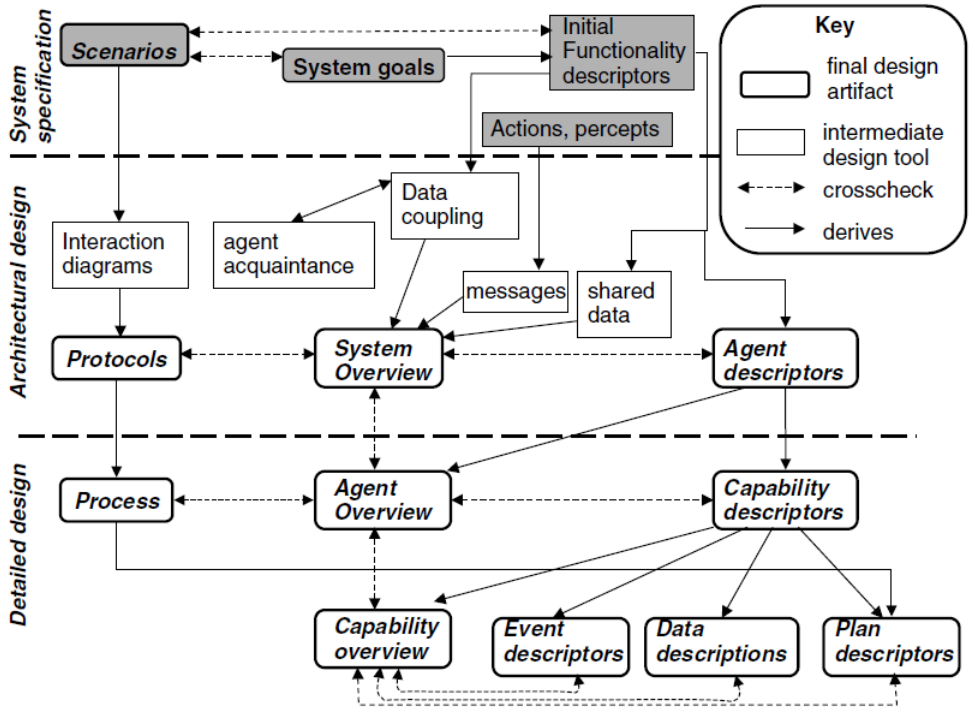


Figura 5: Fases da metodologia Prometheus (PADGHAM; WINIKOFF, 2004)

- Projeto Arquitetural: Utiliza as saídas da fase anterior para determinar quais agentes devem existir no sistema a ser modelado (PADGHAM; WINIKOFF, 2002). O objetivo final dessa etapa da metodologia é especificar completamente a interação entre os agentes.

- Projeto Detalhado: Foca no desenvolvimento da estrutura interna de cada um dos agentes e como os mesmos irão realizar suas tarefas dentro do sistema modelado. Nesta etapa que a metodologia torna-se específica para agentes que usam a arquitetura BDI.

De acordo com PADGHAM; WINIKOFF (2002), uma das vantagens de utilizar essa metodologia é a quantidade de locais onde ferramentas automatizadas podem ser utilizadas para verificar a consistência entre vários modelos produzidos ou processos de projeto. A ferramenta disponível para o Prometheus atualmente chama-se *Prometheus Design Tools* (PDT) (THANGARAJAH; PADGHAM; WINIKOFF, 2005). PDT permite aos usuários criar e modificar projetos desenvolvidos com a metodologia Prometheus. Essa ferramenta assegura que não ocorram certas inconsistências, como utilizar notações em diagramas errados, além de exportar diagramas individuais e gerar um relatório com o projeto completo. A metodologia não leva em conta a plataforma de desenvolvimento até a fase de projeto detalhado. Entretanto, a mesma permite que seja gerado código para a linguagem JACK.

2.4.1.2 Tropos

Segundo BRESCIANI et al. (2004), a metodologia Tropos (MYLOPOULOS; KOLP; CASTRO, 2001) (BERGENTI; GLEIZES; ZAMBONELLI, 2004) (COSSENTINO et al., 2005) tem como propósito apoiar todas as atividades de análise e projeto do desenvolvimento de software orientado a agentes. Ela foi desenvolvida com base no *framework* i*, que permite modelar o sistema multiagente com base nos conceitos de ator, objetivo e dependência entre atores. Tropos é composto por cinco fases distintas no seu processo de desenvolvimento, como apresenta a Figura 6.

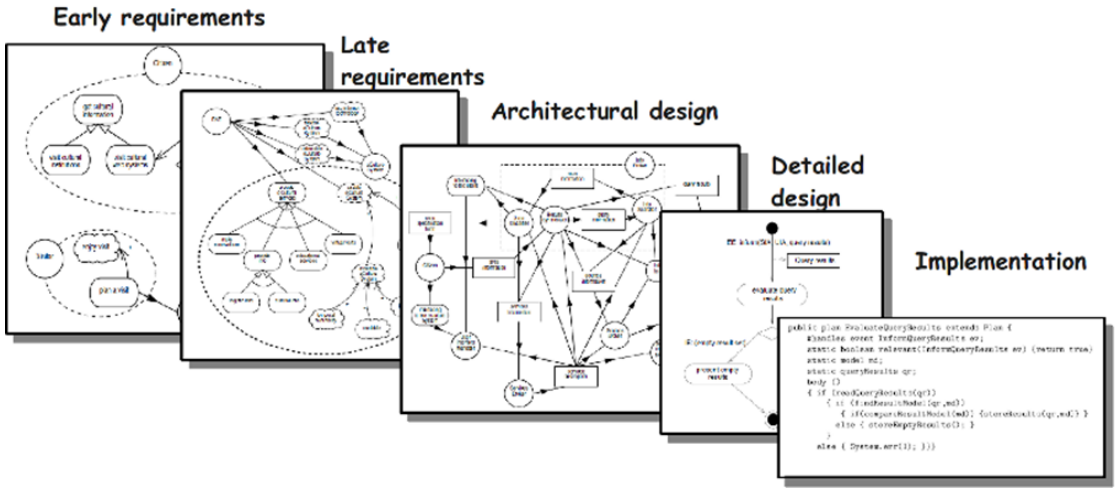


Figura 6: Fases da metodologia Tropos (SCIENZA DELL'INFORMAZIONE, 2015)

Conforme UEZ (2013), a principal característica dessa metodologia consiste na ênfase dada a análise dos requisitos. Essa primeira etapa é dividida em duas fases. Na fase dos requisitos iniciais, o foco é identificar os agentes de domínio e modelá-los como atores sociais, dependentes uns dos outros para atingirem seus objetivos, planos e compartilharem dos mesmos recursos. Já na fase de requisitos finais, o modelo é estendido, incluindo-se uma série de dependências entre atores do ambiente.

A fase de projeto de sistema também é dividida em duas etapas. A primeira, intitulada projeto arquitetural, consiste na definição da arquitetura global do sistema (subsistemas, interligados através de dados e fluxos de controle), também prevê um mapeamento dos atores do sistema para um conjunto de agentes de software, cada um caracterizado por atividades específicas. Na segunda fase, denominada projeto detalhado, são delegados o detalhamento dos agentes, das suas capacidade e das interações entre os agentes. Em virtude da plataforma de implementação já ter sido escolhida nessa etapa, deve-se especificar o agente levando em consideração a linguagem na qual o mesmo será implementado.

A fase final visa a implementação do sistema em uma linguagem de desenvolvimento própria para tal finalidade. Segundo COSSENTINO et al. (2005), a metodologia Tropos permite o mapeamento direto da modelagem para a linguagem JACK. Todavia, utilizando

a ferramenta TAOM4E, pode-se também dispor da geração automática de códigos para a linguagem Jadex (MORANDINI; PENSERINI; PERINI, 2008).

2.4.1.3 Multiagent Systems Engineering (MaSE)

A metodologia *Multiagent Systems Engineering* (MaSE) foi proposta por DELOACH (1999). É uma metodologia desenvolvida para análise e projeto de sistemas multiagente. MaSE utiliza da abstração fornecida por SMA para ajudar projetistas a desenvolver sistemas de software distribuídos inteligentes (HENDERSON-SELLERS; GIORGINI, 2005). Segundo HENDERSON-SELLERS; GIORGINI (2005), MaSE é construído sobre as técnicas orientadas a objetos existentes. Porém, é especializado para o projeto de SMA.

A metodologia MaSE consiste de duas fases principais que resultam na criação de um conjunto de modelos complementares (HENDERSON-SELLERS; GIORGINI, 2005). A primeira fase é análise e a segunda é projeto. Detalhes de cada uma das fases são ilustrados na Figura 7.

Phases	**Models**
1. Analysis Phase a. Capturing Goals b. Applying Use Cases c. Refining Roles	Goal Hierarchy Use Cases, Sequence Diagrams Concurrent Tasks, Role Model
2. Design Phase a. Creating Agent Classes b. Constructing Conversations c. Assembling Agent Classes d. System Design	Agent Class Diagrams Conversation Diagrams Agent Architecture Diagrams Deployment Diagrams

Figura 7: Fases da metodologia MaSE (HENDERSON-SELLERS; GIORGINI, 2005)

A primeira fase inclui três etapas: captura de objetivos, aplicação de casos de uso e refinamento de papéis. A segunda fase é composta por quatro etapas: criação de classes de agentes, construção das conversações, montagem das classes dos agentes e o projeto do sistema.

A finalidade da primeira etapa da fase de análise é capturar os objetivos do sistema, extraindo-os da especificação inicial do sistema. Na etapa de aplicação de casos de uso, o analista transforma os objetivos e sub-objetivos em casos de uso. A terceira etapa tem como propósito garantir que foram identificados todos os papéis necessários no sistema e desenvolver as tarefas que definem o comportamento dos papéis e os padrões de comunicação.

Na segunda fase, a primeira etapa identifica as classes de agentes a partir dos papéis refinados. A próxima fase é a construção de conversações. O objetivo desta etapa é definir os detalhes de comunicação, baseado nos detalhes internos de tarefas simultâneas. Os elementos internos dos agentes são projetados durante a etapa de montagem das classes dos agentes, que inclui duas sub-etapas: a definição da arquitetura de agentes e a definição

dos componentes da arquitetura. Projeto do sistema é o último passo da metodologia MaSE. Nessa etapa, é utilizado o diagrama de implantação para demonstrar os números, tipos e locais das instâncias dos agentes no sistema.

Para auxiliar os projetistas a utilizarem a metodologia, foi construído uma ferramenta chamada agentTool. Esta serve de plataforma de validação para MaSE. A ferramenta é baseada em gráficos e completamente interativa (DELOACH; WOOD; SPARKMAN, 2001). Entretanto, a ferramenta ainda não possui geração de códigos para nenhuma plataforma de desenvolvimento[1].

2.4.1.4 Ingenias

Segundo HENDERSON-SELLERS; GIORGINI (2005), a metodologia Ingenias fornece uma notação para modelagem de SMA através de uma coleção bem definida de atividades, para orientar o processo de desenvolvimento. Especificamente, nas tarefas de análise, projeto, verificação e implementação. Essas etapas são apoiadas por um conjunto integrado de ferramentas chamado *Ingenias Development Kit* (IDK).

A metodologia Ingenias foi construído baseando-se em cinco metamodelos que definem vários pontos de vista. A metodologia se propõe em organizar esses conceitos sob cinco pontos de vista distintos: a organização; o agente; tarefas/objetivos; interação e o ambiente. Segundo HENDERSON-SELLERS; GIORGINI (2005), a organização descreve o espaço em que os agentes, recursos, tarefas e objetivos existem; o ponto de vista de agentes está preocupado com os objetivos, responsabilidades e capacidades dos agentes; o ponto de vista de tarefas/objetivos considera a decomposição de metas e tarefas, e descrevem as consequências de executar uma tarefa além do por quê de ela ser realizada; o ponto de vista de interação aborda a troca de informações ou pedidos entre os agentes, ou entre agentes e usuários humanos; O ponto de vista do ambiente, definem as entidades com as quais o SMA interage, podendo ser: recursos, outros agentes ou aplicações.

Ingenias propõe o desenvolvimento em três etapas: análise, projeto e implementação. As duas primeiras etapas são divididas em três fases: inserção, elaboração e construção. A fase de implementação consiste em mapear os conceitos modelados para conceitos do *framework* JADE. A ferramenta Ingenias Development Kit (IDK) oferece suporte a modelagem dos sistemas usando a metodologia Ingenias, além de permitir a criação de módulos a geração de código em outras linguagens.

2.4.1.5 MESSAGE

Segundo CAIRE et al. (2002), MESSAGE toma a UML como um ponto de partida e adiciona conceitos de agente, papel, tarefa para suprir as necessidades relativas a sistemas multiagente. Conforme UEZ (2013), esta metodologia propõe a análise e projeto de um SMA com base em cinco pontos de vista: organização, objetivos/tarefas, agentes/papéis,

[1] http://agenttool.cis.ksu.edu/

interação e domínio.

O ponto de vista da organização foca em demonstrar as entidades (agentes, organizações, papéis e recursos) do sistema e as relações existentes entre eles. Já o ponto de vista de objetivos/tarefas tem como finalidade apresentar os objetivos que os agentes devem buscar e as tarefas necessárias para atingir cada objetivo. O ponto de vista de agentes/papéis apresenta para cada agente ou papel do sistema, os objetivos pelos quais esse agente ou papel é responsável, os eventos que o agente deve sentir, os recursos que deverá usar e as tarefas que sabe executar. O ponto de vista de interação descreve cada interação entre agentes e papéis, demonstrando o gatilho da interação, os colaboradores e outras informações relevantes. Por último, o ponto de vista do domínio apresenta informações ou relacionamentos específicos do domínio do sistema que está sendo modelado.

A metodologia MESSAGE possui duas fases de desenvolvimento, análise e projeto. Na primeira fase, o objetivo 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. Esse modelo visa ajudar a comunicação entre o time de desenvolvimento e o cliente, fornecendo a base a partir da qual o projeto poderá continuar. Os modelos dessa etapa são produzidos a partir de refinamentos separados por etapas. Portanto, nessa etapa o foco consiste na produção de uma especificação que descreva a organização, objetivos e os papéis do sistema.

A próxima fase da metodologia é intitulada projeto e é divida em duas etapas: Projeto de alto nível e projeto de baixo nível. Na primeira fase, o nível 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. Essa etapa não leva em consideração a plataforma que o sistema será desenvolvido. Na próxima fase são definidas a estrutura e o comportamento de entidades, tais como organização, agentes, tarefas, além de entidades de domínio de objetivos. Nessa fase a plataforma de implementação é considerada e os conceitos modelados na etapa de projeto de alto nível são mapeados para os elementos computacionais da plataforma destino (UEZ, 2013).

2.4.1.6 PASSI

De acordo com COSSENTINO; POTTS (2002), *Process for Agent Societies Specification and Implementation* (PASSI) é uma metodologia para projetar e desenvolver sociedades multiagente integrando modelos de projeto e conceitos de engenharia de software e inteligência artificial, usando a notação UML. Além disso, PASSI baseia-se na arquitetura FIPA para a modelagem de agentes. A metodologia é dividida em cinco modelos:

- Modelo de requisitos do sistema: Um modelo antropomórfico dos requisitos do sistema em termos de agência e destino.

- Modelo de sociedade de agente: Um modelo das interações e dependências sociais

entre os agentes que participam na solução.

- Modelo de implementação de agente: Um modelo de arquitetura de solução em termos de classes e métodos.
- Modelo de código: Um modelo de solução a nível de código.
- Modelo de implantação: Um modelo da distribuição das partes do sistema sobre unidades de hardware.

Para apoiar o modelo de código, existe a ferramenta PTK (PASSI Toolkit). A ferramenta também oferece bibliotecas de padrões que podem ser utilizados durante a geração de código e permite testes do sistema antes da implantação do software. O método permite a geração de código para o *framework* JADE.

2.4.1.7 GAIA

A metodologia GAIA (WOOLDRIDGE; JENNINGS; KINNY, 2000) foi projetada para tratar aspectos de análise e projeto de sistemas baseados em agentes. Esta metodologia trata ambos os aspectos de nível macro (sociedade) e micro (agentes,) e mostra-se neutra em relação ao domínio de arquitetura de agentes. Segundo UEZ (2013), GAIA foi desenvolvida para permitir que o SMA fosse analisado e projetado com suficiente detalhamento para permitir a sua implementação.

GAIA é composta por três fases: análise, projeto arquitetural e projeto detalhado. A primeira fase tem como finalidade organizar os requisitos e definir o ambiente no qual o sistema está inserido. Na fase de projeto arquitetural são definidos a estrutura organizacional do SMA; complementação do modelo de papéis, incluindo os papéis organizacionais necessários; complementação do modelo de protocolos, incluindo os protocolos organizacionais necessários. A fase de projeto detalhado visa definir os agentes e os serviços que irão servir de base para a implementação do sistema.

Segundo GUEDES (2012), a metodologia GAIA produz artefatos textuais, não possuindo uma notação gráfica particular, embora seja recomendado a utilização da AUML para preencher essa lacuna.

2.4.2 Análise das Metodologias

O objetivo desta subseção é analisar as metodologias sob dois pontos de vista distintos: ferramenta gráfica para apoio a utilização da metodologia e geração de código a partir da especificação.

2.4.2.1 Ferramenta gráfica para apoio a utilização da metodologia

O objetivo de desenvolvimento de uma ferramenta gráfica (editor gráfico) é possibilitar que o usuário faça uso de uma gama de notações gráficas criadas pelo detentor da

metodologia a qual o editor deseja apoiar. A Tabela 1 exibe as metodologias estudadas neste trabalho e as ferramentas gráficas que apoiam o seu uso respectivamente.

Tabela 1: Metodologias x Ferramentas de Apoio.

Metodologia	**Ferramenta de Apoio**
Prometheus	PDTools
Tropos	TAOM4E
MaSE	agentTool
Ingenias	IDK
MESSAGE	-
PASSI	PTK
GAIA	-

A finalidade dessa tabela é demonstrar que existem muitas ferramentas gráficas para modelar SMA e que, embora cada metodologia possua suas peculiaridades, sabe-se que algumas delas modelam um SMA além das dimensões de agentes. Portanto, a partir desse comparativo, conclui-se que o principal problema em relação a especificação/implementação de um SMA consiste na transformação do *work products* em código desenvolvido de maneira automática.

2.4.2.2 Geração de código a partir da especificação

A geração de códigos tem como objetivo transformar as especificações definidas no *work products* em código fonte. Entretanto, existem alguns problemas ao realizar essa transformação, sendo que o principal deles consiste no *gap* conceitual existente entre as abstrações utilizadas durante a especificação e aquelas utilizadas pela plataforma de desenvolvimento (NUNES et al., 2011).

Segundo UEZ (2013), algumas metodologias fornecem regras e instruções que definem como deve ocorrer a tradução das especificações em código. A autora explica também que essa transformação pode ocorrer de forma manual, sendo feita pelos próprios usuários da metodologia ou de forma automática, através de uma ferramenta para auxiliar a geração de código.

As metodologias descritas anteriormente, em sua grande parte, permitem a geração automática de códigos através de ferramentas ou fornecem documentações que tem como propósito ajudar ao usuário a fazer isso de forma manual. A Tabela 2 resume as metodologias estudadas e as plataformas de desenvolvimento que possibilitam a geração de código de forma automática.

A Tabela 2 demonstra que das metodologias existentes, a maioria gera códigos para os *frameworks* Jade ou Jack, com exceção de Tropos que também gera códigos para a plataforma Jadex. Entretanto, ao analisar as plataformas, percebe-se que todas focam sua implementação no âmbito de agentes, ocasionando perda das demais informações presentes no *work products* projetado anteriormente. Além disso, conforme DEMAZEAU

Tabela 2: Metodologias x plataformas de desenvolvimento que são gerados códigos automaticamente a partir da ferramenta apoiadora da metodologia.

Metodologia	**Plataforma de Desenvolvimento**	**Dimensões do SMA atingidas**
Prometheus	Jack	Agentes
Tropos	Jadex e Jack	Agentes
MaSE	-	-
Ingenias	Jade	Agentes
MESSAGE	-	-
PASSI	Jade	Agentes
GAIA	-	-
Prometheus AEOlus	JaCaMo	Agentes, Ambiente e Organização

(1995), um SMA possui outras dimensões como ambiente e organização que não são levados em consideração por essas plataformas, resultando em um projeto com diversas deficiências.

Das plataformas estudadas na Seção 2.3, a única que possui em sua essência a preocupação com as dimensões de agente, ambiente e organização é o *framework* JaCaMo (BOISSIER et al., 2013). Além disso, nenhuma das metodologias estudadas na Seção 2.4.1 foca sua modelagem em todas as dimensões presentes no SMA, com exceção da metodologia Prometheus AEOlus, que será descrita a seguir.

Em virtude da peculiaridade encontrada na metodologia Prometheus AEOlus, o objetivo deste trabalho consiste na criação de uma plataforma de desenvolvimento que a apoie, de modo a possibilitar que o utilizador da mesma consiga modelar SMAs levando em consideração todas as dimensões descritas por (DEMAZEAU, 1995). Alternativamente, o trabalho descrito à seguir, apresenta uma nova abordagem para modelar SMA, utilizando de conceitos descritos na Seção 4.1, porém, com enfoque distinto do desejado na elaboração deste trabalho.

3 METODOLOGIAS AOSE E INTEGRAÇÃO COM A PLATAFORMA JACAMO

Este capítulo tem como objetivo apresentar trabalhos que possibilitam a modelagem de SMA e a interligação com alguma das ferramentas do *framework* JaCaMo.

Este capítulo está dividido como segue: A seção 3.1 apresenta o trabalho de PANTOJA; CHOREN (2013), além de uma análise ao trabalho desenvolvido; a seção 3.2 descreve a metodologia Prometheus AEOlus (UEZ, 2013), cuja qual é a base desta dissertação; na seção 3.3 são feitas considerações em relação aos trabalhos apresentados.

3.1 Tropos para JaCaMo

GUINELLI; PANTOJA; CHOREN (2015) desenvolveram um trabalho que teve como objetivo integrar o *Model Driven Architecture* (MDA) proposto em (PANTOJA; CHOREN, 2013) com a ferramenta de modelagem gráfica TAOM4E, a qual possibilita que sejam modelados diagramas aderentes a metodologia Tropos. A solução desenvolvida é ilustrada na Figura 8.

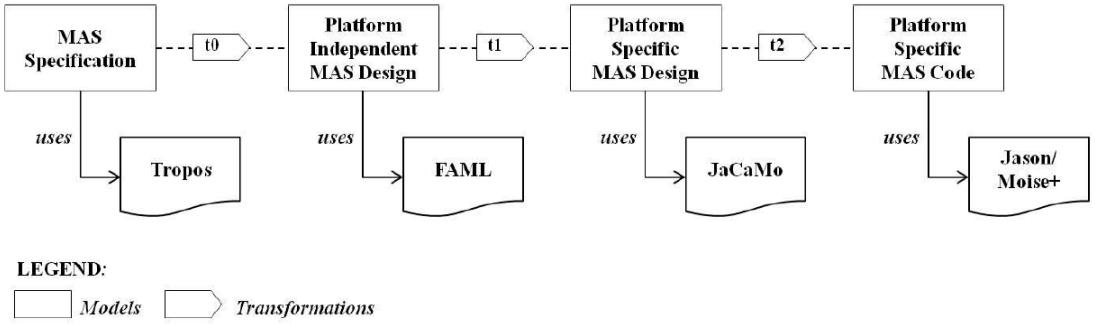


Figura 8: Metodologia MDA para SMA (GUINELLI; PANTOJA; CHOREN, 2015).

Primeiramente, em (PANTOJA; CHOREN, 2013), foi desenvolvido um MDA para desenvolvimento genérico de SMA. O MDA em questão é uma ferramenta que contém um conjunto de *plug-ins* desenvolvidos para a plataforma Eclipse. Para tanto, SOLEY et al. (2000) propuseram especificações e linguagens de modelagem que possibilitam que

sejam realizadas esse tipo de solução. Sobre o MDA proposto em PANTOJA; CHOREN (2013), este possui como solução central o metamodelo chamado FAML.

O FAML é um metamodelo genérico proposto em BEYDOUN et al. (2009), o qual unifica diferentes linguagens de modelagem orientadas a agentes (PANTOJA; CHOREN, 2012). O FAML é dividido em dois âmbitos: o tempo de projeto, onde o conceito central é o sistema do agente e o tempo de execução, onde o conceito passa ser o ambiente onde este agente está situado. Os agentes também são conceitos dentro do FAML, definidos em dois âmbitos: interno e externo. Portanto, o metamodelo possui quatro níveis, sendo: ambiente, agentes e níveis de definição dos agentes, podendo ser o âmbito de definição interna do agente e externa (PANTOJA; CHOREN, 2012).

Segundo GUINELLI; PANTOJA; CHOREN (2015), o FAML foi criado pensando nas mais utilizadas metodologias e linguagens de modelagens orientadas a agentes, como Tropos, Prometheus e GAIA. Por esse motivo, FAML reune todos os conceitos de Tropos e de algumas outras metodologias no mesmo metamodelo, viabilizando a sua utilização dentro do MDA proposto em PANTOJA; CHOREN (2013). Conforme os autores, a abordagem proposta no trabalho possibilita que seja escolhida uma tecnologia de projeto preferida entre as metodologias, e linguagens de modelagem aderentes ao FAML.

No trabalho de GUINELLI; PANTOJA; CHOREN (2015), a primeira parte da solução é a escolha da metodologia preferida pelo utilizador para a realização da modelagem do SMA, que conforme a Figura 8, faz parte do *MAS Specification*. No trabalho, foi escolhida a metodologia Tropos, a qual possui como ferramenta gráfica de modelagem um *plug-in* chamado TAOM4E.

Posterior a modelagem do SMA utilizando o TAO4ME, é necessário que os conceitos modelados sejam transformados. Nesse ponto que entra em evidência o MDA proposto em PANTOJA; CHOREN (2013). Para tanto, os autores utilizaram a especificação *Query View Transformation* (QVT) para fazer o mapeamento do modelo Tropos para o modelo FAML. Após a realização dessa transformação, é gerado o metamodelo do FAML mapeado com os conceitos do Tropos. Para a realização dessa transformação, foi utilizado um *plug-in* do Eclipse chamado *QVT Operationals*. Além disso, para a descrição dos metamodelos utilizados, os autores exploraram uma ferramenta chamada Ecore, que é parte integrante do *plug-in* do Eclipse chamado *Eclipse Modeling Framework* (EMF). O Ecore permite que sejam criadas, de forma mais interativa, instâncias dos metamodelos em XML. A transformação em evidência corresponde ao quadro T0 da Figura 8. Como forma de esclarecimento, todos os metamodelos utilizados na solução proposta por GUINELLI; PANTOJA; CHOREN (2015) foram implementados utilizando o Ecore.

Após a obtenção do metamodelo do FAML com os conceitos anteriormente modelados em Tropos, é necessário a escolha de uma plataforma específica de desenvolvimento (GUINELLI; PANTOJA; CHOREN, 2015). No trabalho de GUINELLI; PANTOJA; CHOREN (2015), a plataforma escolhida foi o JaCaMo.

A segunda parte proposta no trabalho de GUINELLI; PANTOJA; CHOREN (2015) consiste na transformação do modelo mapeado em FAML para o metamodelo da plataforma escolhida para desenvolvimento. Para realizar essa transformação, GUINELLI; PANTOJA; CHOREN (2015) desenvolveram algumas transformações no metamodelo do JaCaMo. As modificações consistem na simplificação da dimensão ambiental e na extensão da dimensão organizacional.

A primeira simplificação do trabalho de GUINELLI; PANTOJA; CHOREN (2015) ocorreu em virtude da metodologia FAML não utilizar em seu metamodelo artefatos para implementar o ambiente de agentes, não sendo preciso a utilização de CartAgO, de modo a possibilitar a simplificação da dimensão ambiental. Esta facilidade ocorreu mantendo a classe *environment* para representar o ambiente e a classe *Percept* para representar as percepções, ambas no metamodelo do JaCaMo, excluindo o uso do Cartago.

A segunda modificação consistiu em adicionar uma extensão na dimensão organizacional do metamodelo JaCaMo. Para isto, adicionou-se a classe *Link* e criou-se um auto-relacionamento como *supertype* na classe *Role*. Criou-se também um *monitoringScheme* na classe *Scheme* e *subGoal* na classe *Goal*. Segundo GUINELLI; PANTOJA; CHOREN (2015), todas essas modificações ocorreram com a intensão de estruturar a codificação do arquivo organizacional Moise+, considerando as especificações funcionais, estruturais e normativas.

GUINELLI; PANTOJA; CHOREN (2015) explicam que a classe *Link* foi adicionada para representar a relação entre grupos e funções da especificação do grupo do modelo estrutural. Já o *monitoringScheme* foi criado para permitir a identificação do sistema de monitoramento de cada *Scheme* existente. Sobre os auto-relacionamentos, *superType* e *subGoal* foram criados para representar a hierarquia de funções e a hierarquia de objetivos da especificação funcional, respectivamente.

Posterior a realização das modificações no metamodelo do JaCaMo, viabilizou-se o mapeado dos conceitos de Tropos presentes no FAML para o metamodelo do JaCaMo. Para isso, utilizou-se o mesmo processo anteriormente realizado, ou seja, o *plug-in QVT Operationals* com as especificações QVT, a fim de mapear os conceitos presentes no FAML para o metamodelo JaCaMo. Esse passo corresponde a transformação T1, vista na Figura 8. A partir desse modelo, a solução proposta passa para a fase *Plataform Specific Model* (PSM).

Após a geração do metamodelo do JaCaMo e a especificação da solução para a fase PSM, o passo restante a ser realizado é a transformação T2, conforme Figura 8. Para realizá-la, foi feito o uso de um *plug-in* chamado Acceleo, que utiliza a especificação Model-To-Text. Este *plug-in* recebe como entrada o metamodelo especificado no Ecore, faz o mapeamento deste, e gera códigos para Jason/Moise+.

A solução proposta viabilizou a geração automática de código de forma independente, permitindo que a geração ocorresse através de um ambiente de modelagem gráfica, no

caso deste trabalho, o TAOM4E. Este processo aconteceu através da utilização do MDA desenvolvido no trabalho de PANTOJA; CHOREN (2013). Além disso, o trabalho de GUINELLI; PANTOJA; CHOREN (2015) separou as especificações do SMA em diferentes níveis hierárquicos, viabilizando a independência entre os modelos utilizados.

GUINELLI; PANTOJA; CHOREN (2015) explicam que a utilização de metamodelos viabilizam diversas ligações para soluções já existentes. Além disso, a metodologia MDA integrada a ferramenta TAOM4E não visa projetar mais uma ferramenta para modelagem de SMA. O objetivo do trabalho é a integração de soluções e transformações de diferentes modelos de abstração até a obtenção de uma implementação de software, que segundo (GUINELLI; PANTOJA; CHOREN, 2015), é o foco principal de um MDA.

3.1.1 Análise do Trabalho

Embora o trabalho de GUINELLI; PANTOJA; CHOREN (2015) esteja bem fundamentado e apresente a solução para um problema em potencial, ele demonstra uma deficiência em relação a modificação da modelagem do Tropos para as demais transformações.

Através de uma análise mais aprofundada sobre as transformações realizadas e as ferramentas utilizadas, percebe-se que não foi levada em consideração a arquitetura de agentes BDI necessária para desenvolvimento de agentes na plataforma Jason. GUINELLI; PANTOJA; CHOREN (2015) delegaram a responsabilidade de definição da arquitetura do agente para o MDA, especificamente para o FAML, em virtude do mesmo ser aderente a metodologia de desenvolvimento utilizada.

Entretanto, ao realizar a transformação de TAOM4E para o MDA, nota-se a ausência de informações a respeito dos agentes modelados. Esse fato traz incompletude da transformação. Para solucionar o problema, GUINELLI; PANTOJA; CHOREN (2015) definiram que toda comunicação inicial entre um agente e outro se torna uma intenção de envio de mensagens, esperando que um agente informe isso ao outro. Embora seja uma solução, ela não é a ideal, visto que para funcionar adequadamente, é necessário que um agente informe o outro em relação a comunicação que irá ocorrer entre eles. O certo seria essa intenção já vir previamente definida no momento da geração do código do JaCaMo para o Jason/Moise+, respeitando a arquitetura BDI.

Além da incompletude em relação as informações dos agentes para a geração de códigos para o Jason, o trabalho proposto por GUINELLI; PANTOJA; CHOREN (2015) fez adaptações para que não fosse necessário a utilização das ferramentas Moise+ e CartAgO, inviabilizando que a modelagem do SMA apresentasse uma solução possível de ser implementada sobre todas as dimensões que compõem um SMA.

3.2 Método Prometheus AEOLus

Segundo UEZ (2013), a metodologia Prometheus AEOLus foi desenvolvida baseando-se em duas tecnologias: a metodologia Prometheus e o *framework* JaCaMo. A autora afirma que o propósito da metodologia é fazer uma extensão da metodologia Prometheus de modo a incluir a especificação de Ambiente e Organização, recebendo por isso o nome de Prometheus AEOLus.

De acordo com UEZ (2013), a utilização do *framework* JaCaMo nesta extensão serviu de base para a inclusão dos aspectos relacionados ao ambiente e a organização como abstrações de primeira classe. Essa abordagem tende a facilitar a compreensão dos diferentes aspectos envolvidos em um SMA, aumentando a legibilidade e simplificando a alteração do sistema desenvolvido (UEZ, 2013). Outro ponto a destacar na utilização deste *framework* é a possibilidade de geração de código a partir dos *work products* desenvolvidos, na medida em que fornece para a modelagem do problema as mesmas abstrações utilizadas para o desenvolvimento de um SMA utilizando o *framework* JaCaMo.

O desenvolvimento inicial da metodologia preocupou-se em definir os conceitos relevantes sobre cada dimensão de um SMA e como ocorre o relacionamento entre os mesmos. Com base nesses conceitos, definiu-se metamodelos que servem como base para a utilização da metodologia. Para realizar essa definição entre os conceitos das duas tecnologias, foi feita uma comparação entre os metamodelos. Os metamodelos são explicados a seguir.

3.2.1 Metamodelos

3.2.1.1 Metamodelo do Ambiente

Na Figura 9 é apresentado o metamodelo do ambiente gerado a partir da combinação da metodologia Prometheus e o *framework* JaCaMo. A figura é dividida ao meio, sendo que do lado esquerdo, ficam os conceitos referentes ao ambiente definidos pelo JaCaMo. Do lado direito da figura ficam os conceitos da dimensão de agentes apresentados pela metodologia Prometheus, além de alguns conceitos que foram incluídos na dimensão de agentes (*ExternalAction*, *InternalAction* e *TriggerEvent*). Os conceitos de *Actor* e *Data* do Prometheus conflitam com os conceitos de ambiente do JaCaMo. No Prometheus, o conceito de *Actor* é utilizado para representar as entidades externas com as quais o sistema interage, como usuários ou outro sistemas. Já o conceito *Data*, representa as crenças, bem como informações persistentes, armazenadas em arquivos de texto ou banco de dados, e recursos com os quais o agente integrado, como impressoras, sensores, entre outros. Ambos os conceitos conflitantes são tratados na metodologia Prometheus AEOlus como *Artifacts*.

Ao analisar a Figura 9, percebe-se que há algumas semelhanças entre conceitos, embora estejam em dimensões diferentes (UEZ, 2013). Na figura esses conceitos foram

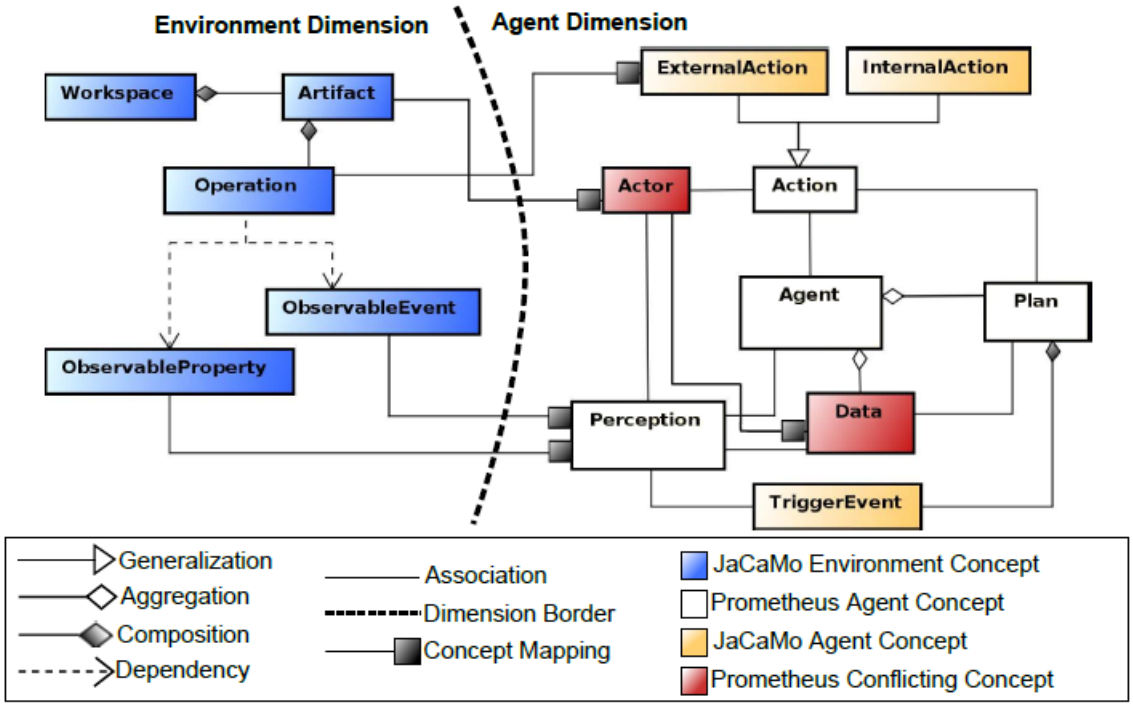


Figura 9: Metamodelo do Ambiente (UEZ, 2013).

unidos através de uma linha finalizada por um quadrado. Um exemplo é o conceito *ExternalAction* utilizado pelo metamodelo de agente, que pode ser compreendido como uma *Operation* quando observado no metamodelo do ambiente. Outros casos também apresentam essa peculiaridade, como a *Perception* no metamodelo do agente com a *ObservableProperty* ou um *ObservableEvent* no metamodelo do Ambiente, ou os conceitos de *Actor* e *Data* que representam entidades externos ao sistema na metodologia Prometheus e no metamodelo do ambiente são representados como um *Artifact*.

3.2.1.2 Metamodelo da Organização

Segundo UEZ (2013), o JaCaMo define a organização baseando-se em três pontos de vista: o estrutural, no qual são explicitados os *Roles* que os agentes podem assumir na organização e os *Groups* aos quais cada *Role* pertence; o funcional, que determina o *SocialScheme*, os *Goals* e as *Missions* do sistema; e o normativo, onde são definidas *Norms* que relacionam *Roles* e *Missions*.

Conforme UEZ (2013), a análise dos metamodelos permite visualizar que a integração entre a organização do JaCaMo e o agente Prometheus é relativamente simples. No Prometheus, os conceitos de *Goals* e *Role* já são utilizados. Os *Goals* determinam o que o sistema deve fazer. A definição desses é a base da modelagem do sistema nesse método. Os *Roles* no Prometheus são identificados a partir de *Goals* correlatos e são utilizados como base para definição dos Agentes que farão parte do sistema.

A Figura 10 permite visualizar que os conceitos de organização apresentados no meta-

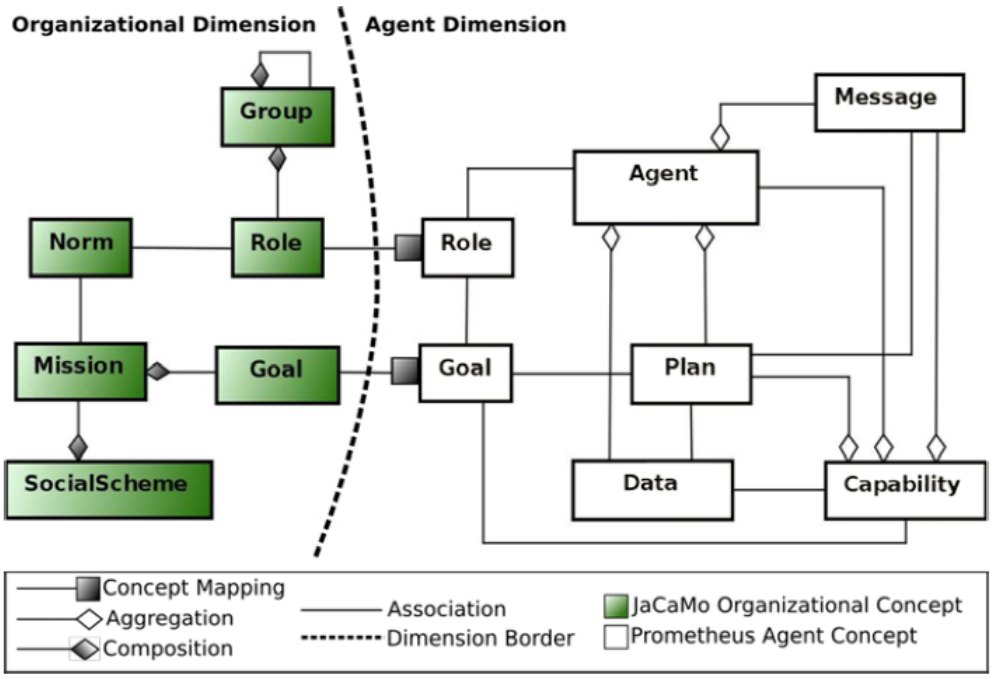


Figura 10: Metamodelo da Organização (UEZ, 2013).

modelo do JaCaMo podem ser incluídos no metamodelo do Prometheus sem que seja necessário alterar conceitos já existentes. Nessa figura, os conceitos referentes a organização definidos no metamodelo do JaCaMo são apresentados ao lado esquerdo, enquanto que os conceitos referentes aos agentes definidos no metamodelo do Prometheus são apresentados ao lado direito. Sendo assim, os conceitos organizacionais, *Group*, *Role*, *Goal*, *Norm*, *Mission* e *SocialScheme*, são apresentados ao lado esquerdo e os conceitos do Prometheus referentes à modelagem do agente, ou seja, *Role*, *Goal*, *Agent*, *Plan*, *Data*, *Message* e *Capability*, são apresentados ao lado direito.

Através da Figura 10 pode-se relatar a correlação entre os conceitos *Goal* e *Role* existentes em ambos os metamodelos. Essa semelhança é representada pela linha finalizada por um quadrado no final e indica que o *Role* assumido pelo agente possivelmente deve ter sido especificado pela organização. Da mesma maneira, todo *Goal* que deve ser atingido pela organização também deve ser delegado para os agentes individualmente.

3.2.1.3 Metamodelo Prometheus AEOlus

Segundo UEZ (2013), o metamodelo da metodologia Prometheus AEOlus foi definido com base nos metamodelos apresentados anteriormente, ambiente e organização, porém foram feitas algumas alterações que se fizeram necessárias. O metamodelo da metodologia é apresentado na Figura 11.

Na metodologia Prometheus AEOLus, conforme UEZ (2013), os *Agents* podem executar dois tipos diferentes de *Action*: *ExternalAction* e *InternalAction*. As *Actions* do tipo *ExternalAction* são executadas pelo ambiente. Esses conceitos foram inclusos na

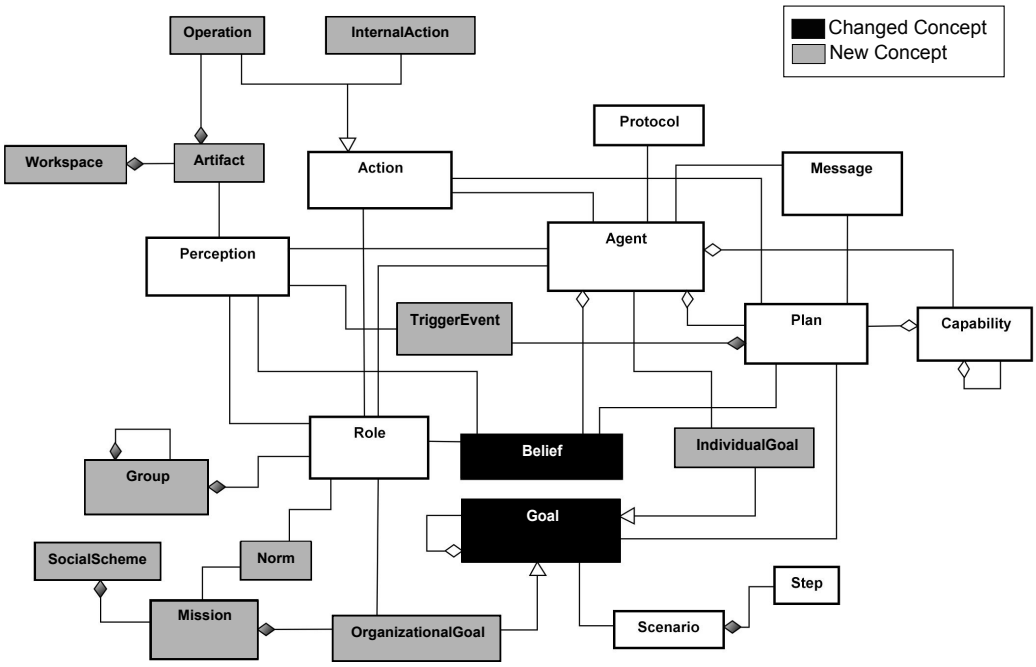


Figura 11: Metamodelo da Metodologia Prometheus AEOLus (UEZ, 2013).

metodologia Prometheus AEOlus, não sendo utilizados na metodologia Prometheus. Os conceitos *Workspace*, *Artifact* e *Operation* são conceitos relacionados ao ambiente que foram definidos com base no metamodelo da metodologia JaCaMo. Segundo UEZ (2013), uma *ExternalAction* consiste na execução de uma *Operation* em um *Artifact* no ambiente do SMA. Além disso, como consequência da execução de uma *Operation*, pode-se transformá-la em uma *Belief* de um *Agent* ou um *TriggerEvent* em um *Plan*. O *TriggerEvent* não fazia parte do metamodelo da metodologia Prometheus. Já o conceito *Belief*, foi definido a partir da alteração do conceito *Data* que existia em Prometheus e que representa, além das crenças, recursos externos ao sistema que eram utilizados pelos agentes (UEZ, 2013).

Nesta mesma Figura, existem conceitos relativos a organização de um SMA, sendo que *Group*, *Norm* e *SocialScheme* foram definidos com base no metamodelo do JaCaMo. O agente só pode assumir um *Role* se este foi definido na organização do SMA (UEZ, 2013). Ao assumir um *Role*, o agente torna-se membro de um *Group* e deve respeitar as *Norms* definidas para o papel que ele desempenhará. O *Agent* pode ter *Goal* definido por intermédio da organização, através de uma *Mission*, e também *Goals* que não estão ligados à organização. Em razão disto, o conceito de *Goal* foi especializado em dois subconceitos: *OrganizationalGoal*, que representa os objetivos definidos pela organização e atribuídos aos agentes através dos papéis que assume, e *IndividualGoal*, que representam os objetivos que o agente pode ter independente da organização (UEZ, 2013).

3.2.2 Linguagem de Modelagem

3.2.2.1 Notação e Conceitos

A Figura 12 apresenta a notação utilizada pela metodologia Prometheus AEOlus (UEZ, 2013). Cada conceito é representado por um símbolo único, no qual está inserido o nome da entidade.

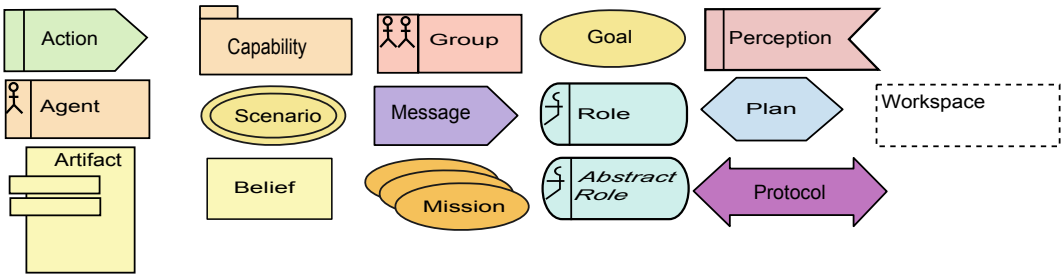


Figura 12: Notação que representa os conceitos utilizados na metodologia Prometheus AEOlus (UEZ, 2013).

Os principais conceitos representados pelo método são (UEZ, 2014):

- Ação (*Action*): Ações são coisas que os agentes são capazes de realizar. Elas podem ser "externas", quando alteram o ambiente, ou internas, quando alteram somente as crenças do agente, sem modificar nada no âmbito do ambiente do SMA.
- Agente (*Agent*): Um agente é uma entidade que possui diversas características, dentre elas, autonomia, reatividade e proatividade.
- Artefato (*Artifact*): Artefatos são recursos ou ferramentas que estão no ambiente e provêem operações que podem ser utilizadas pelos agentes.
- *Capability*: Uma *capability* é uma biblioteca de planos que podem ser compartilhados por mais de um agente.
- Cenário (*Scenario*): Os cenários descrevem o comportamento esperado do sistema em uma determinada situação, seja em sua execução normal ou em algum fluxo alternativo.
- Crença (*Belief*): Uma crença é uma informação que o agente possui sobre o ambiente, sobre outros agentes ou até mesmo sobre si mesmo.
- Grupo (*Group*): Um grupo é um conjunto de agentes que possuem afinidades ou objetivos similares.
- Mensagem (*Message*): A mensagem é a forma básica de comunicação entre os agentes.

- Missão (*Mission*): Uma missão é um conjunto coerente de objetivos que deve ser atingido por um agente.

- Objetivo (*Goal*): Os objetivos determinam aquilo que o sistema - como um todo - e os agentes - individualmente - devem fazer.

- Papel (*Role*): Um papel define um conjunto de comportamentos que é esperado do agente enquanto este fizer parte de uma organização.

- Papel Abstrato (*Abstract Role*): Papéis abstratos são utilizados somente para auxiliar na especificação da organização, não sendo possível nenhum agente exercer diretamente esse tipo de papel. A notação do papel abstrato é semelhante a utilização para o papel, com exceção de que no papel abstrato a entidade é descrita em itálico.

- Percepção (*Perception*): Uma percepção é uma informação que o agente recebe sobre o estado do ambiente.

- Plano (*Plan*): Um plano é um conjunto ordenado de ações que o agente deve executar para satisfazer um objetivo.

- Protocolo (*Protocol*): Um protocolo é uma sequência de mensagens trocadas entre dois ou mais agentes, em forma de conversação.

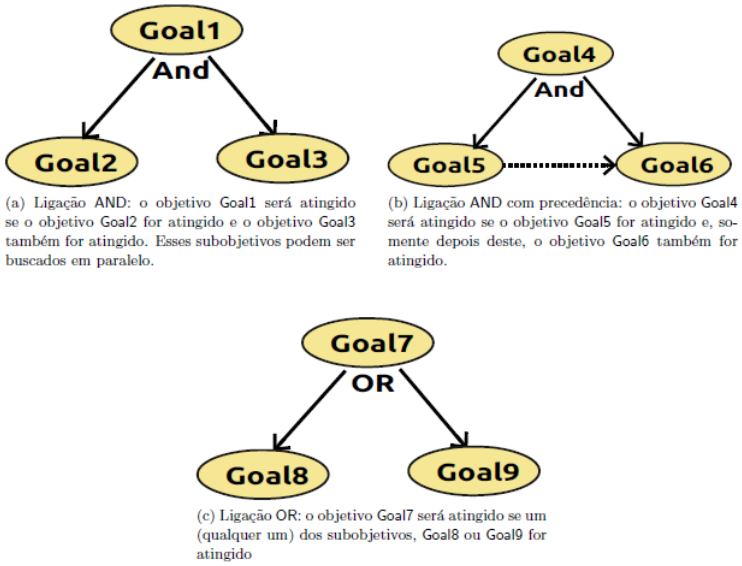
- *Workspace*: Workspaces são contêineres lógicos que contêm um conjunto de artefatos.

3.2.3 Diagramas e Descritores

Conforme UEZ (2014), a metodologia Prometheus AEOlus contêm um conjunto com doze diagramas e oito descritores, apresentados a seguir.

3.2.3.1 Diagrama de Objetivos do Sistema

Segundo UEZ (2013), o diagrama de objetivos do sistema, ilustrado na Figura 13, permite a análise dos objetivos em forma de árvore de decomposição *AND/OR*. Nesse diagrama, os objetivos são refinados gerando subobjetivos que podem ser interligados através de ligações do tipo *AND* ou ligações do tipo *OR*. A ligação *AND* indica que o objetivo será atingido se todos os seus subobjetivos também forem atingidos. Já a ligação do tipo *OR* indica que o objetivo será atingido assim que um dos seus subobjetivos forem atingidos. Nos caso das ligações *AND*, também é possível expressar a ordem na qual os subobjetivos devem ser atingidos, através de uma seta pontilhada que parte do subobjetivo que deve ser atingido antes até o subobjetivo que a sucede (UEZ, 2013).



(a) Ligação AND: o objetivo Goal1 será atingido se o objetivo Goal2 for atingido e o objetivo Goal3 também for atingido. Esses subobjetivos podem ser buscados em paralelo.

(b) Ligação AND com precedência: o objetivo Goal4 será atingido se o objetivo Goal5 for atingido e, somente depois deste, o objetivo Goal6 também for atingido.

(c) Ligação OR: o objetivo Goal7 será atingido se um (qualquer um) dos subobjetivos, Goal8 ou Goal9 for atingido

Figura 13: Exemplo Diagrama de Objetivos do Sistema (UEZ, 2014).

3.2.3.2 Diagrama de Visão Geral dos Papéis

Segundo UEZ (2014), o diagrama de Visão Geral dos Papéis tem como objetivo representar as ligações realizadas entre papéis e as percepções e ações que cada papel recebe ou executa. A Figura 14 ilustra um exemplo deste diagrama. Conforme UEZ (2014), a ligação de um papel com uma percepção acontece por intermédio de uma seta simples partindo da percepção até o papel. Para ligar o papel com uma ação, a seta simples parte do papel até a ação.

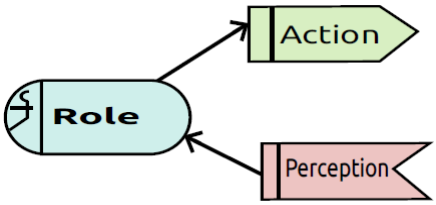
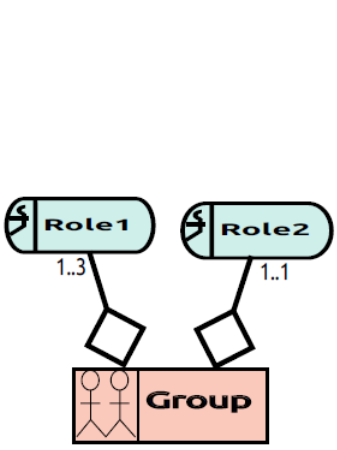


Figura 14: Exemplo Diagrama de Visão Geral de Papéis (UEZ, 2014).

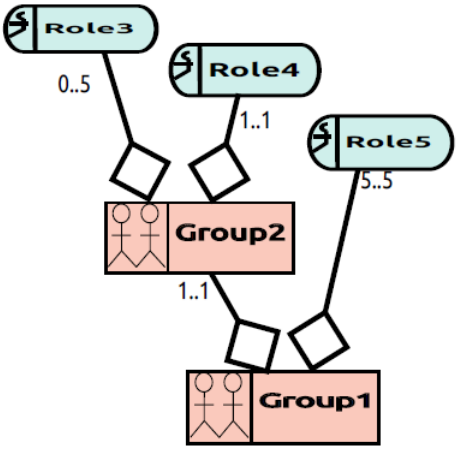
3.2.3.3 Diagrama Estrutural

Segundo UEZ (2014), neste diagrama são definidos os grupos, os papéis que compõem cada grupo e as ligações existentes entre esses grupos, ou seja, este diagrama tem como objetivo descrever a estrutura da organização. Os grupos são conjuntos de agentes que possuem afinidades mais refinadas e objetivos mais próximos. A Figura 15 ilustra um exemplo deste diagrama. Segundo UEZ (2013), neste diagrama os grupos são descritos com base nos papéis que podem ser assumidos pelos agentes na organização e das cardinalidades (mínimas e máximas) de instâncias de agentes que podem assumir cada papel. Complementarmente, é possível especificar subgrupos e cardinalidades para

esses subgrupos. Um grupo está bem formado quando todas as cardinalidades dos papéis e subgrupos que o compõem forem respeitadas (UEZ, 2014).



(a) O grupo Group é formado pelos papéis Role1 e Role2. Para ser bem formado, no mínimo um e no máximo três agentes devem assumir o papel Role1 e um único agente deve assumir o papel Role2.



(b) O grupo Group1 é formado pelo papel Role5, que deve ser assumido por cinco agentes, e pelo subgrupo Group2. Nesse subgrupo, um agente deve assumir o papel Role4. O papel Role3 pode não ser assumido por nenhum ou por até cinco agentes.

Figura 15: Exemplo Diagrama Estrutural (UEZ, 2014).

Neste diagrama é possível expressar quatro tipos de ligações que podem ser utilizadas para descrever os relacionamentos entre os papéis:

- Conhecimento: Uma ligação de conhecimento entre o papel X e Y indica que os agentes que assumem o papel X tem conhecimento sobre os agentes que assumem o papel Y.
- Comunicação: Uma ligação de comunicação entre o papel X e Y indica que os agentes que assumem o papel X podem se comunicar com os agentes que assumem o papel Y.
- Autoridade: Se os agentes que assumem o papel X forem ligados aos agentes que assumem o papel Y através de uma ligação de autoridade, os agentes que compõem o papel X passam a ter autoridade sobre os que compõem o papel Y.
- Compatibilidade: Se o papel X for ligado ao papel Y por intermédio de uma relação de compatibilidade, significa que agentes que assumem o papel X podem também assumir o papel Y.

UEZ (2014) explica que essas ligações são representadas por tipos diferentes de setas. Uma seta simples indica conhecimento; um círculo preenchido indicada comunicação; um triângulo preenchido representa autoridade; e um losango preenchido indica compatibilidade. Além disso, é necessário delimitar o escopo dessas ligações. Essas ligações

podem ter validade somente para agentes que façam parte do mesmo grupo (ligação intra-grupo) ou podem também ser válidas mesmo que os agentes pertençam a grupos distintos (ligação inter-grupo). Para representar uma ligação do tipo intra-grupo é utilizada linhas pontilhadas, enquanto que para representar ligações inter-grupo são utilizadas linhas contínuas.

Outra característica que pode ser expressada através do diagrama estrutural é a relação de herança entre papéis. Conforme UEZ (2013), essa relação é semelhante a expressada na orientação a objetos. Neste caso, o diagrama possibilita que sejam modelados papéis que podem ser herdados por outros papéis, dizendo que um papel é uma especialização do outro papel ao qual foi herdado as características. Além disso, a metodologia possibilita, para facilitar a especificação da organização, que sejam criados papéis abstratos. Segundo UEZ (2014), esses papéis não podem ser diretamente exercidos por nenhum agente, entretanto, podem conter características que são herdadas por outros papéis.

3.2.3.4 Diagrama de Missões

Segundo UEZ (2013), o diagrama de Missões tem como finalidade demonstrar o agrupamento de objetivos em missões, conforme Figura 16. Neste diagrama, as missões são conjuntos de objetivos correlatos que são atribuídos a um agente através do papel que este exercer. Vale ressaltar que quando o agente se compromete com uma missão, é necessário que ele atinga todos os objetivos que compõem essa missão para ter êxito (UEZ, 2014). Para modelar o diagrama de missões é utilizado como base o diagrama de objetivos do sistema, conectando cada missão ao objetivo que a compõem.

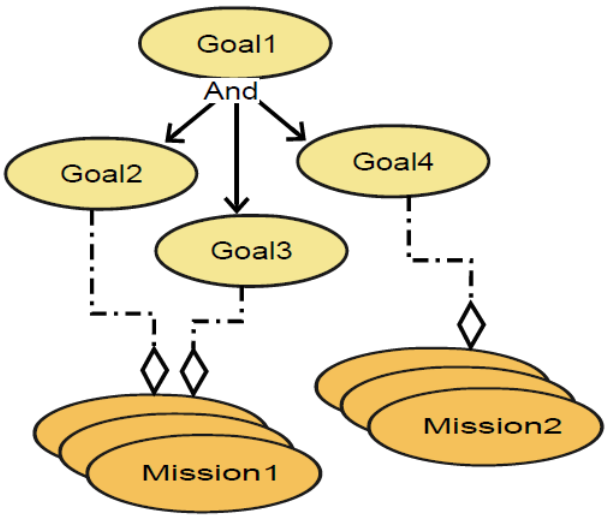


Figura 16: Exemplo Diagrama Missão (UEZ, 2014).

3.2.3.5 *Diagrama Normativo*

Segundo UEZ (2013), esse diagrama tem como objetivo apresentar as normas que definem quais missões cada papel poderá ou deverá se comprometer. Isto significa que quando um agente assumir um papel, este deve passar a ser responsável pelas missões que foram atribuídas para aquele papel. Existem dois tipos de ligações possíveis entre papéis e missões, conforme Figura 17. O primeiro tipo, intitulada ligação de obrigação, determina que o agente que assumir o papel será obrigado a se comprometer com aquela missão. Conforme a Figura, essa ligação é representada por uma seta preenchida. O segundo tipo, intitulada ligação de permissão, indica que o agente que assumir o papel tem a permissão de se comprometer com aquela missão. Por ser uma ligação mais abrangente, é representada por uma seta simples. Além disso, a relação de obrigação possibilita também a de permissão, ou seja, um papel obrigado a se comprometer com uma missão, também tem permissão para se comprometer com a mesma missão.

(a) Obrigação: Qualquer agente que assumir o papel Role1 passa a ter obrigação de assumir a missão Mission1.

(b) Permissão: Qualquer agente que assumir o papel Role2 poderá assumir a missão Mission2.

Figura 17: Ligações possíveis do Diagrama Normativo (UEZ, 2014).

3.2.3.6 *Diagrama de Relacionamento entre Papéis e Agentes*

Segundo UEZ (2013), o diagrama de Relacionamento entre Papéis e Agentes demonstra quais papéis cada tipo de agente pode assumir. Neste diagrama também pode se definir quais agentes são efetivamente implementados. Um exemplo do diagrama é ilustrado na Figura 18. A ligação entre o papel e o agente é feita por intermédio de uma seta simples que liga o agente no papel.

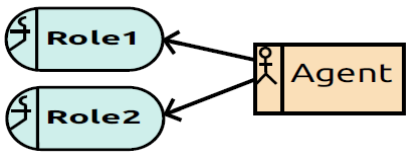


Figura 18: Exemplo Diagrama de Relacionamento Papel Agente (UEZ, 2013).

3.2.3.7 *Diagrama de Visão Geral do Ambiente*

Segundo UEZ (2013), o diagrama de Visão Geral do Ambiente tem como finalidade apresentar os artefatos que fazem parte do ambiente, as ações que podem ser executadas sobre esse artefato e as percepções que este artefato provê para os agentes. Neste diagrama

os artefatos ficam agrupados em *workspaces* e, para ocorrer a interação com um artefato, o agente precisa entrar em um *workspace*. Para ligar as ações aos artefatos e os artefatos as percepções são utilizadas setas simples.

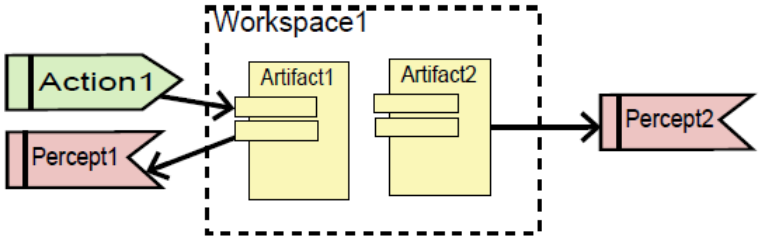
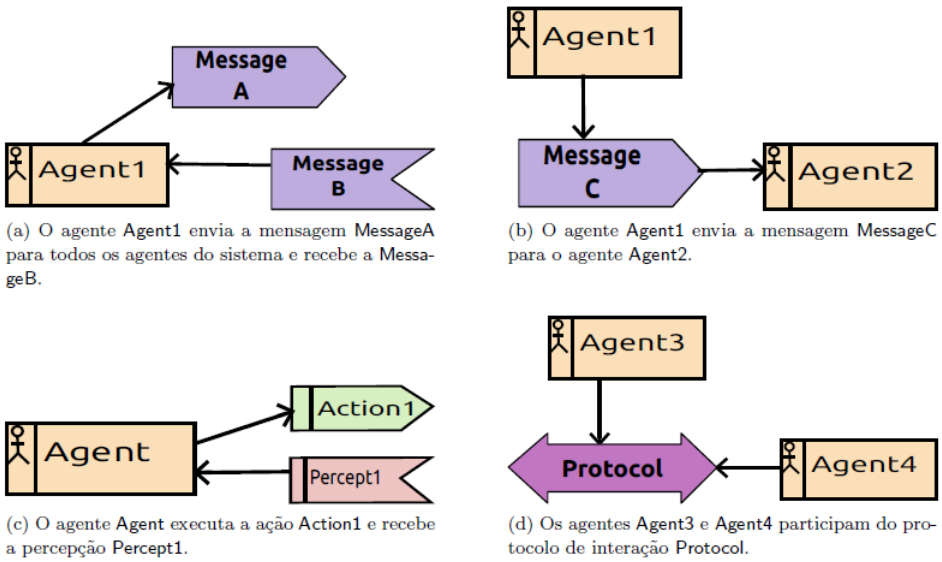


Figura 19: Exemplo Diagrama de Visão Geral da dimensão do Ambiente (UEZ, 2014).

3.2.3.8 Diagrama de Visão Geral do Sistema

Segundo UEZ (2013), o Diagrama de Visão Geral do Sistema tem como propósito demonstrar a estrutura global do sistema. Sendo assim, este diagrama demonstra agentes, suas ações e percepções e as interações entre esses agentes. A forma de interação dos agentes é por meio de troca de mensagens e de protocolos de comunicação. A Figura 20 ilustra as ligações permitidas neste diagrama. Uma seta partindo de um agente e indo até uma ação significa que o agente executa a ação e uma seta partindo de uma percepção até um agente indica que essa percepção é recebida pelo agente. Da mesma maneira, para saber se o agente enviou ou recebeu uma mensagem, basta verificar a direção da seta, sendo que quando o agente envia a mensagem a seta parte do agente em direção à mensagem. Já quando o agente recebe uma mensagem, a seta parte da mensagem em direção ao agente.



(a) O agente Agent1 envia a mensagem MessageA para todos os agentes do sistema e recebe a MessageB.

(b) O agente Agent1 envia a mensagem MessageC para o agente Agent2.

(c) O agente Agent executa a ação Action1 e recebe a percepção Percept1.

(d) Os agentes Agent3 e Agent4 participam do protocolo de interação Protocol.

Figura 20: Ligações permitidas no Diagrama de Visão Geral do Sistema (UEZ, 2014).

Para representar o envio de mensagens para todos os agentes (envio em *broadcast*) basta enviar a mensagem e não ligar a nenhum receptor. Pode-se também identificar os protocolos de comunicação. Para isso, é necessário apenas que seja ligado os protocolos aos agentes que deles participam.

3.2.3.9 Diagrama Descrição dos Protocolos

Segundo UEZ (2014), o diagrama de Descrição de Protocolos é usado para representar possíveis sequências de mensagens, em forma de conversação, trocadas entre agentes que compõem o SMA. A descrição de protocolos é feita com base no diagrama de Sequência e Eventos da linguagem AUML, onde o protocolo é representado por um conjunto de linhas da vida, cada qual representando um agente.

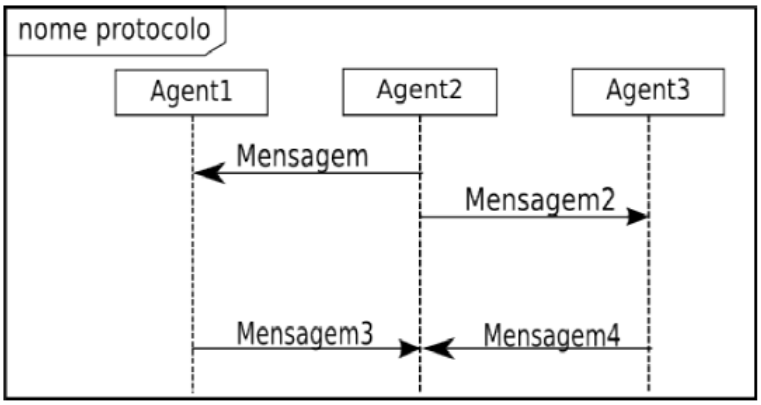


Figura 21: Exemplo de Diagrama de Descrição de Protocolos (UEZ, 2014).

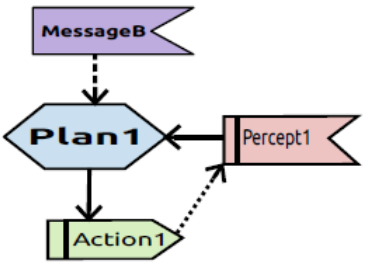
A Figura 21 apresenta um exemplo deste diagrama. No topo da linha da vida está o nome do agente que ela representa. As mensagens são simbolizadas por setas nomeadas entre as linhas da vida, mostrando que em determinados momentos, uma mensagem foi transmitida de um agente a outro (UEZ, 2014). A linha da vida do topo (nome do agente) até a base, representa o tempo. Além disso, protocolos podem também conter caixas onde são descritos comportamentos alternativos ou opcionais.

3.2.3.10 Diagrama de Visão Geral do Agente

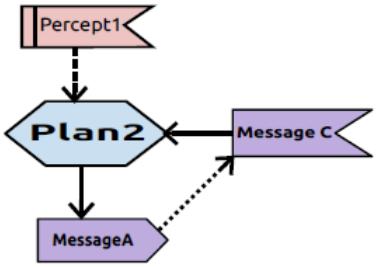
Segundo UEZ (2014), o diagrama de Visão Geral do Agente descreve o agente internamente, ou seja, detalhando seus planos, suas habilidades e suas crenças, bem como as *capabilities* que o agente pode ter. Os planos tem como finalidade indicar quais ações o agente deve executar para atingir um objetivo. Cada plano deve ter um evento *trigger* que iniciará a execução do plano. Eventos *triggers* podem ser desde mensagens recebidas ou até mesmo uma percepção do ambiente que é ligado ao plano através de uma seta pontilhada. Conforme UEZ (2014), os planos podem conter envio de mensagens, ações, outras percepções recebidas ou crenças. Quaisquer desses elementos são ligados

ao plano através de setas simples e contínuas e são ligados entre si por setas pontilhadas que indicam a ordem em que essas ações devem ocorrer.

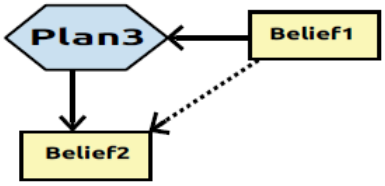
Conforme UEZ (2014), a forma como as crenças, as percepções e as mensagens se conectam com o plano alteram a maneira como esse elemento é interpretado. Por exemplo, crenças que não estão ligadas a um plano representam coisas que o agente tem conhecimento antes de iniciar o sistema, ou seja, crenças iniciais. Se existir uma seta que liga o plano até a crença, essa crença será incluída no repositório de crenças do agente (inclusão de crença). Caso ocorra uma ligação entre uma seta que inicia na crença e vai até o plano, isto significa que será feita uma consulta a base de crenças para verificar se a crença em questão já existe.



(a) Plano Plan1: iniciado quando o agente recebe a mensagem MessageB. O agente executará a ação Action1 e, após, fará uma consulta à sua base de crenças para verificar a informação Percept1.



(b) Plano Plan2: iniciado quando o agente recebe a percepção Perception1. O agente irá enviar a mensagem MessageA e adicionar o conteúdo da mensagem MessageC à sua base de crenças.



(c) Plano Plan3: inclui a crença Belief1 na base de crenças do agente e verifica se a crença Belief2 faz parte da base de crenças.

Figura 22: Ligações entre planos, mensagens, percepções e crenças que podem ser utilizadas tanto no Diagrama de Visão Geral do Agente quanto no Diagrama de *Capability* (UEZ, 2014).

As percepções e as mensagens recebidas durante a execução de um plano, sem ser as do evento *trigger* do plano, indicam uma consulta a base de crenças, no caso da percepção, ou uma adição de crença, no caso da mensagem. A Figura 22 ilustra as ligações possíveis de serem realizadas dentro dos planos.

3.2.3.11 Diagrama de Capability

Segundo UEZ (2014), uma *capability* é uma biblioteca de planos que pode ser utilizada pelo agente. No diagrama de Capability, essa biblioteca é descrita. As *capabilities* são utilizadas para modularizar o desenvolvimento do agente, permitindo que planos relacionados sejam agrupados e utilizados por mais de um agente, sem a necessidade de descrevê-los novamente. Esse diagrama segue as mesmas regras do Diagrama de Visão Geral de Agente, portanto as notações da Figura 22 são válidas também para este diagrama. Para criar este digrama, é necessário incluir uma *capability* no diagrama de Visão Geral do Agente. Porém, uma *capability* não precisa estar ligada a nenhum plano, além de ser possível incluir uma *capability* dentro de outra, novamente reaproveitando os planos descritos.

3.2.3.12 Diagrama de Visão Geral de Análise

Segundo UEZ (2013), este diagrama na metodologia Prometheus era utilizado para representar a relação entre o sistema e os atores externos. Já no Prometheus AEOlus, o diagrama de visão geral de Análise tem uma função diferente, sendo utilizado para permitir a identificação dos cenários de uso do sistema. A Figura 23 apresenta um exemplo deste diagrama.

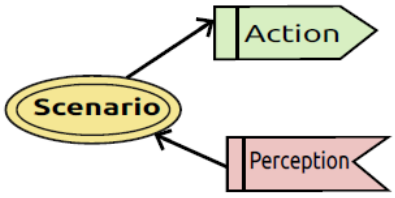


Figura 23: Exemplo de Diagrama de Visão Geral de Análise (UEZ, 2013).

No diagrama de visão geral de análise os cenários são definidos a partir das ações e percepções que são utilizadas em cada cenário (UEZ, 2013). Percepções utilizadas pelos cenários são ligadas através de uma seta simples que vai da percepção até o cenário. Da mesma maneira, uma seta simples entre o cenário e uma ação representa que aquela ação é executada pelo cenário.

3.2.3.13 Descrição de Cenários

A Figura 24 apresenta o modelo para descrição de cenários. Segundo UEZ (2013), os cenários descrevem como as coisas devem acontecer em determinadas situações durante o uso do sistema. Portanto, além do comportamento normal, os cenários também podem descrever o que deve acontecer no caso de erros no sistema.

O descritor de cenários é composto pelo nome do cenário, uma descrição do contexto no qual este cenário se desenvolve, o gatilho que inicia a execução do cenário e

Nome do Cenário	
Descrição	Breve descrição do contexto do cenário.
Gatilho	Evento que dá início ao cenário
Passos	* Tipo de passo: Lista de passos que descreve como o sistema deve reagir. * Tipo de passo: Descrição do Passo * Papel: Papel envolvido com o cenário.

Figura 24: Modelo de Descrição de Cenários (UEZ, 2014).

um conjunto sequencial de passos que indicam como o sistema reagirá. Segundo UEZ (2013), esses passos podem ser desde ações, até percepções, objetivos ou outros cenários do sistema e compõem a parte mais importante de um cenário.

3.2.3.14 Descrição das Ações

A Figura 25 apresenta o modelo para descrição das ações. Este descritor é composto pelo nome da ação, uma descrição, a função que deve ser executada pela ação e os parâmetros que devem ser passado para essa função (UEZ, 2014).

Nome da Ação	
Descrição	Breve descrição da ação. Pode conter informações sobre o motivo da ação, o que espera que ela faça, etc.
Função	Chamada da função que executa a ação, juntamente com seus parâmetros. Essa ação pode ser tanto uma ação interna quanto uma ação externas.

Figura 25: Modelo de Descrição das Ações (UEZ, 2014).

3.2.3.15 Descrição das Percepções

A Figura 26 apresenta o modelo de descritor das percepções. Conforme UEZ (2014), para descrever as percepções podem ser informados o nome, uma descrição e qual é a informação que essa percepção carrega. Normalmente, a informação de uma percepção tem a forma propriedade (objeto), indicando que o objeto possui a propriedade. Este descritor também pode ter uma anotação, na forma de [fonte], que pode apresentar dados adicionais em relação a origem da percepção.

3.2.3.16 Descrição das Mensagens

A Figura 27 ilustra o modelo de descrição das mensagens. Segundo UEZ (2014), para descrever as mensagens podem ser informados o nome, uma descrição da mensagem, seu

Nome da Percepção	
Descrição	Breve descrição da percepção.
Informação	Informação carregada por essa percepção.

Figura 26: Modelo de Descrição das Percepções (UEZ, 2014).

propósito e o conteúdo da mensagem. O propósito representa o tipo desta mensagem, seguindo a regra presente na linguagem *Knowledge Query and Manipulation Language* (KQML) (FININ et al., 1994).

Nome da mensagem	
Descrição	Breve descrição da mensagem.
Propósito	Indica a performativa a ser utilizada. Pode ser tell, untell, achieve, unachieve, askone, askall, tellhow, untellhow e *askhow*
Conteúdo	Conteúdo da mensagem, tal e qual deve ser enviada ou recebido pelos agentes.

Figura 27: Modelo de Descrição das Mensagens (UEZ, 2014).

3.2.3.17 Descrição dos Planos

A Figura 28 ilustra o modelo de descrição dos planos. Segundo UEZ (2014), para descrever os planos é necessário informar o nome do plano, é recomendado indicar uma descrição do plano e o contexto do plano. O contexto tem como finalidade definir as pré-condições, ou seja, o que deve ser verdadeiro no ambiente para que o plano possa ser executado.

Nome do Plano	
Descrição	Breve descrição do Plano.
Contexto	Pré-condições para que o plano seja executado. Caso dois planos tenham o mesmo evento *trigger*, o contexto determina qual dos dois será executado.

Figura 28: Modelo de Descrição dos Planos (UEZ, 2014).

3.2.3.18 Descrição das Crenças

A Figura 29 ilustra o modelo de descrição das crenças. Segundo UEZ (2014), para descrever as crenças deve-se informar o nome, uma descrição da crença e a informação inicial que aquela crença possui. Esta informação é representada na forma propriedade (objeto), indicando que o objeto possui a propriedade. Uma anotação, no formato [anota-

cao] pode ser adicionada à crença. A finalidade dessas anotações é apresentar informações adicionais a respeito da crença.

Nome da Crença	
Descrição	Breve descrição da Crença.
Informação Inicial	Informação armazenada pela crença na forma propriedade(objeto).

Figura 29: Modelo de Descrição de Crenças (UEZ, 2014).

3.2.3.19 Descrição dos Agentes

A Figura 30 ilustra o modelo de descrição dos agentes. Segundo UEZ (2014), a descrição dos agentes visa definir o número de instâncias que será inicializado para cada agente ao início do sistema. Para atingir esse objetivo, o descritor do agente possui as informações do nome e descrição do agente, além do campo cardinalidade. Caso não seja informada a cardinalidade, por padrão, somente uma instância do agente será iniciada. No caso de ser informado cardinalidade com valor 0, significa que nenhuma instância do agente será iniciada juntamente com o sistema.

Nome do Agente	
Descrição	Breve descrição do Agente.
Cardinalidade	Quantidade de instâncias do agente que será criada. Por padrão, o valor é 1. 0 indica que nenhuma instância deve ser criada.

Figura 30: Modelo de Descrição do Agente (UEZ, 2014).

3.2.3.20 Descrição dos Artefatos

A Figura 31 ilustra o modelo de descrição dos artefatos do ambiente. Segundo UEZ (2014), para descrever os artefatos é necessário informar seu nome, uma breve descrição, lista de parâmetros, a lista de operações disponibilizadas pelo artefato, as propriedades observáveis e os eventos observáveis.

3.2.4 Fases de Desenvolvimento da Metodologia

Conforme UEZ (2014), a metodologia Prometheus AEOlus possui quatro fases de desenvolvimento, ilustradas através da Figura 32. A fase inicial chama-se especificação do sistema (*system specfication*); a segunda fase intitula-se projeto de arquitetura (*architectural design*); a terceira fase é denominada projeto detalhado (*detailed design*); e por último, a fase de implementação (*implementation*).

Nome do Artefato	
Descrição	Breve descrição do artefato. Pode conter informações e novos detalhes sobre o artefato e seu uso.
Operações	Lista de operações disponibilizadas pelo artefato. Essas operações representam ações que o agente pode executar.
Parâmetros	Tipo e nome dos parâmetros que são informados na criação do artefato. Esses parâmetros devem ser separados por vírgula.
Propriedades Observáveis	Lista de propriedades observáveis disponibilizadas pelo artefato. Essas propriedades serão recebidas como percepções pelo agente.
Eventos Observáveis	Lista de eventos observáveis disponibilizados pelo artefato. Esses eventos serão recebidos pelos agentes como percepções.

Figura 31: Modelo de Descrição dos Artefatos do Ambiente (UEZ, 2014).

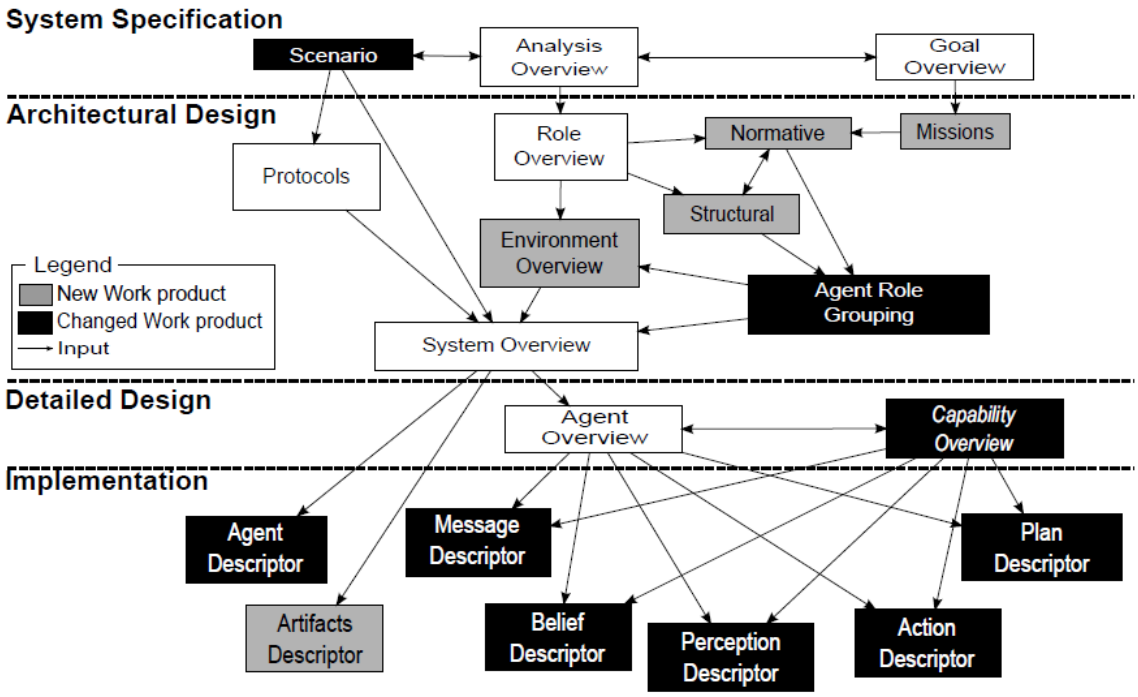


Figura 32: Fases de Desenvolvimento da Metodologia Prometheus AEOlus e *Work products* gerados em cada fase. (UEZ, 2014).

Segundo UEZ (2014), a fase de especificação do sistema visa criar uma definição clara e detalhada do sistema a ser desenvolvimento, descrevendo o que este deve fazer. Para tanto, é necessário identificar os cenários de uso do sistema e os objetivos do mesmo. Tanto os cenários de uso do sistema quanto os objetivos devem ser obtidos baseando-se na descrição inicial do sistema.

A segunda fase do processo de desenvolvimento da metodologia, chamada de projeto de arquitetura, tem como finalidade começar a definição acerca do comportamento do sistema. Por isso, nessa fase são identificados os tipos de agente que farão parte do sistema e os artefatos do ambiente. Além disso, são definidos os aspectos organizacionais e como os agentes interagem entre si. Esta interação é descrita baseando-se nas mensagens e nos protocolos. Para UEZ (2013), essas definições podem acarretar na revisão dos cenários e objetivos definidos na fase anterior.

A fase de projeto detalhado tem por objetivo descrever os planos dos agentes. É nesta etapa que os agentes são projetados com base na arquitetura BDI, ao contrário do que ocorre nas etapas anteriores, onde a arquitetura dos agentes ainda não é levada em consideração (UEZ, 2014). Ao final dessa fase, deve-se ter detalhado a estrutura interna dos agentes que compõem o SMA, especificando o que deve ser feito para que seus objetivos possam ser atingidos.

A última fase da metodologia é intitulada implementação. Nesta fase, alguns aspectos do sistema são refinados de modo que facilite a codificação do SMA projetado nas etapas anteriores da metodologia.

Segundo UEZ (2014), Prometheus AEOlus utiliza um processo iterativo que pode ser dividido em doze atividades: 1) especificação dos cenários; 2) especificação dos objetivos; 3) especificação das ações; 4) especificação das percepções; 5) especificação dos papéis; 6) especificação da estrutura organizacional; 7) especificação das missões; 8) especificação das normas; 9) especificação do ambiente; 10) especificação das interações; 11) especificação dos agentes; 12) especificação dos planos. Em cada uma dessas atividades, que podem ser desenvolvidas em mais de uma interação no decorrer das quatro fases de desenvolvimento, pelo menos um diagrama ou descritor é criado.

A tabela apresentada na Figura 33 ilustra o processo de execução das atividades, demonstrando as quatro fases de desenvolvimento, bem como as doze atividades que compõem essas fases e os passos que são realizados em cada uma delas. A execução dessas atividades envolve quatro passos: inicia, define, revisa e refina (UEZ, 2014). Neste contexto, segundo UEZ (2014), a palavra **inicia** indica que alguns conceitos podem ser pensados a partir dessa fase. Porém, essa ação é feita de forma conceitual e ainda não gera nenhum diagrama ou descritor. A palavra **define** indica que os diagramas e descritores referentes àquela atividade são desenvolvidos. A palavra **revisa** indica que os *work products* gerados nas fases anteriores para a atividade devem ser revisados, alterados ou complementados. Por último, a palavra **refina** indica que os *work products* são refina-

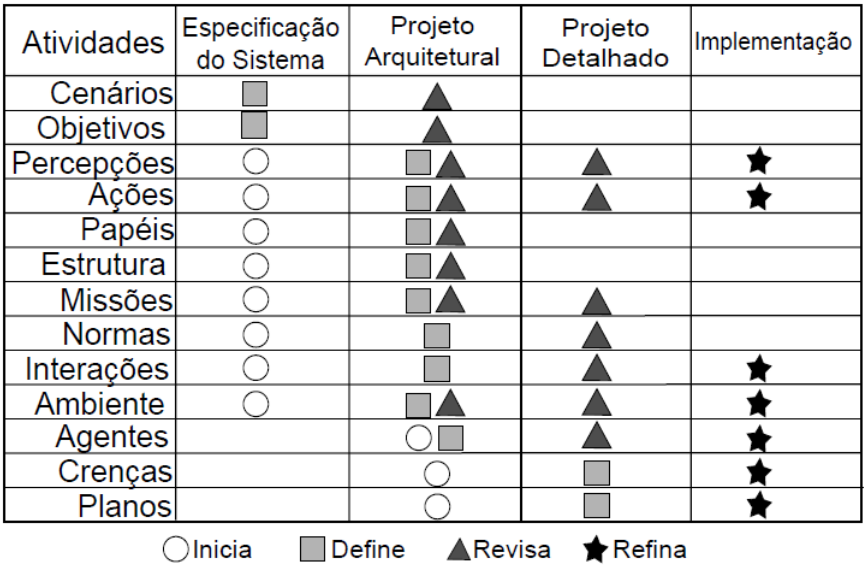


Figura 33: Atividades presentes em cada fase da metodologia Prometheus AEOlus (UEZ, 2014).

dos visando a transformação dos diagramas e descritores em códigos. Mais detalhes e exemplos de utilização da Metodologia Prometheus AEOlus podem ser encontrados nas referências (UEZ, 2013), (UEZ, 2014) e (UEZ; HÜBNER, 2014).

3.3 Considerações Finais

A seção 3.2 apresentou o trabalho realizado por UEZ (2013), o qual consiste em uma extensão da metodologia Prometheus (PADGHAM; WINIKOFF, 2002), com o objetivo de adicionar a especificação do problema conceitos referentes a organização e as características do ambiente que o SMA está inserido. Segundo ZANLORENCI; BURNETT (2003), essas dimensões são cruciais para o desenvolvimento de um SMA.

A metodologia Prometheus AEOlus, conforme já falado anteriormente, é uma extensão da metodologia Prometheus. Por esse motivo, o metamodelo de Prometheus AEOlus foi desenvolvido a partir do metamodelo do Prometheus (UEZ, 2013). Além disso, as notações gráficas utilizadas na metodologia também foram derivadas da metodologia Prometheus. A Figura 34 apresenta as devidas notações. Através desta figura percebe-se que, foram utilizadas as notações presentes no Prometheus e foram adicionadas algumas novas notações, permitindo uma visão mais ampla da metodologia.

Dentre essas novas notações mencionadas na metodologia Prometheus AEOlus, adicionou-se os conceitos de missão, papel abstrato, grupo, artefatos e *workspace*. Também foram incluídos novos diagramas para uma melhor especificação do SMA, sendo-os: Diagrama Estrutural, Diagrama de Missões, Diagrama Normativo e Diagrama de Visão Geral do Ambiente, bem como o formulário descritor de Artefatos. Em relação aos formulários descritores, esses foram sintetizados, de modo que só apareçam

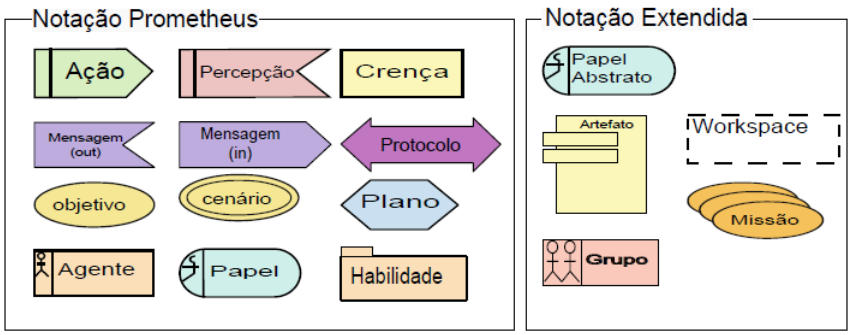


Figura 34: Notação da Metodologia Prometheus AEOlus (UEZ, 2013).

informações que sejam relevantes para a posterior geração automática de códigos.

Em uma análise preliminar do trabalho de UEZ (2013), pode-se concluir que o mesmo é enxuto, potencializando a especificação de um SMA no âmbito da organização e do ambiente, melhorando de forma significativa essa representação se comparado com as demais metodologias estudadas na Seção 2.4.1. Além disso, o trabalho tem como objetivo posterior a geração de códigos para o *framework* JaCaMo. Até o presente momento, não existe nenhuma metodologia que favoreça a geração automática de códigos para as demais dimensões de um SMA, fazendo com que o trabalho apresente uma melhora significativa na área em que está inserida.

4 MATERIAIS E MÉTODOS

Este capítulo apresenta as tecnologias que foram utilizadas para o desenvolvimento deste trabalho e está dividido como segue: A Seção 4.1 define os conceitos relacionados ao desenvolvimento de aplicações em diversas camadas interligadas; a Seção 4.2 descreve os procedimentos para o desenvolvimento de *plug-ins* para a IDE de programação Eclipse; A Seção 4.3 elenca os passos necessários para o desenvolvimento de *plug-ins* gráficos utilizando como base o *framework* de desenvolvimento de *plug-ins* gráficos chamado *Graphical Modelling Framework*.

4.1 Model-Driven Engineering

Segundo RUBE (2013), a *Model-Driven-Engineering* (MDE) ou Engenharia Dirigida a Modelos é um novo enfoque na área de ES. MDE utiliza modelos como artefatos de software. Modelos são um conjunto de elementos que descrevem um sistema (MELLOR, 2004) com grau de abstração maior que o próprio sistema (KLEPPE; WARMER; BAST, 2003). Para MILLER; MUKERJI et al. (2001), um modelo pode ser definido como uma especificação formal de uma função, estrutura e/ou comportamento de um sistema.

Para RUBE (2013), através da MDE é possível desenvolver otimizadores, validadores e compiladores de código baseado em modelos. O objetivo do modelo é facilitar o trabalho e reduzir o tempo de desenvolvimento e o número de erros no software gerado. Segundo RUBE (2013), a MDE tem algumas aplicações, sendo: *Model-Driven Development* (MDD), Engenharia Reversa, *Software Process Engineering* (SPE), *Domain Specific Language* (DSL) e *Model-Driven Integration* (MDI).

Nesse trabalho, foi utilizada a DSL. Segundo VAN DEURSEN; KLINT (2002), a DSL é uma linguagem específica de domínio que fornece uma notação adaptada para um domínio de aplicação e baseia seus conceitos e características em um domínio relevante. Assim, uma DSL é um meio para descrever e gerar membros de uma família de programas baseados em um domínio.

Conforme (RUBE, 2013), uma DSL serve para desenvolvimento de linguagens textuais e visuais. Ela é classificada em duas categorias: representação e implementação.

Na categoria representação, existe uma subclassificação do tipo textual ou visual. Já na categoria implementação, ela é subclassificada em interno e externo.

Para apoiar as fases de desenvolvimento de um DSL, é utilizado ferramentas para ajudar com este processo (RUBE, 2013). Segundo RUBE (2013), uma ferramenta que apoia as fases de desenvolvimento de DSL é a plataforma Eclipse. Ela contêm mecanismos que facilitam a edição de editores textuais e visuais.

4.2 Plataforma Eclipse para o Desenvolvimento de *Plug-ins*

A plataforma Eclipse é baseada em *plug-ins* que são utilizados para ampliar as funcionalidades da IDE (FOUNDATION, 2014a). Esses *plug-ins* são codificados na linguagem de programação Java e podem oferecer diversas modalidades de serviço, como biblioteca de códigos, guias de documentação ou uma extensão da própria plataforma (DESRIVIERES; WIEGAND, 2004).

As bibliotecas de código auxiliam os programadores com funcionalidades já implementadas e podem ser oferecidas pelo *plug-in* em formato de API. A documentação auxilia os desenvolvedores nos aspectos técnicos da linguagem de programação, bibliotecas de código, demonstrando os recursos disponíveis e sua aplicações. Na extensão da plataforma, os desenvolvedores utilizam os componentes e recursos da plataforma Eclipse como interface gráfica, mecanismos de interpretação textual e de compilação para desenvolverem soluções tecnológicas para outras finalidades, como uma ferramenta de apoio a uma nova linguagem de programação.

4.2.1 Arquitetura da Plataforma Eclipse

A arquitetura da plataforma Eclipse é composta por *Workspace*, *Workbench*, *JFace*, *Standard Widget Toolkit* (SWT), *Help*, *Team* e os demais *plug-ins* acoplados. Na Figura 35 são apresentados esses elementos e sua organização na plataforma.

Workspace

É o espaço de trabalho do desenvolvedor, uma pasta destinada ao armazenamento dos projetos de alto-nível, como projetos escritos em Java, PHP ou C++ (DESRIVIERES; WIEGAND, 2004).

Workbench

É o ambiente principal da ferramenta, construído com a utilização dos componentes *JFace* e SWT. Proporciona recursos de edição de código, execução, depuração, visualização das estruturas do projetos (localizados no *workspace*), adição de *plug-ins*, criação de novos projetos, entre outros.

JFace

Define um conjunto de ferramentas de interface de usuário, como área de preferências,

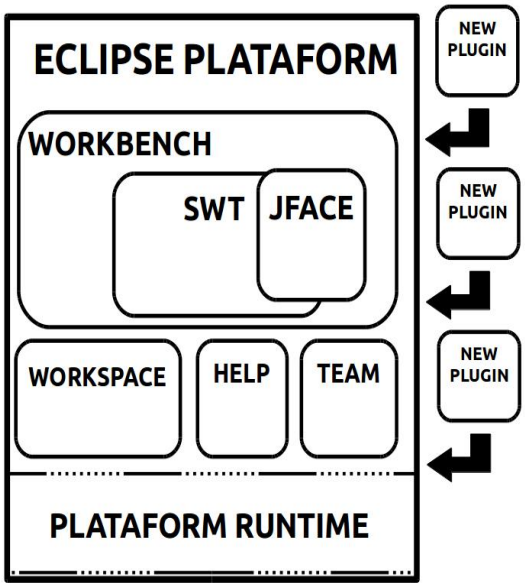


Figura 35: Arquitetura da Plataforma Eclipse (DESRIVIERES; WIEGAND, 2004).

módulo de configuração do ambiente de desenvolvimento e de projeto (DESRIVIERES; WIEGAND, 2004).

Standard Widget Toolkit

Biblioteca de componentes gráficos compatível com diferentes Sistemas Operacionais. Através dela é possível construir interfaces gráficas, como a *JFace*.

Help

Mecanismo de auxílio para utilizadores do ambiente, através dele é possível acessar guias e documentações referentes ao ambiente ou ao *plug-in* utilizado.

Team

Mecanismo que permite o versionamento de projeto através da inicialização, clone, configuração e gerenciamento de repositórios remotos.

Plataforma de tempo de execução

É acionada na inicialização da IDE. Sua tarefa é descobrir quais *plugins* estão disponíveis e interpretar o seus arquivos de configuração.

4.2.2 Arquitetura de um *Plug-in* Eclipse

Um *plug-in* é composto por um Java *Archive* (JAR) e um arquivo denominado *plug-in*, escrito em XML. Na Figura 36 é possível visualizar a organização desses elementos e seus itens.

Arquivo JAR

Um arquivo JAR contém os códigos java e os arquivos utilizados pela aplicação como imagens, dados e configurações. Os arquivos JAR são interpretados pela máquina virtual

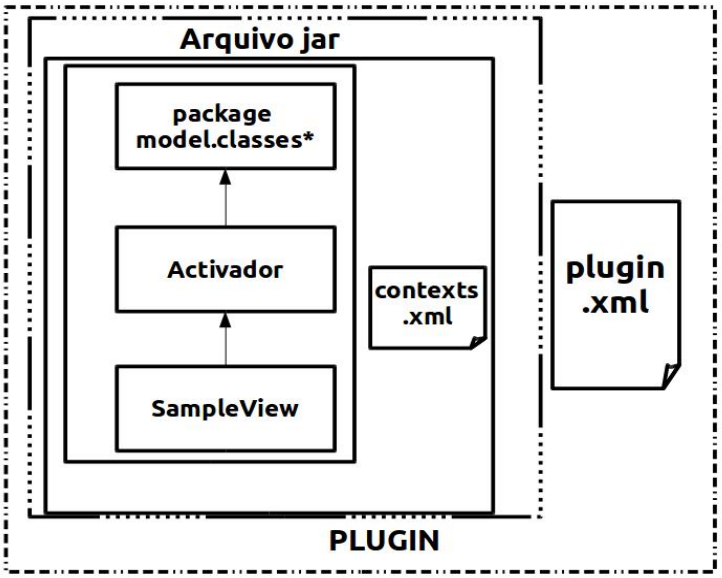


Figura 36: Arquitetura de um *plugin* eclipse.

Java, podendo ser utilizados como bibliotecas, sistemas executáveis ou componentes de software.

No desenvolvimento de *plug-ins*, o arquivo JAR é utilizado como componente de software que será acoplado ao ambiente de desenvolvimento Eclipse. O JAR *plug-in* pode ser dividido em classes java e arquivo *contexts*.

As classes são organizadas de acordo com suas finalidades, podendo ser definidos níveis arquiteturais para o controle dos recursos da aplicação, manipulação de componentes de interface gráfica e tratamento das operações que envolvem processamento de dados. Na classe *Activator* é realizado o controle do ciclo de execução do *plug-in* e o acesso das preferências do ambiente Eclipse para realizar possíveis configurações (MARKUS, 2010). A classe *SampleView* é utilizada para registrar e controlar os elementos da interface gráfica do ambiente. As classes de processamento de dados são utilizadas para gerenciar as informações da aplicação (MARKUS, 2010).

O JAR também é composto pelo arquivo *contexts* que é escrito em XML. Através dele é possível registrar as principais informações sobre o *plug-in*, para auxiliar o usuário no processo de instalação, utilização e configuração do mesmo no ambiente Eclipse (MARKUS, 2010).

Arquivo de Configuração

Conforme apresentado na figura a arquitetura do *plug-in*, existe um arquivo de configuração definido como “plugin.xml”. Nele se registram as dependências, os pacotes utilizados pelo *plug-in*, as extensões e os pontos de extensão (MARKUS, 2010).

4.2.3 *Plug-in Development Environment*

O PDE (*Plug-in Development Environment*) permite o desenvolvimento de *plug-ins* para a plataforma eclipse incluindo atividades de teste, depuração e implantação de *plug-ins* (FOUNDATION, 2014a). Esse *plug-in* é composto por componentes de interface gráfica de usuário, ferramentas da API e *Build*.

Os componentes de interface gráfica de usuário fornecem editores de codificação, assistentes de editor e manifesto, recursos de exportação, módulos de configuração de *plug-in*, etc. Já as ferramentas da API são destinadas ao versionamento, manutenção de incompatibilidades, otimizações das API fornecidas pelo *plug-in*. Além disso, o *Build* automatiza o desenvolvimento de *plug-ins*, gerando arquivos de configuração como "build.properties" e "plugin.xml". Esse componente é muito utilizado no empacotamento e distribuição do *plug-in* desenvolvido, que pode ser disponibilizado em um repositório remoto (FOUNDATION, 2014a).

4.3 Graphical Modelling Framework

Segundo FOUNDATION (2014b), o *Graphical Modelling Framework* (GMF) é um arcabouço para desenvolvimento de editores gráficos para modelos de domínio dentro da plataforma Eclipse. Ele foi baseado em outros dois arcabouços: *Graphical Editing Framework* (GEF), utilizado para a criação de editores gráficos genéricos; e *Eclipse Modelling Framework* (EMF), que permite ao desenvolvedor construir metamodelos e gerar códigos Java referidos ao mesmo.

Conforme GRONBACK (2009), EMF é um arcabouço de modelagem e geração de código para ferramentas de construção e outras aplicações baseadas em um modelo de dados estruturado. Para STEINBERG et al. (2008), EMF é uma estrutura de modelagem que explora as facilidades oferecidas pelo Eclipse. DUARTE (2015) explica que o EMF permite a modelagem do domínio utilizando um metamodelo próprio, mais simples do que a especificação *Meta-Object Facility* (MOF) da MDA, chamado Ecore.

Segundo KOLOVOS et al. (2010), GEF fornece o suporte gráfico necessário para a construção de um editor de diagramas. Diagramas trazem duas adições vitais para a experiência de modelagem: em primeiro lugar, o cérebro humano é muito melhor na interpretação de figuras, diagramas gráficos de texto, árvores ou tabelas; em segundo lugar, diagramas mostram múltiplas relações entre objetos muito melhor do que os formatos de texto ou tabela. Em FOUNDATION (2014c), GEF é definido como um fornecedor de tecnologia para criar ricos editores gráficos e visualizações.

4.3.1 Processo de Desenvolvimento de um Editor Gráfico utilizando o *plug-in* GMF Eclipse

Para explicar o processo de desenvolvimento do *plug-in* deste trabalho, utiliza-se a Figura 37. Esta figura demonstra o fluxo de trabalho necessário para gerar um editor gráfico utilizando o *plug-in* GMF do eclipse.

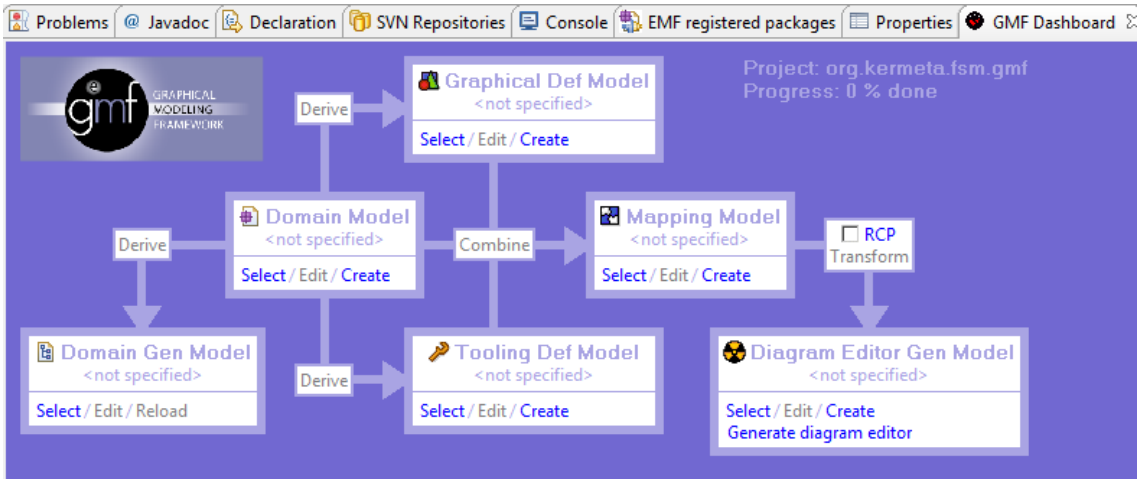


Figura 37: Dashboard do GMF Eclipse.

O processo de desenvolvimento consiste na concepção de seis arquivos. O primeiro chama-se Ecore e representa o *Domain Model*. É neste arquivo que é especificado o metamodelo do editor. O metamodelo serve para regulamentar todas as ações posteriores que o editor gráfico possa ter, por exemplo, quais entidades terão relacionamento e quais não terão. Embora não apareça na figura em questão, seguinte a modelagem do Ecore é possível gerar um arquivo **.diagram**, que representa respectivamente um diagrama de classes do metamodelo desenvolvido.

O próximo passo na elaboração do editor gráfico é derivar o *Domain Gen Model* do editor. Por esse motivo existe o botão *Derive* ao lado esquerdo da caixa *Domain Model* no *dashboard*. Conforme DUARTE (2015), posterior ao clique de derivar, é feita uma transformação do metamodelo gerado no Ecore para um metamodelo específico com extensão **.genmodel**, para que a partir deste, sejam gerados códigos na linguagem de programação Java. Em FOUNDATION (2014d) é explicado que o arquivo *genmodel* contém informações adicionais para a geração de código, como o caminho e informações do arquivo. Além disso, ele também contém o parâmetro de controle de como o código deve ser gerado na linguagem java.

O metamodelo estipulado no *Domain Model* é a base para a geração do restante dos artefatos do projeto de geração de um editor gráfico, utilizando o GMF eclipse. Os dois primeiros passos descritos anteriormente, correspondem ao EMF. Servem para regulamentar os passos restantes.

Posterior a elaboração do metamodelo e a geração de códigos em java, pode-se esco-

lher entre criar a paleta de componentes que será utilizada pelo editor gráfico ou o campo de desenho que servirá para modelar o diagrama. O botão *Derive* localizado na parte inferior e superior da caixa do *Domain Model* gera esses elementos, respectivamente.

Sobre o campo de desenho utilizado para modelar os diagramas, é derivado o *Domain Model* e corresponde a caixa chamada *Graphical Del Model*. O arquivo gerado nesta transformação tem a extensão **.gmfgraph**. Além disso, através da plataforma eclipse pode-se editar este arquivo, adicionando pontos provenientes de figuras geométricas em cada entidade derivada do Ecore (GRONBACK, 2009). Pode-se também alterar a cor da figura estipulada, adicionar caixa de texto para que o usuário informe o nome da entidade a ser modelada, dentre diversas outras possibilidades (GRONBACK, 2009). É neste arquivo que se configura toda a caixa de desenho, onde depois de pronta, será utilizada pelo usuário para modelar seu diagrama por intermédio do editor gráfico.

Em relação a paleta de componentes que irá do lado do campo de desenho do editor gráfico desenvolvido, este é concebido também por meio do *Domain Model*. Após a solicitação de derivação do *Domain Model* em relação a caixa chamada *Tooling Del Model*, é gerado um arquivo com a extensão **.gmftool**. Este arquivo contem a respectiva configuração de como serão disponibilizados os componentes gráficos para serem utilizados no campo de desenho do editor gráfico (GRONBACK, 2009). Sua configuração é simples, apenas pode-se alterar as figuras que correspondem aos respectivos campos e agrupar os campos em categorias diversas. Pode-se também criar categorias.

O próximo passo da elaboração da ferramenta é fazer a combinação entre todos os arquivos gerados anteriormente. Por intermédio do botão *Combine* é feita essa combinação, gerando um arquivo na caixa chamada *Mapping Model*, ilustrada na Figura 37. O arquivo gerado nessa caixa tem a extensão **.gmfmap**. Segundo GRONBACK (2009), talvez o mais importante de todos os modelos em GMF é o *Mapping Model*. Nele, elementos de definição do diagrama (nós e links) são mapeados para o modelo de domínio e elementos de ferramentas atribuídas. O *Mapping Model* representa o diagrama de definição real e é usado para criar um modelo de gerador. Tipicamente, existe um mapeamento um-para-um entre um modelo de mapeamento, o seu modelo de gerador e um campo de desenho em particular.

O último passo no desenvolvimento do editor gráfico é fazer a geração do arquivo que ficará responsável em fazer a junção de todos os outros componentes abordados no decorrer do desenvolvimento. O nome da caixa equivalente a esse processo é *Diagram Editor Gen Model* ilustrada na Figura 37. A partir do arquivo **.gmfmap** e arquivo **.ecore** é gerado o arquivo **.gmfgen**. Este arquivo é responsável por gerar as configurações de onde será gerado os códigos fontes para a execução do editor gráfico modelado no decorrer de todo esse processo. Posterior a configuração desse arquivo, inicia-se o processo de transformação, gerando o código fonte correspondente as etapas anteriores realizadas. No momento da finalização dessa transformação, tem-se os códigos fontes que poderão

ser executados para utilizar o editor gráfico, finalizando o processo.

5 DESENVOLVIMENTO DA FERRAMENTA PROMETHEUS AEOLUS

Este capítulo apresenta os procedimentos realizados para o desenvolvimento da ferramenta para apoio a metodologia Prometheus AEOlus.

A arquitetura geral da ferramenta Prometheus AEOlus é ilustrado na Figura 38. Cada pacote representa um *plug-in* gerado para incorporar funções a ferramenta desenvolvida. O pacote chamado *gerador*, representa o *plug-in* responsável pela geração de códigos da ferramenta. Dentro desse *plug-in*, são agrupadas as classes responsáveis por realizar a função designada ao *plug-in*. Seu desenvolvimento é descrito em detalhes na Seção 5.3. Esse pacote contém uma ligação de dependência com o pacote *diagram* devido a necessitar de informações fornecidas pelo mesmo.

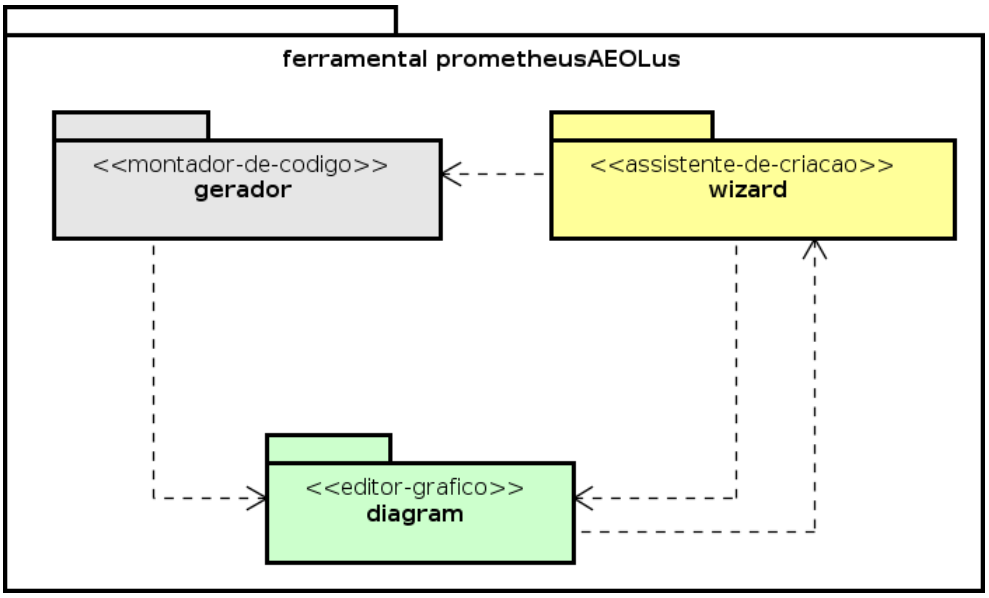


Figura 38: Arquitetura Geral do *plug-in* Prometheus AEOlus.

O pacote *diagram* comporta o *plug-in* editor gráfico. Esse pacote compila todos os arquivos de configuração e os códigos gerados por todo o desenvolvimento proveniente do *plug-in* GMF Eclipse. Detalhes de seu processo de desenvolvimento são descritos na Seção 4.3. As Seções 5.1 e 5.2 retratam de forma minuciosa o procedimento adotado para

desenvolvimento do *plug-in* editor gráfico da metodologia Prometheus AEOlus. Diferente dos demais *plug-ins* que compõem esta ferramenta, o gráfico não é da categoria PDE, explicada na Seção 4.2.3. Em virtude disso, o *plug-in* intitulado diagram possui uma dependência ao *plug-in* wizard para ser inicializado.

O pacote *wizard* aglomera o *plug-in* responsável por fazer a junção entre os outros dois *plug-ins* que constituem a ferramenta desenvolvida. Este *plug-in* é do tipo *PDE*, consequentemente, pode iniciar os demais *plug-ins*, por ter sua própria categoria no menu *runtime* da Ide Eclipse. Esse é o *plug-in* que reúne as funcionalidades dos outros *plug-ins* descritos, por esse motivo possui dependência aos demais. A peculiaridade desse *plug-in* em relação aos demais consiste na inexistência de códigos-fontes, apenas pontos de extensão adicionados no arquivo XML de configuração do *plug-in*, onde são informados quais *plug-ins* ocorre a relação de dependência.

5.1 Metamodelos implementados para a Geração do *plug-in* editor gráfico da metodologia Prometheus AEOlus

Esta seção tem por objetivo demonstrar os metamodelos desenvolvidos para regulamentar cada diagrama implementado pelo *plug-in* deste trabalho. O desenvolvimento da ferramenta gráfica que apoie a Metodologia Prometheus AEOlus não teve como objetivo validar a semântica desenvolvida por intermédio da utilização do *plug-in* gráfico. Sendo assim, nesta primeira versão de desenvolvimento, o utilizador pode modelar os *work products* da maneira que achar conveniente, apenas utilizando das notações presentes no diagrama, ficando sob sua responsabilidade a validação semântica dos mesmos.

5.1.1 Diagrama de Objetivos do Sistema

O diagrama de Objetivos do Sistema foi explicado na seção 3.2.3.1. O metamodelo desenvolvido correspondente a ele é ilustrado na Figura 39 e 40. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore, já a segunda, o diagrama gerado a partir deste Ecore.

Pela Figura 39, a entidade *Domain* representa o domínio que este diagrama está inserido, ou seja, o todo. Tudo que é adicionado neste metamodelo tem que estar referenciado na classe *Domain*, seja direta ou indiretamente (por meio de ligações). A classe *Domain* tem o mesmo significado pra todos os diagramas restantes.

A classe *Domain* contém uma ligação de cardinalidade *0..** , denominada *domain_goals*, cujo o tipo é *Goal*. Esta ligação representa que o *Domain* contém um conjunto de N valores de *Goals*.

A entidade *Goal* representa o objetivo do diagrama. O *label* que está incluso é um atributo que representa o nome que será adicionado ao *Goal* no momento de sua utilização no campo de desenho do *plug-in* desenvolvido. O atributo é do tipo *EString*,

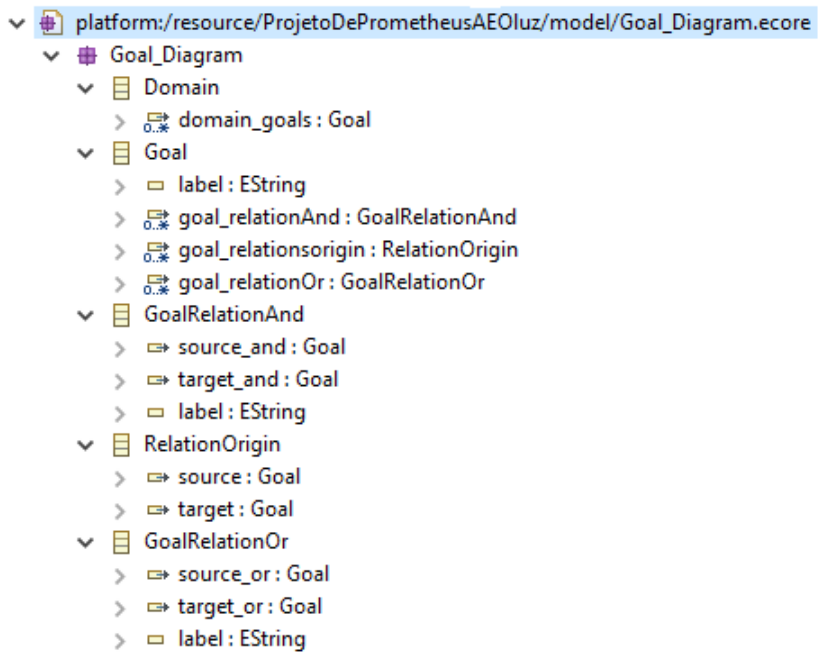


Figura 39: Metamodelo do Diagrama de Objetivos do Sistema na perspectiva do Ecore.

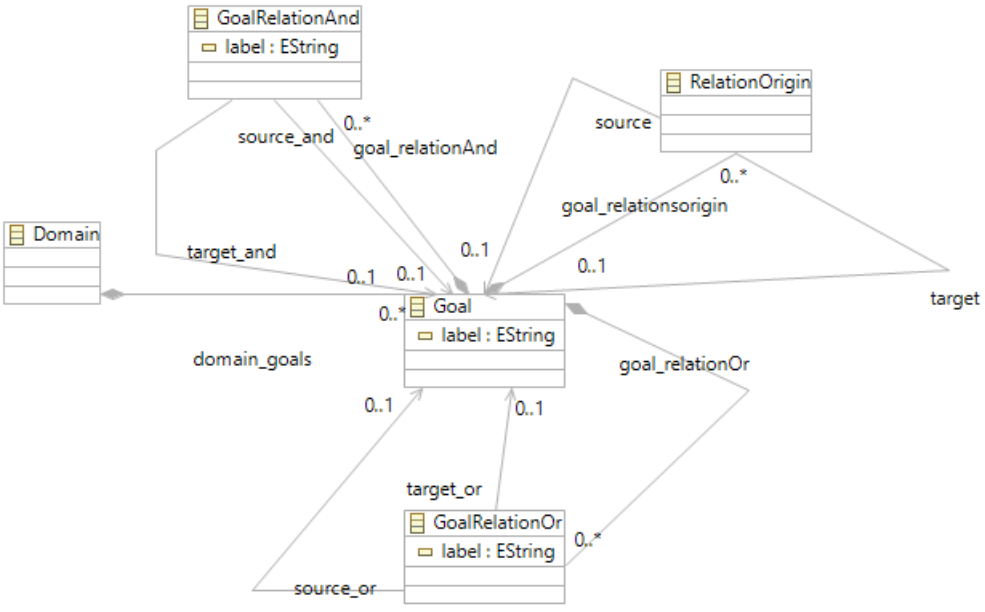


Figura 40: Metamodelo do Diagrama de Objetivos do Sistema na perspectiva de Diagrama.

por ser uma caixa de texto livre. Essa entidade possui ainda três relações, denominadas *goal_relationAnd*, *goal_relationorigin* e *goal_relationOr*, que representam respectivamente uma ligação do tipo *And*, *Origin* e *Or*. Detalhes desses relacionamentos são explicados na Seção 3.2.3.1.

A entidade *GoalRelationAnd* corresponde ao relacionamento do tipo *And*. Ela possui duas relações denominadas *source_and* e *target_and*, que representam, respectivamente, o *Goal* que irá sair a ligação e o *Goal* que chegará a ligação. Em virtude disso, ambos os relacionamentos são do tipo *Goal*. Essa entidade também possui um atributo do tipo *label* para ser possível denominar um nome para o relacionamento no campo de desenho.

A entidade *RelationOrigin* corresponde ao relacionamento do tipo *Origin*. Ela possui duas relações denominadas *source* e *target Goal* que irá sair a ligação e o *Goal* que chegará a ligação. Em virtude disso, ambos os relacionamentos são do tipo *Goal*. Diferente dos outros relacionamentos da entidade *Goal*, o do tipo *Origin* não contém um atributo do tipo *String*.

A entidade *GoalRelationOr* corresponde ao relacionamento do tipo *Or*. Ela possui duas relações denominadas *source_or* e *target_or* que representam respectivamente o *Goal* que irá sair a ligação e o *Goal* que chegará a ligação. Em virtude disso, ambos os relacionamentos são do tipo *Goal*. Essa entidade também possui um atributo do tipo *label* para ser possível denominar um nome para o relacionamento no campo de desenho.

Uma outra forma de visualização desse metamodelo é ilustrado pela Figura 40. Este diagrama possui a mesma explicação descrita na Figura 39.

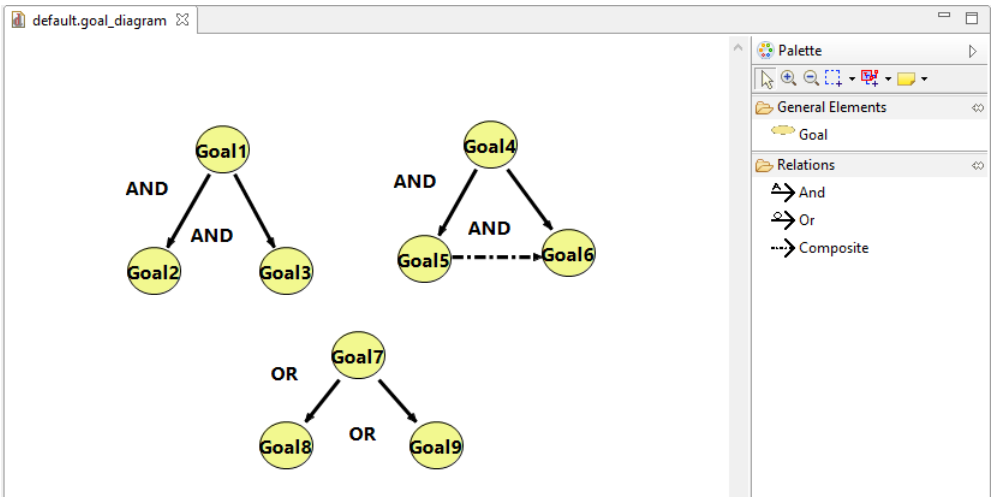


Figura 41: Exemplo do Diagrama de Objetivos do Sistema na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.1 é ilustrado um exemplo da modelagem deste diagrama, por meio da Figura 13. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 41. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama

fielmente, ou seja, sem perda de nenhuma informação.

5.1.2 Diagrama de Visão Geral dos Papéis

O diagrama de Visão Geral dos Papéis foi explicado na seção 3.2.3.2. O metamodelo desenvolvido correspondente é ilustrado pelas Figuras 42 e 43. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

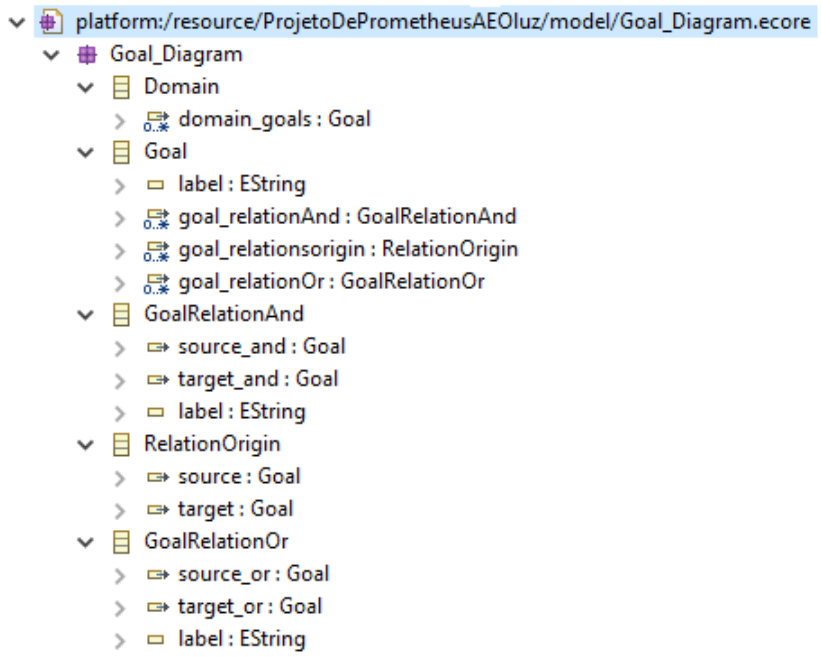


Figura 42: Metamodelo do Diagrama de Visão Geral dos Papéis na perspectiva do Ecore.

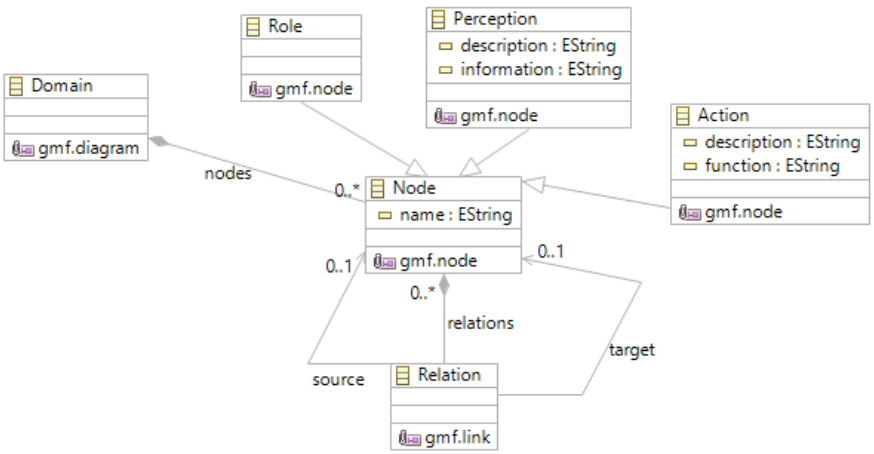


Figura 43: Metamodelo do Diagrama de Visão Geral dos Papéis na perspectiva de Diagrama.

Devido a este metamodelo ter mais entidades associadas, optou-se por fazer uma relação de herança. Desse modo, a entidade *Node* representa o conjunto de entidades

presentes no metamodelo, sendo que todas as entidades que a herdarem, herdarão o atributo *Name* e o relacionamento *relations*.

A entidade *Role* é herdada da entidade *Node*. Adicionou-se uma *Annotation* em sua entidade, em virtude de se acreditar que isto poderia ajudar o restante do *plug-in* a se localizar melhor com as relações de herança. No decorrer do desenvolvimento, este é o único atributo opcional do modelo, e não influencia no comportamento geral do sistema.

A entidade *Perception* possui, além do atributo *name* derivado da entidade *Node*, outros dois atributos: *description* e *information*. Ambos os atributos são do tipo *EString*, ou seja, armazenam informações textuais. Esses dois atributos foram retirados do descritor das percepções, descrito na Seção 3.2.3.15.

A entidade *Action* possui, além do atributo *name* derivado da entidade *Node*, outros dois atributos: *description* e *function*. Ambos os atributos são do tipo *EString*, ou seja, armazenam informações textuais. Esses dois atributos foram retirados do descritor das ações, descrito na Seção 3.2.3.14.

A entidade *Relation* é uma entidade genérica utilizada pela entidade *Node*. Ela representa os relacionamentos que podem existir entre as entidades do metamodelo. Os atributos *source* e *target* representam, respectivamente, o *Node* de origem do relacionamento e o *Node* de destino. Ambos são do tipo *Node*, pois todos os atributos concretos do metamodelo são derivados da entidade *Node*.

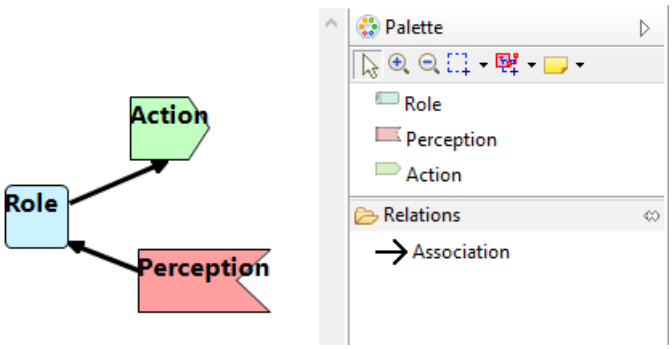


Figura 44: Exemplo do Diagrama de Visão Geral de Papéis na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.2 é ilustrado um exemplo deste diagrama, por meio da Figura 14. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 44. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

5.1.3 Diagrama Estrutural

O diagrama Estrutural foi explicado na seção 3.2.3.3. O metamodelo desenvolvido correspondente é ilustrado pelas Figuras 45 e 46. A primeira figura corresponde ao di-

agrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

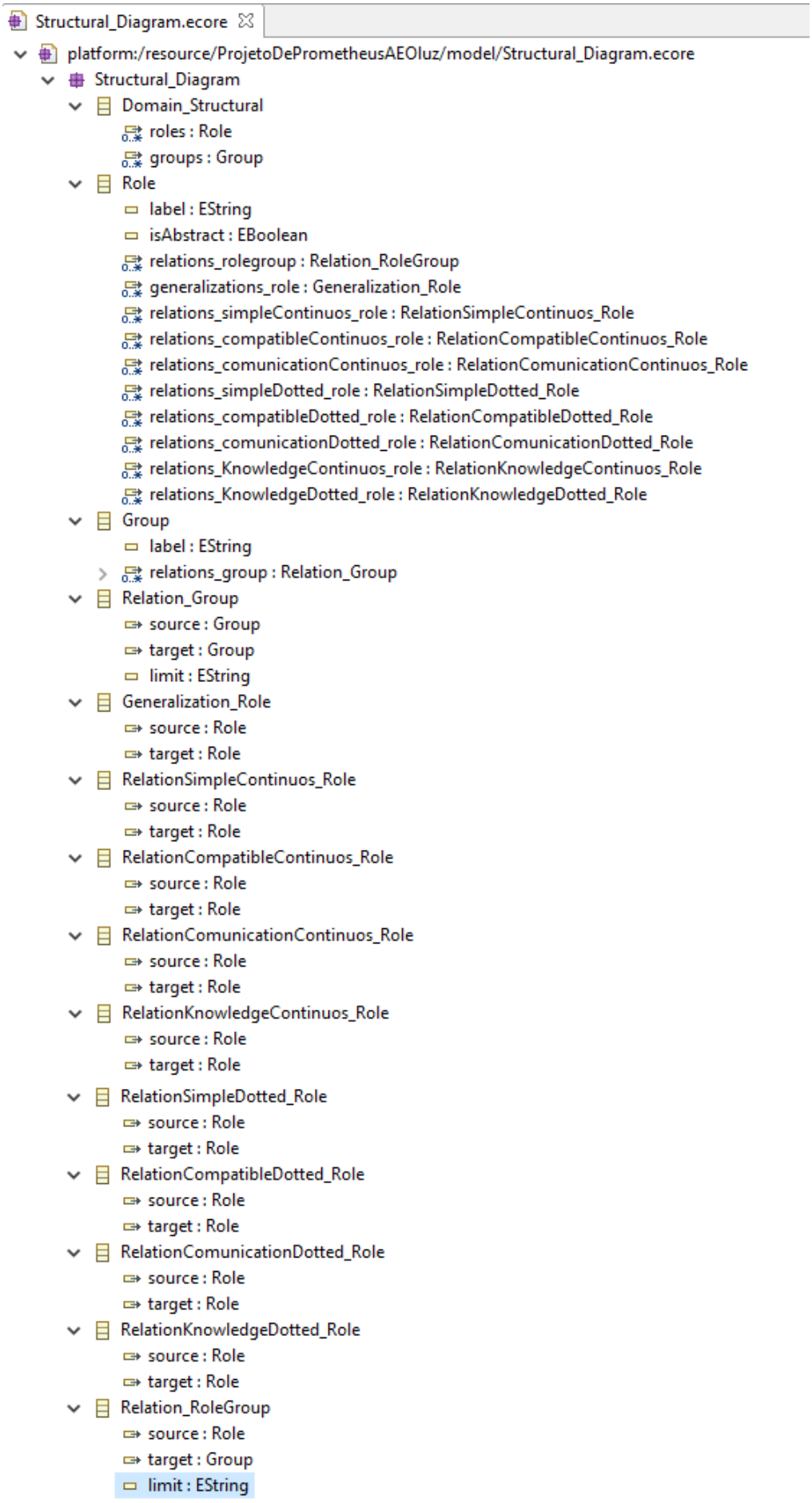


Figura 45: Metamodelo do Diagrama Estrutural na perspectiva do Ecore.

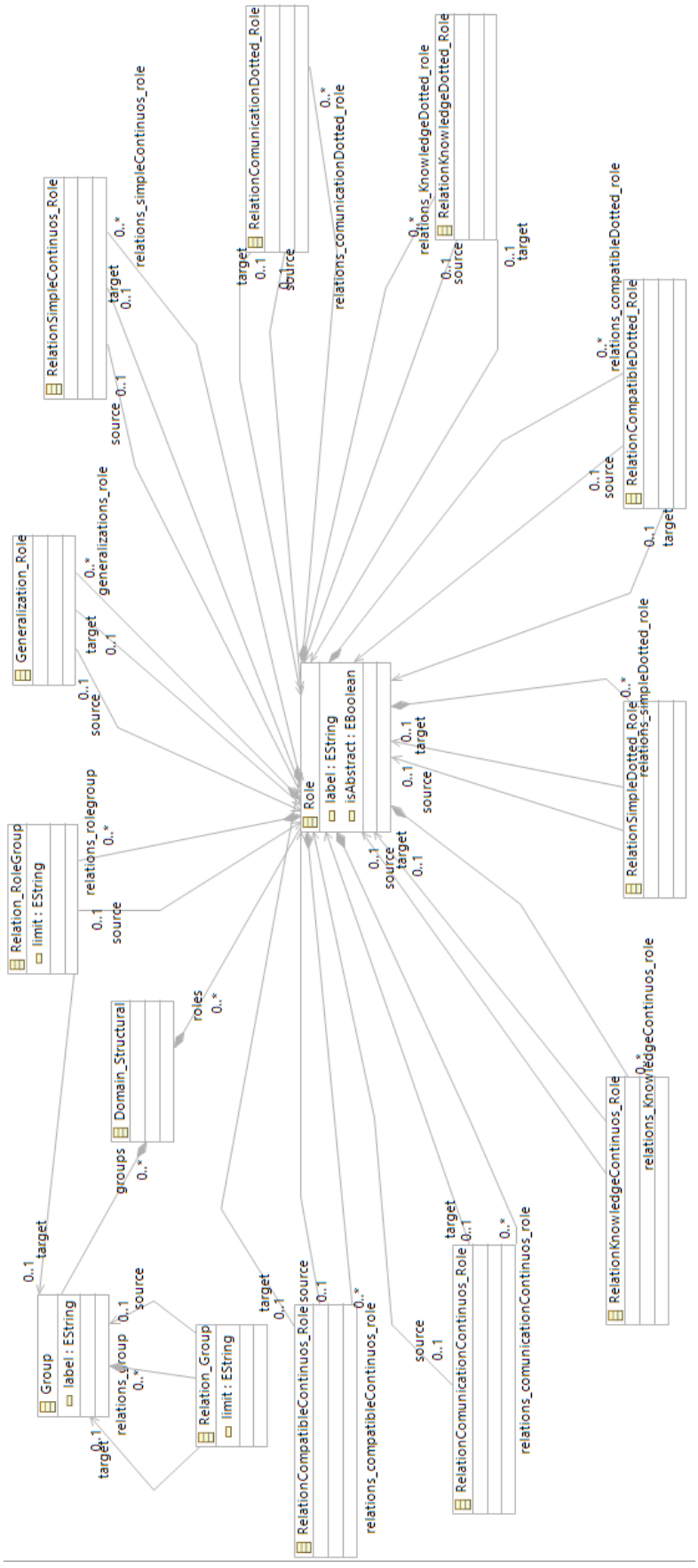


Figura 46: Metamodelo do Diagrama Estrutural na perspectiva de Diagrama.

Este diagrama tem como objetivo demonstrar as ligações existentes entre os papéis do sistema e entre os papéis e grupos. Devido aos relacionamentos serem diferentes entre esses dois conceitos, não foi possível utilizar herança neste metamodelo.

A entidade *Role* contém dois atributos: *label* e *isAbstract*. O primeiro atributo é do tipo *EString* e armazena um texto livre que representa o nome dado a entidade *Role* no campo de desenho. O segundo é um atributo do tipo *EBoolean*, que informará se o *Role* criado é abstrato ou não.

A respeito dos relacionamentos, da metodologia Prometheus AEOlus, este é o diagrama que contém mais relacionamentos. A entidade *Role* possui os relacionamentos descritos na Tabela 3.

Tabela 3: Relacionamentos da entidade *Role*.

Relacionamento	**Tipo de Relacionamento**
relations_rolegroup	*Relation_RoleGroup*
generations_role	*Generation_Role*
relations_simpleContinuos_role	*RelationSimpleContinuos_Role*
relations_compatibleContinuos_role	*RelationCompatibleContinuos_Role*
relations_comunicationContinuos_role	*RelationComunicationContinuos_Role*
relations_compatibleDotted_role	*RelationCompatibleDotted_Role*
relations_comunicationDotted_role	*RelationComunicationDotted_Role*
relations_KnowledgeContinuos_role	*RelationKnowledgeContinuos_Role*
relations_KnowledgeDotted_role	*RelationKnowledgeDotted_Role*

Dos relacionamentos descritos na Tabela 3, todos possuem os atributos chamados *source* e *target*, ambos são do tipo *Role*. Cada relacionamento possui um significado, cuja explicação ocorreu na seção 3.2.3.3. O único relacionamento dos descritos que possui uma entidade a mais chama-se *Relation_RoleGroup*, onde existe o atributo *limit* é do tipo *EString*, o qual representa a cardinalidade que poderá assumir essa ligação no campo de desenho do diagrama.

A entidade *Group* possui o atributo *label* do tipo *EString*, que tem como finalidade. Além disso, essa entidade detém do relacionamento *relations_group*, do tipo *Relation_Group*, que representa as relações que podem existir entre os grupos modelados. A respeito desse relacionamento, ele também apresenta os atributos *source* e *target*, bem como o atributo *limit* que representa a cardinalidade entre o relacionamento de *groups* distintos.

Na Seção 3.2.3.3 é ilustrado um exemplo deste diagrama (Figura 15). Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 47. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

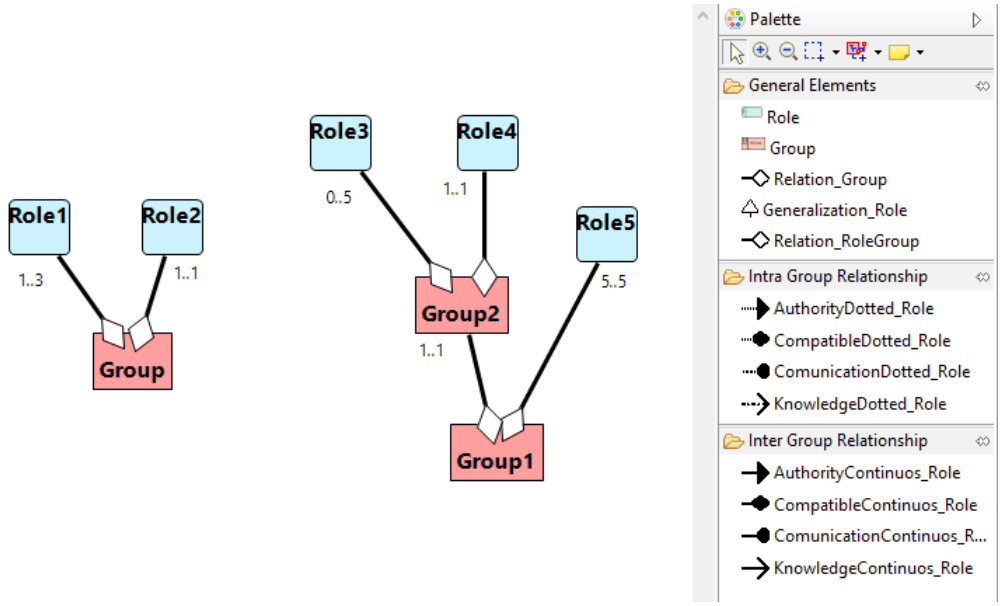


Figura 47: Exemplo do Diagrama Estrutural na perspectiva do *plug-in* desenvolvido.

5.1.4 Diagrama de Missões

O diagrama de Missões foi explicado na seção 3.2.3.4. O metamodelo desenvolvido correspondente a ele é ilustrado pelas Figuras 48 e 49. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

As entidades *DomainMission*, *Goal* e *GoalRelationAnd* tem o mesmo papel das entidades *Domain* e *Goal*, do metamodelo explicado na Seção 5.1.1.

A entidade *Mission* tem um atributo chamado *Label* do tipo *EString*, que tem como finalidade armazenar o nome da entidade no momento de sua utilização no campo de desenho do *plug-in* desenvolvido. Além disso, essa entidade possui um relacionamento denominado *relationsmissiongoal* do tipo *RelationMissionGoal* que tem cardinalidade *0..**. Esse relacionamento denota as relações entre as entidades *Mission* e *Goals*.

A entidade *RelationMissionGoal* representa a ligação entre *Mission* e *Goal*. Essa entidade possui dois atributos, denominados *source* e *target*. O primeiro atributo é do tipo *Mission* e o segundo é do tipo *Goal*. Isto significa que o relacionamento só pode ocorrer partindo de uma entidade *Mission* com destino a uma entidade *Goal*.

Na Seção 3.2.3.4 é ilustrado um exemplo deste diagrama, por meio da Figura 16. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 50. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

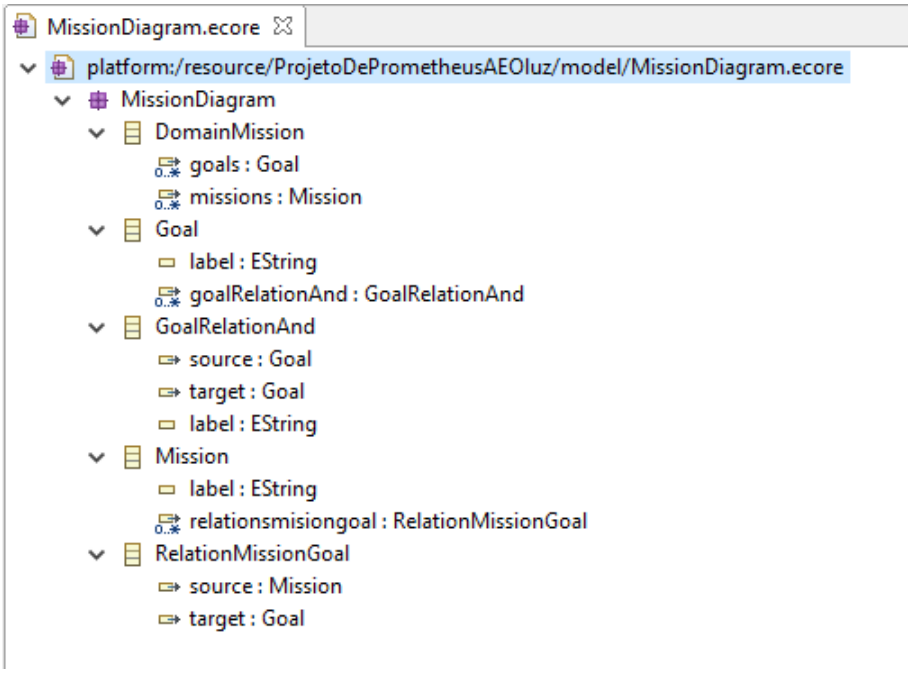


Figura 48: Metamodelo do Diagrama de Missões na perspectiva do Ecore.

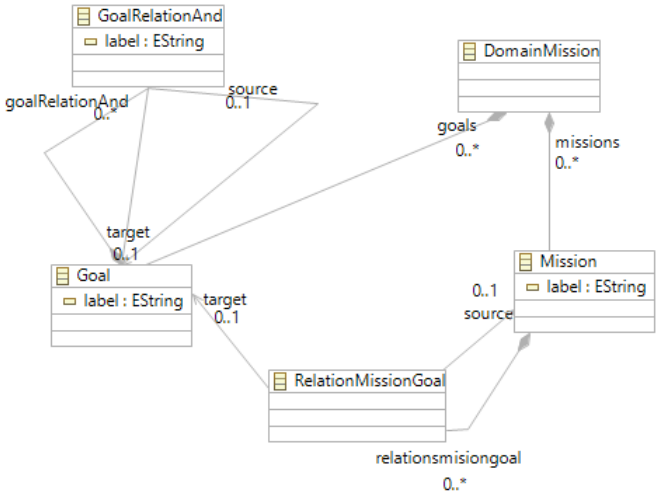


Figura 49: Metamodelo do Diagrama de Missões na perspectiva de Diagrama.

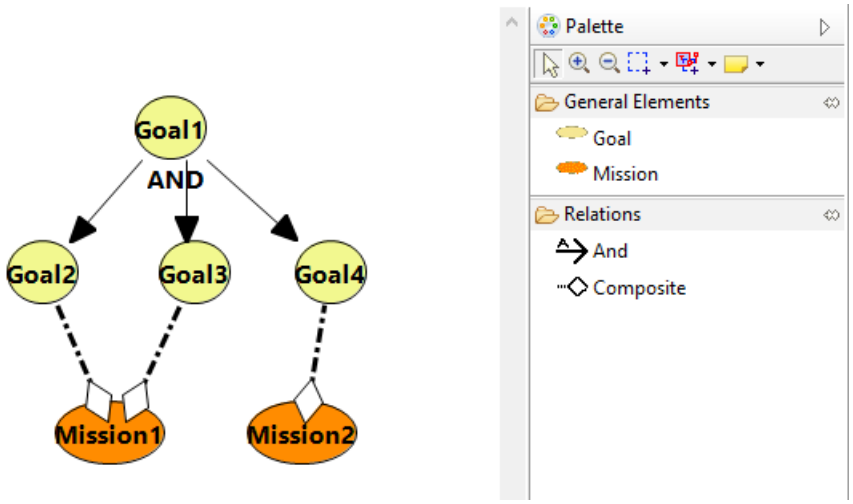


Figura 50: Exemplo do Diagrama de Missões na perspectiva do *plug-in* desenvolvido.

5.1.5 Diagrama Normativo

O diagrama Normativo foi explicado na seção 3.2.3.5. O metamodelo desenvolvido correspondente é ilustrado nas Figuras 51 e 52. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

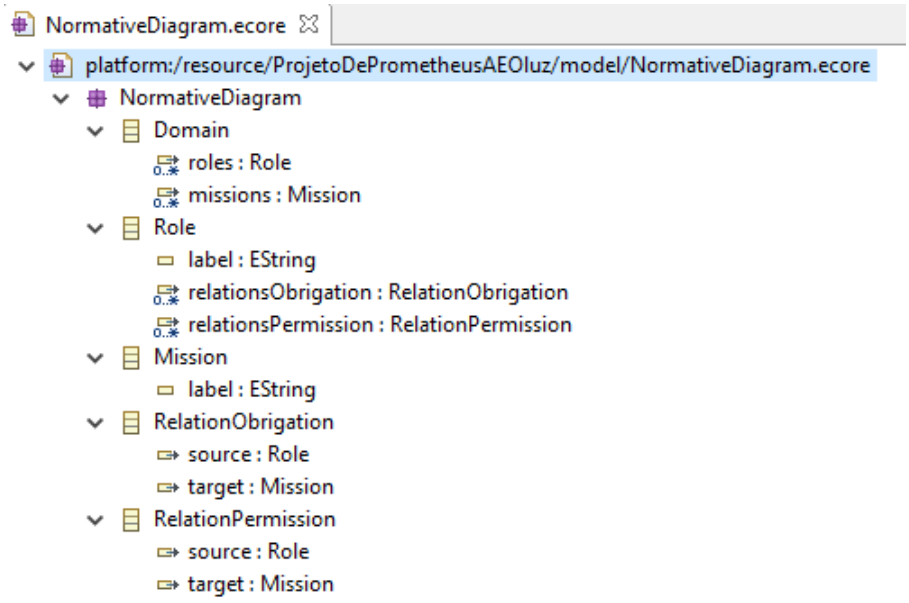


Figura 51: Metamodelo do Diagrama Normativo na perspectiva do Ecore.

A entidade *Role* contém um atributo chamado *label*, que igualmente aos demais diagramas, tem como finalidade armazenar o nome intitulado a entidade utilizada no campo de desenho do *plug-in* desenvolvido. Além disso, esta entidade possui dois relacionamentos, chamados de *relationsObrigation* e *relationsPermission* que são do tipo *RelationObrigation* e *RelationPermission* respectivamente. Ambos tem a cardinalidade *0..**.

A entidade *Mission* possui o atributo *label* do tipo *EString*, que tem a mesma função

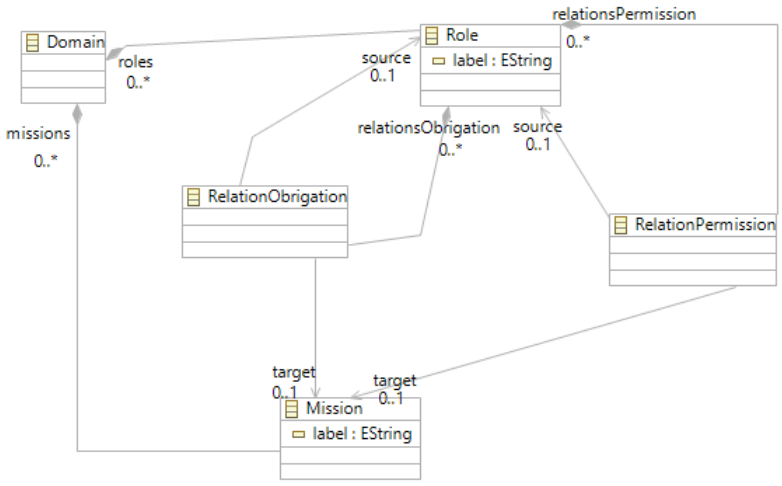


Figura 52: Metamodelo do Diagrama Normativo na perspectiva de Diagrama.

dos outros atributos com este mesmo nome.

As entidades *RelationObrigation* e *RelationPermission* representam os relacionamentos *Obrigation* e *Permission* respectivamente. Ambas as entidades possuem dois atributos, chamados de *source* e *target*. O primeiro é do tipo *Role* e o segundo do tipo *Mission*. Esses relacionamentos significam que uma relação pode partir de uma entidade *Role* em direção a uma entidade *Mission*. A diferença de um para o outro está em sua seta de ligação, mas esse detalhe não é especificado a nível de metamodelo.

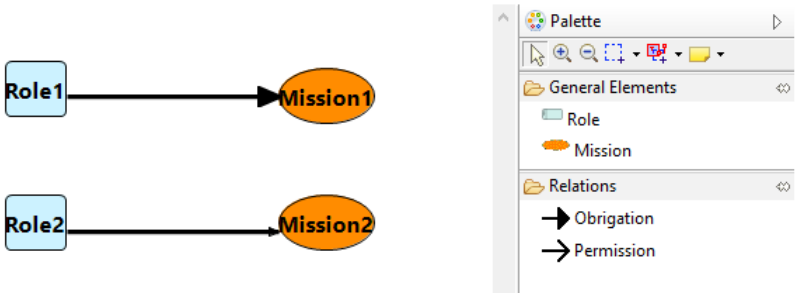


Figura 53: Exemplo do Diagrama Normativo na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.5 é ilustrado um exemplo deste diagrama, pela Figura 17. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 53. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

5.1.6 Diagrama de Relacionamento entre Papéis e Agentes

O diagrama de Relacionamento entre Papéis e Agentes foi explicado na seção 3.2.3.6. O metamodelo desenvolvido correspondente a ele é ilustrado nas Figuras 54 e 55. A primeira figura correspondea o diagrama descrito na perspectiva do Ecore; a segunda, ao

diagrama gerado a partir deste Ecore.

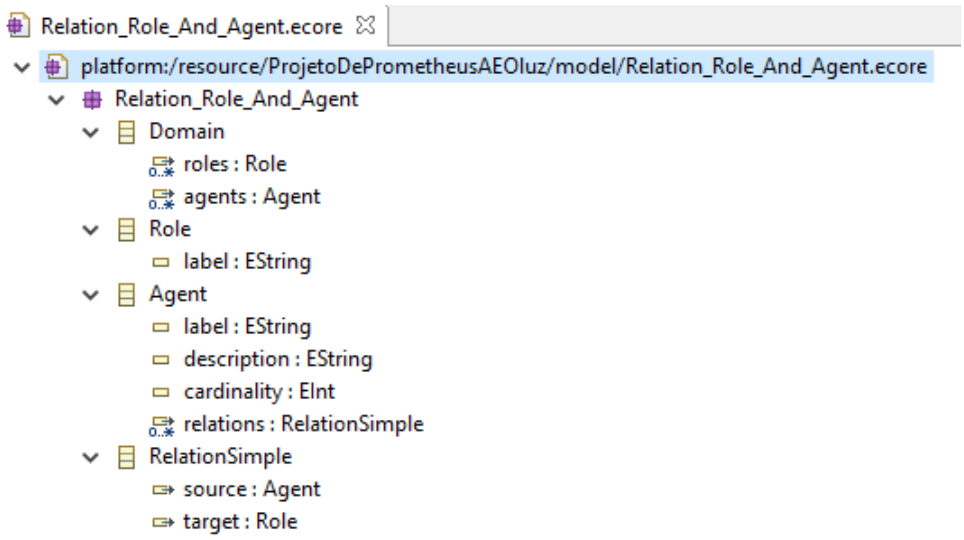


Figura 54: Metamodelo do Diagrama de Relacionamento entre Papéis e Agentes na perspectiva do Ecore.

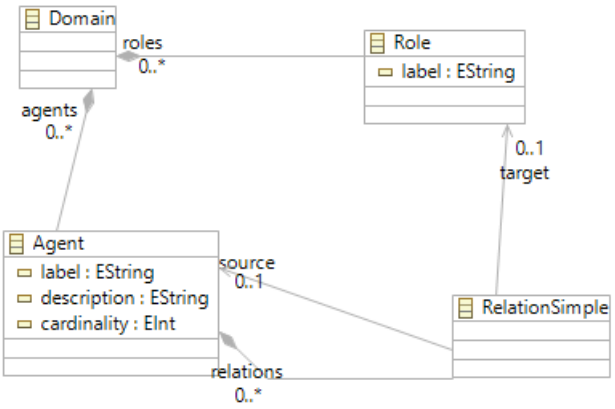


Figura 55: Metamodelo do Diagrama de Relacionamento entre Papéis e Agentes na perspectiva de Diagrama.

Neste metamodelo, a entidade *Role* representa os papéis e possui o atributo *label* do tipo *EString*, que tem como finalidade armazenar o texto estipulado no momento da criação da entidade no campo de desenho do *plug-in* desenvolvido.

A entidade *Agent* possui os atributos *label*, que tem a mesma função do estipulado na entidade *Role*; o atributo *description* do tipo *EString*; e o atributo *cardinality* do tipo *EInt*. O atributo *description* tem a função de armazenar a descrição do agente. Já o atributo *Cardinality* tem a função de armazenar a cardinalidade (número de representações) daquele agente no sistema.

A entidade *RelationSimple* representa a relação entre as entidades *Agent* e *Role*. Por esse motivo, ela possui dois atributos: *source* e *target*, do tipo *Agent* e *Role* respecti-

vamente. Isto significa que a ligação só poderá partir de um agente com destino a um papel.

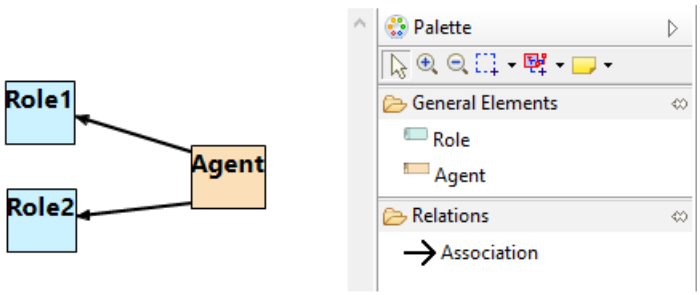


Figura 56: Exemplo do Diagrama de Relacionamento entre Papéis e Agentes na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.6 é ilustrado um exemplo deste diagrama, pela Figura 18. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 56. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

5.1.7 Diagrama Visão Geral do Ambiente

O diagrama de Visão Geral do Ambiente foi explicado na seção 3.2.3.7. O metamodelo desenvolvido correspondente a ele é ilustrado nas Figuras 57 e 58. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

A entidade *Artifact* é derivada da entidade *Node*. Assim como no diagrama de visão geral dos papéis, neste diagrama optou-se por criar uma entidade genérica chamada *Node* que tem um relacionamento comum a todas as entidades. Nessa entidade *Node* não foi possível fazer herança do atributo *label*, devido ao GMF apresentar inconsistências no momento de fazer a herança do atributo nas entidades que herdam dela.

A entidade *Artifact* tem os atributos *label*, *description*, *operation*, *parameter*, *observable_property* e *observable_event*, ambos do tipo *EString*. A explicação para cada um dos atributos encontra-se na Seção 3.2.3.20. Além dos atributos, essa entidade possui o relacionamento denominado *relationArtifactWorkspace*, do tipo *RelationArtifactWorkspace*. Este representa a ligação entre artefatos e *Workspaces*.

A entidade *Action* também herda o relacionamento da entidade *Node*. Ela possui os atributos *label*, *description* e *function*, todos do tipo *EString*. Esses atributos são explicados na Seção 3.2.3.14.

A entidade *Perception* possui os atributos *label*, *description* e *information*, todos do tipo *EString*. A explicação para cada um dos atributos é encontrada na Seção 3.2.3.15.

A entidade *Workspace* possui somente o atributo *label*, do tipo *EString*. Este atri-

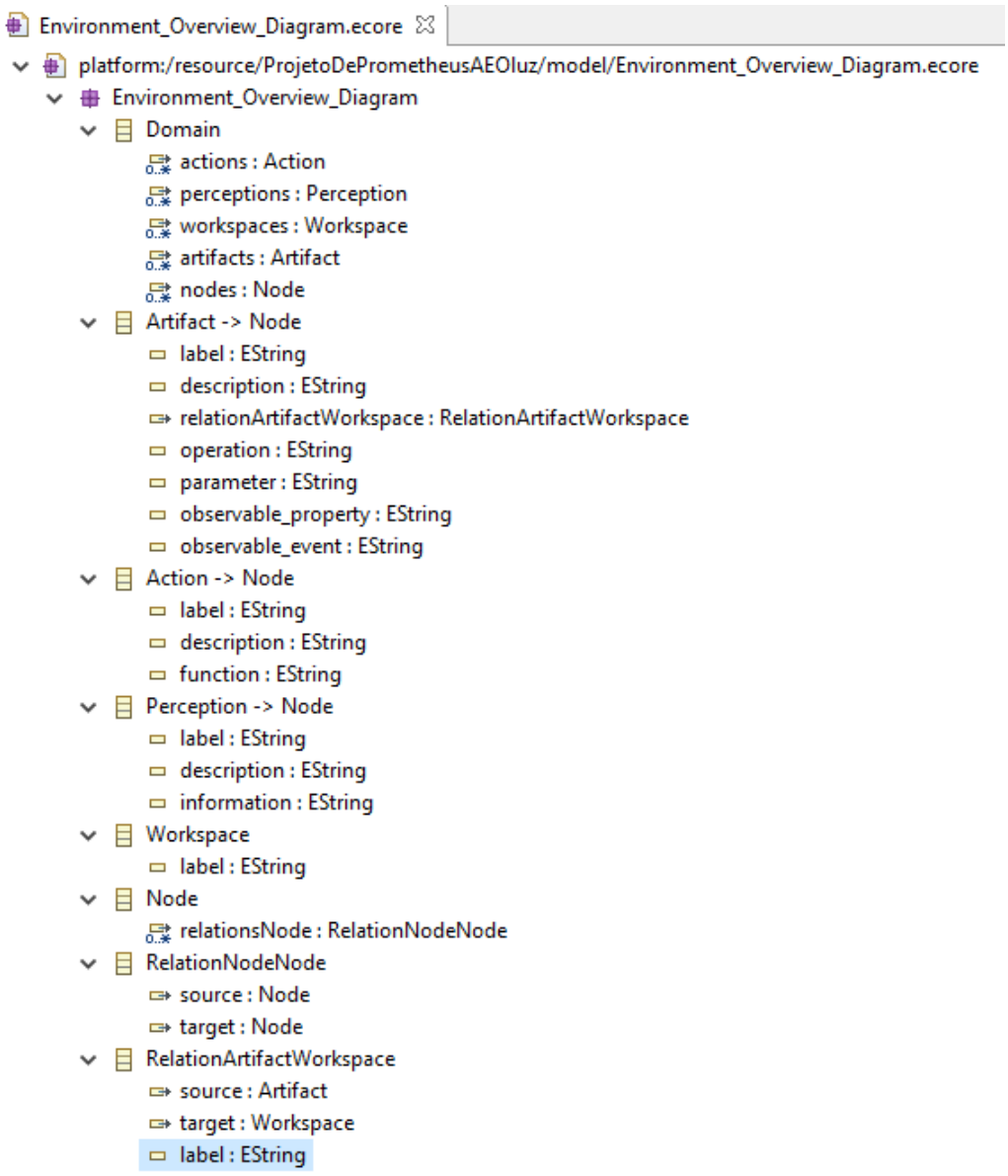


Figura 57: Metamodelo do Diagrama de Visão Geral do Ambiente na perspectiva do Ecore.

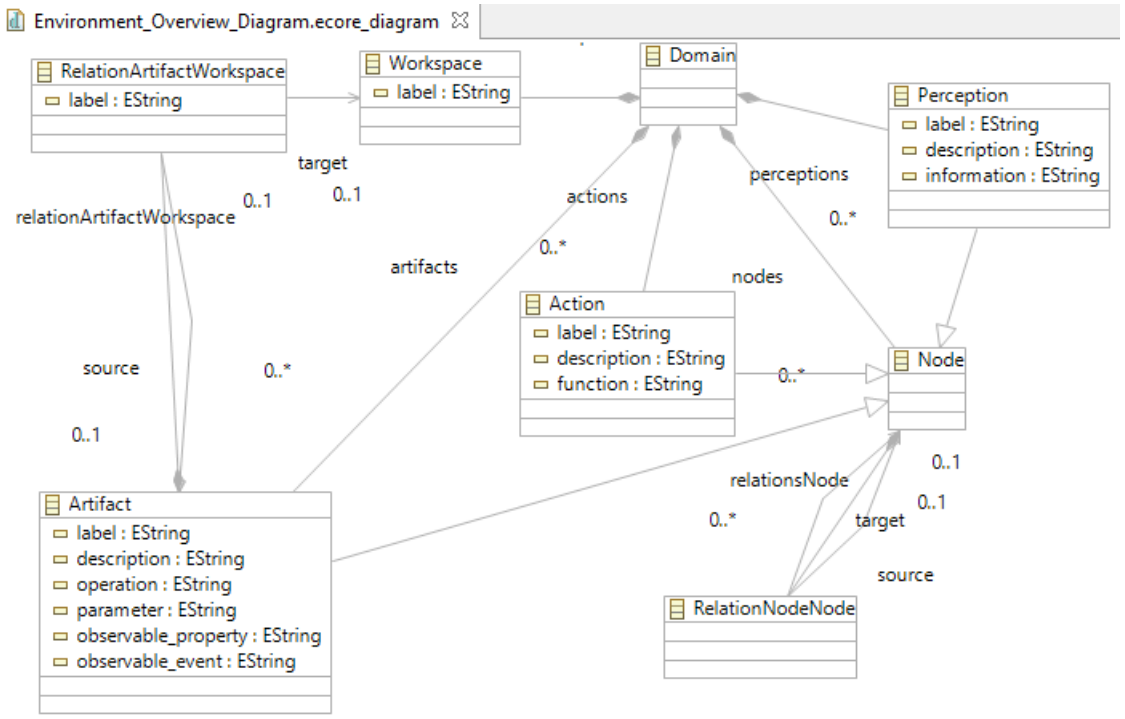


Figura 58: Metamodelo do Diagrama de Visão Geral do Ambiente na perspectiva de Diagrama.

buto tem finalidade semelhante aos demais do mesmo nome, ou seja, armazenar o texto informado a entidade no momento de sua utilização no campo de desenho do *plug-in*.

A entidade *RelationNodeNode* representa o relacionamento entre a entidade *Node*. Possui os atributos *source* e *target*, ambos do tipo *Node*. A peculiaridade dessa ligação é ser utilizada pela entidade *Node*, sendo que, todas as entidades que herdam da entidade *Node* acabam por herdar também essa relação. Dessa forma, por intermédio desse recurso é garantido um relacionamento entre todas as entidades que herdam da entidade *Node* no metamodelo.

A última entidade chama-se *RelationArtifactWorkspace*. Esta entidade representa a relação proveniente entre as entidades *Artifact* e *Workspace*. Ela possui os atributos *source*, do tipo *Artifact*; *target*, do tipo *Workspace*; e *label*, do tipo *EString*. Isto implica que o relacionamento só poderá ser iniciado em uma entidade *Artifact* e terminado em uma entidade *Workspace*.

Na Seção 3.2.3.7 é ilustrado um exemplo deste diagrama, pela Figura 19. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 59. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido. A única peculiaridade na implementação deste diagrama foi a impossibilidade de colocar dentro das *workspaces* as entidades *artifacts*. Por esse motivo, foi criado um novo relacionamento: *RelationArtifactWorkspace*, que faz esse papel de ligação entre as entidades referenciadas. Além disso, após fazer a ligação no campo de desenho, é incluído automaticamente o texto *its inside*, significando que a entidade

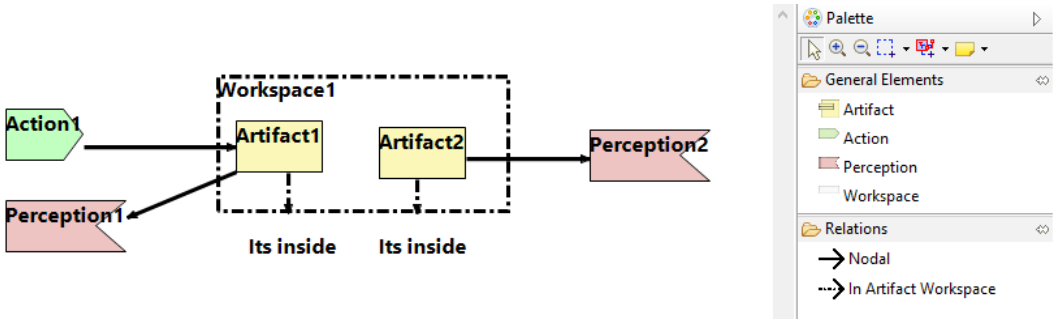


Figura 59: Exemplo do Diagrama de Visão Geral de Ambiente na perspectiva do *plug-in* desenvolvido.

ligada está dentro do *workspace*. Acredita-se que esta modificação não altera o nível de expressividade do diagrama gerado.

5.1.8 Diagrama Visão Geral do Sistema

O diagrama de Visão Geral do Sistema foi explicado na seção 3.2.3.8. O metamodelo desenvolvido correspondente a ele é ilustrado nas Figuras 60 e 61. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

A entidade *Action* representa uma ação e herda o relacionamento da entidade *Node*, assim como ocorre em vários outros metamodelos. Esta entidade contém os atributos *label*, *description* e *function*, todos do tipo *EString*. Os atributos são explicados na Seção 3.2.3.14.

A entidade *Perception* também herda o relacionamento da entidade *Node*. Ela possui os atributos *label*, *description* e *information*, todos do tipo *EString*. Os atributos desta entidade são explicados na Seção 3.2.3.15.

A entidade *Message* representa uma mensagem no metamodelo. Essa entidade também herda o relacionamento da entidade *Node*. Além disso, ela possui os atributos *label*, *description*, *propose* e *content*. Com exceção do atributo *propose*, todos os outros são do tipo *EString*, portanto, armazenam textos. O atributo *propose* é do tipo *MessageType*, que é uma entidade do tipo *Enumeration*, ao qual representa os tipos de mensagens que podem ser enviados pelos agentes. Esses tipos de mensagem, bem como os outros atributos que compõem a entidade *Message* são explicados na Seção 3.2.3.16.

A entidade *Protocol* representa um protocolo no metamodelo. Ela possui o atributo *label* do tipo *EString*, que tem finalidade de guardar o texto informado pelo usuário no campo de desenho. Assim como as entidades desse diagrama, ela também herda o relacionamento da entidade *Node*.

A entidade *Agent* herda o relacionamento da entidade *Node*. Ela contém os atributos *label* e *description*, do tipo *EString* e *cardinality*, do tipo *EInt*. Esses atributos são explicados na Seção 3.2.3.19.

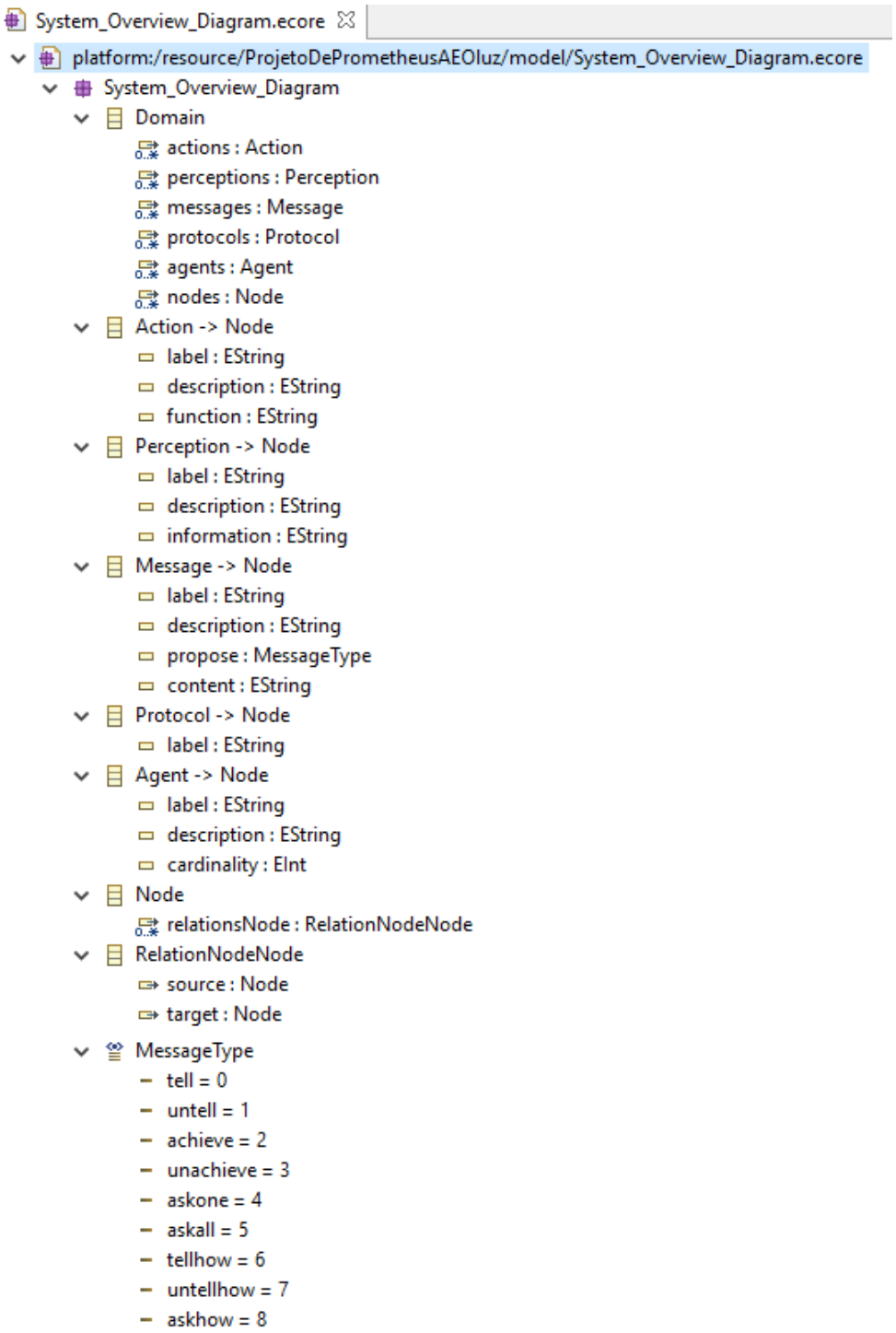


Figura 60: Metamodelo do Diagrama de Visão Geral do Sistema na perspectiva do Ecore.

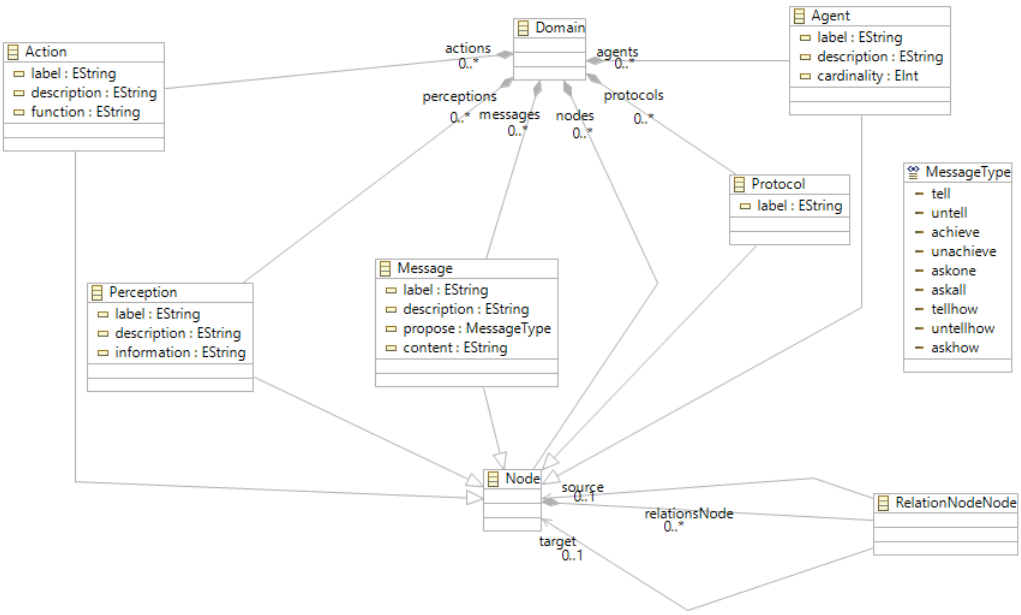


Figura 61: Metamodelo do Diagrama de Visão Geral do Sistema na perspectiva de Diagrama.

A entidade *Node* é uma entidade genérica utilizada para criar um relacionamento comum a todas as outras entidades que a herdam. Assim como ocorreu nos outros metamodelos, infelizmente não foi possível colocar o atributo *label* nesta entidade para que fosse possível ser herdado por todas as outras, evitando o retrabalho.

A entidade *RelationNodeNode* representa o relacionamento entre as entidades do tipo *Node*. Por esse motivo, os atributos *source* e *target* são do tipo *Node*. Importante salientar que todas as entidades que herdam da entidade *Node* acabam incorporando esse relacionamento.

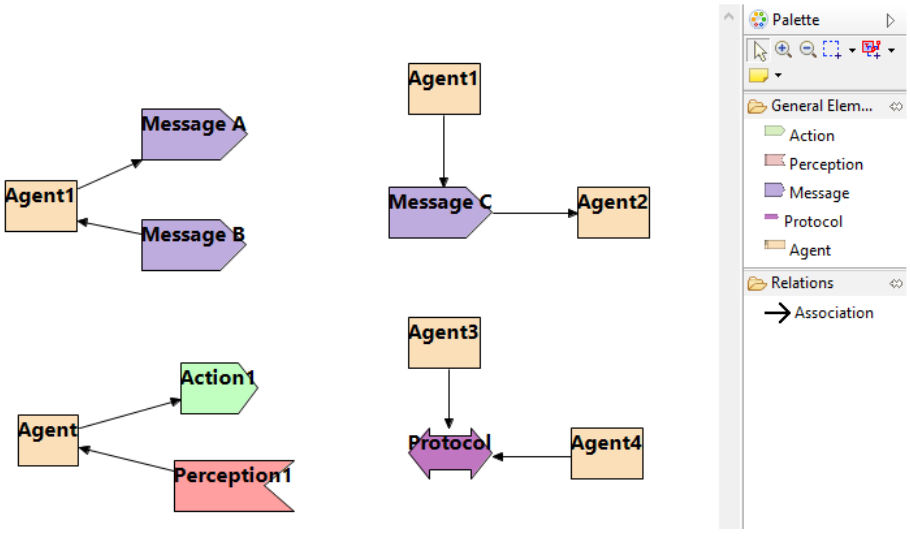


Figura 62: Exemplo do Diagrama de Visão Geral do Sistema na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.8 é ilustrado um exemplo deste diagrama, pela Figura 20. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 62. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

5.1.9 Diagrama Visão Geral do Agente

O diagrama de Visão Geral do Agente foi explicado na seção 3.2.3.10. O metamodelo desenvolvido correspondente a ele é ilustrado nas Figuras 63 e 64. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

Figura 63: Metamodelo do Diagrama de Visão Geral do Agente na perspectiva do Ecore.

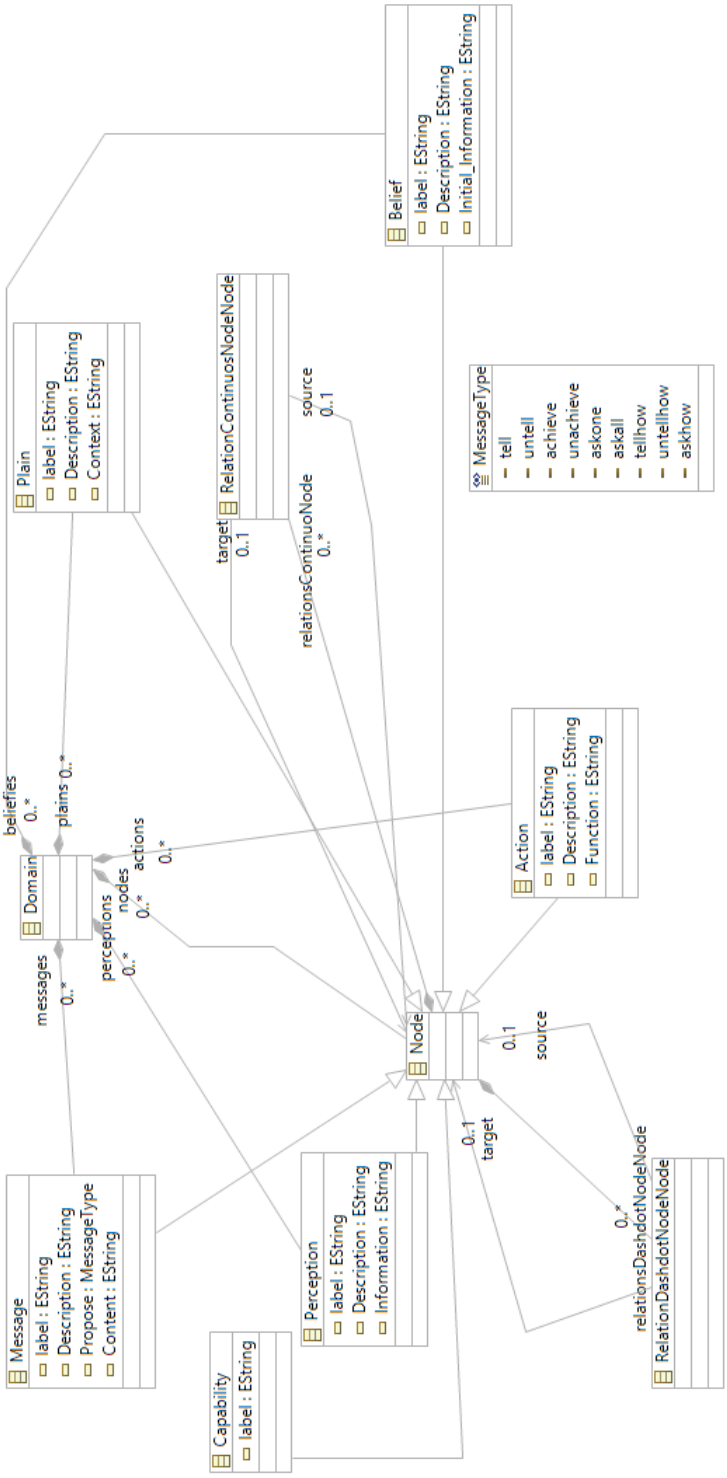


Figura 64: Metamodelo do Diagrama de Visão Geral do Agente na perspectiva de Diagrama.

A entidade *Node* tem papel semelhante aos demais metamodelos explicados. O objetivo é ser uma entidade genérica que isola atributos e relacionamentos comuns as demais entidades do metamodelo. Sua diferença, em relação as demais, consiste no número de relacionamentos. A entidade *Node*, neste metamodelo, possui dois relacionamentos: *relationsContinuoNode*, corresponde as ligações que utilizam uma seta simples e contínua; *relationsDashdotNodeNode*, representa os relacionamentos que utilizam uma seta pontilhada entre duas entidades que herdam da entidade *Node*.

A entidade *RelationContinuosNodeNode* e *RelationDashdotNodeNode* possuem os atributos *source* e *target*, que representam a origem e o destino das relações. Ambos os atributos são do tipo *Node*, justamente para serem reaproveitados nas entidades que herdam os relacionamentos da entidade *Node*.

As entidades *Action*, *Perception* e *Message* foram explicadas na Seção 5.1.8 e exercem a mesma função. A entidade *Plain* representa os planos que os agentes terão. Essa entidade possui os atributos *label*, *description* e *context*, todos do tipo *EString*. Além disso, essa entidade herda os relacionamentos presentes na entidade *Node*.

A entidade *Belief* representa as crenças no campo de desenho. Essa entidade possui os atributos *label*, *description* e *inicial_information*, todos do tipo *EString*. A explicação dos atributos encontra-se na Seção 3.2.3.18. O atributo *label* tem papel semelhante a outros metamodelos, ou seja, armazenar o nome dado a entidade utilizada no campo de desenho.

A entidade *Capability* não aparece na descrição deste diagrama, localizado na Seção 3.2.3.10. No momento de desenvolvimento desse metamodelo percebeu-se que sua única distinção em relação ao metamodelo do Diagrama de Capacidade era a entidade *Capability*. Portanto, optou-se por modelar a entidade *Capability* no metamodelo deste diagrama. A entidade *Capability* possui somente o atributo *label* cujo o tipo é *EString*.

O *Enumeration MessageType* tem papel de elencar todos os possíveis tipos de mensagens. Sua função é a elencar os tipos de mensagens possíveis.

Na Seção 3.2.3.10 é ilustrado um exemplo deste diagrama, pela Figura 22. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 65. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação. Além disso, acrescentou-se a entidade *Capability*, tornando o modelo coeso e completo.

5.1.10 Diagrama de Visão Geral de Análise

O diagrama de Visão Geral de Análise foi explicado na seção 3.2.3.12. O metamodelo desenvolvido correspondente a ele é ilustrado nas Figuras 66 e 67. A primeira figura corresponde o diagrama descrito na perspectiva do Ecore; a segunda, o diagrama gerado a partir deste Ecore.

As entidades *Domain*, *Perception*, *Action*, *Relation* e *Node* foram explicados sem

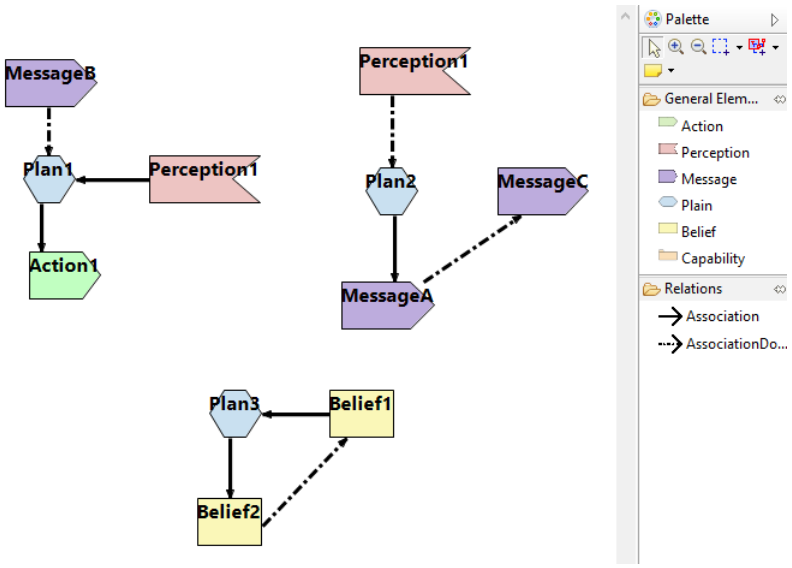


Figura 65: Exemplo do Diagrama de Visão Geral do Agente na perspectiva do *plug-in* desenvolvido.

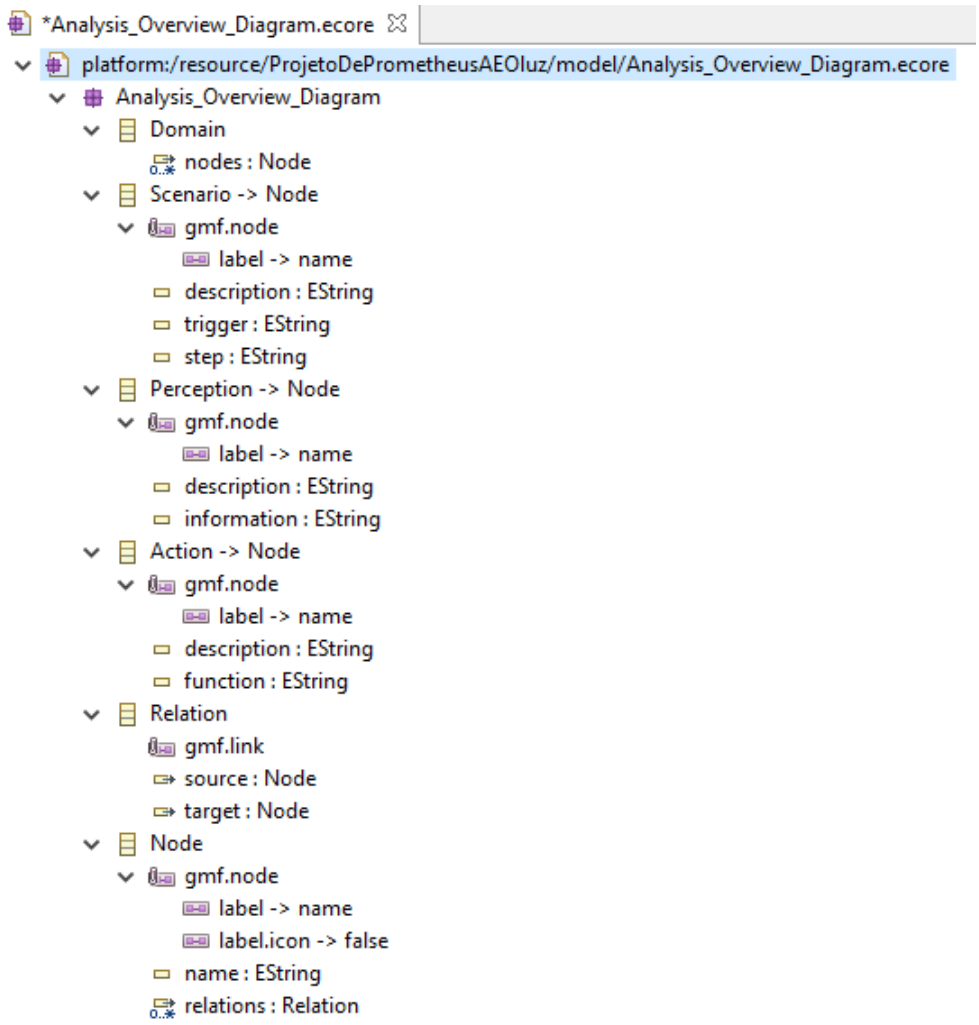


Figura 66: Metamodelo do Diagrama de Visão Geral de Análise na perspectiva do Ecore.

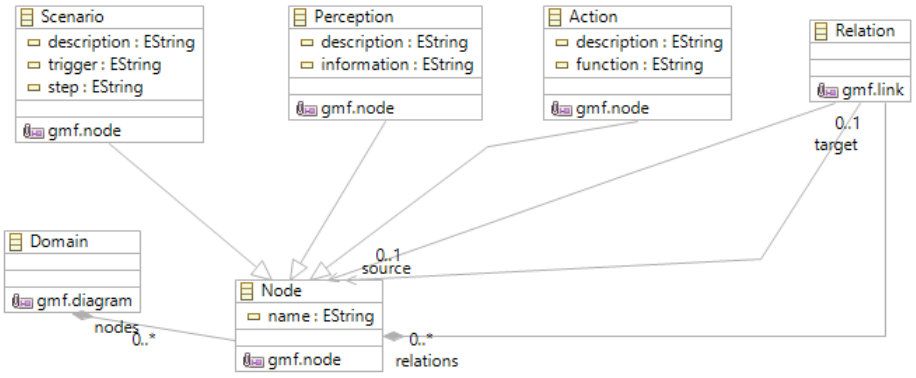


Figura 67: Metamodelo do Diagrama de Visão Geral de Análise na perspectiva de Diagrama.

subseções anteriores e seguem tendo o mesmo propósito. A entidade *Scenario* é peculiar somente ao metamodelo do diagrama de Visão Geral de Análise. Esta entidade possui os atributos *label*, *description*, *trigger* e *step*, todos do tipo *EString*. Assim como as demais entidades deste metamodelo, *Scenario* também herda o relacionamento da entidade *Node*.

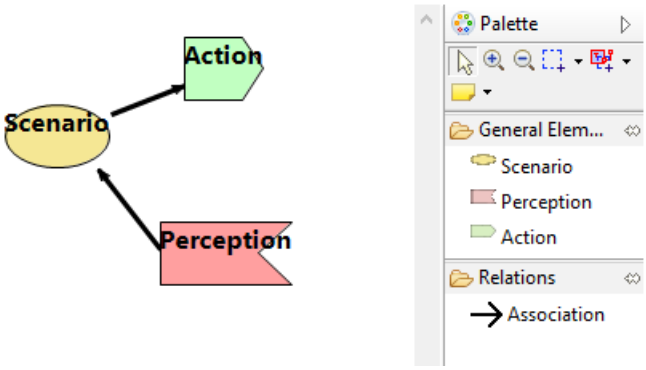


Figura 68: Exemplo do Diagrama de Visão Geral de Análise na perspectiva do *plug-in* desenvolvido.

Na Seção 3.2.3.12 é ilustrado um exemplo deste diagrama, pela Figura 23. Para exemplificar a utilização do *plug-in*, modelou-se o mesmo exemplo, conforme Figura 68. Através desta figura, mostra-se a viabilidade de modelar este diagrama por intermédio do *plug-in* desenvolvido, onde foi possível reproduzir o mesmo diagrama fielmente, ou seja, sem perda de nenhuma informação.

5.1.11 Diagrama de Modelo Geral Prometheus AEOlus

O diagrama de Modelo Geral da metodologia Prometheus AEOlus não se trata de um diagrama oficial da metodologia. Este diagrama surgiu devido aos problemas relatados na Seção 7.2, onde constatou-se que seria inviável realizar a geração de códigos fontes para a plataforma Jason sem que houvesse uma alternativa que pudesse minimizar os problemas explanados.

A solução encontrada foi criar um diagrama que unisse todas as entidades e relacionamentos presentes na metodologia, de modo que fosse possível modelar por intermédio de um modelo as informações necessárias para que houvesse posteriormente a geração de códigos. Deste modo, criou-se um metamodelo, ilustrado na Figura 69, onde realizou-se a unificação citada. Por questões dimensionais não foi aberto os ramos das entidades na Figura 69. Entretanto, as informações acerca dos atributos de cada uma das entidades não foram alteradas, portanto, já foram explicadas nas subseções anteriores.

O diagrama implementado por intermédio do *plug-in* é demonstrado na Figura 70. Através da figura, percebe-se que é possível modelar quaisquer diagramas presentes na metodologia. Ressalta-se que este diagrama trata-se de um diagrama de rascunho, onde é possível dar liberdade ao usuário modelar da forma que desejar.

5.2 Persistência de Dados no *Plug-in* GMF Eclipse

O GMF Eclipse utiliza de um arquivo semelhante ao XML para fazer sua persistência de dados. Segundo ZISMAN (2000), o XML é é uma linguagem de descrição de dados, um subconjunto da Norma Generalizadas *Markup Language* (SGML). Entretanto, o arquivo utilizado pelo GMF é uma derivação do XML, denominado *XML Metadata Interchange* (XMI).

Conforme ROUSE (2005), XMI é uma proposta de uso da linguagem XML destinada a fornecer uma maneira padrão para programadores e outros usuários trocarem informações sobre metadados (essencialmente, informação sobre um conjunto de dados e como é organizado). Um exemplo de um XMI gerado pelo GMF é apresentado pela Figura 71.

A Figura 71 representa um arquivo XMI que contém diversas *tags*, que tornam o processo de extração de informações mais acessível. Por exemplo, da linha 6 até a linha 17 são descritos entidades que foram modeladas no campo de desenho, sendo que o GMF começa a contagem das entidades a partir do número 0. A linha 21 representa um relacionamento entre duas entidades, no caso, entre a entidade da linha 7 (Role) e a da linha 17 (Agent). Por intermédio das *tags* também é possível identificar o relacionamento, que neste caso, é do tipo *EdgeSimples*, ou seja, seta contínua. As demais informações encontradas no início do arquivo correspondem ao cabeçalho padrão de arquivos XML e XMI.

No processo de geração de códigos para a plataforma Jason utilizou-se arquivos XMI gerados pelo GMF. Estes arquivos continham informações referentes a modelagem de digramas do gênero Modelo Geral da metodologia Prometheus AEOlus, ou seja, o diagrama que reune todas as entidades e relacionamentos presentes na metodologia. O objetivo deste processo é fazer a leitura dos arquivos XMI de modo que tornasse viável a extração das informações provendo a geração automatizada de códigos.

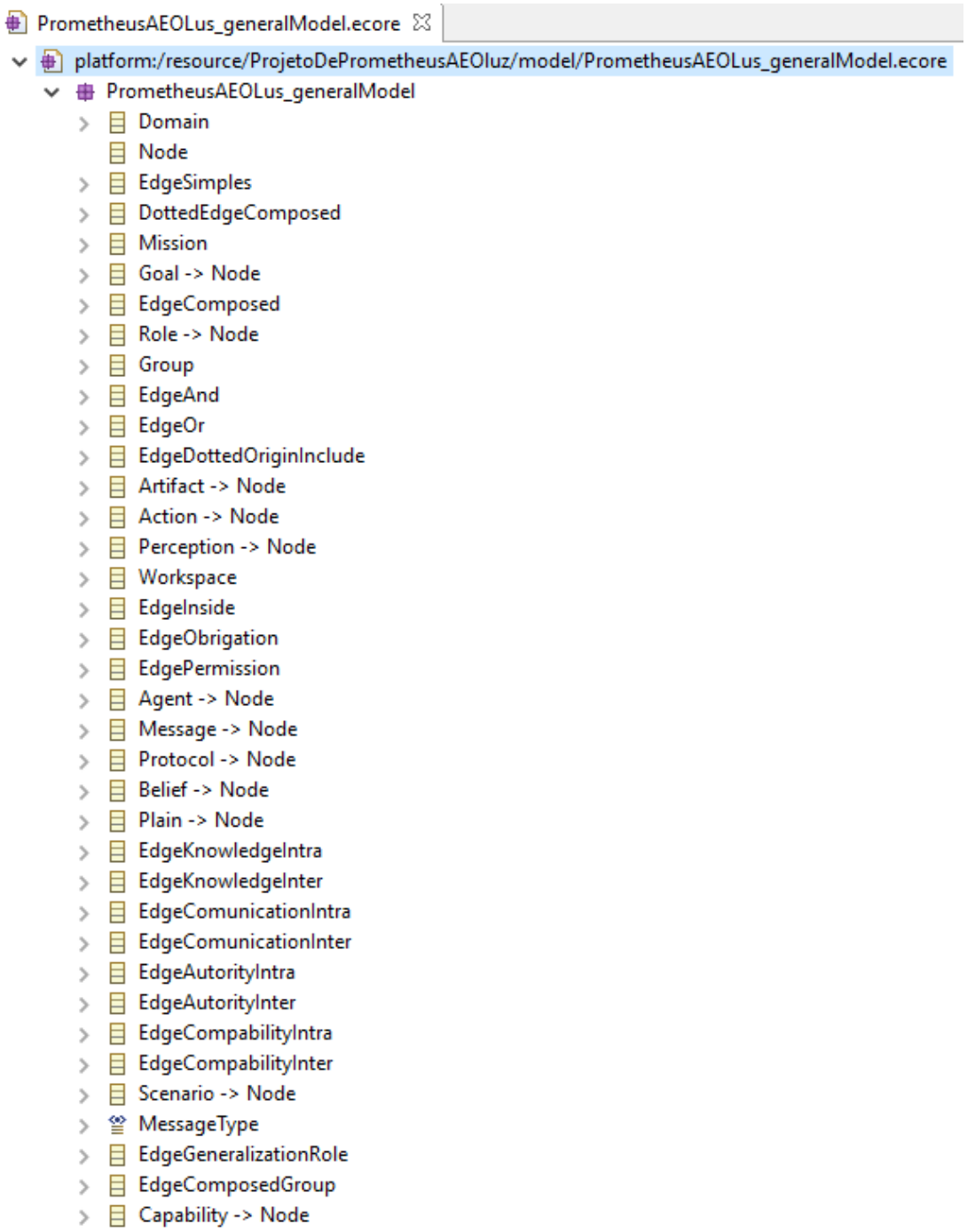


Figura 69: Metamodelo do Diagrama de Modelo Geral na perspectiva do Ecore.

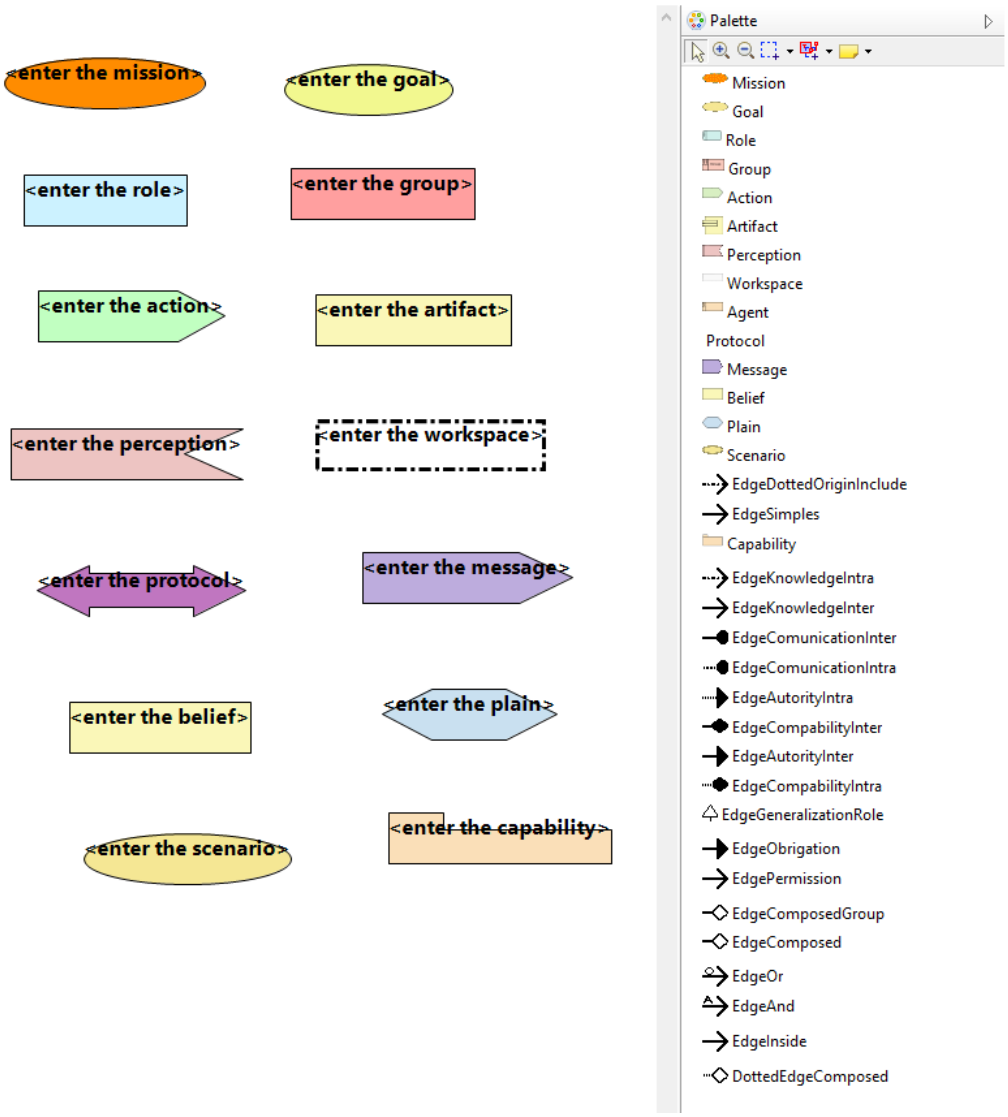


Figura 70: Exemplo do Diagrama de Modelo Geral na perspectiva do *plug-in* desenvolvido.

```
<?xml version="1.0" encoding="UTF-8"?>
<PrometheusAEOLus_generalModel:Domain xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:PrometheusAEOLus_generalModel="PrometheusAEOLus_generalModel">
  <nodes xsi:type="PrometheusAEOLus_generalModel:Goal"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Role"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Action"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Artifact"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Perception"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Protocol"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Message"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Belief"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Plain"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Scenario"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Capability"/>
  <nodes xsi:type="PrometheusAEOLus_generalModel:Agent"/>
  <missions/>
  <groups/>
  <workspaces/>
  <edgesSimple source="//@nodes.11" target="//@nodes.1"/>
</PrometheusAEOLus_generalModel:Domain>
```

Figura 71: Exemplo do Diagrama de Modelo Geral na perspectiva do *plug-in* desenvolvido.

5.3 *Plug-in* responsável pela geração de códigos para a plataforma Jason

A Figura 72 ilustra a arquitetura utilizada no desenvolvimento do *plug-in* gerador de códigos. Este *plug-in* trabalha juntamente com o *plug-in* descrito na Seção 5.1. Ambos foram desenvolvidos como projetos separados e integrados por meio de um *wizard*, tornando-os unificados.

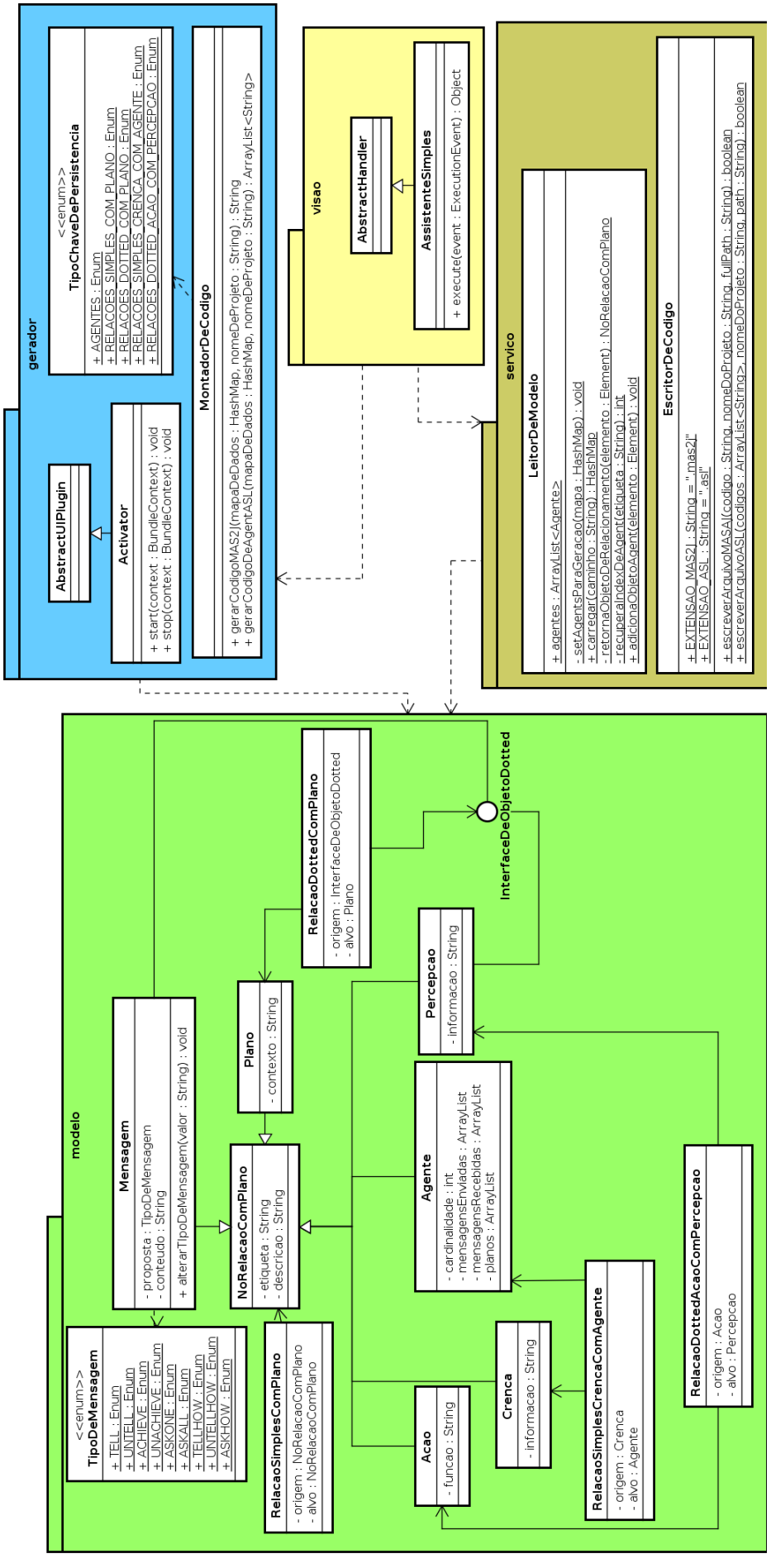


Figura 72: Diagrama de Arquitetura do *plug-in* gerador de Códigos

O *plug-in* gerador de códigos é dividido em 4 camadas: *modelo*, *gerador*, *visao* e *servico*. A primeira camada é responsável por representar através de objetos as entidades utilizadas pelos diagramas da metodologia Prometheus AEOlus. Além disso, essa camada também possui classes que ajudam o *plug-in* a encontrar informações presentes no XMI, conforme explicado na Seção 5.2. Por exemplo, as classes *NoRelacaoComPlano*, *RelacaoDottedComPlano*, *RelacaoSimplesCrencaComAgente* e *RelacaoDottedAcaoComPercepcao* não representam entidades utilizadas na metodologia, todavia exercem função de descrever possíveis ligações que façam sentido para a geração de códigos automatizada.

A camada *gerador* engloba as classes que centralizam as tarefas do *plug-in* com o ambiente Eclipse. Essas classes são as intituladas *AbstractUIPlugin* e *Activador*. Sobretudo, nessa camada também está incluída a classe *MontadorDeCodigo*, cuja função é escrever a *string* que conterá os códigos que devem ser persistidos no arquivo que o *plug-in* gerará posteriormente, sendo portanto uma das classes principais da aplicação.

A camada *servico* é responsável por ler o arquivo XMI explicado na Seção 5.2, o qual é gerado pelo *plug-in* descrito na Seção 5.1. Cabe ressaltar que o *plug-in* não faz a leitura de todos os arquivos XMI gerados pelo *plug-in* editor gráfico, sendo interpretado somente o arquivo que corresponde as informações modeladas no diagrama de Modelo Geral Prometheus AEOlus, conforme já dito anteriormente. A classe que realiza essa leitura, chama-se *LeitorDeModelo*, ficando responsável também por transformar todas as *tags* do arquivo XMI em objetos, retornando uma estrutura de dados do tipo *hashmap*, que contem os respectivos objetos transformados.

A camada *visao* tem como propósito gerenciar a utilização dos serviços e a geração de códigos pelo *plug-in* desenvolvido. Por intermédio de métodos provenientes das classes dessa camada, é possível ativar as funcionalidades do *plug-in*. A ativação ocorre através de eventos que são disparados pelos usuários.

Em síntese, o *plug-in* gerador de códigos segue os seguintes passos: o leitor de modelos, disponível na camada serviço, explora a *Application Pogramming Interface* (API) JDOM2 para realizar a leitura do arquivo XMI. Ressalta-se que essa API foi desenvolvida por terceiros, sendo somente utilizada neste projeto. O mapa de dados gerado a partir da leitura do arquivo é utilizado como dado de entrada pela classe *EscritorDeCodigo*. Esta percorre as informações para montar o código na linguagem AgentSpeak, formando arquivos com extensão ".asl". Ao final da execução do *plug-in*, são gerados arquivos ".asl"para cada agente modelado no diagrama e um arquivo de configuração da plataforma com extensão ".mas2j', finalizando o processo. Ressalta-se que o *plug-in* gerador de códigos preocupou-se em gerar códigos que fossem condizentes a sintaxe da linguagem AgentSpeak, não levando em consideração a versão da plataforma Jason que o código funcionará posteriormente.

6 VALIDAÇÃO DA FERRAMENTA DESENVOLVIDA

Este Capítulo apresenta os procedimentos realizados para a validação da ferramenta desenvolvida para apoiar a metodologia Prometheus AEOlus.

Este Capítulo está organizado como segue: A Seção 6.1 descreve os testes criados para validar o mecanismo de geração de códigos para a linguagem *agentspeak*; Na Seção 6.2 é demonstrado um estudo de caso, onde objetivou-se o desenvolvimento de *work products* para mostrar a eficiência da ferramenta construída.

6.1 Validação do Gerador de Código

O Teste Unitário ou Teste de Unidade é a primeira fase do processo de teste de software e é considerado um dos fundamentos principais do desenvolvimento de software (BIASI, 2006). Para CAMPOS (2014), os testes unitários são testes realizados ao nível de unidades individuais de software, ou de grupos de unidades relacionados. Segundo PRESSMAN (2011), um teste de unidade é realizado na menor parte funcional de um software (uma função, uma classe, um procedimento, etc). O objetivo do teste unitário é comparar funcionalidades de um determinado módulo com o que foi descrito em sua especificação, de forma a assegurar que o mesmo não o contradiga (PFLEEGER, 2004).

Ao total, foram codificados 63 casos de teste em JUnit 4x. Estes, foram agrupados em duas categorias: a primeira, destina-se aos testes de geração de código para arquivos de configuração (com extensão “mas2j”), arquivo pelo qual é especificado os agentes que compõe o sistema. A segunda, é responsável por verificar se os códigos de extensão “asl” estão conforme o esperado, respeitando as regras de produção impostas pela gramática Jason. Esse arquivo é destinado a especificação das crenças, objetivos, planos, ações e troca de mensagens entre os agentes que compõem o sistema.

A cobertura na teoria de testes de software permite verificar quais zonas de código foram executadas por um conjunto de testes realizados (BINDER, 2000). Isso permite verificar fluxos de códigos desnecessários, os quais podem ser removidos para otimizar ainda mais o código-fonte testado. Outro ponto positivo da técnica de análise de cobertura é que a mesma possibilita verificar se os testes unitários e de integração atingem todas as

condicionais codificadas, fazendo com que novos testes sejam confeccionados para testar outros dados de entrada.

A ferramenta de análise de cobertura utilizada foi a EclEmma 2.3.2[1]. Esta, foi explorada juntamente com o *framework* de testes JUnit 4x[2]. Foram realizados duas análises de cobertura. A primeira, destinou-se a testar a primeira categoria de testes, do arquivo com extensão "mas2j". Já a segunda, foi designada a testar arquivos com extensão "asl". A seguir os gráficos são apresentados nas figuras 73 e Anexo B.

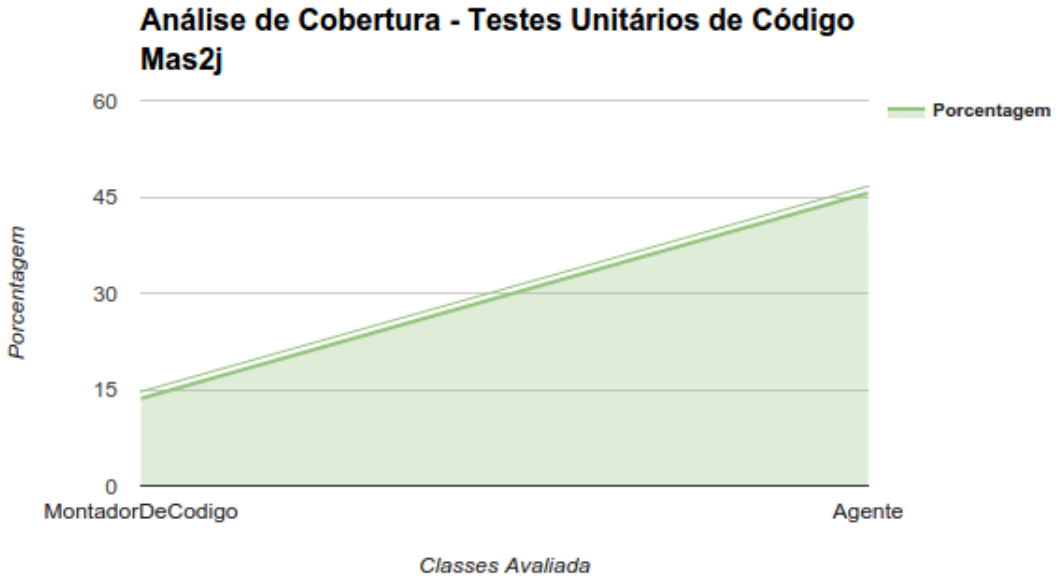


Figura 73: Cobertura de testes unitários para geração de código mas2j.

Na Figura 73 é apresentado a cobertura do conjunto de testes unitários codificados para averiguar a geração de código "mas2j". As classes avaliadas nessa análise foram a *MontadorDeCodigo* e *Agente*. No gráfico é possível visualizar que a classe *MontadorDeCodigo* possui apenas 13,6% de cobertura, essa porcentagem corresponde ao valor esperado, visto que a classe analisada é composta por vários métodos que são destinados também a geração de código "asl".

A classe *Agente* apresenta uma cobertura de 45,6%, além dos métodos analisados, a cobertura é aceitável, pois, nessa classe existe uma quantidade de métodos de encapsulamentos (*getters* e *setters*) que não precisam ser alcançados pelos testes. Em comparação, as duas classes analisadas, possuem cobertura de aproximadamente 90% para os métodos avaliados. Um ponto a analisar em um trabalho futuro é a divisão das responsabilidades da classe *MontadorDeCodigo* em duas classes, resultando em uma cobertura de testes relativamente maior em ambos os conjuntos de testes analisados. Detalhes sobre os testes realizados nas classes que provêm os arquivos com extensão "asl", encontram-se no Anexo B.

[1]http://eclemma.org/
[2]http://junit.org/

6.2 Validação da Ferramenta Gráfica

Para avaliar a utilização da ferramenta gráfica, optou-se por modelar dois estudos de casos distintos. O primeiro, abaixo, chama-se *Build a House* e é fruto do trabalho de (UEZ, 2013). Já o segundo, na Apêndice A, aborda um sistema de gerenciamento de conferência intitulado *Conference Management System* (CMS). Este último, trata-se de um estudo de caso mais complexo, sendo adaptado do trabalho de (GIORGINI, 2009).

6.2.1 Estudo de Caso - *Build a House*

Em BOISSIER et al. (2013), foi explicado um estudo de caso chamado *Build a House*. Para comprovar a potencialidade do *plug-in*, nesta subseção é demonstrado a modelagem desse estudo de caso utilizando a ferramenta gráfica desenvolvida. O estudo de caso em questão trata-se de um personagem chamado *Giacomo*, que deseja construir sua casa. Para isso, ele precisa contratar empresas que possam executar as tarefas relacionadas com a construção e coordenar o trabalho dessas empresas. A contratação será feita por meio de uma licitação, na qual a empresa que apresentar o menor preço para a tarefa será contratada. As tarefas de construção envolvem: preparar o terreno, fazer as fundações, construir as paredes, construir o telhado, colocar as portas e as janelas, instalar o sistema elétrico, instalar o encanamento, pintar as paredes externas e pintar as paredes internas.

Os diagramas demonstrados são frutos do trabalho de UEZ (2013), sendo portanto modelados naquele trabalho. O objetivo é mostrar a viabilidade de transpassar os conceitos envolvidos no trabalho de UEZ (2013) para uma ferramenta de modelagem gráfica desenvolvida justamente para apoiar a metodologia criada. Além disso, após a modelagem dos diagramas, mostra-se a geração de códigos para a plataforma Jason.

A Figura 74 representa o diagrama de objetivo do sistema modelado para o estudo de caso em questão. Percebe-se com a modelagem desse diagrama que os objetivos encontram-se em uma árvore hierarquia, sendo portanto necessário concluir os objetivos dos ramos inferiores para findar os superiores.

A Figura 75 representa o diagrama de Visão Geral de Análise modelado para o estudo de caso em questão. Este diagrama apresenta um resumo dos cenários, percepções e ações que terão o sistema modelado. As cores que representam cada uma das entidades são amarelo, verde e vermelho respectivamente. Entende-se, por meio deste diagrama, que um cenário representa uma quantidade N de subobjetivos que são necessários para atingir os objetivos gerais, definidos na Figura 74. Uma falha destacada nesse diagrama são as ações terem exatamente os mesmos nomes dos cenários, podendo ser resolvido através de uma convenção onde os cenários já tenham em sua descrição uma forma de executá-los.

A Figura 76 representa o diagrama de Visão Geral de Papéis modelado para o estudo de caso em questão. Este diagrama ilustra os papéis que compõem o sistema bem como as

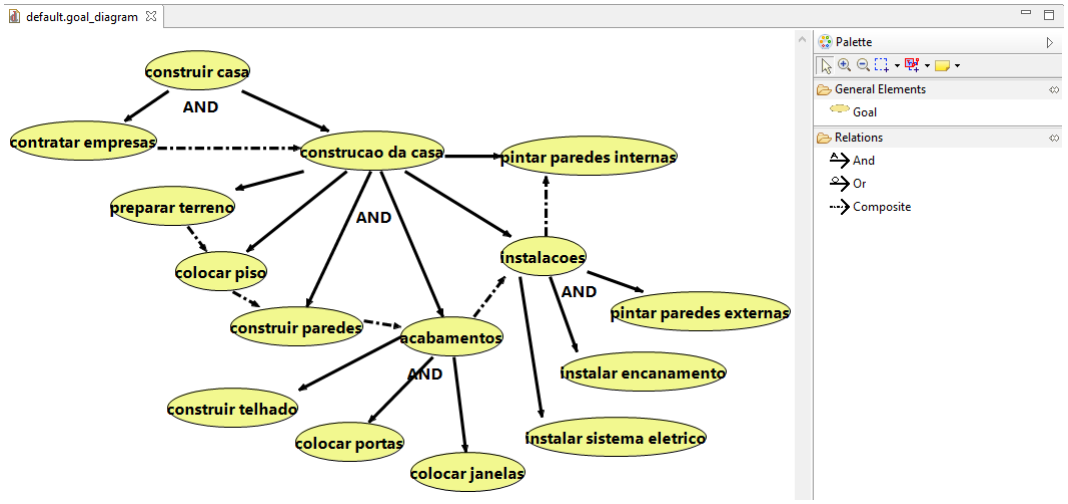


Figura 74: Diagrama de Objetivo do Sistema - Estudo de Caso *Build a House*

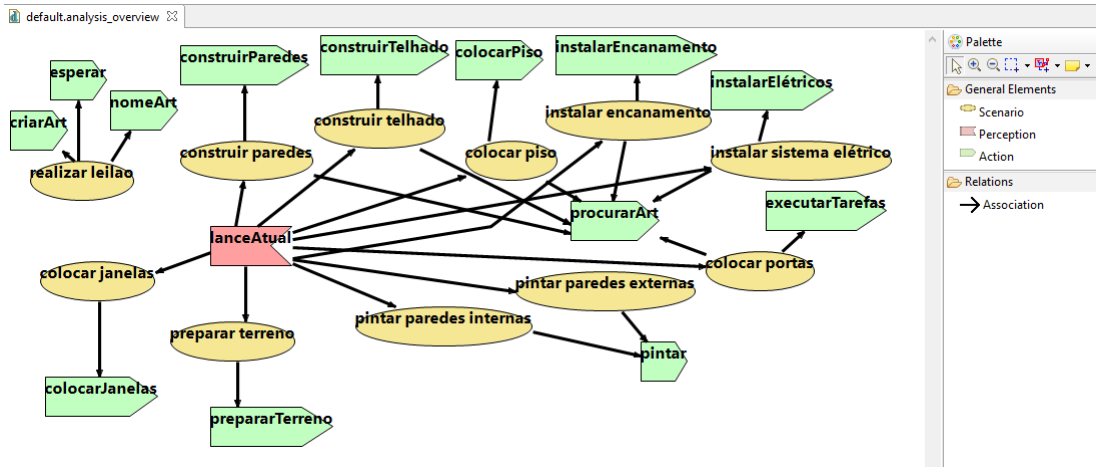


Figura 75: Diagrama de Visão Geral de Análise - Estudo de Caso *Build a House*

percepções e ações que são captadas e executadas respetivamente pelos papéis. Acredita-se que a visibilidade das informações melhoraria se fosse unificado com o diagrama de Relacionamento entre papéis e agentes, além de minimizar a complexidade da utilização da metodologia em questão.

A Figura 77 representa o diagrama de Missões modelado para o estudo de caso em questão. Este diagrama ilustra os objetivos e missões que compõem o sistema a ser modelado. Conforme já dito anteriormente, existe outro diagrama, o de Objetivos, que também modela os objetivos do sistema. Seria interessante unificar o diagrama de Missões e o de Objetivos, resultando na redução da quantidade de diagramas presentes na metodologia.

A Figura 78 representa o diagrama Estrutural modelado para o estudo de caso em questão. Este diagrama representa as ligações entre papéis e entre grupos. É o diagrama que contém mais possibilidades de ligações. Acredita-se que deveria ter um estudo mais aprofundado na elaboração deste diagrama, visando a diminuição da quantidade de possíveis ligações.

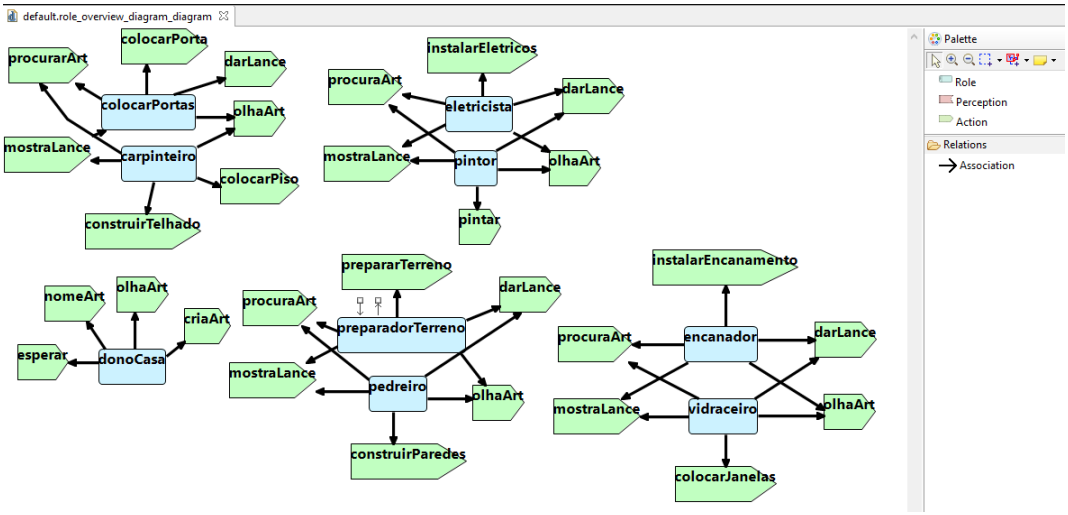


Figura 76: Diagrama de Visão Geral de Papéis - Estudo de Caso *Build a House*

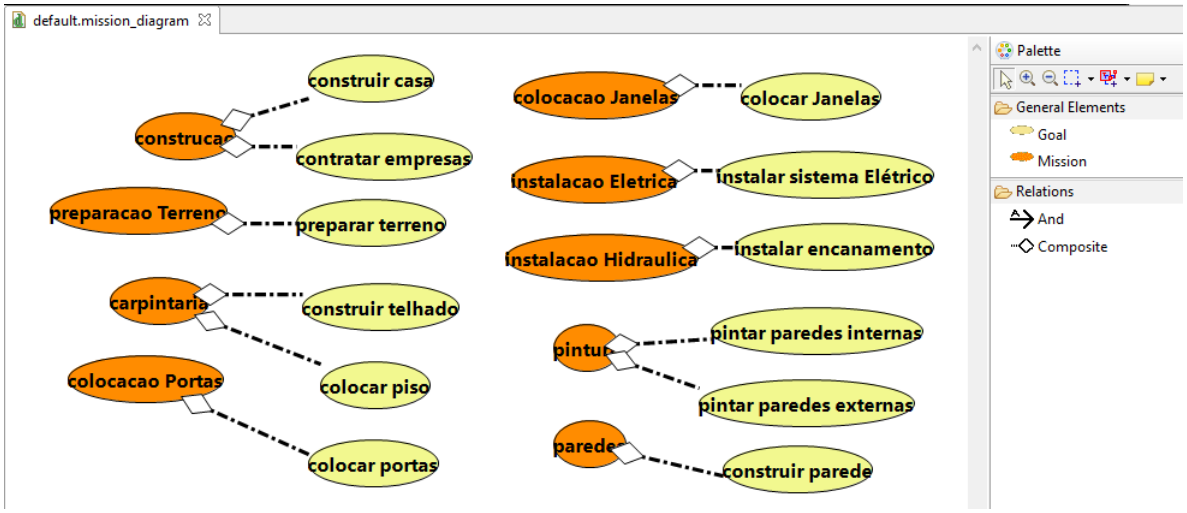


Figura 77: Diagrama de Missão - Estudo de Caso *Build a House*

A Figura 79 representa o diagrama Normativo modelado para o estudo de caso em questão. Este diagrama tem como finalidade demonstrar as missões e os papéis que executarão essas missões. Ao analisar esse diagrama, percebe-se que, sua diferença em relação ao diagrama de missões é a entidade *Role* e a entidade *Goal*. Por este motivo, acredita-se que é possível fazer uma unificação entre os diagramas de Objetivo, Missões e Normativo, sem perda nenhuma de expressividade. Além disso, teria uma diminuição considerável na complexidade de utilização da metodologia Prometheus AEOlus.

A Figura 80 representa o diagrama de Relacionamento entre Papéis e Agentes modelado para o estudo de caso em questão. Este diagrama ilustra os agentes que assumirão os papéis designados pelo sistema. É um dos diagramas cruciais na modelagem de um sistema, pois sem saber os papéis atribuídos para cada agente, torna-se inviável sua utilização em um SMA. Entretanto, o diagrama estrutural descreve os papéis, suas ligações entre si, e suas ligações com grupos distintos. Sendo assim, acredita-se que é possível unificar o

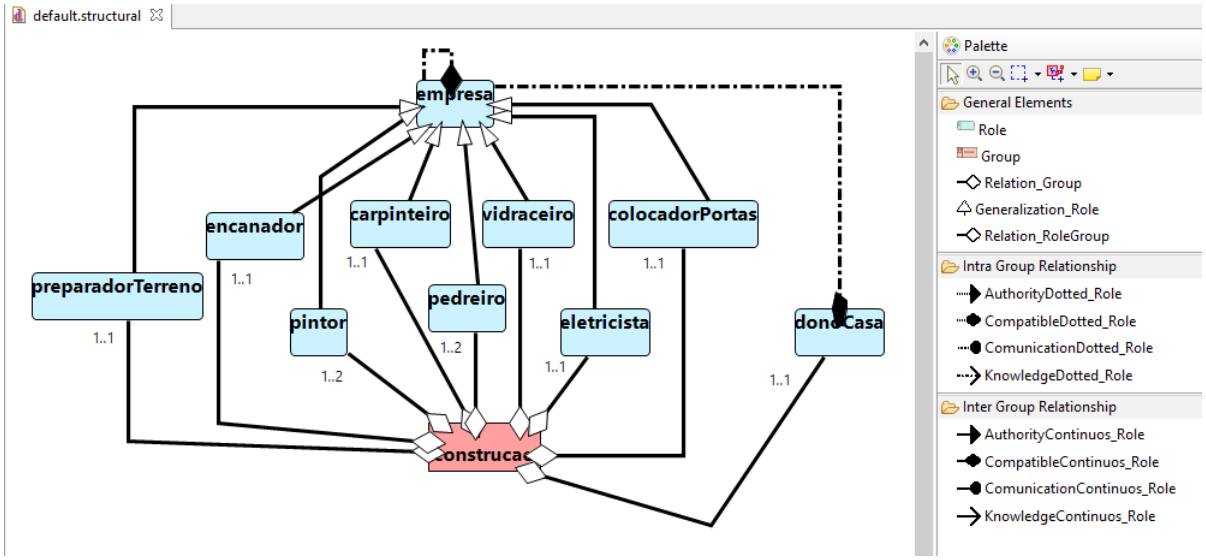


Figura 78: Diagrama Estrutural - Estudo de Caso *Build a House*

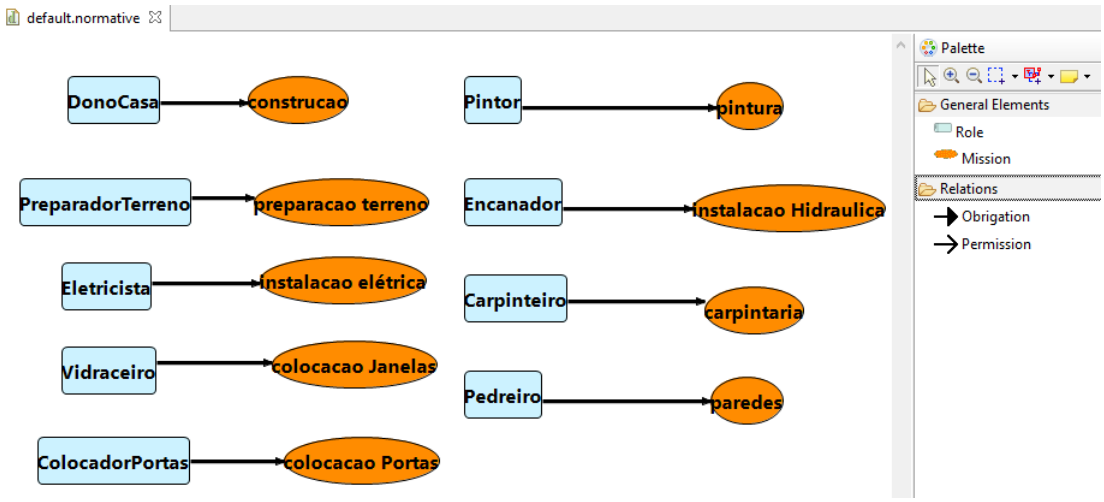


Figura 79: Diagrama Normativo - Estudo de Caso *Build a House*

diagrama intitulado *Relacionamento entre Papéis e Agentes* com o diagrama Estrutural, mantendo as informações de ambos e diminuindo a quantidade de diagramas necessários para a conclusão da modelagem.

A Figura 81 representa o diagrama de Visão Geral do Ambiente modelado para o estudo de caso em questão. Este diagrama possui um equívoco em sua modelagem. Em (UEZ, 2014) é afirmado que o diagrama ilustra somente as ações, as percepções, os artefatos e o *workspace* em sua modelagem. Entretanto, em (UEZ, 2013), o mesmo diagrama é modelado com o conceito de agentes. Este diagrama apresenta de forma sintetizada as entidades que compõem o ambiente do sistema.

A Figura 82 representa o diagrama de Visão Geral do Sistema modelado para o estudo de caso em questão. Em uma versão ampliada sobre as entidades modeladas no sistema, torna-se plausível unificar este diagrama com o de visão geral do ambiente. Entretanto,

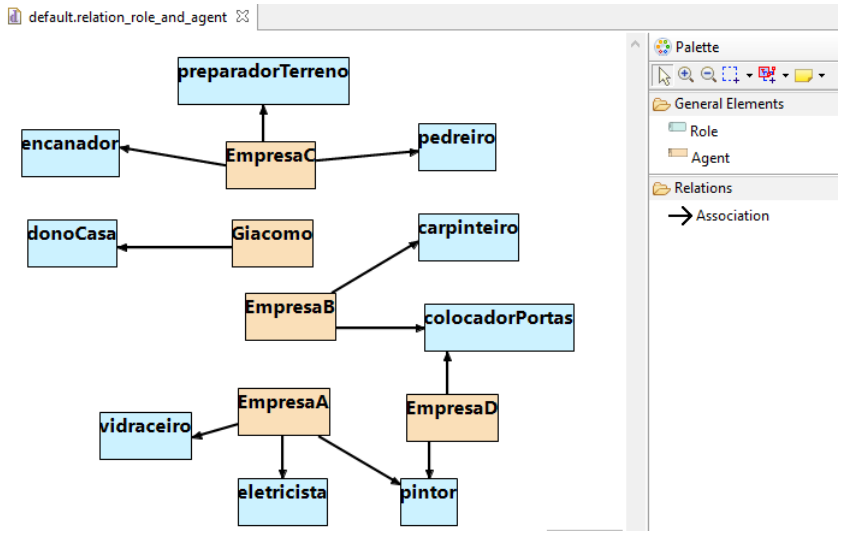


Figura 80: Diagrama de Relacionamento entre Papéis e Agentes - Estudo de Caso *Build a House*

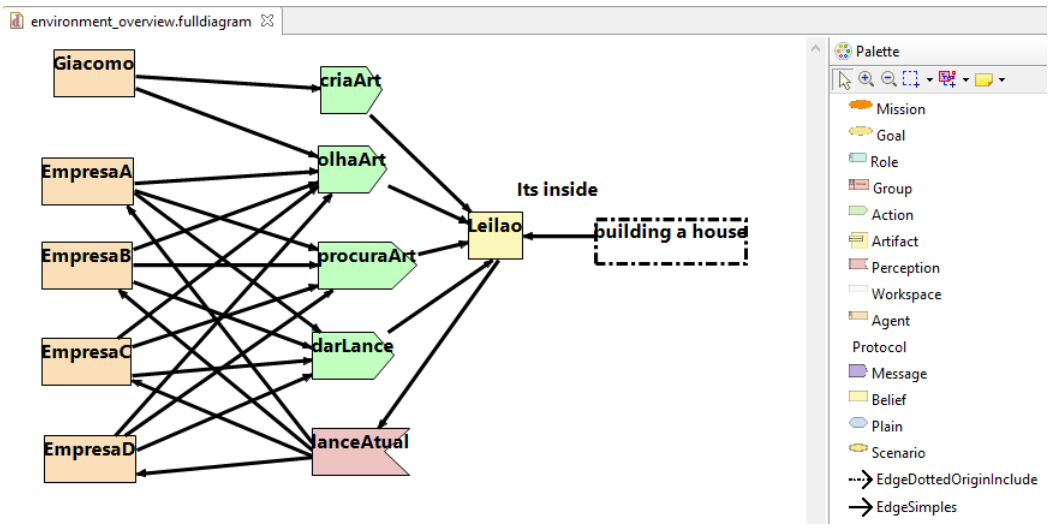


Figura 81: Diagrama de Visão Geral do Ambiente - Estudo de Caso *Build a House*

caso seja necessário modelar em detalhes os conceitos do ambiente, a separação torna-se a melhor opção.

As Figuras 83, 84, 85, 86 e 87 representam o diagrama de visão geral do Agente Giacomo, EmpresaA, EmpresaB, EmpresaC e EmpresaD, respectivamente modelados para o estudo de caso em questão. Estes diagramas são oriundos do Diagrama de Visão Geral do sistema, onde os agentes são modelados de forma simplificada, sendo-os estendidos e enriquecidos através dos diagramas de visão geral do Agente. Em questões de modelagem, torna-se custoso modelar mais diagramas, entretanto, aumenta o nível de detalhamento e a qualidade das informações fornecidas pela modelagem.

A Figura 88 representa o diagrama de Visão Geral de Capacidade modelado para o estudo de caso em questão. Este diagrama é proveniente dos diagramas de Visão geral de Agentes, modelados anteriormente seguindo a ordem da metodologia. O objetivo deste

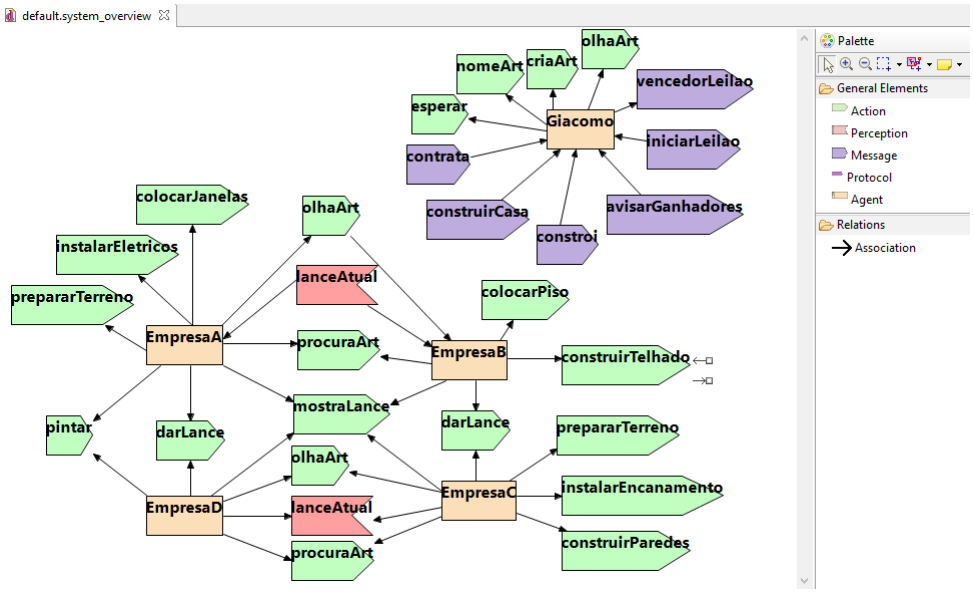


Figura 82: Diagrama de Visão Geral do Sistema - Estudo de Caso *Build a House*

diagrama é isolar entidades repetidas em vários diagramas de Visão geral de Agente, de modo que tornem-se isolados em um arquivo, semelhante ao processo utilizado por um método utilizado em vários objetos no paradigma de programação orientado a objetos. Ao analisar os Diagramas de Visão Geral do Sistema, Diagrama de Visão Geral do Agente e Diagrama de Visão Geral de Capacidade a fundo, percebe-se que tornaria mais fácil ao utilizador da metodologia se fosse possível modelá-los somente em um diagrama. Pois ambos possuem uma relação de independência, portanto, modelá-los de forma unificada simplificaria o processo. Essa alternativa facilitaria o processo de modelagem, porém, perderia na qualidade das informações, visto que um diagrama desse porte teria dimensões consideráveis.

6.2.1.1 Geração de Código para o Estudo de Caso Build a House

Conforme dito na subseção 5.1.11, a geração de códigos para a plataforma Jason é realizada por meio da modelagem do diagrama de Modelo Geral da metodologia Prometheus AEOlus. Nesta mesma subseção explicou-se os motivos da criação deste diagrama, principalmente de conter todas as informações reunidas em um mesmo local.

A Seção 3.2 explica a metodologia Prometheus AEOlus. Nessa seção é explanado que a finalidade principal da metodologia é ser aderente ao *framework JaCaMo*. Para este trabalho, conforme subseção 1.2, o objetivo é desenvolver a ferramente gráfica que apoie a metodologia Prometheus AEOlus e gerar um protótipo executável que possibilite que ocorra a geração de códigos para a plataforma Jason. Esta plataforma, concentra-se a codificação e execução dos agentes modelados pela metodologia. Portanto, as demais dimensões presentes no *framework* serão desconsideradas a fins de geração de código, contudo podem ser modeladas utilizando a ferramenta gráfica desenvolvida neste traba-

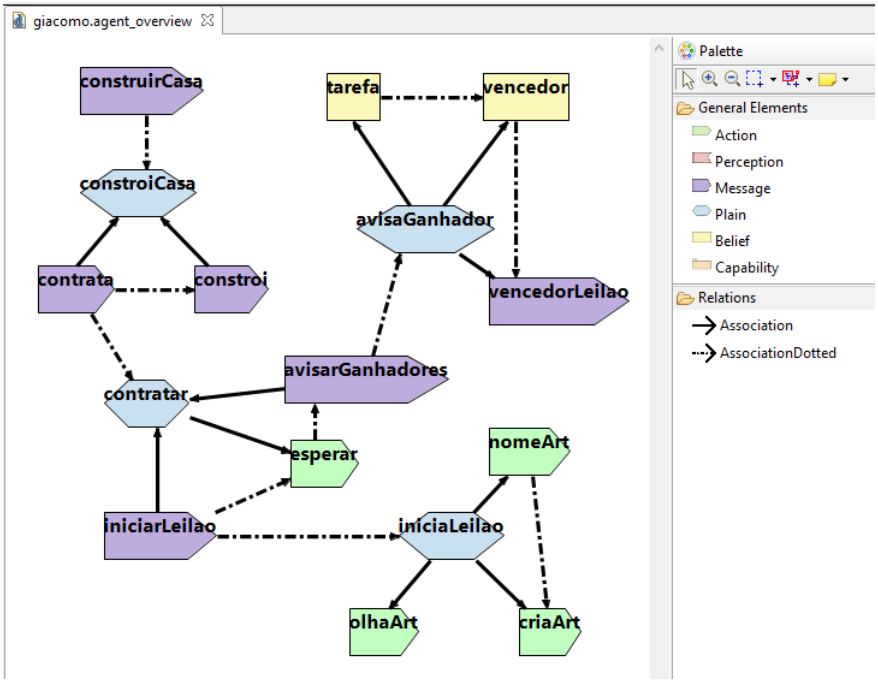


Figura 83: Diagrama de Visão Geral do Agente Giacomo - Estudo de Caso *Build a House*

lho.

Para este estudo de caso, foram modelados cinco agentes, sendo-os: *Giacomo*, *EmpresaA*, *EmpresaB*, *EmpresaC* e *EmpresaD*. A Figura 89 ilustra a modelagem do Agente *Giacomo*. Este diagrama foi também modelado na Figura 83. Percebe-se na comparação das figuras que existe uma disparidade entre as mesmas. As entidades que são ligadas ao plano apresentam, as vezes, uma relação do tipo *Origin*, a qual representa uma ordem de ocorrência das entidades no plano a ser executado. Por tratar-se de um protótipo, preferiu-se não desenvolver essa peculiaridade, visto que o objetivo principal do desenvolvimento dessa parte do projeto é demonstrar a viabilidade da metodologia ser aderente a plataforma Jason.

Outra característica implementada na geração de códigos foi a obrigatoriedade de se modelar os agentes no diagrama e ligá-los nos planos e mensagens definidas no estudo de caso utilizado. Além de ligar as mensagens aos respectivos planos que elas compõem, é necessário também ligá-las aos agentes que emitem e recebem essas mensagens. Este fato ocorre devido a impossibilidade de coletar essas informações de outra maneira sem ser a ligação das respectivas entidades no diagrama modelado. A outra especialidade trata da impossibilidade do *plug-in* gerador de códigos inferir os objetivos presentes em cada um dos agentes que compõem o sistema, sendo que mais detalhes sobre este problema serão relatados na Seção 7.2.

A última especificidade não desenvolvida para o protótipo em questão foi a geração de *capability* no diagrama de modelo geral da metodologia. Através da modelagem é até possível diagramar essas entidades. Entretanto, o motor de geração de códigos para

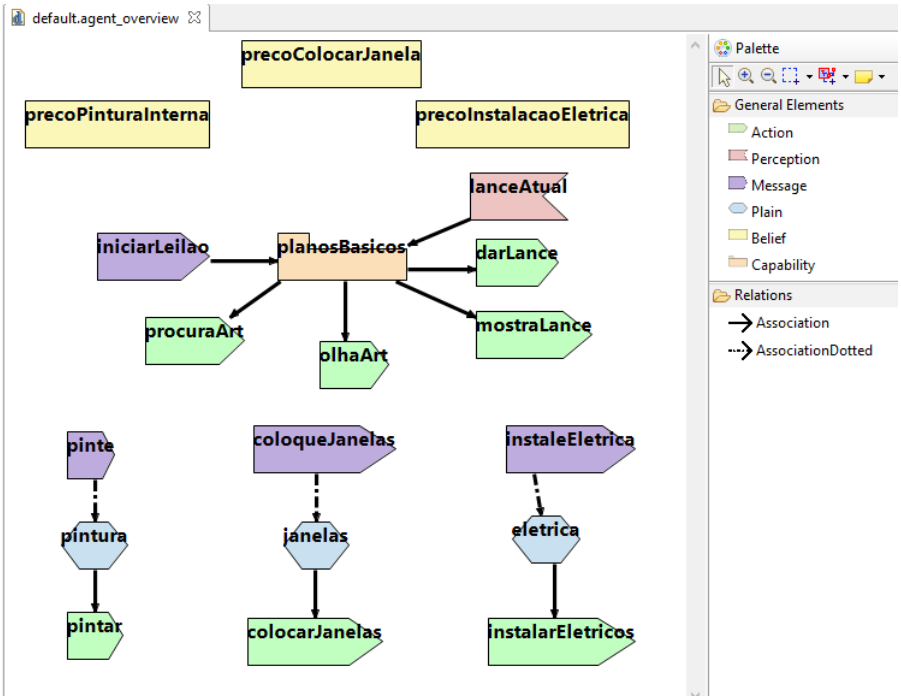


Figura 84: Diagrama de Visão Geral do Agente Empresa A - Estudo de Caso *Build a House*

a linguagem Jason ainda não é capaz de agrupá-las em arquivos distintos e importá-los nos agentes que a utilizam. Acredita-se que isso não invalida a aplicação, visto que é possível diagramar o problema excluindo essa peculiaridade. Todos os outros agentes desenvolvidos neste estudo de caso respeitam as características do *plug-in* gerador de códigos elencadas acima.

A Figura 90 exemplifica a geração de códigos ocorrida para o agente *Giacomo*. Esta geração é proveniente da modelagem do diagrama da Figura 89. Além disso, toda a parametrização das propriedades de cada um dos elementos foi baseado no estudo de caso de (UEZ, 2013). A linha 2 refere-se a uma mensagem disponibilizada ao utilizador do *plug-in*, para que o mesmo informe os objetivos individuais de cada um dos agentes gerados. Posteriormente, são informados os planos dos agentes, conforme já falado anteriormente. Cabe lembrar que na linguagem *AgentSpeak*, a sintaxe de um plano ocorre da seguinte forma:

Trigger : context <- body

As transformações realizadas dos diagramas em códigos-fontes foram baseadas em (UEZ, 2013) e (UEZ, 2014), cujo trabalhos demonstram, em diversos exemplos práticos, todas as possibilidades de transformação das entidades diagramadas em códigos-fontes.

As linhas 4, 9, 15 e 21 da Figura 90 representam a notação para representar que abaixo iniciará um plano. A linha 5 representa a *Trigger* e após os : o corpo de um plano. A *Trigger* é a situação para iniciar o plano. Conforme (UEZ, 2013), somente duas coisas podem representar uma *trigger* de um plano, sendo-as: Entidades do tipo *Mensagem* ou

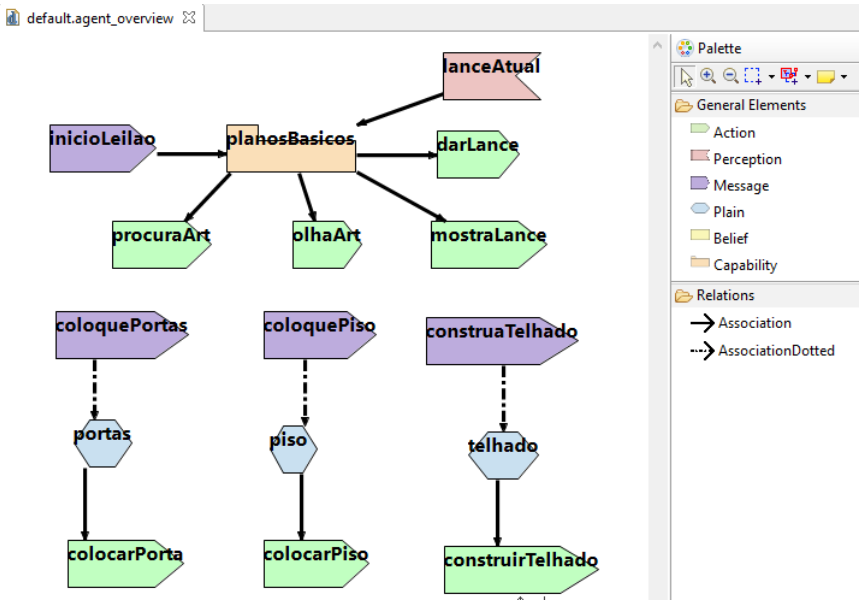


Figura 85: Diagrama de Visão Geral do Agente Empresa B - Estudo de Caso *Build a House*

Percepção ligadas ao plano através de setas pontilhadas. O *True* posterior aos : indica o contexto do plano, neste caso estipulado por parâmetros pelo diagramador. As linhas 6 e 7 representam respectivamente a inclusão de novos objetivos.

As linhas 10, 11, 12 e 13 representam o plano *contratar*. Conforme é possível visualizar, após o <- ocorre a concatenação de todas as entidades que compõem a *body* de um plano, sendo que a novidade deste plano é a entidade ação denominada *esperar* cuja *function* é *.wait(500)*.

As linhas 15, 16, 17 e 18 representam o plano *avisaGanhador*. A palavra *broadcast* refere-se a uma mensagem enviada para todos os agentes que compõem o sistema. Percebe-se que isso ocorre pois na entidade *vencedorLeilao* não existem ligações originadas da entidade e ligadas a outras entidades, consolidando a ideia da mensagem não ter um remetente exclusivo. Além disso, através da Figura 89, nota-se que as ligações entre as entidades mensagens e planos representam as mensagens que devem ser trocadas pelo sistema. Entretanto, é necessário que a mensagem também tenha a ligação com o agente remetente e os destinatários, quando houver o segundo caso. As linhas 18 e 19 representam consultas as crenças do plano e são originadas das ligações entre planos e crenças.

Por último, as linhas 22, 23, 24 e 25 representam o plano *iniciaLeilao*. Da linha 23 em diante, é apresentada uma concatenação de ações que compõem o plano em questão, findando assim a geração de códigos para o agente *Giacomo*.

A Figura 91 representa a modelagem do agente *EmpresaA* utilizando o diagrama de modelo geral da metodologia. Este diagrama é similar ao modelado na Figura 84. A disparidade entre as modelagens consiste nas peculiaridades necessárias para o motor de

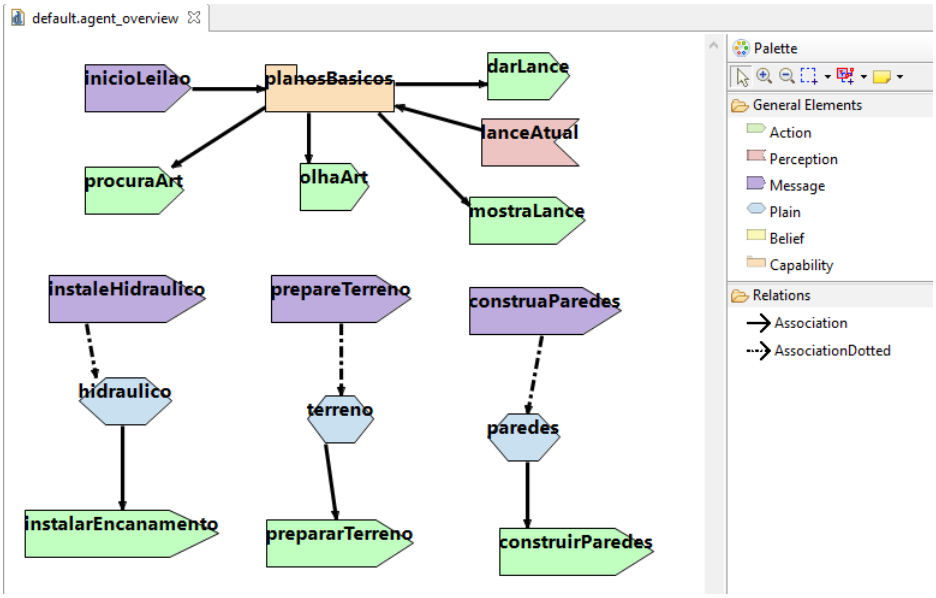


Figura 86: Diagrama de Visão Geral do Agente Empresa C - Estudo de Caso *Build a House*

geração de códigos funcionar, conforme explicado anteriormente. Além disso, neste e demais agentes explanados a seguir, excluiu-se de sua modelagem as *capabilities*. As entidades presentes nestas foram incluídas na concepção dos digramas dos agentes.

A Figura 92 ilustra os códigos gerados para o agente *EmpresaA*. As linhas 4, 5 e 6 representam as crenças iniciais que o agente *EmpresaA* possui. As linhas 8, 9, 10 e 11 evidenciam o plano *participaLeilao*. As linhas 13, 14, 15, 16 e 17 denotam o plano *lances*. Ambos os planos compõem a *capability planosBasicos* que foi excluída. Todos os demais diagramas demonstrados a seguir apresentarão esses planos, devido aos respectivos agentes possuírem ligação com a *capability* referenciada. Da linha 19 a 29 são descritos 3 planos referentes ao agente *EmpresaA*.

Os códigos gerados para os agentes Empresa B, C e D são ilustrados nas Figuras 93, 94 e 95, respectivamente. Optou-se por omitir suas representações de modelagem em virtude de serem semelhantes a ilustrada para o agente *EmpresaA* (Figura 91). A diferença consiste nos planos ligados a cada um desses agentes. Entretanto, mesmo apresentando planos diferentes, sua sintaxe permanece a mesma. Além disso, assim como o agente *EmpresaA*, todos os demais agentes referenciados aqui possuem originalmente a ligação com a *capability planosBasicos*. Portanto, apresentam os planos *participaLeilao* e *lances* inclusos.

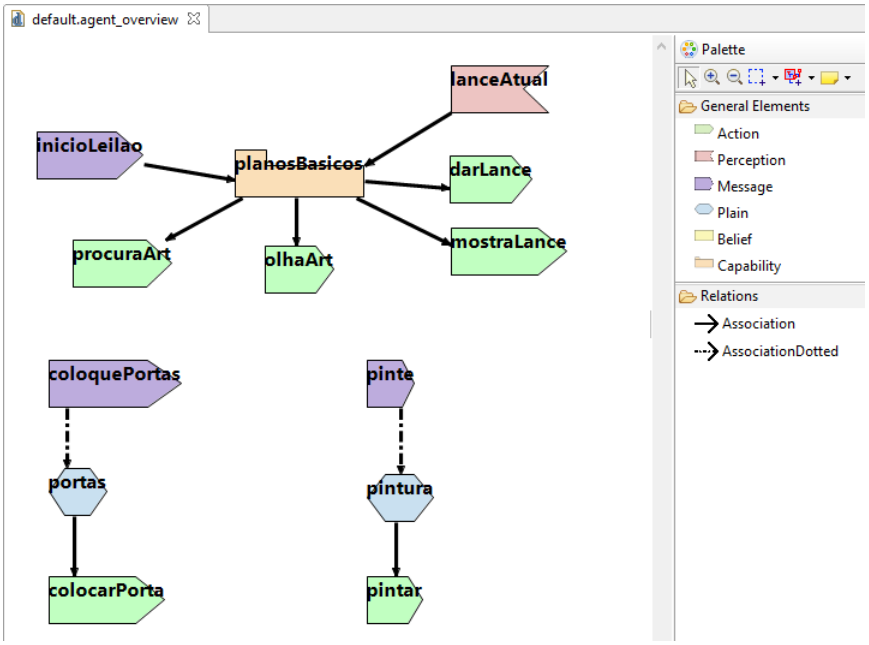


Figura 87: Diagrama de Visão Geral do Agente Empresa D - Estudo de Caso *Build a House*

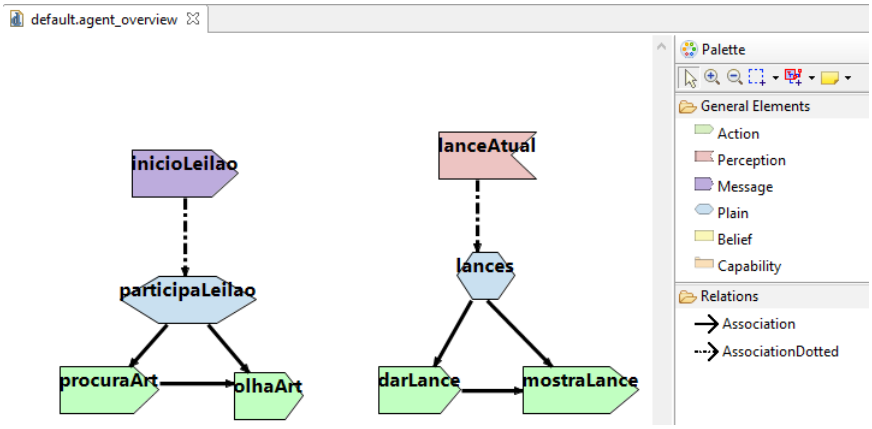


Figura 88: Diagrama de Visão Geral de Capacidade - Estudo de Caso *Build a House*

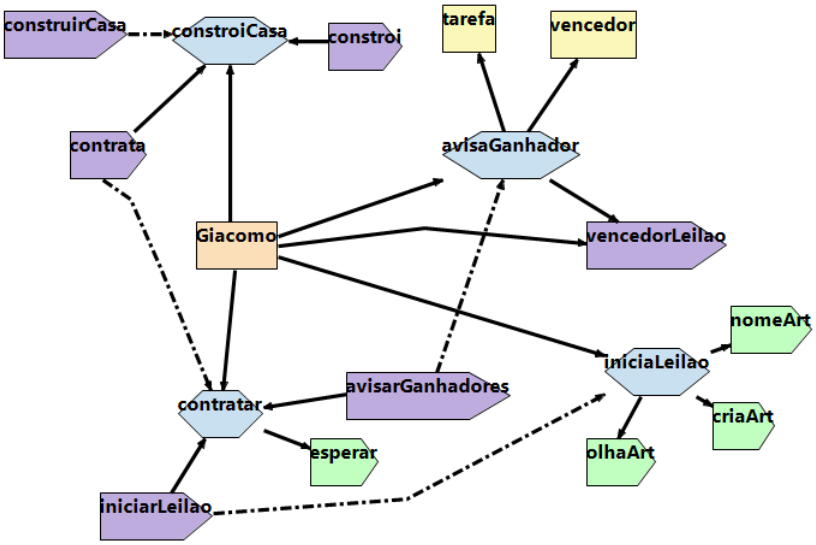


Figura 89: Diagrama de Modelo Geral do Agente Giacomo - Estudo de Caso *Build a House*

```
1
2  //insert here the goals to boot your agent
3
4  @constroiCasa
5  +!construirCasa :true
6  <-!contrata;
7  !constroi.
8
9  @contratar
10 +!contrata :true
11 <-!iniciarLeilao(x);
12 !avisarGanhadores;
13 .wait(500).
14
15 @avisaGanhador
16 +!avisarGanhadores :true
17 <-.broadcast(tell, vendedor(Tarefa, Vencedor));
18 ?tarefa;
19 ?vencedor.
20
21 @iniciaLeilao
22 +!iniciarLeilao(x) :true
23 <-.concat("leilao_",Tarefa,NomeArtefato);
24 focus(id);
25 makeArtifact(NomeArtefato, "leilao", [Tarefa,ValorMaximo],Id).
26
```

Figura 90: Código gerado para o Agente Giacomo - Estudo de Caso *Build a House*

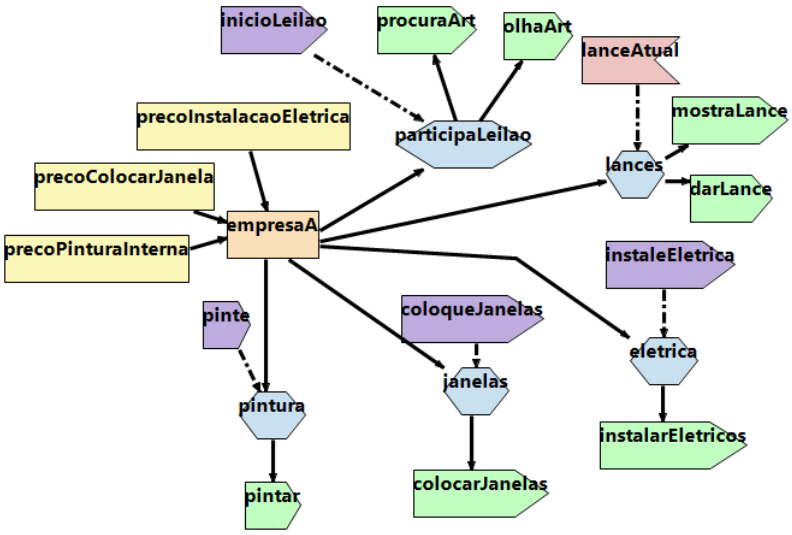


Figura 91: Diagrama de Modelo Geral do Agente EmpresaA - Estudo de Caso *Build a House*

```
2  //insert here the goals to boot your agent
3
4  preco("pintar_paredes_internas",1900);
5  preco("colocar_janela",1900);
6  preco("instalacao_eletrica",1900).
7
8  @participaLeilao
9  +!inicioLeilao :true
10 <-lookupArtifact(NomeArtefato,Id);
11 focus(Id).
12
13 @lances
14 +ultimoLance(LanceAtual)[artifact_id(Art)]:tarefaArtefato(Tarefa)
15 [artifact_id(Art)] & preco(Tarefa, Valor) & Valor < LanceAtual
16 <-lance(Valor);
17 .print("Meu lance foi" , Valor).
18
19 @eletrica
20 +!instaleEletrica :true
21 <-instalarEletricos.
22
23 @janelas
24 +!coloqueJanelas :true
25 <-colocarJanelas.
26
27 @pintura
28 +!pinte(Lugar) :true
29 <-pintar(Lugar).
```

Figura 92: Código gerado para o Agente EmpresaA - Estudo de Caso *Build a House*

```
2  //insert here the goals to boot your agent
3
4  @portas
5  +!coloquePortas :true
6  <-colocarPorta.
7
8  @piso
9  +!coloquePiso :true
10 <-colocarPiso.
11
12 @telhado
13 +!construaTelhado :true
14 <-construirTelhado.
15
16 @participaLeilao
17 +!inicioLeilao :true
18 <-lookupArtifact(NomeArtefato,Id);
19 focus(Id).
20
21 @lances
22 +ultimoLance(LanceAtual)[artifact_id(Art)]:tarefaArtefato(Tarefa)
23 [artifact_id(Art)] & preco(Tarefa, Valor) & Valor < LanceAtual
24 <-.print("Meu lance foi" , Valor);
25 lance(Valor).
```

Figura 93: Código gerado para o Agente EmpresaB - Estudo de Caso *Build a House*

```
2  //insert here the goals to boot your agent
3
4  @hidraulico
5  +!instaleHidraulico :true
6  <-instalarEncanamento.
7
8  @terreno
9  +!prepareTerreno :true
10 <-prepararTerreno.
11
12 @paredes
13 +!construaParedes :true
14 <-construirParedes.
15
16 @participaLeilao
17 +!inicioLeilao :true
18 <-lookupArtifact(NomeArtefato,Id);
19 focus(Id).
20
21 @lances
22 +ultimoLance(LanceAtual)[artifact_id(Art)]:tarefaArtefato(Tarefa)
23 [artifact_id(Art)] & preco(Tarefa, Valor) & Valor < LanceAtual
24 <-.print("Meu lance foi" , Valor);
25 lance(Valor).
```

Figura 94: Código gerado para o Agente EmpresaC - Estudo de Caso *Build a House*

```
2  //insert here the goals to boot your agent
3
4  @portas
5  +!coloquePortas :true
6  <-colocarPorta.
7
8  @pintura
9  +!pinte(Lugar) :true
10 <-pintar(Lugar).
11
12 @participaLeilao
13 +!inicioLeilao :true
14 <-lookupArtifact(NomeArtefato,Id);
15 focus(Id).
16
17 @lances
18 +ultimoLance(LanceAtual)[artifact_id(Art)]:tarefaArtefato(Tarefa)
19 [artifact_id(Art)] & preco(Tarefa, Valor) & Valor < LanceAtual
20 <-.print("Meu lance foi" , Valor);
21 lance(Valor).
```

Figura 95: Código gerado para o Agente EmpresaD - Estudo de Caso *Build a House*

7 CONCLUSÃO

Esta dissertação propõe a criação de um *plug-in* de modelagem gráfica que tem como objetivo principal apoiar a metodologia Prometheus AEOlus. Esta metodologia foi desenvolvida por (UEZ, 2013) e tem como finalidade permitir a modelagem integrada das três dimensões que envolvem um SMA: agente, organização e ambiente. Esta integração visa a interligação dos conceitos modelados nos *work products* com o *framework* JaCaMo de desenvolvimento de SMA. Para colaborar com o processo de integração com o *framework* em questão, desenvolveu-se agregado ao *plug-in* gráfico, um mecanismo de varredura de informações que propicia a geração automática de códigos fontes para a linguagem *AgentSpeak*, interligando o *plug-in* ao ambiente de desenvolvimento de agentes chamado Jason, que faz parte do *framework* JaCaMo. As dimensões de ambiente (CartAgO) e organização (Moise+), as quais compõem o restante do *framework* JaCaMo não foram acopladas ao mecanismo de geração de códigos presente neste trabalho, pois deverá ser feito um estudo aprofundado acerca desse processo de interação entre as demais dimensões presentes no SMA, visto é um processo complexo. Entretanto, mesmo não sendo acopladas no mecanismo de geração de códigos, são plausíveis de serem modeladas utilizando a ferramenta gráfica desenvolvida.

Para avaliar o *plug-in* desenvolvido, forma realizados dois testes distintos. O primeiro teste, intitulado unitário, teve como objetivo avaliar a capacidade do *plug-in* em gerar códigos fontes funcionais para a linguagem de programação *agentspeak*. Essa avaliação ocorreu através de métodos de análise de cobertura dos códigos referentes as classes responsáveis por tal finalidade no *plug-in*. Os resultados foram descritos em 6.1. Constata-se que o *plug-in*, no quesito geração de códigos, apresenta um bom desempenho, gerando esqueletos de códigos funcionais para seus utilizadores. O segundo teste, teve como finalidade avaliar a eficácia do *plug-in* em apoiar os diagramas especificados na metodologia Prometheus AEOlus. Realizou-se dois estudos de casos, descritos em 6.2 e apêndice A, onde utilizou-se toda a gama de diagramas presentes na metodologia, comprovando que o *plug-in* é capaz de gerar *work products* fiéis aos diagramas especificados na descrição da metodologia.

7.1 Contribuições

A principal contribuição dessa dissertação está no desenvolvimento de uma ferramenta computacional que apoia a metodologia Prometheus AEOlus (UEZ, 2013). DEMAZEAU (1995) propõe a divisão de um SMA em quatro componentes, conforme descrito em 2.2. Através de um comparativo realizado em 2.4.2, constatou-se que nenhuma outra metodologia era capaz de suportar todas as dimensões presentes em um SMA. Desta forma, a metodologia criada supre uma necessidade da área de desenvolvimento de SMA. Entretanto, a mesma não tinha nenhuma ferramenta para suporte a suas especificações, fazendo-o deste trabalho uma contribuição para o avanço da metodologia.

Outra contribuição importante da ferramenta desenvolvida consiste na possibilidade de interligação dos diagramas desenvolvidos ao *framework* JaCaMo, especificamente ao ambiente de desenvolvimento Jason. Através do trabalho realizado, torna-se evidente a viabilidade de interligar os *work products* produzidos por usuários ao ambiente Jason, fazendo-o com que o usuário da metodologia e da ferramenta ganhe qualidade e eficiência em sua utilização. As demais dimensões do *framework* não foram interligadas neste trabalho, conforme já dito anteriormente.

Esse trabalho também gerou algumas publicações (aceitas ou submetidas):

- Engenharia de Software Orientada a Agentes: Um Estudo Comparativo entre UML e Metodologias que Suportam o Processo de Desenvolvimento de Sistemas Multiagente - Workshop-Escola de Agentes, seus Ambientes e ApliCações (WESAAC) - 2015 - (**aceito**).

- Engenharia de Software Orientada a Agentes: Um Estudo Comparativo entre Metodologias que Suportam o Processo de Desenvolvimento de Sistemas Multiagente - The 2nd Latin-American School on Software Engineering (ELA-ES) - 2015 - (**aceito**).

- Agent Oriented Software Engineering: A Comparative Study between methodologies that support the Development of Multi-Agent Systems - The 7th International Conference on ManagEment of Digital EcoSystems (MEDES) - 2015 - (**aceito**).

- Integrando uma Ferramenta Gráfica aderente ao Prometheus e o Framework Jason utilizando Arquitetura Orientada a Modelos - Mostra Produção Universitária da FURG (MPU) - 2015 - (**aceito**).

- Uma Proposta de Arquitetura para um Editor Gráfico de Apoio à metodologia Prometheus AEOlus - Revista de Informática Teórica e Aplicada (RITA) - 2015 - (**submetido**).

- Processo de Desenvolvimento de uma Ferramenta Gráfica de Apoio a Metodologia Prometheus AEOlus - Workshop-Escola de Agentes, seus Ambientes e ApliCações (WESAAC) - 2016 - (**submetido**).

- A Graphic Tool to Support Prometheus AEOlus Methodology - The 28th International Conference on Software Engineering and Knowledge Engineering (SEKE) - 2016 - (**submetido**).

7.2 Análise da Metodologia Prometheus AEOlus

A metodologia Prometheus AEOlus é uma metodologia de desenvolvimento de SMA que possui caráter inovador e indiscutivelmente contribuiu para o avanço no quesito modelagem e desenvolvimento de SMA. Entretanto, após seu entendimento e implementação em uma ferramenta gráfica, ela possui algumas deficiências que poderão ser corrigidas em versões futuras:

- A entidade plano é descrita no diagrama de Visão Geral do Agente, sendo necessário anteriormente modelar os agentes através do Diagrama de Visão Geral do Sistema.

- Em (UEZ, 2013), enfatiza-se apenas a necessidade de desenvolver o diagrama de Visão Geral do Sistema e o de Visão Geral dos Agentes para a geração de códigos para a linguagem *agentspeak*. Verificou-se, nesse trabalho, que além desses diagramas é preciso o desenvolvimento do diagrama de *capability* para concluir o processo de geração de códigos.

- As entidades mensagens são ligadas aos agentes através do diagrama de Visão Geral do Sistema e posteriormente interligadas aos planos através do diagrama de Visão Geral dos Agentes. Esse fato provoca uma interdependência entre os diagramas da metodologia, tornando-a mais difícil de ser utilizada.

- As entidades capacidades são descritas através do diagrama de Visão Geral do Agente e posteriormente detalhadas em seus próprios diagramas, intitulado diagrama de *Capability*. Outro fator que acarreta a interdependência entre diagramas.

- A entidade mensagem possui como forma de envio o tipo *ASK*, o qual é caracterizado como uma mensagem síncrona. Entretanto, a metodologia não oferece nenhuma forma de validação que obrigue ao seu utilizador informar uma resposta para a mesma.

- As entidades mensagens ligadas as *capabilities* não possuem nenhuma relação com o agente. Torna-se inviável descobrir qual agente é receptor e/ou destinatário dessas mensagens.

- A metodologia tem como proposta principal a interligação com o *framework* JaCaMo. Parte deste *framework* é o ambiente de desenvolvimento de agentes chamado Jason. Este, utiliza como base a linguagem *agentspeak*. Esta linguagem adota como premissa que os agentes possuam objetivos individuais. Entretanto, a metodologia Prometheus AEOlus trata os objetivos como pertencentes ao sistema como um todo. Esses objetivos se tornam objetivos de agentes posteriormente a modelagem de missões e a interligação de missões com papéis e de desses papéis com agentes. Acredita-se que deveria ter uma forma simplificada de interligar objetivos com agentes, possibilitando a simplificação de informações caso o usuário da metodologia necessitasse apenas modelar agentes.

Além disso, a metodologia Prometheus AEOlus é composta por doze diagramas. Sugere-se uma redução considerável em sua gama de diagramas, levando em consideração a replicação de diversas informações entre os mesmos. Todos detalhes desses diagramas foram descritos em 3.2 e as possibilidades de unificação em 6.2.

- O diagrama de Visão Geral de Papéis poderia ser unificado com o diagrama de Relacionamento entre papéis e agentes devido a apresentar informações que se completam.

- O diagrama de Missões e o de objetivos poderiam ser unificados, visto que a única diferença entre ambos consiste na entidade Missão.

- O diagrama de Missões, Objetivos e Normativos poderiam também ser unificados, visto que apresentam informações similares e complementares.

- O diagrama de Relacionamento entre Papéis e Agentes poderia ser unificado com o diagrama Estrutural, ambos tratam das ligações possíveis entre papéis.

- O diagrama de Visão Geral do Sistema, Visão Geral do Agente e *Capability* poderia ser pensado de forma unificada. Na ferramenta desenvolvida criou-se o diagrama de modelo geral da metodologia Prometheus AEOlus que torna essa possibilidade viável. Entretanto, acredita-se que deveria ser feito um estudo aprofundado para averiguar o impacto que causará se essa unificação fosse oficializada.

7.3 Trabalhos Futuros

A ferramenta desenvolvida para o apoio a metodologia Prometheus AEOlus é um protótipo e, apresenta uma série de possibilidades de melhorias, dentre elas:

- Implementar a interligação dos *work products* gerados pelo usuário com a plataforma CartAgO e Moise+.

- Exportar o *plug-in* desenvolvido para arquivos com a extensão .jar, tornando-o independente da IDE Eclipse.
- Implementar validações aos diagramas desenvolvidos por intermédio do *plug-in*, de modo que seja possível validar semanticamente os *work products* produzidos pelo usuário antes de realizar a geração de códigos para o *framework* JaCaMo.
- Realizar testes de usuário em ambientes reais para provar a eficiência do *plug-in* em resolver problemas existentes no mundo atual.
- Implementar a possibilidade de exportar por meio de imagens ou relatórios os *work produts* desenvolvidos no *plug-in* utilizado.

APÊNDICE A ESTUDO DE CASO - *CONFERENCE MANAGEMENT SYSTEM* (CMS)

O estudo de caso *Conference Management System* foi proposto por (GIORGINI, 2009). O objetivo dele é descrever um a criação e gerenciamento de um sistema de conferências em um processo multi-fases que envolvem várias pessoas e grupos. Durante a fase de submissão, os autores precisam ser informados que o seu papel foi recebido (um número de apresentação atribuído). Uma vez que o prazo foi passado, o comitê de programa (PC) tem de gerir a revisão dos documentos: entrar em contato com potenciais colaboradores; pedindo-lhes para rever uma série de documentos. Os revisores podem decidir aceitar ou não rever um artigo e, em caso de aceitação, eles têm que realizar uma revisão por um determinado prazo. Os comentários são recolhidos e utilizados para decidir sobre a aceitação do artigo pelo PC. Os autores serão notificados pelo PC sobre a aceitação/rejeição do artigo. Em caso de aceitação, eles são convidados a produzir uma versão revista do artigo (*camera-ready*). Por fim, a editora tem que receber todos os artigos aprovados e imprimir os anais do evento.

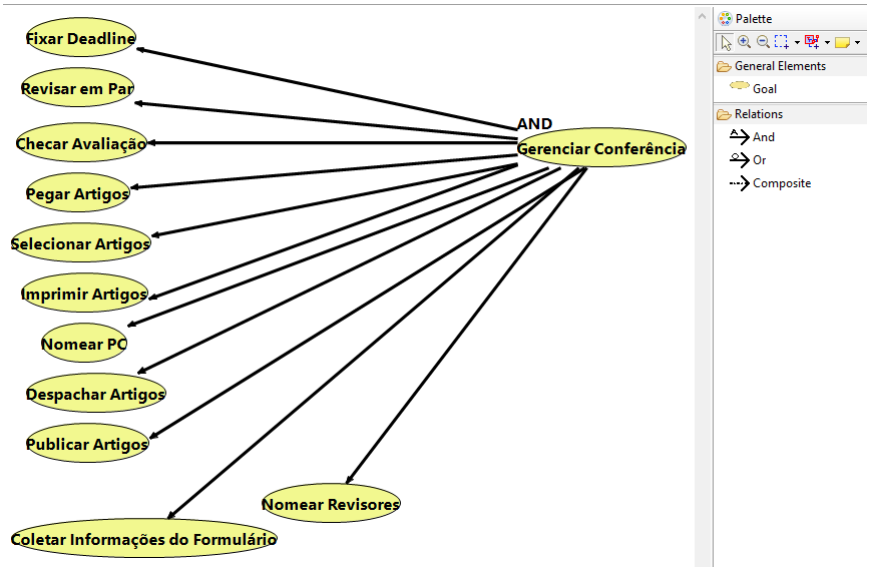


Figura 96: Diagrama de Objetivos - Estudo de Caso CMS

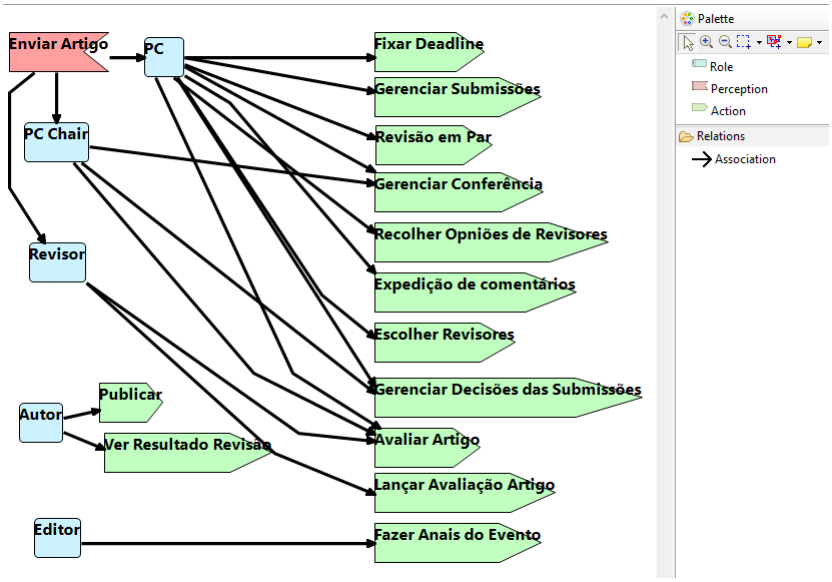


Figura 97: Diagrama de Visão Geral de Papéis - Estudo de Caso CMS

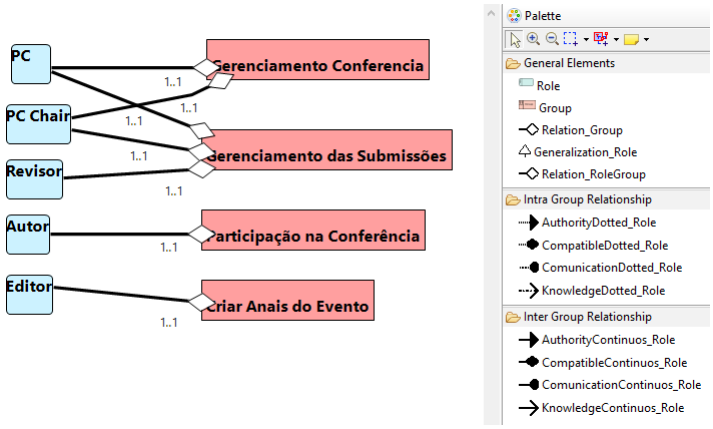


Figura 98: Diagrama Estrutural - Estudo de Caso CMS

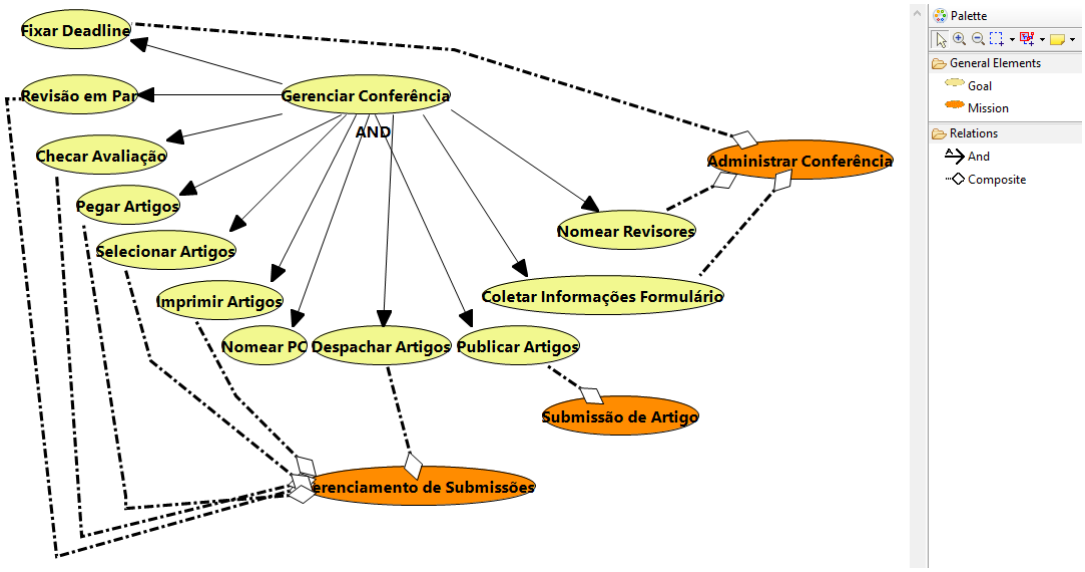


Figura 99: Diagrama de Missões - Estudo de Caso CMS

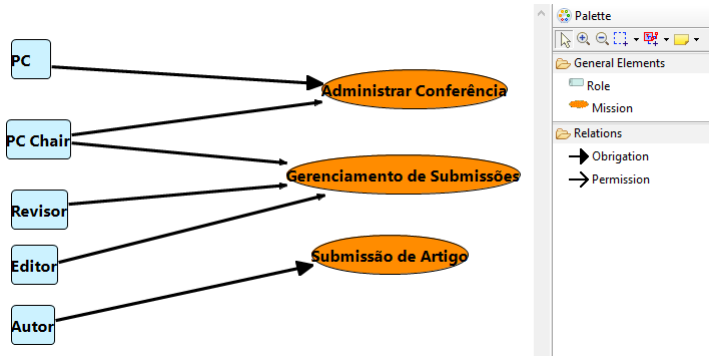


Figura 100: Diagrama Normativo - Estudo de Caso CMS

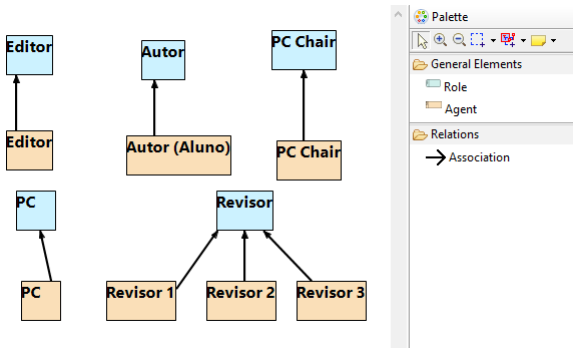


Figura 101: Diagrama de Relacionamento entre Papéis e Agentes - Estudo de Caso CMS

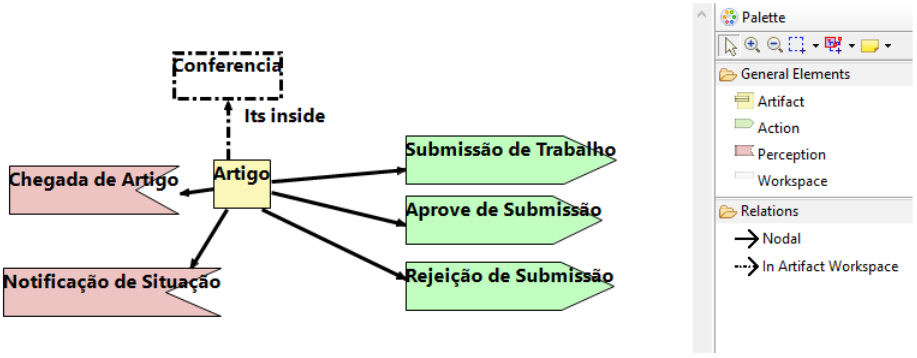


Figura 102: Diagrama de Visão Geral do Ambiente - Estudo de Caso CMS

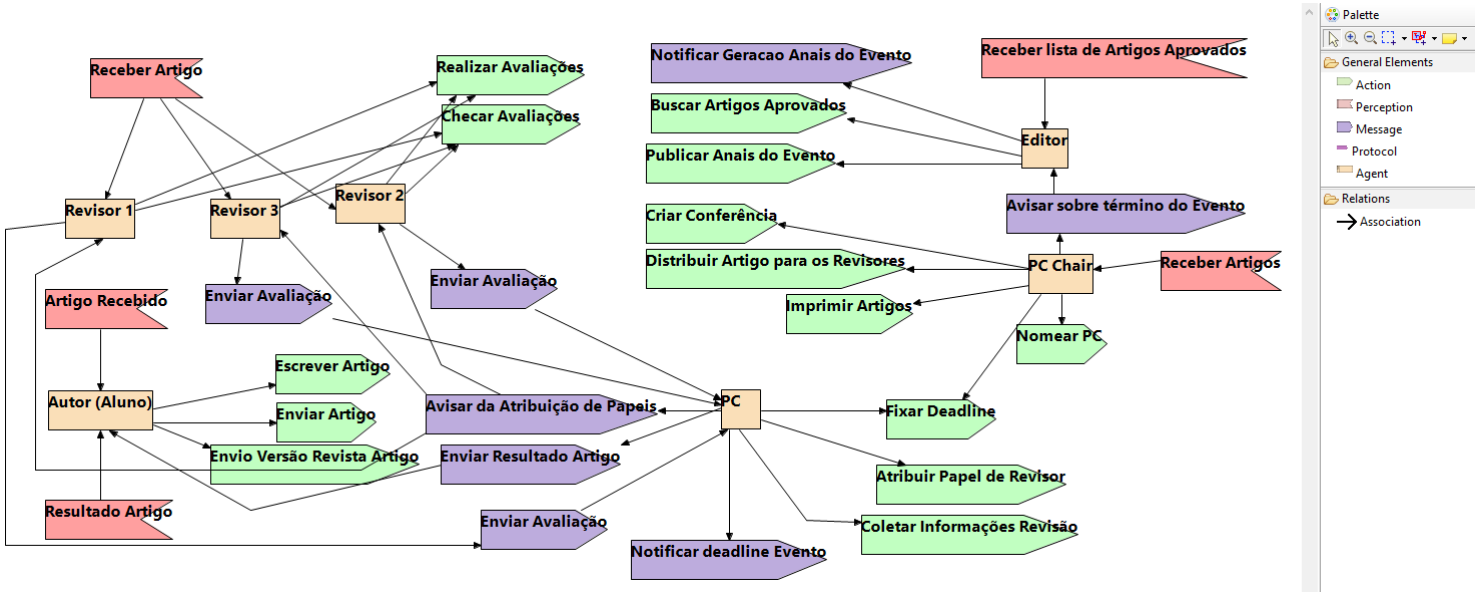


Figura 103: Diagrama de Visão Geral do Sistema - Estudo de Caso CMS

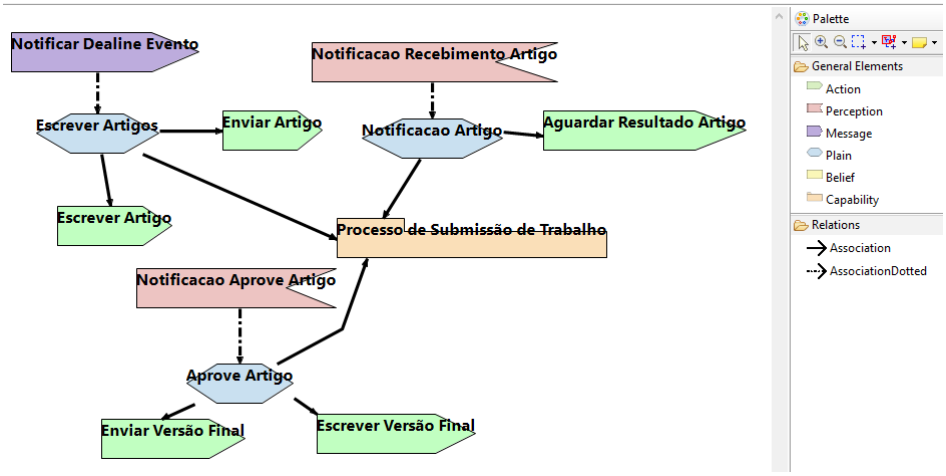


Figura 104: Diagrama de Visão Geral do Agente Autor - Estudo de Caso CMS

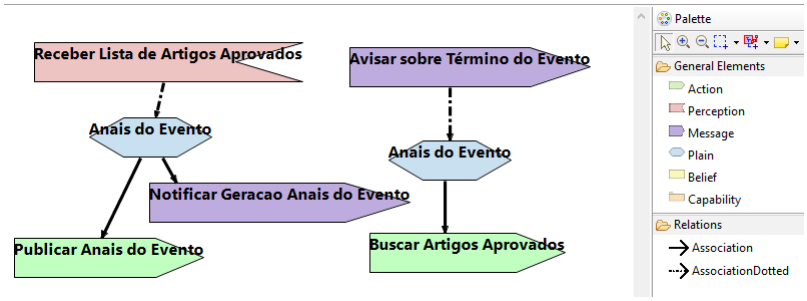


Figura 105: Diagrama de Visão Geral do Agente Editor - Estudo de Caso CMS

A.1 Geração de Códigos para o Estudo de Caso CMS

Nesta seção são demonstrados os códigos gerados para os agentes Autor, Editor, PC, PC Chair e Revisor respectivamente. Salienta-se que os códigos são gerados com base nos diagramas de visão geral do agente descritos na Seção A. Além disso, é ilustrado também o arquivo de configuração necessário para configurar o ambiente de execução do Jason.

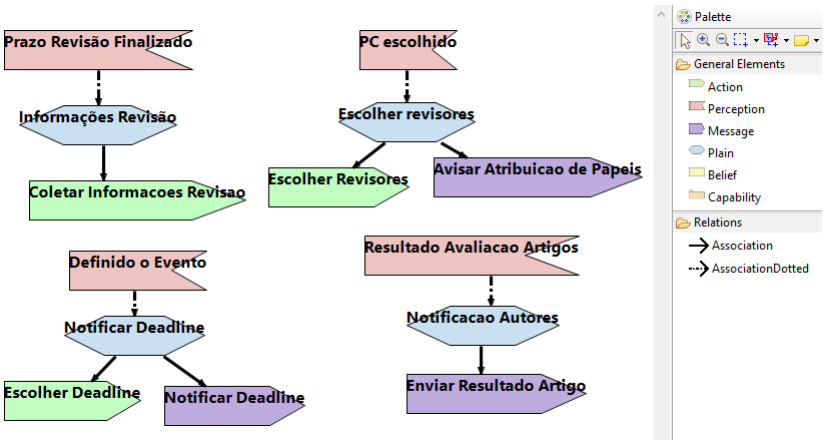


Figura 106: Diagrama de Visão Geral do Agente PC - Estudo de Caso CMS

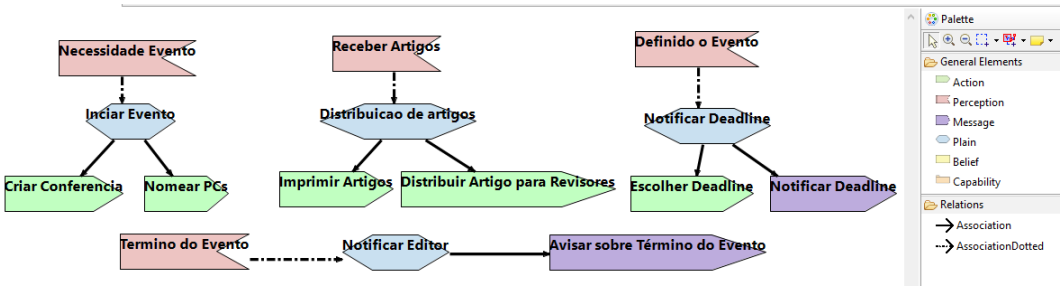


Figura 107: Diagrama de Visão Geral do Agente PC Chair - Estudo de Caso CMS

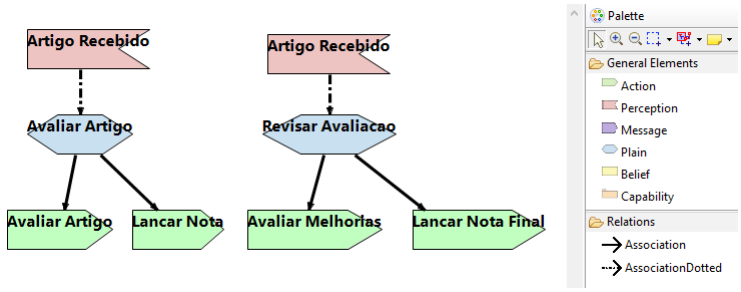


Figura 108: Diagrama de Visão Geral do Agente Revisor - Estudo de Caso CMS

```
MAS CMS
    {
        infraestructure JaCaMo
        agents:
            Autor #1;
            Editor #1;
            PC #1;
            PcChair #1;
            Revisor #3;
        aslSourcePath: "src/asl"
    }
```

Figura 109: Código gerado para configuração - Estudo de Caso CMS

```
1  //insert here the goals to boot your agent
2
3  @escreverArtigos
4  +!notificarDeadlineEvento :true
5  <-escreverArtigo.
6
7  @notificacaoArtigo
8  +notificacaoRecebimentoArtigo:true
9  <-aguardarResultadoArtigo.
10
11 @aproveArtigo
12 +notificacaoAproveArtigo:true
13 <-escreverVersaoFinal;
14 enviarVersaoFinal.
```

Figura 110: Código gerado para o agente Autor - Estudo de Caso CMS

```
1  //insert here the goals to boot your agent
2
3  @anaisEvento
4  +receberListaArtigosAprovados:true
5  <-.broadcast(tell, notificar(Autor,PC,PCChair,Revisor));
6  publicarAnaisEvento;
7  buscarArtigosAprovados.
8
9  @anaisEvento
10 +!avisarSobreTerminoEvento :true
11 <-publicarAnaisEvento;
12 buscarArtigosAprovados.
```

Figura 111: Código gerado para o agente Editor - Estudo de Caso CMS

```
1  //insert here the goals to boot your agent
2
3  @informacoesRevisao
4  +prazoRevisaoFinalizado:true
5  <-coletarInformacoesRevisao.
6
7  @notificarDeadline
8  +definidoEvento:true
9  <-.broadcast(tell, notificarDeadline);
10 escolherDeadline.
11
12 @notificacaoAutores
13 +resultadoAvaliacaoArtigo:true
14 <-.broadcast(tell, enviarResultadoArtigo).
15
16 @escolherRevisores
17 +pcEscolhido:true
18 <-.broadcast(tell, avisarAtribuicaoPapel);
19 escolherRevisores.
```

Figura 112: Código gerado para o agente PC - Estudo de Caso CMS

```
1  //insert here the goals to boot your agent
2  @iniciarEvento
3  +necessidadeEvento:true
4  <-criarConferencia;
5  nomearPCs.
6
7  @distribuicaoArtigos
8  +receberArtigos:true
9  <-imprimirArtigos;
10 distribuirArtigosParaRevisores.
11
12 @notificarEditor
13 +terminoEvento:true
14 <-.broadcast(tell, avisarSobreTerminoEvento(Editor)).
15
16 @notificarDeadline
17 +definidoEvento:true
18 <-.broadcast(tell, notificarDeadline);
19 escolherDeadline.
```

Figura 113: Código gerado para o agente PC Chair - Estudo de Caso CMS

```
1
2  //insert here the goals to boot your agent
3
4  @avaliarArtigo
5  +artigoRecebido:true
6  <-lancarNota;
7  avaliarArtigo.
8
9  @revisarAvaliacao
10 +artigoRecebido:true
11 <-lancarNotaFinal;
12 avaliarMelhorias.
```

Figura 114: Código gerado para o agente Revisor - Estudo de Caso CMS

APÊNDICE B VALIDAÇÃO DO GERADOR DE CÓDIGOS - ARQUIVOS .ASL

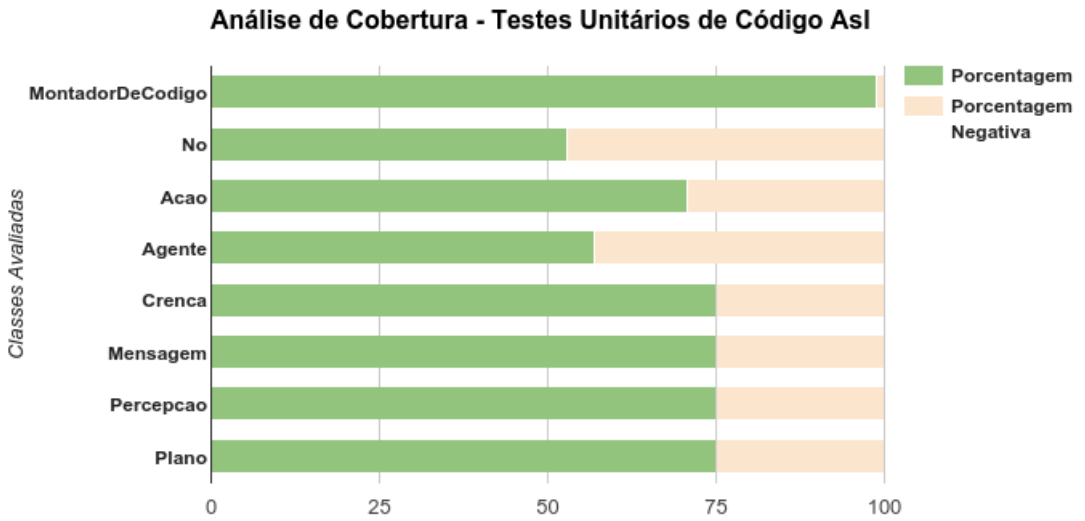


Figura 115: Cobertura de testes unitários para geração de código asl.

Na Figura 115 é apresentada a cobertura de testes para geração de código “asl”. As classes analisadas foram *MontadorDeCodigo, No, Acao, Agente, Crenca, Mensagem, Percepcao* e *Plano*. A maior porcentagem de cobertura encontrada foi de 98,8%, com a classe *MontadorDeCodigo*. A porcentagem negativa desta classe ficou em cerca de 1,2%, o que responde o baixo índice de cobertura para geração de arquivos “mas2j”. Justifica-se esse índice em virtude da classe possuir uma quantidade de códigos superior para a geração de arquivos “asl”. Quanto as demais classes avaliadas, a porcentagem obtida foi aproximadamente 69%, considerando também apenas as estruturas de geração de código para “asl”.

REFERÊNCIAS

BELLIFEMINE, F.; BERGENTI, F.; CAIRE, G.; POGGI, A. JADE: a Java Agent Development Framework. In: MULTI-AGENT PROGRAMMING, 2005. **Anais...** Springer, 2005. p.125–147.

BERGENTI, F.; GLEIZES, M.-P.; ZAMBONELLI, F. **Methodologies and software engineering for agent systems**: the agent-oriented software engineering handbook. [S.l.]: Springer, 2004. v.11.

BEYDOUN, G.; LOW, G.; HENDERSON-SELLERS, B.; MOURATIDIS, H.; GOMEZ-SANZ, J. J.; PAVÓ, J.; GONZALEZ-PEREZ, C. FAML: a generic metamodel for MAS development. **Software Engineering, IEEE Transactions on**, [S.l.], v.35, n.6, p.841–863, 2009.

BIASI, L. B. **Geração automatizada de drivers e stubs de teste para JUnit a partir de especificações U2TP**. 2006. Dissertação (Mestrado em Ciência da Computação) — Pontifícia Universidade Católica do Rio Grande do Sul.

BINDER, R. **Testing object-oriented systems**: models, patterns, and tools. [S.l.]: Addison-Wesley Professional, 2000.

BOISSIER, O.; BORDINI, R. H.; HÜBNER, J. F.; RICCI, A.; SANTI, A. Multi-agent oriented programming with JaCaMo. **Science of Computer Programming**, [S.l.], v.78, n.6, p.747–761, 2013.

BORDINI, R. H.; BRAUBACH, L.; DASTANI, M.; SEGHROUCHNI, A. E. F.; GOMEZ-SANZ, J. J.; LEITE, J.; O'HARE, G.; POKAHR, A.; RICCI, A. A survey of programming languages and platforms for multi-agent systems. **Informática**, [S.l.], v.30, n.1, 2006.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming multi-agent systems in AgentSpeak using Jason**. [S.l.]: John Wiley & Sons, 2007. v.8.

BORDINI, R. H.; VIEIRA, R. Linguagens de Programação Orientadas a Agentes: uma introdução baseada em AgentSpeak (L). **Revista de informática teórica e aplicada. Porto Alegre. Vol. 10, n. 1 (2003), p. 7-38**, [S.l.], 2003.

BRANDÃO, A. A. F. **Apresentação de oficina no WESAAC 2014 - Engenharia de Software Orientada a Agente**. Enviado por e-mail pela autora (anarosabrandao@gmail.com), em 17 julho 2014.

BRATMAN, M. **Intention, plans, and practical reason**. Harvard University Press, Cambridge, MA.

BRAUBACH, L.; LAMERSDORF, W.; POKAHR, A. **Jadex**: Implementing a BDI-infrastructure for JADE agents. [S.l.]: Citeseer, 2003.

BRAZIER, F. M.; JONKER, C. M.; TREUR, J. et al. **Principles of compositional multi-agent system development**. [S.l.]: na, 1998.

BRESCIANI, P.; PERINI, A.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J. Tropos: An agent-oriented software development methodology. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.8, n.3, p.203–236, 2004.

BRIOT, J.-P.; DEMAZEAU, Y. et al. **Principes et architecture des systèmes multi-agents**. [S.l.]: Hermès Science Publications, 2001. v.217.

BUSETTA, P.; RÖNNQUIST, R.; HODGSON, A.; LUCAS, A. Jack intelligent agents-components for intelligent agents in java. **AgentLink News Letter**, [S.l.], v.2, n.1, p.2–5, 1999.

CAIRE, G.; COULIER, W.; GARIJO, F.; GOMEZ, J.; PAVÓN, J.; LEAL, F.; CHAINHO, P.; KEARNEY, P.; STARK, J.; EVANS, R. et al. Agent oriented analysis using MESSAGE/UML. In: **Agent-oriented software engineering II**. [S.l.]: Springer, 2002. p.119–135.

CAMPOS, T. F. **Ferramenta para Geração Automática de Testes Unitários a partir de Especificações Algébricas usando Alloy e SMT**. 2014. Dissertação (Mestrado em Ciência da Computação) — Faculdade de Engenharia Universidade de Porto.

COSSENTINO, M.; POTTS, C. PASSI: A process for specifying and implementing multi-agent systems using UML. **Retrieved October**, [S.l.], v.8, p.2007, 2002.

COSSENTINO, M.; SEIDITA, V.; MICILETTO, N.; RUBINO, R. Tropos: Processo e frammenti. **Rapporto Tecnico N.: RT-ICAR-PA-05-06, Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni**, [S.l.], 2005.

DELOACH, S. A. **Multiagent systems engineering**: a methodology and language for designing agent systems. [S.l.]: DTIC Document, 1999.

DELOACH, S. A.; WOOD, M. F.; SPARKMAN, C. H. Multiagent systems engineering. **International Journal of Software Engineering and Knowledge Engineering**, [S.l.], v.11, n.03, p.231–258, 2001.

DEMAZEAU, Y. From interactions to collective behaviour in agent-based systems. In: IN: PROCEEDINGS OF THE 1ST. EUROPEAN CONFERENCE ON COGNITIVE SCIENCE. SAINT-MALO, 1995. **Anais...** [S.l.: s.n.], 1995.

DENNETT, D. C. **The intentional stance**. [S.l.]: MIT press, 1989.

DESRIVIERES, J.; WIEGAND, J. Eclipse: A platform for integrating development tools. **IBM Systems Journal**, [S.l.], v.43, n.2, p.371–383, 2004.

DIGNUM, V.; DIGNUM, F. Modelling agent societies: Co-ordination frameworks and institutions. In: **Progress in artificial intelligence**. [S.l.]: Springer, 2001. p.191–204.

DUARTE, A. **Tutorial GMF - Graphical Modelling Framework**. Disponível em: http://docslide.com.br/documents/tutorial-gmf-por-alabe-duarte.html. Acesso em 19 de janeiro de 2016.

FERBER, J. **Les Systémes Multi-Agents**: Vers une Inteliigence Collective. [S.l.]: Paris, France: InterEdittions, 1995.

FININ, T.; FRITZSON, R.; MCKAY, D.; MCENTIRE, R. KQML as an agent communication language. In: INFORMATION AND KNOWLEDGE MANAGEMENT, 1994. **Proceedings...** [S.l.: s.n.], 1994. p.456–463.

FOUNDATION, E. **Eclipse documentation - Current Release**. Disponível em: http://help.eclipse.org/luna/index.jsp. Acesso em 19 de janeiro de 2016.

FOUNDATION, E. **Graphical Modeling Framework**. Disponível em: http:/wiki.eclipse.org/Graphical_Modeling_Framework. Acesso em 19 de janeiro de 2016.

FOUNDATION, E. **Graphical Editing Framework**. Disponível em: https://eclipse.org/gef/. Acesso em 19 de janeiro de 2016.

FOUNDATION, E. **Graphical Modeling Framework/ Models/ GMFGen**. Disponível em: https:/wiki.eclipse.org/Graphical_Modeling_FrameworkModelsGMFGen. Acesso em 20 de janeiro de 2016.

GARCIA, A. C. B.; SICHMAN, J. S. Agentes e Sistemas Multiagentes. In: LTDA, M. (Ed.). **Sistemas inteligentes**: fundamentos e aplicações. [S.l.]: Editora Manole Ltda, 2005. p.269–304.

GIORGINI, P. **Tropos**: basics - Agent-Oriented Software Engineering Course. Disponível em: http://www.troposproject.org/files/8-Tropos-Basics.pdf. Acesso em 15 de fevereiro de 2016.

GRONBACK, R. C. **Eclipse modeling project**: a domain-specific language (DSL) toolkit. [S.l.]: Pearson Education, 2009.

GUEDES, G. T. A. **Um metamodelo UML para a modelagem de requisitos em projetos de sistemas multiagentes**. 2012. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

GUINELLI, J. V.; PANTOJA, C. E.; CHOREN, R. Integrating a Tropos Modeling Tool with a MDA Methodology for Engineering Multi-agent Systems. **Workshop - Escola de Sistemas de Agentes, seus Ambientes e aplicações**, [S.l.], 2015.

HENDERSON-SELLERS, B.; GIORGINI, P. **Agent-oriented methodologies**. [S.l.]: IGI Global, 2005.

HOWDEN, N.; RÖNNQUIST, R.; HODGSON, A.; LUCAS, A. JACK intelligent agents-summary of an agent infrastructure. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 5., 2001. **Anais...** [S.l.: s.n.], 2001.

HÜBNER, J. F.; BOISSIER, O.; KITIO, R.; RICCI, A. Instrumenting multi-agent organisations with organisational artifacts and agents. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.20, n.3, p.369–400, 2010.

HUBNER, J. F.; SICHMAN, J. S. Aplicação de Organização de Sistemas Multiagentes em Futebol de Robôs. **São Paulo**, [S.l.], p.1–19, 2003.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: **Advances in artificial intelligence**. [S.l.]: Springer, 2002. p.118–128.

IGLESIAS, C. A.; GARIJO, M.; GONZÁLEZ, J. C. A survey of agent-oriented methodologies. In: **Intelligent Agents V**: Agents Theories, Architectures, and Languages. [S.l.]: Springer, 1999. p.317–330.

KHALLOUF, J.; WINIKOFF, M. The goal-oriented design of agent systems: a refinement of Prometheus and its evaluation. **International Journal of Agent-Oriented Software Engineering**, [S.l.], v.3, n.1, p.88–112, 2009.

KLEPPE, A. G.; WARMER, J. B.; BAST, W. **MDA explained**: the model driven architecture: practice and promise. [S.l.]: Addison-Wesley Professional, 2003.

KOLOVOS, D. S.; ROSE, L. M.; ABID, S. B.; PAIGE, R. F.; POLACK, F. A.; BOTTERWECK, G. Taming EMF and GMF using model transformation. In: **Model Driven Engineering Languages and Systems**. [S.l.]: Springer, 2010. p.211–225.

LEMAÎTRE, C.; EXCELENTE, C. B. Multi-agent organization approach. In: II IBEROAMERICAN WORKSHOP ON DAI AND MAS, 1998. **Proceedings...** [S.l.: s.n.], 1998.

LIND, J. Issues in agent-oriented software engineering. In: AGENT-ORIENTED SOFTWARE ENGINEERING, 2001. **Anais...** [S.l.: s.n.], 2001. p.45–58.

MARKUS, B. A Quick-Start Tutorial to Eclipse Plug-in Development. , [S.l.], 2010.

MELLOR, S. J. **MDA distilled**: principles of model-driven architecture. [S.l.]: Addison-Wesley Professional, 2004.

MILLER, J.; MUKERJI, J. et al. Model driven architecture (mda). **Object Management Group, Draft Specification ormsc/2001-07-01**, [S.l.], 2001.

MORANDINI, M.; PENSERINI, L.; PERINI, A. Automated mapping from goal models to self-adaptive systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 2008., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.485–486.

MYLOPOULOS, J.; KOLP, M.; CASTRO, J. UML for agent-oriented software development: The Tropos proposal. In: **UML 2001-The Unified Modeling Language. Modeling Languages, Concepts, and Tools**. [S.l.]: Springer, 2001. p.422–441.

NUNES, I.; CIRILO, E.; LUCENA, C. J. de; SUDEIKAT, J.; HAHN, C.; GOMEZ-SANZ, J. J. A survey on the implementation of agent oriented specifications. In: **Agent-Oriented Software Engineering X**. [S.l.]: Springer, 2011. p.169–179.

NUNES, I. O. de. **Implementação do Modelo e da Arquitetura BDI**. [S.l.]: Pontifícia Universidade Católica do Rio de Janeiro, 2007.

OMICINI, A.; ZAMBONELLI, F. Coordination for Internet application development. **Autonomous Agents and Multi-agent systems**, [S.l.], v.2, n.3, p.251–269, 1999.

PADGHAM, L.; WINIKOFF, M. Prometheus: A Methodology for Developing Intelligent Agents. **John Wiley & Sons**, [S.l.], 2002.

PADGHAM, L.; WINIKOFF, M. Developing Intelligent Agent Systems: A Practical Guide. **John Wiley & Sons**, [S.l.], 2004.

PADGHAM, L.; WINIKOFF, M. **Developing intelligent agent systems**: A practical guide. [S.l.]: John Wiley & Sons, 2005. v.13.

PANTOJA, C. E.; CHOREN, R. A MDA Approach for Agent-oriented Development using FAML. In: ICEIS (2), 2012. **Anais...** [S.l.: s.n.], 2012. p.415–420.

PANTOJA, C. E.; CHOREN, R. A MDA Methodology to Support Multi-Agent System Development. In: ICAART (1), 2013. **Anais...** [S.l.: s.n.], 2013. p.393–396.

PFLEEGER, S. L. Engenharia de software: teoria e prática. **2ª Edição, Prentice Hall**, [S.l.], 2004.

PRESSMAN, R. S. **Engenharia de software**. [S.l.]: McGraw Hill Brasil, 2011.

RAO, A. S. AgentSpeak (L): BDI agents speak out in a logical computable language. In: **Agents Breaking Away**. [S.l.]: Springer, 1996. p.42–55.

RICCI, A.; PIUNTI, M.; VIROLI, M. Environment programming in multi-agent systems: an artifact-based perspective. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.23, n.2, p.158–192, 2011.

RICCI, A.; VIROLI, M.; OMICINI, A. The A&A programming model and technology for developing agent environments in MAS. In: **Programming multi-agent systems**. [S.l.]: Springer, 2008. p.89–106.

ROUSE, M. **XMI (XML Metadata Interchange) definition**. Disponível em: http://searchsoa.techtarget.com/definition/XMI . Acesso em 26 janeiro de 2016.

RUBE, R. I. **Introducción a la ingeneriería del software dirigida por modelos**. Disponível em: https://ocw.uca.es/pluginfile.php/2487/mod_resource/content/0/T1Acesso em 20 de janeiro de 2016.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence**: A Modern Approach. [S.l.]: Prentice Hall Press, 2009. v.3.

SCIENZA DELL'INFORMAZIONE, D. di Ingegneria e. **The Tropos Methodology**. Disponível em: http://www.troposproject.org/, acesso em 15 de setembro de 2015.

SICHMAN, J. S. **Raciocínio social e organizacional em sistemas multiagentes**: avanços e perspectivas. 2003. Tese (Doutorado em Ciência da Computação) — Universidade de São Paulo.

SOLEY, R. et al. Model driven architecture. **OMG white paper**, [S.l.], v.308, n.308, p.5, 2000.

STEINBERG, D.; BUDINSKY, F.; MERKS, E.; PATERNOSTRO, M. **EMF**: eclipse modeling framework. [S.l.]: Pearson Education, 2008.

THANGARAJAH, J.; PADGHAM, L.; WINIKOFF, M. Prometheus design tool. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.127–128.

UEZ, D. M. **Método para o desenvolvimento de software orientado a agentes considerando o ambiente e a organização**. 2013. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina.

UEZ, D. M. **Descrição do Método Prometheus AEOlus.** Disponível em: http://www.uez.com.br/aeolus/docs/aeolus_11112014.pdf. Acesso em 23 de setembro de 2015.

UEZ, D. M.; HÜBNER, J. F. Environments and organizations in multi-agent systems: From modelling to code. In: **Engineering Multi-Agent Systems**. [S.l.]: Springer, 2014. p.181–203.

VAN DEURSEN, A.; KLINT, P. Domain-specific language design requires feature descriptions. **CIT. Journal of computing and information technology**, [S.l.], v.10, n.1, p.1–17, 2002.

WEISS, G. **Multiagent systems**: a modern approach to distributed artificial intelligence. [S.l.]: MIT press, 1999.

WEYNS, D.; PARUNAK, H. V. D.; SHEHORY, O. The future of software engineering and multi-agent systems. **International Journal of Agent-Oriented Software Engineering**, [S.l.], v.3, n.4, 2009.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. **The knowledge engineering review**, [S.l.], v.10, n.02, p.115–152, 1995.

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The Gaia methodology for agent-oriented analysis and design. **Autonomous Agents and multi-agent systems**, [S.l.], v.3, n.3, p.285–312, 2000.

ZAMBONELLI, F.; OMICINI, A. Challenges and research directions in agent-oriented software engineering. **Autonomous agents and multi-agent systems**, [S.l.], v.9, n.3, p.253–283, 2004.

ZANLORENCI, E. P.; BURNETT, R. C. A Abordagem de Engenharia de Requisitos em Software Legado. In: WER, 2003. **Anais...** [S.l.: s.n.], 2003. p.270–284.

ZISMAN, A. An overview of XML. **Computing & Control Engineering Journal**, [S.l.], v.11, n.4, p.165–167, 2000.