PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

EM BUSCA DE UM META-MODELO PARA A UNIFICAÇÃO DE DIFERENTES ABORDAGENS DE REPRESENTAÇÃO DE AGENTES DE SOFTWARE E PARA A GERAÇÃO DE CÓDIGO EM PLATAFORMAS DE DESENVOLVIMENTO DE SISTEMAS MULTIAGENTES

IVAN LUIZ PEDROSO PIRES

Dissertação apresentada com requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

P667e Pires, Ivan Luiz Pedroso

Em busca de um meta-modelo para a unificação de diferentes abordagens de representação de agentes de software e para geração de código em plataformas de desenvolvimento de sistemas multiagentes / Ivan Luiz Pedroso Pires. – Porto Alegre, 2009.

191 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Em Busca de um Meta-Modelo para a Unificação de Diferentes Abordagens de Representação de Agentes de Software e para a Geração de Código em Plataformas de Desenvolvimento de Sistemas Multiagentes", apresentada por Ivan Luiz Pedroso Pires, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sisitemas de Informação, aprovada em 21/12/09 pela Comissão Examinadora:

Prof. Dr. Marcelo Blois Ribeiro – Orientador — PPGCC/PUCRS

Prof. Dr. Ricardo Melo Bastos – — PPGCC/PUCRS

Prof. Dr. Rafael Heitor Bordini — UFRGS

Homologada em..18../..05../..10...., conforme Ata No. 08/10.. pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Dedico este trabalho à minha querida família, em especial à minha esposa Laine, meu pai Jorge e minha mãe Margarethe.

AGRADECIMENTOS

Agradeço à minha querida e amada esposa Elaine Alves da Rocha por toda dedicação, renúncia, apoio, força, fé e perseverança que tivestes por nós dois, por todas as noites sem dormir me acompanhando, me levantando com sua sempre auto-estima e pelos choques de ânimo quando não havia mais força pra continuar e broncas pra quando eu perdia o foco. Por toda leitura que fizestes comigo, todas as escritas, correções, todos os erros de códigos que encontrava, enfim por tudo. Muito obrigado Laine! Este trabalho é mais seu do que meu, e, pra mim, você já é Mestra há muito tempo.

Aos meus pais Jorge Luiz de Souza Pires e Margarethe Pedroso Lino Pires que, mesmo longe, sempre estiveram tão perto de mim com tanto carinho, força e conselhos. Eu me construí através de vossos exemplos, e sempre caminho tendo vocês como modelo de ser humano. Agradeço também à minha irmã Joizeanne Pedroso Pires pelo carinho e sempre estar presente nas horas que necessito.

A minha nova família Colidense, Cleusa Morandi, Dino, Elisângela e Eliane. Obrigado pela compreensão e incentivo.

Ao professor Dr. Marcelo Blois Ribeiro, por aceitar ser meu orientador neste trabalho, por compreender meus defeitos e reconhecer o meu esforço, por auxiliar o desenvolvimento do meu aprendizado e me mostrar o caminho da pesquisa científica.

Aos professores Dr. Ricardo Melo Bastos e Dr. Rafael Bordini por terem aceitado avaliar este trabalho.

Aos colegas do Minter e da Unemat/Colíder, em especial Benevid, André, Tales, Maicon, Toni e Luciano (tchê) que estavam presentes em toda esta caminhada. Ao departamento de Computação de Colíder, pelo apoio e compreensão à minha ausência.

Aos colegas do CDPE, em especial ao Maurício Escobar pela tamanha presteza em me ajudar nos momentos de programação.

À Universidade do Estado do Mato Grosso (Unemat) e Fundação de Amparo a Pesquisas do Mato Grosso (FAPEMAT) por financiar meus estudos e me possibilitar o título de Mestre. A Unemat sempre esteve presente em minha vida em todas as áreas, e me considero um Filho desta Instituição.

Meu Muito Obrigado.

EM BUSCA DE UM META-MODELO PARA A UNIFICAÇÃO DE DIFERENTES ABORDAGENS DE REPRESENTAÇÃO DE AGENTES DE SOFTWARE E PARA A GERAÇÃO DE CÓDIGO EM PLATAFORMAS DE DESENVOLVIMENTO DE SISTEMAS MULTIAGENTES

RESUMO

Os Sistemas Multiagentes têm apresentado grande crescimento na área de desenvolvimento de software como um paradigma promissor para enfrentar a complexidade dos cenários atuais de tecnologia da informação. Muitas abordagens surgem no intuito de consolidar formas e meios de desenvolver um SMA, que podem ser classificadas como Metodologias, Linguagens de Modelagem e Plataformas de Implementação. Além disso, alguns trabalhos são propostos como tentativa de unificar os conceitos envolvidos e as notações que simbolizam estes conceitos, havendo uma gama enorme de simbologias divergentes e muitos conceitos em comuns entre estas abordagens. Este trabalho apresenta comparações entre estes conceitos e notações visuais no intuito de encontrar um meio de mapear estas abordagens, mostrando suas divergências e convergências. Como um primeiro passo no sentido da unificação, esse trabalho estende o Meta-modelo de Representação Interna de Agentes para permitir o mapeamento completo da metodologia Tropos. Assim, demonstra-se como um meta-modelo pode prover a interoperabilidade entre diferentes abordagens desvinculando a criação de um SMA das exigências de específicas metodologias ou linguagens de modelagem. Também é gerado um esqueleto de código fonte a partir deste meta-modelo para o framework SemantiCore através de um protótipo, sendo possível estendê-lo para que suporte a geração de código-fonte para qualquer plataforma de implementação. Este processo de mapeamento e geração de código é demonstrado através da aplicação de um exemplo presente da literatura da área.

Palavras Chave: Agentes de software, Sistema Multiagentes, Mapeamento de Conceitos de SMA, Geração de Código de SMA.

TOWARDS OF A META-MODEL FOR THE UNIFICATION OF DIFFERENT APPROACHES TO REPRESENT SOFTWARE AGENTS AND FOR CODE GENERATION IN MULTI-AGENMT SYSTEMS DEVELOPMENT PLATFORMS.

ABSTRACT

Multi-agent systems have shown great growth in the area of software development as a promising paradigm to deal with the complexity of current scenarios of information technology. Many approaches arise in order to consolidate ways and means to develop an MAS, which can be classified as methodologies, modeling languages and implementation platforms. In addition, some studies are offered as an attempt to unify the concepts involved and the notations that represent these concepts, with an enormous range of different symbologies, and many concepts in common between these approaches. This paper presents comparisons between these concepts and notations visually in order to find a way to map these approaches, showing their differences and similarities. As a first step towards unification, this work extends the Meta-Model Representation of Internal Agents to allow complete mapping of the Tropos methodology. Thus, we showed how a meta-model can provide interoperability between different approaches severing the creation of a SMA of the requirements of specific methodologies or modeling languages. It's also generated a skeleton source code from this meta-model for SemantiCore framework through a prototype, it is possible to extend it to support the generation of source code for any platform implementation. This process of mapping and code generation is demonstrated by applying an example of the literature.

Keywords: Software agents, Multiagent System, Mapping Concepts MAS, Code Generation for MAS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AOS	*Agent Oriented Sofware*
AUML	*Agent-based Unified Modeling Language*
BDI	*Belief-Desire-Intention*
IDE	*Integrated Development Environment*
IDK	*Ingenias Development Kit*
JAL	*Jack Agent Language*
KQML	*Knowledge Query Manipulation Language*
MAS-ML	*Multi-Agent System Modeling Language*
MMM	*Multi-Agent Meta-Models*
MMI	*Multi-Agent System Meta-models Interchange*
MMI4E	*Multi-Agent System Meta-models Interchange For Eclipse*
MRIA	Meta-modelo de Representação Interna de um Agente
OCL	*Object Constraint Language*
OWL	*Web Ontology Language*
PDT	*Prometheus Projeto Tool*
RDF	*Resource Description Framework*
SMA	Sistema Multiagentes
TAO	*Taming Agents and Objects*
TAOM4E	*Tool for Agent Oriented visual Modeling for the Eclipse platform*
UAVs	*Unmanned Aerial Vehicles*
UML	*Unified Modeling Language*
UP	*Unified Process*
USE	*UML-based Specification Environment*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Bergenti, Gleizes e Zambonelli [BER04], Agentes e Sistemas Multiagentes têm emergido como uma poderosa tecnologia para enfrentar a complexidade de uma variedade de cenários atuais de tecnologia da informação. Nos últimos anos, tem crescido o volume de pesquisas relacionadas à identificação e definição de modelos, ferramentas, e técnicas que suportem o desenvolvimento de SMAs. Estas pesquisas de engenharia de software orientada a agentes resultam em novas abordagens de modelagem, técnicas, metodologias e ferramentas, baseados em agentes. Estas abordagens podem ser classificadas como:

- Metodologias de desenvolvimento: que nos traz propostas de como planejar, organizar e sistematizar um SMA. Por exemplo, Prometheus [PAD02] [PAD02a], Tropos [BRE04] [SUS05] e Ingenias [PAV03] [GOM02].
- Linguagens de Modelagem de SMA: As soluções de meta-modelos apresentam propostas de como deve ser estruturado o agente e os SMAs quanto a suas entidades, relacionamentos e entre estas restrições. Em muitas soluções, este meta-modelo traz consigo uma notação visual que representa estas entidades e relacionamentos, tais como o ANote [CHO04], o MAS-ML [SIL04a] [SIL04] [SIL08] e o Message [MES08] [CAI01].
- Plataformas de implementação: são, geralmente, frameworks que possibilitam a implementação do SMA em uma linguagem de programação, utilizando recursos relacionados. Algumas plataformas suportam características de específicas metodologias, como a plataforma Jack [HOW01] que suporta a metodologia Prometheus [PAD02].

Como muitas abordagens incorporam características dos três tipos apresentados, alguns pesquisadores apresentam propostas para a unificação destas diversas soluções, como a Notação Unificada para representação de SMAs [PAD08], e o Meta-Modelo Unificado para modelagem de SMAs [BER04] [BER05], apresentados no capítulo 3. Embora estas propostas abordem a unificação de meta-modelos, não existe uma solução integrada que incorpore o mapeamento dos meta-modelos de diferentes metodologias e que seja capaz de gerar o esqueleto de código para uma plataforma de implementação, independente da metodologia ou da linguagem de modelagem utilizada.

Este trabalho é um passo preliminar para a busca de integração de soluções apresentado como um esforço inicial para ser referência na incorporação de outras

abordagens pesquisadas e também presentes na literatura. Como o primeiro passo este trabalho apresenta o *MAS Meta-model Interchange* para integrar a metodologia Tropos a partir da extensão do MRIA, a fim de gerar código de software para plataforma SemantiCore.

Nas subseções a seguir, serão apresentados a questão de pesquisa, o objetivo geral, os objetivos específicos e a metodologia e organização da dissertação.

1.1 Questão de Pesquisa

De acordo com [BER04], a integração de soluções que abordam desde a modelagem até a implementação, contribuirá para a construção de SMAs aplicados a indústria em larga-escala. Neste sentido, emerge a questão de pesquisa deste estudo: "**É possível gerar uma abordagem que permita a integração entre diferentes metodologias e linguagens de modelagem de SMAs e os diversos tipos de plataformas de implementação?**".

1.2 Objetivo Geral

O objetivo geral deste trabalho é propor uma abordagem que permita a integração de soluções para o desenvolvimento de SMAs, permitindo a geração automática de esqueletos de código em plataforma de implementação. Além disso, este trabalho possui também o objetivo de contribuir para as metodologias e linguagens de modelagem na identificação de possíveis lacunas conceituais, e como escopo da pesquisa este trabalho utiliza a metodologia Tropos e a plataforma SemantiCore.

1.3 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

- Aprofundar o estudo teórico sobre as metodologias, linguagens de modelagem e plataformas voltadas ao desenvolvimento de SMAs.
- Identificar trabalhos correlatos que possam contribuir para esta pesquisa.
- Verificar a possibilidade de unificação de conceitos das várias metodologias.
- Comparar conceitos das entidades e relacionamentos abordados em metodologias e linguagens de modelagem de SMAs, bem como símbolos utilizados nas

notações visuais para representação dos elementos que compõem o meta-modelo de cada abordagem.

- Propor uma extensão do meta-modelo de representação interna de agentes proposto em SANTOS (2008) que cubra os conceitos e os relacionamentos da metodologia Tropos.
- Verificar a possibilidade de extensão do meta-modelo proposto para suporte às outras abordagens.
- Desenvolver um protótipo que automatize o processo de mapeamento de soluções para o meta-modelo proposto, e posteriormente gere um esqueleto de código da aplicação para o *framework* SemantiCore.
- Apresentar um exemplo de uso publicado na literatura que ilustre o potencial desta abordagem.

1.4 Metodologia e Organização da Dissertação

Esta pesquisa estrutura-se em duas fases. Inicialmente, será feita a revisão bibliográfica, compilação e comparação de diferentes metodologias e linguagens de modelagem de SMA, presentes na literatura. A partir desta análise comparativa, um conjunto de conceitos será definido como base para o desenvolvimento de um meta-modelo que ofereça interoperabilidade entre os conceitos comparados e compilados.

Na segunda etapa, será feita a compatibilização dos conceitos apresentados na linguagem com os conceitos existentes no MRIA. A partir deste mapeamento, será feita uma extensão deste meta-modelo para a construção do MMI. A cada nova metodologia ou linguagem de modelagem suportada será ampliado a semântica do MMI e este será refatorado. Após, será estendido o tradutor do meta-modelo abordado em [SAN08] para traduzir modelos das abordagens suportadas para o MMI, e este para a estrutura de código do ambiente de desenvolvimento. Por fim, será feito um exemplo de uso para a ilustração da proposta.

Desta forma, este trabalho está dividido em sete partes: embasamento teórico, trabalhos relacionados, estudo comparativo de meta-modelos e notações visuais, a extensão do meta-modelo de representação interna de agentes, implementação do protótipo estendido e considerações finais juntamente com trabalhos futuros.

O Capítulo 2 apresenta a base teórica desta pesquisa, com conceitos e definições de agentes e sistemas multiagentes a partir dos principais autores que compõem a

literatura da área. São abordadas as metodologias MASUP [BAS05], Tropos [BRE04], Prometheus [PAD02], Ingenias [GOM02] e MESSAGE [MES08], assim como os meta-modelos e linguagens de modelagem ANote e MAS-ML voltadas ao desenvolvimento de sistemas multiagentes, e as plataformas de implementação de SMAs SemantiCore, Jason e Jack.

O Capítulo 3 aborda dois trabalhos relacionados a esta pesquisa, uma proposta de unificação de meta-modelos e uma proposta de unificação de notações visuais de diferentes metodologias e meta-modelos.

No Capítulo 4 está descrito um estudo comparativo entre meta-modelos e notações visuais. Durante a pesquisa são identificados os conceitos e relacionamentos das metodologias e linguagens de modelagem abordadas, bem como a notação visual que os representam.

No Capítulo 5 é apresentado o MMI, uma proposta de extensão de um meta-modelo para que suporte os conceitos da estrutura interna de um agente e possa receber o mapeamento das metodologias ou linguagens de modelagem pesquisadas. Para demonstração da extensão deste meta-modelo, é utilizada a metodologia de desenvolvimento de SMAs Tropos, com mapeamento de suas entidades e relacionamentos para este meta-modelo.

No Capítulo 6 é apresentado um protótipo estendido que automatiza o mapeamento dos modelos Tropos gerados pelo software TAOM4E [PER04] para os conceitos do meta-modelo estendido proposto, e a posterior geração de código para o framework SemantiCore.

No Capítulo 7, serão apresentadas as conclusões, assim como os possíveis trabalhos futuros. Por fim serão descritas as referências bibliográficas utilizadas nesse trabalho.

2 BASE TEÓRICA

Este capítulo apresenta inicialmente os conceitos de diferentes autores sobre agentes de software e sistemas multiagentes (SMAs). Após, apresenta um levantamento bibliográfico para as metodologias de desenvolvimento de SMAs MASUP, Tropos, Prometheus, Ingenias e MESSAGE, e as linguagens de modelagens ANote e MAS-ML, no intuito de verificar o quanto estas metodologias diferem-se e a existência de um conjunto comum, sendo realizada uma comparação entre estas abordagens no capítulo 4. Esta pesquisa foca na metodologia Tropos, abordando os conceitos e relacionamentos que formam seus meta-modelos, utilizando-o para demonstrar a interoperabilidade entre diferentes abordagens. Por fim, apresenta as plataformas de implementação SemantiCore, Jack e Jason, focando no *framework* SemantiCore por utilizá-lo como base na demonstração de como gerar código de SMA a partir de um meta-modelo interoperável entre metodologias ou linguagens de modelagem.

2.1 Agentes de Software

Agentes de software possuem diversas definições, sendo algumas mais voltadas a Inteligência Artificial e outras mais utilizadas na área de Engenharia de Software. Segundo Russel e Novig [RUS04] um agente é tudo que pode ser considerado capaz de perceber e agir no ambiente através de sensores e atuadores, conforme ilustrado na figura 1. O termo percepção faz referência às entradas do agente em qualquer momento, e sua sequência é a história completa de tudo que o agente já percebeu. O comportamento do agente é descrito pela função de agente, a qual mapeia qualquer seqüência de percepções específicas para uma ação. O comportamento de um agente é implementado pelas ações e percepções dos agentes, e dividem-se em quatro tipos básicos: agentes reativos simples, agentes reativos baseados em modelo, agentes baseados em objetivos e agentes baseados na utilidade.

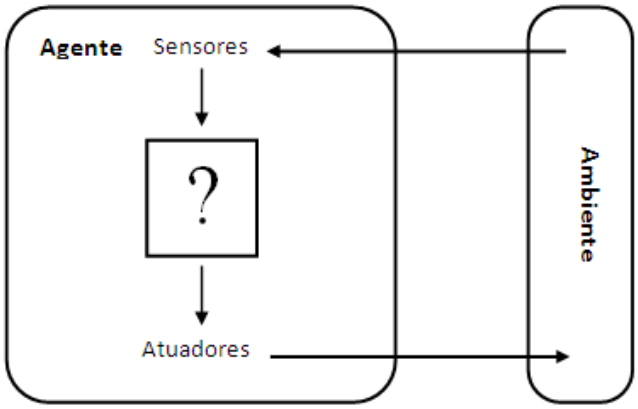


Figura 1 – Interação de agente com o ambiente onde está situado [RUS04].

Agentes reativos simples selecionam ações com base na percepção atual, ignorando o histórico de percepções; os agentes reativos baseados em modelos utilizam um modelo de mundo, no qual o agente deve manter um tipo de estado interno que dependa do histórico de percepções e das informações sobre o modo como o mundo evolui independentemente do agente, e sobre como as ações do próprio agente afetam o mundo; agentes baseados em objetivos possuem uma descrição do estado atual, e informações sobre os resultados de possíveis ações para alcance dos seus objetivos; em um agente baseado na utilidade, a função de utilidade mapeia um estado ou uma sequência de estados que descrevem o grau de satisfação associado ao sucesso no alcance dos objetivos do agente.

Wooldridge [WOO02] caracteriza os agentes conforme sua arquitetura interna, abrangendo os agentes *BDI* (*Belief, Desire and Intention*) e agentes reativos. De acordo com Müller [MUL96], um agente *BDI* é composto por desejos, crenças e intenções, e pode também conter planos e objetivos. Os planos são formados por um conjunto de ações e pode ser executado por um agente no alcance de seus objetivos. Os objetivos são classificados como subconjunto dos desejos, os quais, segundo Odell [ODE00], definem os estados futuros que o agente deve atingir. A crença de um agente, segundo Müller, expressa suas expectativas sobre o estado atual do mundo e a probabilidade de um curso de ação atingir determinados efeitos, enquanto uma intenção é um compromisso para executar um plano. A arquitetura de agentes BDI baseia-se na manipulação de estruturas de dados que representam as crenças, desejos e intenções dos agentes, para a tomada de decisão. Esta arquitetura tem suas bases na tradição filosófica de entender o raciocínio prático, ou seja, o processo de decidir, momento a momento, qual ação deve ser realizada no amparo de seus objetivos.

A arquitetura reativa apóia a implementação de um mapeamento direto da tomada de decisão, seguindo da situação para a ação. Um conjunto de comportamentos para a conclusão das tarefas faz-se necessário, e vários destes comportamentos podem trabalhar simultaneamente.

Nwana [NWA96] classifica agentes de software quanto à sua mobilidade, pela presença ou não de um raciocínio simbólico (agente deliberativo ou reativo), ou pela presença de atributos considerados primários: autonomia, cooperação e aprendizado. A mobilidade em um agente é determinada pela habilidade em se locomover por algum tipo de rede. Os agentes deliberativos são capazes de se engajar em uma negociação por um pensamento, enquanto os reativos possuem seu comportamento dependente de estímulos gerados pelo ambiente onde está inserido. Os agentes autônomos podem executar ações por conta própria para satisfazer seus objetivos. A cooperação permite que os agentes interajam com outros agentes e possivelmente com seres humanos utilizando uma linguagem de comunicação. Por fim, o aprendizado permite ao agente ser "inteligente", devendo aprender com as reações e interações no ambiente externo.

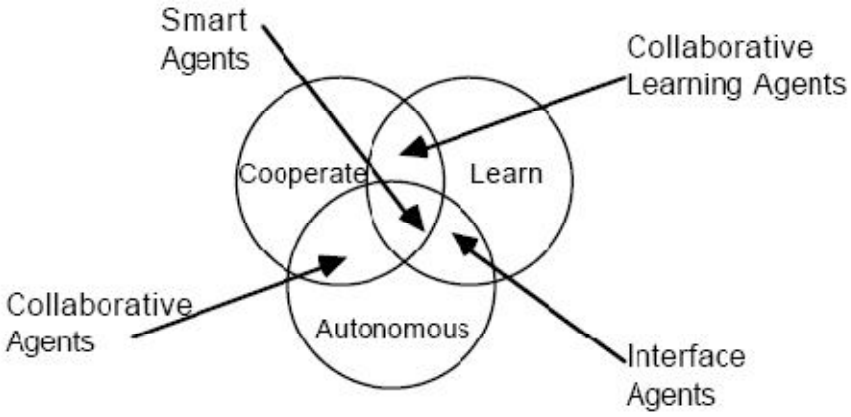


Figura 2 - Tipologia de um agente [NWA96].

Quando um agente apresenta mais de uma das características citadas, é considerado híbrido. Os limites desta classificação não devem ser interpretados como linhas bem definidas, pois o fato de agentes cooperativos possuírem uma ênfase maior em autonomia e cooperação que agentes com capacidade de aprendizado não implica que os agentes cooperativos não possam desenvolver características de aprendizado, conforme ilustrado na figura 2.

Os agentes podem atuar isoladamente no alcance de seu objetivo ou em conjunto com outros agentes formando um sistema multiagentes.

2.2 Sistemas Multiagentes

Os Sistemas Multiagentes (SMAs) são compostos por múltiplos agentes que interagem entre si para atingir um objetivo comum. Os agentes dentro de um sistema multiagentes podem apresentar diferentes atributos, de acordo com o ambiente em que estão inseridos.

Segundo Wooldridge [WOO02], um agente está situado em um ambiente que constitui o contexto em que todas as interações entre os agentes ocorrem com dispersão do controle, dos dados e do conhecimento pela comunidade de agentes. Para Juchen [JUC01], em um projeto de um SMA, é importante considerar o tipo do ambiente no qual os agentes estarão situados, pois a situacionalidade determina a maneira de atuar e de perceber as alterações no ambiente e o tipo de representação de ambiente sobre a qual cada um dos agentes atuará.

Para que haja o processo colaborativo em sistemas multiagentes diferentes, ou até entre agentes do mesmo SMA, é essencial que haja uma forma de comunicação comum, disciplinada e inteligível entre todos agentes para que os objetivos sejam alcançados de forma eficiente. Existem diversas maneiras para agentes se comunicarem entre si em sistemas multiagentes, de acordo com Baker [BAK97]: comunicação direta, federada (também chamada de comunicação assistida), em difusão de mensagens (*broadcast*) e através de *blackboard* com uso de um repositório comum.

Na comunicação direta, cada agente comunica-se diretamente com qualquer outro agente sem intermediários; na federada é utilizado um agente chamado de facilitador que atua como coordenador intermediando a comunicação com outros SMAs; a comunicação por difusão é utilizada em situações onde a mensagem deve ser enviada para todos os agentes do ambiente, ou quando o agente remetente não conhece o agente destinatário e então envia a mensagem a todos os agentes; e por quadro-negro, baseia-se em um modelo de memória compartilhada, ou seja, um repositório onde todos os agentes possuem acesso para receberem e contribuírem através de mensagens a outros agentes para obterem informações sobre o ambiente.

Além das formas de comunicação dos agentes, é necessária uma linguagem que estabeleça um protocolo para comunicação inteligível por todos os agentes que compõem um SMA e comum para o ambiente onde o agente se situa. De acordo com Weiss [WEI01], um protocolo de comunicação especifica os tipos de troca de mensagem entre agentes: propor um curso de ação, aceitar um curso de ação, rejeitar um curso de ação,

cancelar um curso de ação, discordar de um curso de ação proposto e contrapropor um curso de ação. As mais usadas são a KQML [FIN94] e FIPA-ACL [FIP08].

Segundo [BLO07], muitos pesquisadores buscam o desenvolvimento de metodologias para a construção de sistemas multiagentes, com diferentes iniciativas, embora não exista um consenso quanto à melhor metodologia para o desenvolvimento de sistemas multiagentes. Algumas destas abordagens trazem consigo propostas de linguagem de modelagem, com meta-modelo e notação própria que permite a modelagem em um ambiente de desenvolvimento integrado com geração de código e execução da aplicação. Entretanto algumas abordagens limitam-se à metodologia ou plataforma na qual foi projetada.

Estas abordagens podem ser classificadas como: metodologias de desenvolvimento abrangendo como planejar, organizar e sistematizar um SMAs; meta-modelos que modelam o sistema através de entidades e relacionamentos, e podem trazer consigo uma notação visual que os represente; e plataformas de implementação, que são geralmente *framework*s, para a implantação dos SMAs em uma linguagem de programação.

2.3 Visão geral sobre metodologias e linguagens de modelagem de SMAs

Este trabalho concentra-se no estudo das metodologias e linguagens de modelagem de SMAs do ponto de vista de seus conceitos, relacionamentos e restrições. Alguns conceitos presentes nos meta-modelos estudados são representados visualmente segundo notações próprias de cada abordagem.

Uma metodologia de desenvolvimento de SMA nos traz propostas de como planejar, organizar e sistematizar este sistema, ou seja, trata do processo de desenvolvimento do sistema, guiando o desenvolvedor passo a passo. Uma metodologia pode abranger, além do processo, um meta-modelo que guie a sua aplicação.

A linguagem de modelagem é um elemento essencial em tecnologia de software que propicia a sistematização visual e a organização do sistema em modelos, os quais são passíveis de mapeamento para a implementação e a codificação. Uma linguagem de modelagem é definida por um meta-modelo que descreve os conceitos a serem utilizados através das entidades, relacionamentos e restrições entre estes conceitos. Uma linguagem de modelagem além de possibilitar uma melhor compreensão para o desenvolvedor traz junto com o meta-modelo uma notação visual que representa as entidades e relacionamentos deste meta-modelo.

Um meta-modelo é uma representação dos tipos de entidades que podem existir em um modelo, suas relações e restrições de aplicação [CHO04]. Este meta-modelo contém todos os conceitos que podem ser usados para projetar e descrever o sistema estudado e podem ser representados por uma notação visual. Uma notação visual é um sistema técnico de símbolos usados para representar elementos dentro de um sistema [BER04]. Para que uma notação visual tenha melhor aceitação deve apresentar características que a torne utilizável e consistente, sendo que alguns critérios foram propostos por Rumbaugh em [RUM96].

Em um cenário ideal, um desenvolvedor seria capaz de especificar um sistema de agentes sem considerar uma linguagem de modelagem específica para uma determinada arquitetura, na qual o sistema será implementado [BER04].

Estas especificações são encontradas em diversas metodologias para o desenvolvimento de SMAs, com uso de meta-modelo e notação visual. Entretanto, não há uma forma padronizada para modelagem de SMAs, apenas esforços e iniciativas em busca desta padronização, como um meta-modelo unificado baseado em agentes [BER05] e uma notação visual unificada [PAD08].

Esta seção apresenta uma visão geral sobre as metodologias de desenvolvimento e linguagens de modelagem de SMAs MASUP, Tropos, ANote, Prometheus, Ingenias, Message e MAS-ML.

2.3.1 MASUP

Os paradigmas tradicionais de engenharia de software são limitados para representar as características de sistemas multiagentes. Embora muitas abordagens relacionadas ao desenvolvimento de SMA existam, a maioria foca em linguagens de modelagem de SMA, havendo poucas abordagens que combinam processos conhecidos e aceitos de desenvolvimento, como o Processo Unificado, e uma linguagem expressiva em representação.

O *Multi-Agent Systems Unified Process* (MASUP) [BAS05] é uma variação do Processo Unificado (UP) [KRU04] para modelagem de sistemas orientados a agentes, com objetivo principal de identificar as aplicabilidades de soluções baseadas em agentes de software durante a modelagem. O MASUP inicia-se como o UP, mas na fase de análise e projeto deriva diferentes artefatos para modelar características específicas de agentes, conforme ilustrado na figura 3. Desta forma, o MASUP apresenta-se totalmente

compatível com o UP e, portanto, partes que não são aplicadas à agente de software podem ser modeladas usando as técnicas tradicionais do UP.

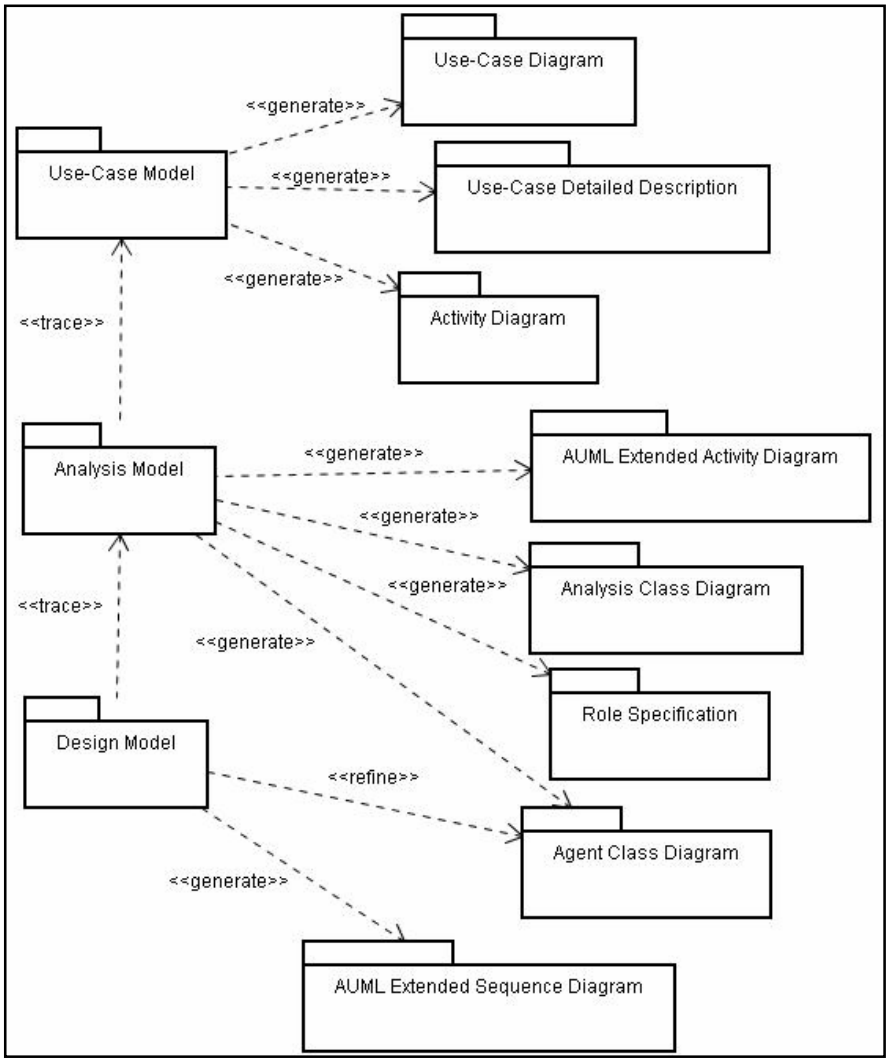


Figura 3 - Modelos do MASUP e artefatos que compõem cada modelo.

Na fase do levantamento de requisitos, o MASUP utiliza a mesma abordagem de casos de uso do Processo Unificado, de modo que o foco desta etapa está na captura de requisitos. Esta fase é responsável por especificar os papéis a serem desempenhados pelos agentes, identificar atores e ações, e detalhar os casos de usos.

Ainda na fase dos requisitos, as funcionalidades envolvidas no sistema são identificadas, e delas deriva diferentes artefatos para modelar as características específicas de cada agente. Para a modelagem destes artefatos, o MASUP utiliza os diagramas da *Agent-based Unified Modeling Language* (AUML) [ODE00].

De acordo com Blois e Santos [BLO07], na fase de análise do MASUP é feito o link entre a fase de requisitos e a fase de projeto, compreendendo as seguintes atribuições: a

revisão dos diagramas de atividades gerados no projeto, de forma a descobrir quais atividades envolvem uma tomada de decisão que necessita ser codificada diretamente no sistema e que na modelagem original é realizada por algum ator; identificação de papéis requeridos para a solução baseada em SMA nos diagramas de atividades gerados para a próxima fase; especificação dos papéis de cada agente e suas atribuições; identificação dos agentes que devem desempenhar funções específicas; e definição das relações entre os agentes que compõem a arquitetura social dos SMA. Esta fase identifica novas responsabilidades capazes de preencher os papéis requeridos para uma solução de sistemas multiagentes, considerando os seguintes aspectos: responsabilidade, informação e conhecimento a serem vinculados a um papel.

Os agentes identificados nesta fase são especificados pela Classe de Agentes, apresentados na figura 4.

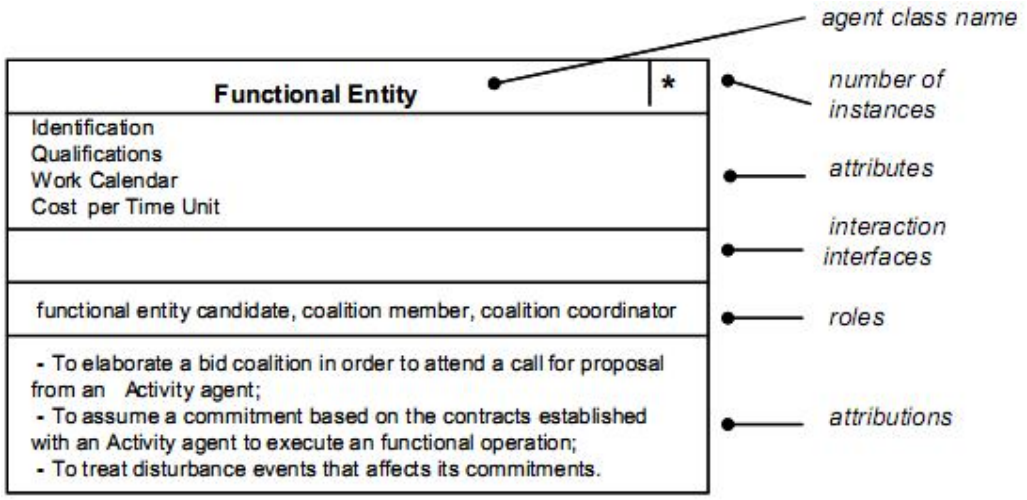


Figura 4 - Especificação de Classe de Agente [BAS05].

A fase de projeto da metodologia MASUP tem como objetivo as seguintes atividades: a especificação do cenário de interação dos agentes; a complementação da especificação da classe agente com as ações de comunicação necessárias para implementar as interações modeladas; e a identificação dos serviços de infra-estrutura envolvidos em um cenário especificado pela interação modelada.

2.3.2 Tropos

O Tropos é uma metodologia de desenvolvimento de SMAs baseada no *framework i** [YU95]. Este *framework* provê noções como atores, objetivos e dependências entre atores que são utilizadas durante todo o ciclo de desenvolvimento.

A metodologia Tropos [BRE04] suporta todas as atividades de análise e projeto no processo de desenvolvimento de software. De acordo com Bresciane e outros, o Tropos introduz cinco principais fases de desenvolvimento: fase inicial de requisitos, fase final de requisitos, projeto arquitetural, projeto detalhado e implementação.

A fase inicial de análise de requisitos consiste em identificar e analisar as partes interessadas (*stakeholders*) do domínio e seus objetivos. Os *stakeholders* são modelados como atores sociais que dependem um do outro para alcançar seus objetivos, realizar seus planos e fornecer seus recursos.

A fase final dos requisitos estende o modelo conceitual incluindo um novo ator que representa o sistema e as dependências com outros atores do ambiente. Estas dependências definem os requisitos funcionais e não funcionais do sistema.

A fase de projeto de arquitetura define a arquitetura global do sistema em termos de subsistemas (atores) interconectados através de dados e fluxos de controles (dependências). Esta fase é dividida em três passos: no primeiro passo é definida toda a arquitetura organizacional, quando novos atores são introduzidos no sistema, e apresentados em um diagrama de atores estendido; no segundo, são identificadas as capacidades necessárias para os atores completarem seus objetivos e planos. As capacidades podem ser facilmente identificadas analisando o diagrama de ator estendido, no qual cada relacionamento de dependência se tornará uma ou mais capacidades iniciadas por um evento externo, e o terceiro passo consiste em definir um conjunto de tipos de agentes e atribuir a cada tipo uma ou mais capacidades diferentes.

Na fase de detalhamento do projeto são especificados em detalhes os objetivos, metas, capacidades e comunicação dos agentes. Nesta fase, são feitas as escolhas da plataforma de desenvolvimento para a implementação do sistema. Durante esta fase, o Tropos faz a utilização do diagrama de atividades da UML para representar a capacidade e os planos, e adota um subconjunto de diagramas proposto pela AUML para especificação do protocolo de agentes.

A última fase da metodologia Tropos é responsável pela implementação do projeto detalhado. Segundo Bresciani [BRE04], para esta fase, o Tropos utiliza a plataforma *BDI* JACK [HOW01] para implementar o SMA.

Segundo Silva [SIL08a], a modelagem realizada em Tropos é bastante confusa e rebuscada, o que dificulta esta fase do processo de desenvolvimento. A fase de projeto detalhado é orientada especificamente à plataforma JACK.

Segundo Bresciane e outros [Bre04], os modelos em Tropos são instâncias de um meta-modelo conceitual que aborda os conceitos de ator, posição, agentes, papéis,

objetivo, planos e recursos e os relacionamentos de dependência, meios-fins (*means-ends*), contribuição e decomposição *AND/OR*. Estes conceitos possuem uma representação em uma notação que Tropos incorpora do *framework i**, demonstrada na figura 5 montada a partir de recortes das figuras 1 e 2 de [Bre04].

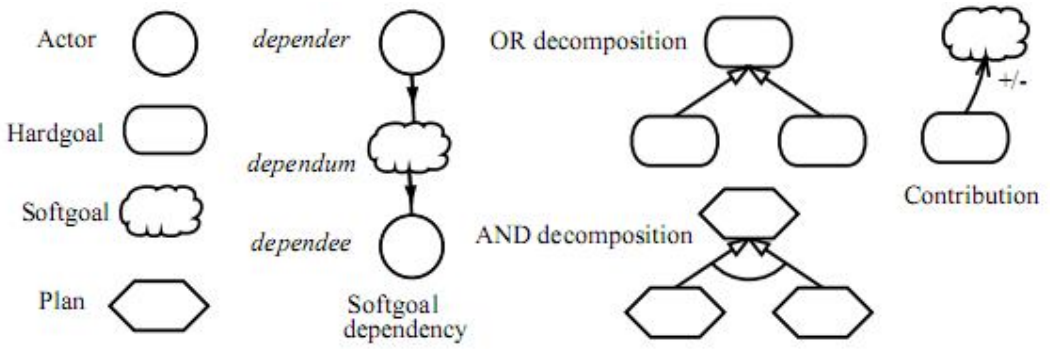


Figura 5 - Notações para os conceitos de Tropos.

O ator, em Tropos, representa um agente, um papel ou uma posição. A metodologia Tropos define conceitos de agente, papel e posição, como: o Tropos adota o conceito clássico da Inteligência Artificial para agentes, que o define por suas propriedades de autonomia, habilidade social, reatividade, pró-atividade; o papel é definido como uma caracterização abstrata do comportamento de um ator social dentro de um contexto específico; e a posição representa um conjunto de papéis, tipicamente executado por um agente. Um agente pode ocupar uma posição enquanto uma posição cobre o papel, conforme representado no meta-modelo de Tropos com enfoque no conceito de ator, ilustrado na figura 6.

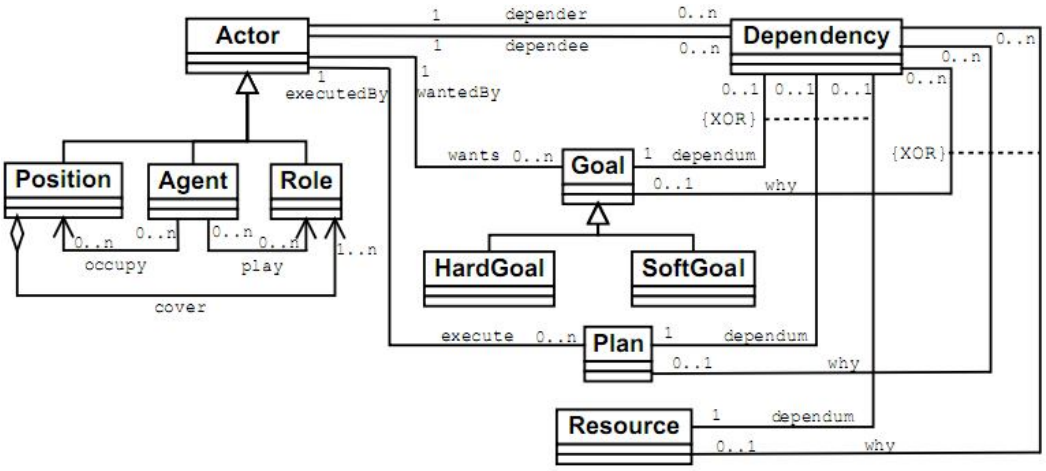


Figura 6- Conceito de ator em Tropos e relação de dependência [SUS05].

O objetivo representa interesses estratégicos do ator. O Tropos distingue objetivo em *Hardgoal*, referente ao objetivo propriamente dito, e *Softgoal*, o qual não possui uma definição clara ou critério para decisão quanto a sua satisfação, sendo tipicamente utilizado para modelar requisitos não-funcionais.

O plano representa, em um nível abstrato, o caminho para se fazer algo. A execução do plano pode ser o meio para satisfazer um objetivo. Um recurso representa uma entidade física ou informacional, enquanto que a crença representa o conhecimento de mundo do ator. Já a capacidade representa a habilidade de um ator definir, escolher e executar um plano para alcançar um objetivo, dada certas condições do mundo e na presença de um evento.

A dependência, em Tropos, indica um relacionamento entre dois atores no qual um ator depende do outro por alguma razão, seja para alcançar um objetivo, executar um plano, ou entregar um recurso. O ator que depende de outro é chamado de *depender*, e o outro ator desta relação de dependência é chamando de *dependee,* sendo objeto de dependência (seja um plano, recurso ou objetivo) chamando de *dependum*. Estes conceitos focado no ator são mostrados no meta-modelo do Tropos, conforme a figura 7.

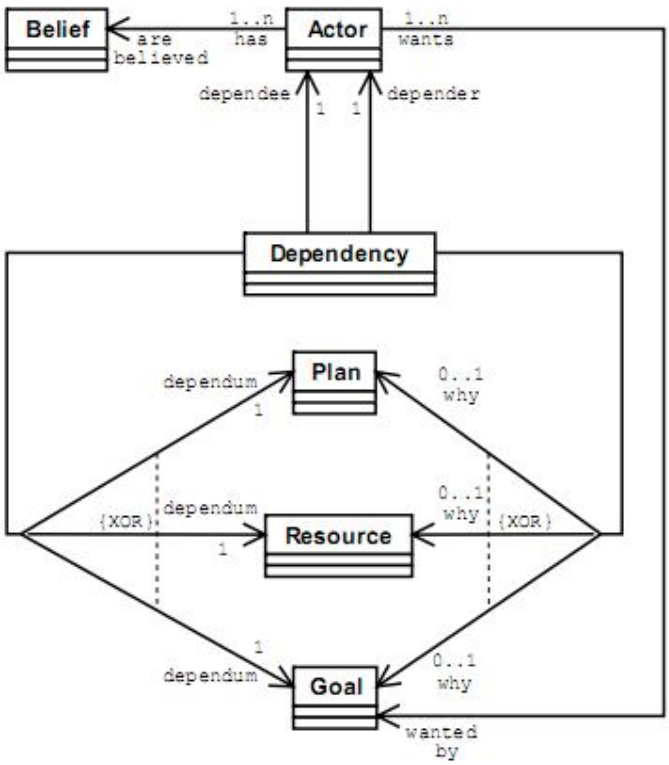


Figura 7 – Relações de dependência e crença do ator [BRE04].

A contribuição é uma relação ternária entre o ponto de vista de um ator, e dois objetivos. Esta contribuição identifica o quanto um objetivo contribui positivamente ou negativamente para outro, e é mensurada utilizando métricas qualitativas denotadas por + (contribuição positiva parcial), ++ (contribuição positiva suficiente), - (contribuição negativa parcial) e -- (contribuição negativa suficiente). Já a decomposição *AND/OR* é uma relação ternária que define a decomposição de objetivos em subobjetivos.

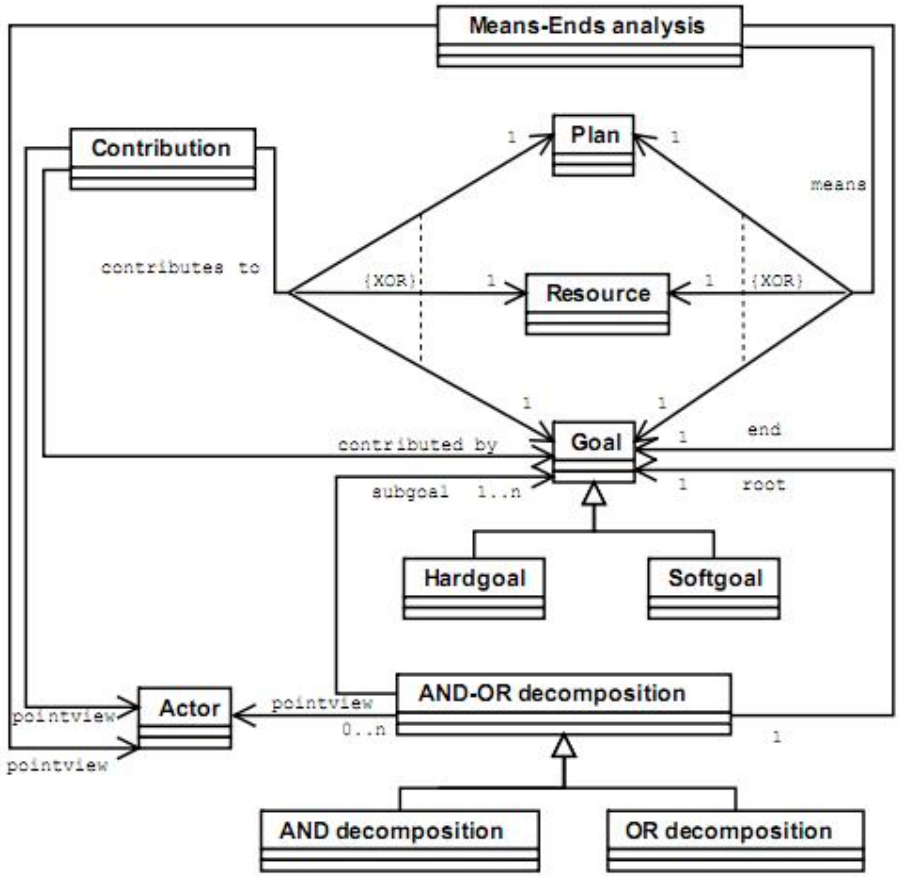


Figura 8 - Conceito de objetivo no meta-modelo do Tropos [BRE04].

No conceito de objetivo, Bresciane e outros demonstram duas especializações de objetivo para representar a entidade *Hardgoal* e *Softgoal*, e como objetivos podem ser analisados de um ponto de vista de um ator através da análise de meios-fins (*Means-ends Analysis*), da contribuição e da decomposição. Estes conceitos no meta-modelo de Tropos estão representados na figura 8.

A análise de meios-fins é uma relação ternária entre um ponto de vista de um ator, um objetivo como fim, e um plano. No conceito de plano, a análise de meios-fins e as decomposições *AND/OR* definidas sob objetivos são aplicadas também aos planos.

Assim, segundo Bresciane, a decomposição *AND/OR* é utilizada para modelar a estrutura do plano. Este conceito está representado na figura 9 [BRE04].

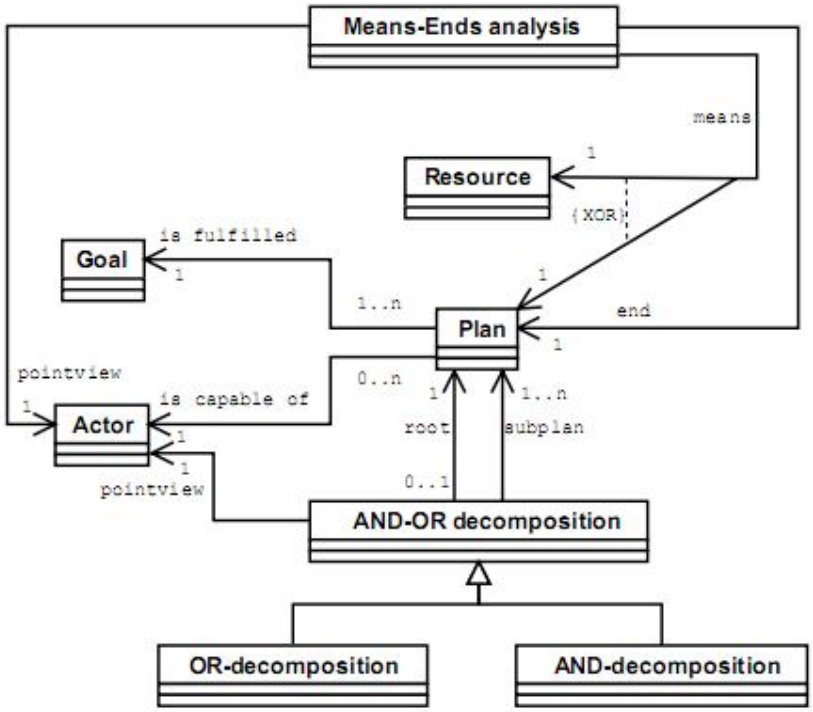


Figura 9 - Conceito de plano no meta-modelo do Tropos.

O meta-modelo de Tropos traz os conceitos de posição, agente e papel como especializações de ator, onde o ator pode ser um agente e ocupar posições na organização, bem como exercer papéis. O conceito de dependência em Tropos traz uma abordagem de interação dos agentes em um SMA, no qual há o *depender*, o *dependee* e o *dependum*. Além da dependência, Tropos traz os relacionamentos de contribuição para representar de que forma uma entidade contribui para outra no alcance de um objetivo. E por fim, o relacionamento de decomposição de Tropos permite decompor objetivos em subobjetivos e planos em subplanos, e especifica o tipo destas decomposições com o operador lógico *AND* ou *OR*. Estas entidades e relacionamentos contribuem para a proposta deste trabalho, através de adição destes conceitos e um processo de mapeamento dos modelos Tropos para a proposta, abordado no capítulo sobre o meta-modelo estendido.

2.3.3 ANote

O ANote [CHO04] oferece uma forma padrão de descrever os conceitos relacionados ao processo de modelagem de sistemas multiagentes, e fornece aos

usuários uma expressiva linguagem de modelagem visual para desenvolver e trocar modelos baseados nos conceitos de agentes. Segundo Choren e Lucena, o ANote possui um meta-modelo, que funciona como guia para a modelagem do sistema, coberto por uma notação visual que representa as entidades e relacionamentos, no intuito de facilitar ao desenvolvedor a construção do SMAs.

Os conceitos da linguagem de modelagem ANote são apresentados em [CHO04]. Os principais conceitos do ANote são: agente, organização, recurso, objetivo, cenário, ação e mensagem, representados em seu meta-modelo, como ilustrado na figura 10.

No ANote, um agente é um módulo que está habilitado a desempenhar ações, sendo o principal bloco na construção da organização do sistema. Um agente age no sistema no sentido de alcançar um objetivo, com execução de ações e interações com outros agentes durante a execução da ação. Um agente possui uma limitação de percepção ou conhecimento do ambiente do sistema.

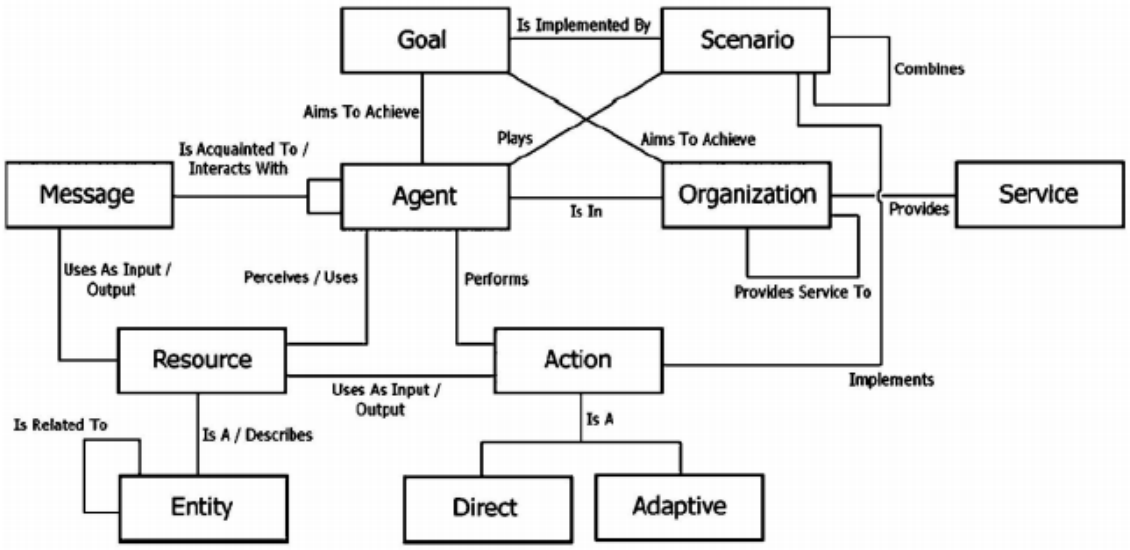


Figura 10 - Meta-modelo conceitual de ANote [CHO04].

Organização é um grupo de um ou mais agentes trabalhando juntos para prover um serviço. Trata-se de uma entidade virtual que age como um container de agentes, não havendo uma entidade computacional que a represente, embora seus serviços sejam providos e seus objetivos alcançados coletivamente pelos agentes que a compõe. Em um SMA pode haver várias organizações que provêem serviços umas às outras, conectadas por um relacionamento de *provider/customer* que define como um agente em uma organização pode *depender* ou interagir com um agente de outra organização.

Recursos são utilizados para representar entidades não-autônomas a serem manipuladas pelo agente enquanto ele desempenha uma ação, como um programa externo ou um banco de dados. Os recursos descrevem a ontologia do agente e são utilizados para modelar o ambiente do sistema, sendo modelados por conceitos orientados a objetos.

A entidade objetivo representa o objetivo do sistema, alcançado por um ou mais agentes associados com as ações referenciadas por um cenário que implementa este objetivo. Os objetivos são o ponto de partida na modelagem de SMAs e podem ser combinados em várias alternativas de subobjetivos.

Um cenário ilustra o objetivo que constitui uma situação no sistema, ou seja, ilustra uma sequência de ações de um agente com intenção de realizar um objetivo específico em um contexto. O cenário descreve o contexto no qual cada agente age, classificando-se como um contexto usual que mostra a execução das ações habituais de um agente, ou um contexto variante, o qual exige a adaptação do agente para execução de possíveis novas ações.

Ação é uma computação que resulta em alteração do estado do agente, vinculada a uma pré-condição. Quando a ação for executada e a pré-condição for válida é esperada uma transição associada, que juntas formam os planos de ação do agente. A ação é dividida em dois tipos: ação direta (*DirectAction*) e ação adaptativa (*AdaptiveAction*).

A ação direta é usualmente executada por um agente enquanto ele participa de um cenário (ou contexto) para o alcance de um objetivo, enquanto que a ação adaptativa é executada quando o contexto requer a adaptação do agente por motivo de alguma funcionalidade.

A mensagem é o envio de informação de um agente para o outro em um alto nível de um tipo de ato de fala para troca de informações. Estas mensagens são construídas através de um protocolo assíncrono que definem os padrões de interação entre os agentes.

Para a especificação do SMA, o ANote fornece um conjunto de modelo ou visões juntamente com uma representação específica de cada visão. Estas visões são agrupadas em estruturais e dinâmicas, e abrangem outras visões como a de objetivo, agentes, cenários, planos, interação, organização e ontologias.

A visão estrutural ou estática define as propriedades estáticas de um sistema multiagentes. Fazem parte da visão estrutural as modelagens de objetivos, agentes e do ambiente do sistema.

A visão de objetivos especifica os objetivos do sistema. É o primeiro passo para o processo de modelagem de sistemas multiagentes. Os objetivos complexos podem funcionalmente ser decompostos em objetivos e fluxos constituintes, fornecendo uma descrição como uma árvore hierárquica dos objetivos. No ANote, um objetivo é um nó na árvore de hierarquia de objetivos e é representado como um retângulo com os cantos arredondados, ilustrado na figura 11.

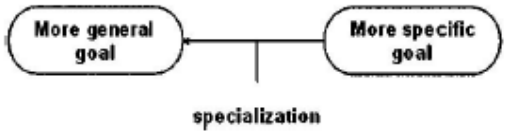


Figura 11 - Notação visual para objetivo e especialização no ANote [CHO04].

A visão de Agentes especifica a estrutura do agente, os tipos de agentes que existem em uma solução multiagentes e seus relacionamentos. Os agentes são vistos como elementos discretos da modelagem, nenhum detalhe sobre seu comportamento é fornecido.

A interação entre agentes é especificada na visão de agente usando o relacionamento de associação. Em ANote, um agente é representado como um retângulo. Uma associação é representada como uma linha, como ilustrado na figura 12.

Figura 12 - Notação visual para classes e associações de agentes [CHO04].

A visão dinâmica define as propriedades comportamentais de um sistema multiagentes. No ANote, a visão dinâmica é usada para modelar os cenários, planos e interações do sistema.

A visão de Cenário captura o comportamento do agente em contextos específicos, tais como a forma que os objetivos podem ser ou não alcançados, em quais circunstâncias o agente pode se adaptar, aprender ou ter um comportamento autônomo. Um cenário elabora caminhos para atingir os objetivos dos agentes em duas fases: pelo curso de comportamento normal, e por trajetos alternativos para cada seqüência de comportamento, sendo que cada trajeto corresponde a um cenário e possui uma descrição textual associada. Um cenário especifica o agente principal, as pré-condições, o plano usual de ação, a interação e os planos alternativos de ações.

As visões de planos especificam as ações que um agente deve executar para realizar um plano de ação descrito em um cenário. Um plano de ação é modelado de maneira que permita que o agente trace as suas ações internas, seqüenciando os eventos para atingir seu objetivo e tomar as decisões baseadas em seu conhecimento atual.

A descrição de plano de ação do agente vem dos cursos de ação (normal e alternativo) descritos na visão de cenário. Os planos de ação são representados como um diagrama de ações muito similar a um diagrama de estados (com estados e transições de ação), como ilustrado na figura 13.

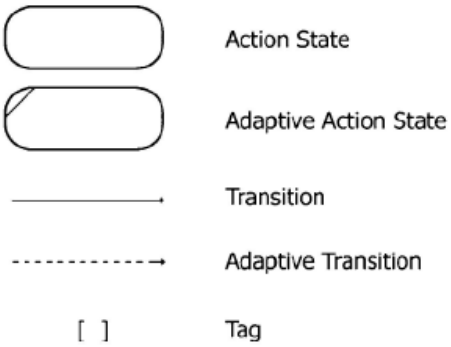


Figura 13 - Notação visual para estado de ações e transições [CHO04].

As transições adaptativas permitem que os desenvolvedores do sistema mostrem quando e sob que circunstâncias um agente deve mudar seu comportamento executando um conjunto de ações especificadas nos planos de ação alternativos de um Cenário.

A Visão de Interação do Sistema representa o conjunto de mensagens que os agentes trocam ao realizar um plano de ação. As interações são representadas como um diagrama da conversação que descreve os discursos entre os agentes, os quais permitem ao desenvolvedor mostrar o estado atual de uma conversação e fazer uma consistência entre as mensagens emitidas por um agente e as mensagens recebidas por outros, como ilustrado na figura 14.

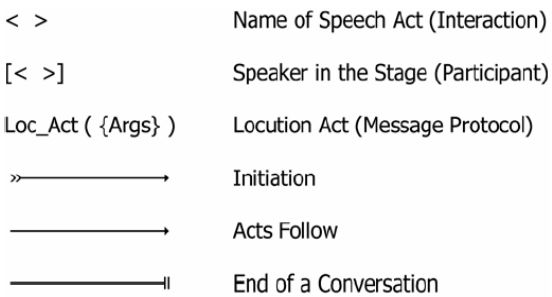


Figura 14 - Notação para mensagem - ato de fala (speech acts) [CHO04].

A visão de organização define a estrutura de um sistema multiagentes, especificando as organizações do sistema e seus relacionamentos. Esta visão modela as organizações de agentes do sistema, e possui caixas como notação visual para demonstrar o conjunto dos agentes que lhe pertencem, como ilustrado na figura 15. Uma dependência mostra que as organizações estão arranjadas em um modelo cliente-servidor, e expressam que um agente de uma organização requer o serviço de um agente em outra organização. Uma seta tracejada representa a dependência entre organizações, indicando o cliente e o servidor.

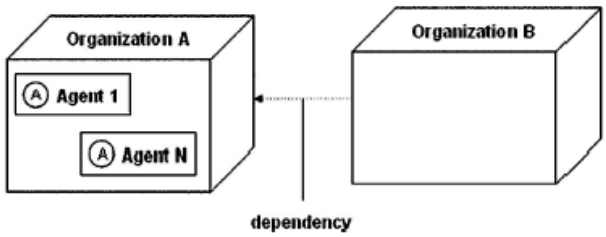


Figura 15 - Notação para organização e dependência [CHO04].

A visão de ontologia identifica o componente não-agente do sistema, e define o mundo onde o agente irá atuar. Além disso, esta visão provê uma descrição estrutural dos recursos de ambiente do agente. Nesta visão, o ANote pode especificar um ambiente de sistema multiagentes através de uma ontologia, representada por recursos.

Através destas visões, o ANote aborda os objetivos, os agentes e o alcance de seus objetivos, os componentes que constroem o ambiente, o contexto no qual o agente alcançará seus objetivos, as ações dos agentes, o diálogo entre os agentes e sua organização lógica.

2.3.4 Prometheus

O Prometheus [PAD02] [PAD02a] é uma metodologia para desenvolvimento de sistemas de agentes inteligentes, criada em colaboração com o *Agent Oriented Software (AOS)*[1]. Trata-se de uma metodologia que atua em todas as atividades requeridas no desenvolvimento de sistemas de agentes inteligentes, voltados a desenvolvedores especialistas ou não.

Segundo Padgham e Winikoff [PAD02], as principais características do Prometheus são: suporte ao desenvolvimento de agentes inteligentes que usam objetivos, crenças,

[1] *http://www.agent-software.com*

planos e eventos; suporte do início ao fim ao desenvolvimento de SMAs, desde a especificação até a implementação do sistema. A metodologia disponibiliza um processo detalhado que especifica os artefatos de projetos construídos e os passos para derivação de artefatos.

Esta metodologia abrange um mecanismo de estrutura hierárquica que permite a construção do projeto em múltiplos níveis de abstração. Ela possui uma abordagem interativa de engenharia de software, e provê uma automática verificação dos artefatos.

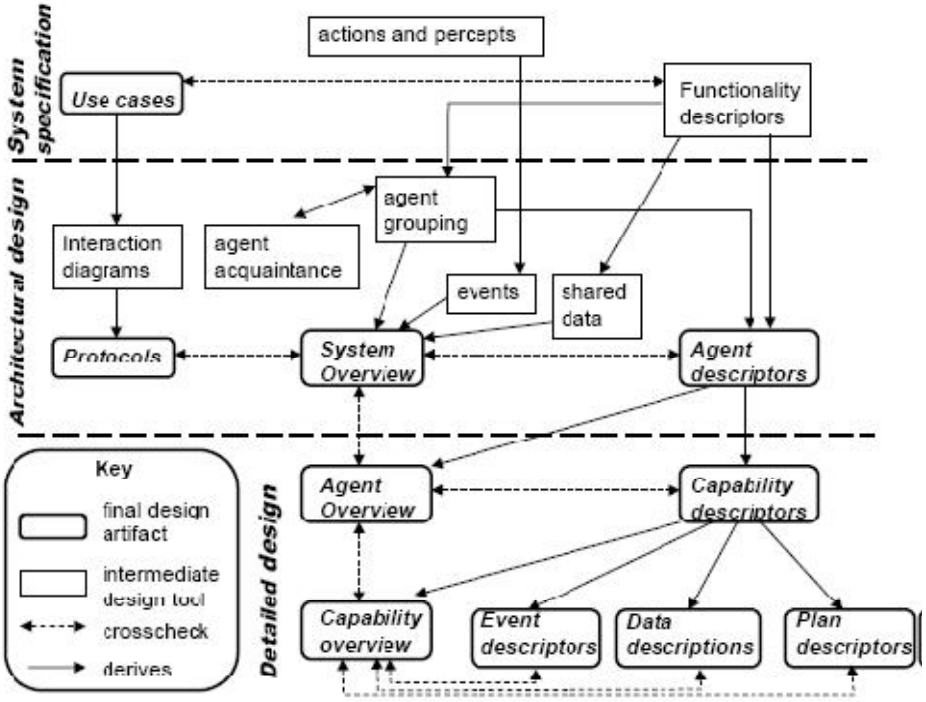


Figura 16 - Fases da metodologia Prometheus [PAD02].

O Prometheus consiste em três fases: especificação, projeto arquitetural e projeto detalhado, como ilustrado na figura 16.

A fase de especificação do sistema é responsável por identificar os objetivos, desenvolver os cenários de casos de uso ilustrando a operação do sistema, identificar as funcionalidades e especificar as ações e percepções.

O projeto arquitetural define quais agentes existirão e como interagirão no ambiente, envolvendo atividades como: definição dos tipos de agentes e das interações entre os agentes de um SMA.

A fase de projeto detalhado abrange a estrutura interna de cada agente e de que forma será realizada sua tarefa dentro do sistema.

De acordo com Silva [SIL08a], são duas as ferramentas que atualmente utilizam o Prometheus: o JACK [HOW01] e PDT *(Prometheus Projeto Tool)* [PDT08]. O ambiente de desenvolvimento do JACK inclui uma ferramenta de modelagem para a construção dos diagramas e resulta na geração do código na linguagem de programação. A ferramenta PDT permite ao usuário inserir e alterar projetos de SMAs, verificar possíveis inconsistências, gerar automaticamente um conjunto de diagramas de acordo com a metodologia e gerar automaticamente a descrição do projeto.

Padghan e Winikoff [PAD02a] e Winikoff e Padghan [WIN04] apresentam uma notação que representa os conceitos utilizados na metodologia Prometheus, embora nas referências eles não sejam claramente definidos.

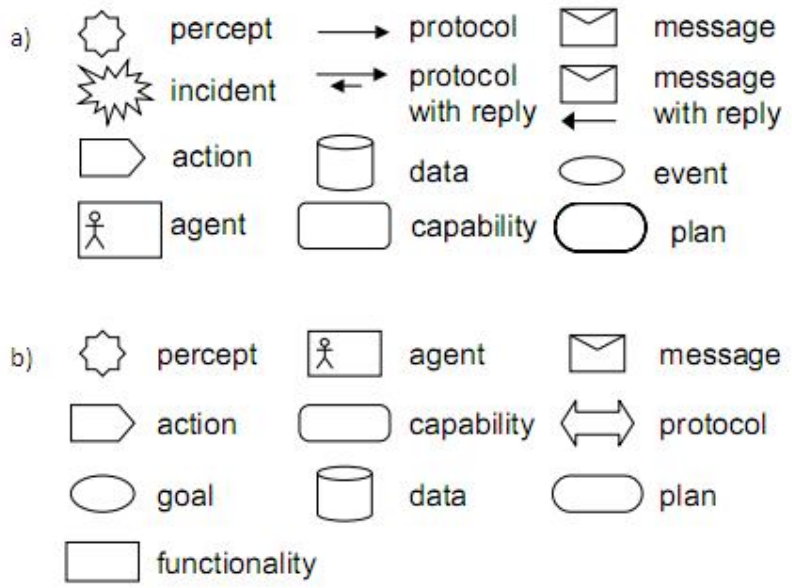


Figura 17 – Divergência entre as notações do Prometheus.

As notações divergem em alguns pontos entre cada referência. Em um trabalho é apresentada uma notação para entidade *incident*, e em outro esta entidade não é abordada; o protocolo possui notações diferentes em cada um das referências; a notação de evento é apresentada somente em [PAD02]; a notação de objetivo é apresentada somente em [WIN04], bem como a funcionalidade. Estas notações estão ilustradas na figura 17.

2.3.5 Ingenias

O Ingenias [GOM02] [PAV03] provê uma linguagem para modelar sistemas multiagentes, e ferramentas de suporte para análise, projeto, verificação e geração de

código pelo *Ingenias Development Kit* (IDK). Estas ferramentas, tão bem quanto à linguagem, são baseadas em especificações do seu meta-modelo que define as diferentes visões e conceitos de como o sistema multiagentes pode ser descrito. Estas visões são divididas em visão organizacional, de agentes, de ambiente, de tarefas e objetivos, e de interação.

A organização é um conjunto de entidades com relacionamentos de agregação e heranças, que define o local onde agentes, recursos, objetivos e tarefas existem. Do ponto de vista de agentes, cada agente é definido por seu propósito (objetivos que um agente tem o compromisso de prosseguir), responsabilidades (quais tarefas tem que executar), e capacidades (papéis que podem executar). Estes propósitos definem o estado mental do agente, a sua gestão e transformação. O estado mental consiste de metas do agente e informações sobre a satisfação dessas metas, conhecimentos do mundo e fatos que refletem a sua experiência passada para a tomada de decisão.

O ambiente é definido pela percepção e atuação dos agentes. Na visão de ambiente, identificam-se os recursos disponíveis e aplicações as quais um agente pode interagir. O ponto de visão de tarefas e objetivos explica como um objetivo alcançado afeta outras objetivos existentes usando relacionamentos de decomposição e dependência. Este ponto de vista também é responsável por descrever as conseqüências de se desempenhar uma tarefa e o porquê ela deve ser executada.

O ponto de visão de interação aborda a troca de informação ou requisições entre agentes, ou entre agente e usuários humanos. No Ingenias é considerada a motivação da interação e seus participantes e as informações sobre o estado mental exigido por cada agente durante a interação, assim como as tarefas executadas no processo. Desta forma, o Ingenias expressa em nível de *projeto* como um agente inicia e continua em uma interação.

Os Protocolos de interação podem ser especificados usando diferentes formalismos: AUML [ODE00], diagrama de colaboração UML [OMG08] e diagramas GRASIA, que é uma especialização do diagrama de colaboração UML para abordar questões intencionais associadas a uma interação.

A metodologia Ingenias adota uma hierarquia de conceitos básicos que possuem uma notação visual, conforme apresentado na figura 18 [GOM04] [GOM02]. Segundo Gomez-Sanz [GOM02], o meta-modelo do Ingenias é uma evolução do meta-modelo do MESSAGE [CAI01], resultando em cinco meta-modelos: do agente, da organização, da interação, de tarefas e objetivos.

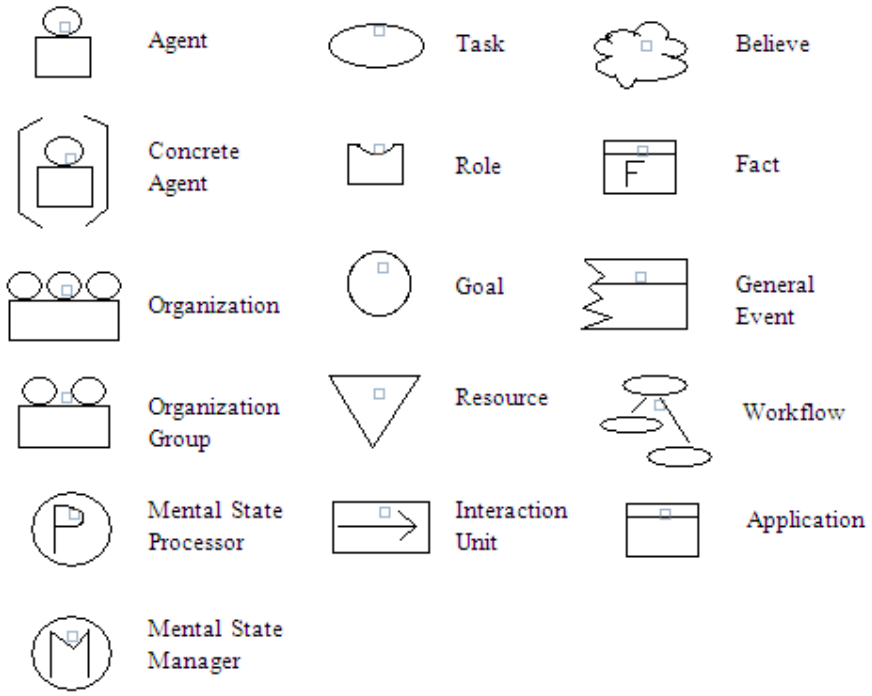


Figura 18 - Notação visual das principais entidades do Ingenias.

O meta-modelo de agentes descreve agentes particulares, com enfoque nas funcionalidades do agente e no projeto de seu controle. No Ingenias, o agente é uma entidade autônoma caracterizada por ter identidade única, propósitos, responsabilidades e capacidades. Este meta-modelo abrange a responsabilidade e o comportamento do agente, apresentado na figura 19.

As responsabilidades no meta-modelo são contempladas pelo uso de papéis no sistema que abrangem as tarefas a serem executadas para o alcance dos objetivos. O papel no Ingenias é uma abstração de um conjunto de funções que possui estado e depende da entidade agente para desempenhá-lo.

O comportamento no Ingenias engloba o controle do agente mediante mecanismos que vão assegurar as execuções das tarefas, através do estado mental do agente (entrada de um conjunto de dados), o qual possibilita a inclusão do aprendizado nas capacidades do agente.

O estado mental do agente são as informações gerenciadas e processadas que permitem ao agente tomar decisões. Desta forma, o Ingenias introduz duas entidades conceituais para representar o estado mental do agente: o gerenciador do estado mental que mantém a coerência do conhecimento armazenado com a evolução do estado mental mediante a criação, destruição, modificação e monitoramento do conhecimento do

agente; e o processador do estado mental que é direcionado a tomada de decisão em busca do alcance de objetivos, resultando na execução de uma sequência de tarefas.

No intuito de descobrir a evolução do estado mental do agente, o Ingenias utiliza a entidade *Consulta de Entidade Autônoma* para fazer referência às propriedades do agente. A consulta ao agente contém uma expressão de restrição que um agente deve satisfazer.

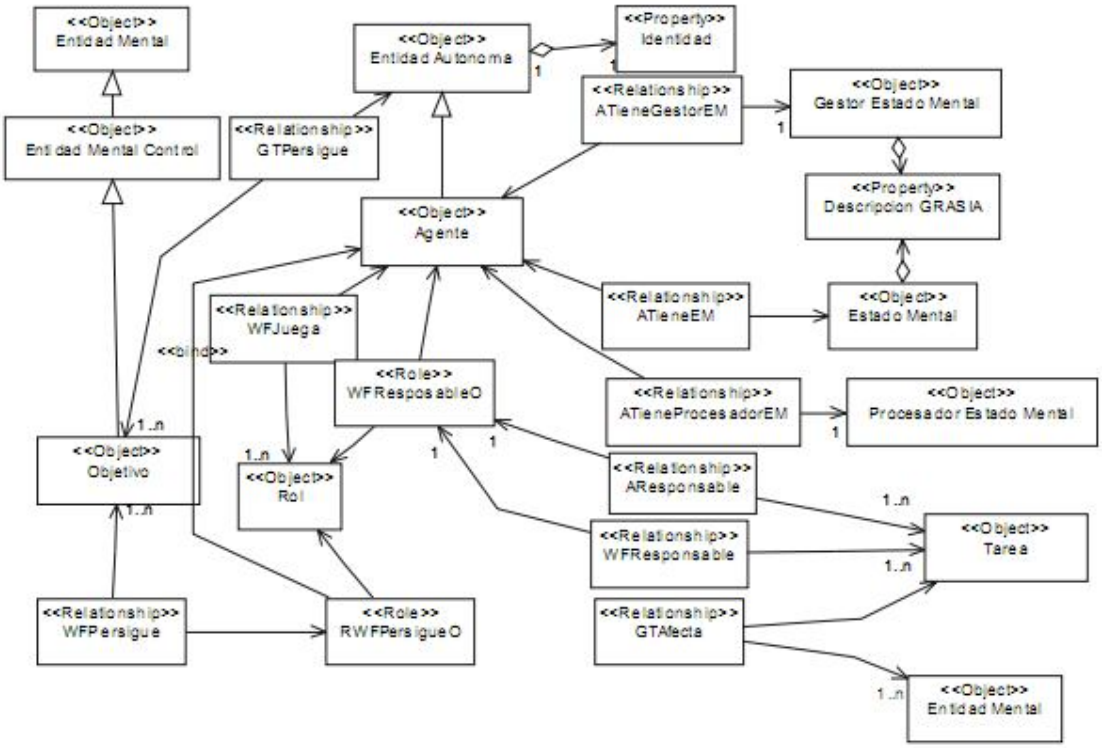


Figura 19 - Meta-modelo de agentes da metodologia Ingenias [GOM02].

O meta-modelo de interação busca definir o contexto das interações com enfoque no objetivo da interação e como será afetado o sistema, bem como quais serão as execuções da interação e quais as mensagens e a ordem em que as mesmas serão aplicadas. Este meta-modelo aborda agentes, papéis, objetivos, interações e unidades de interações. Os agentes e papéis são os atores em uma interação. Este meta-modelo é ilustrado na figura 20.

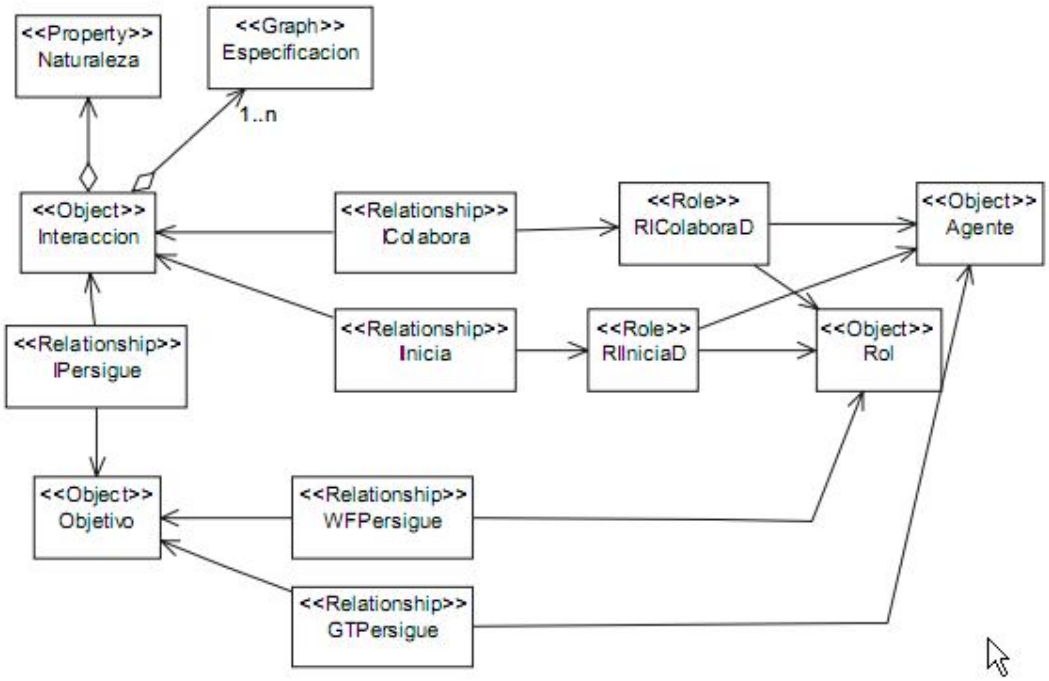


Figura 20 - Meta-modelo de Interação [GOM02].

O meta-modelo de objetivos e tarefas define as ações e responsabilidades identificadas nos meta-modelos de organização, interação e de agentes, como ilustrado na figura 21. Este meta-modelo traz os conceitos de objetivos e tarefas para esta metodologia.

Em Ingenias, uma tarefa é uma unidade transformadora de estado global com pré-condições e pós-condições, sendo ela vista como processo. Para uma simplificação do meta-modelo, o Ingenias optou por restringir o que se pode fazer em uma tarefa, omitindo o estado da tarefa, e colocando-a restrita em um tempo finito.

Nesta metodologia, objetivos são entidades auto-representativas que guiam o comportamento do agente e relacionam-se com as entidades agentes, papéis e organização.

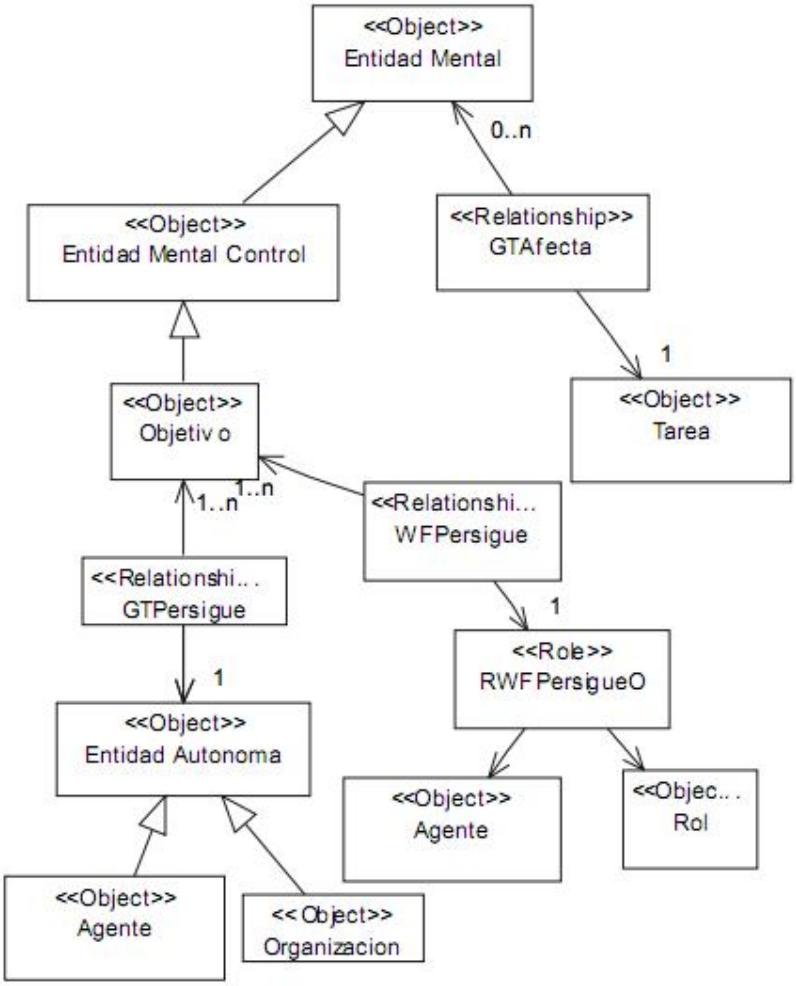


Figura 21 - Meta-modelo de objetivos e tarefas [GOM02].

A organização é separada em três definições: estrutural, funcional e social. Este meta-modelo é apresentado na figura 22.

A visão estrutural da organização proporciona a decomposição da organização em grupos e fluxos de trabalhos, onde cada grupo contém agentes, recursos, aplicações e papéis relacionados em um fluxo de trabalho.

Na descrição funcional, o fluxo de trabalho estabelece como são utilizados os recursos, quais tarefas são necessárias para a execução de um objetivo, e quem são os responsáveis por executá-lo. O fluxo de trabalho apresenta como as tarefas são associadas e executadas. O fluxo de trabalho pode ser decomposto em outros fluxos de trabalho ou tarefas. As tarefas são executadas por agentes diretamente ou através de seus papéis.

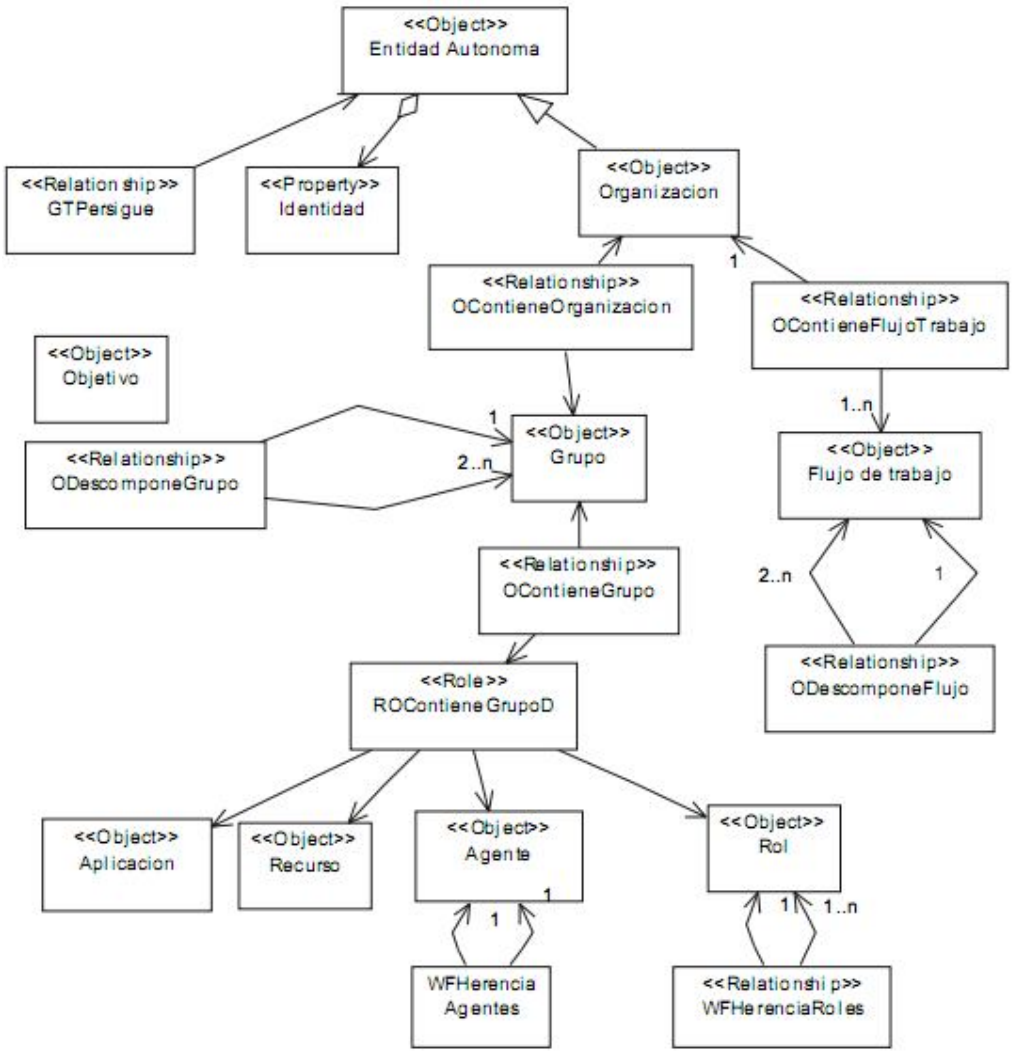


Figura 22 - Meta-modelo de Organização [GOM02].

Em uma descrição social de organização, são consideradas as relações de restrições sociais entre organizações, agentes e grupos, indicando a interação entre estas entidades. Estas relações são relações de subordinação, prestação de serviços e requisitos de um serviço.

Em uma relação de subordinação o subordinado cumpre todas as ordens do subordinador, em contradição com a característica de liberdade de atuação dos agentes. Esta relação é decomposta em relação de obediência incondicional e condicionada. A relação condicional é similar a um contrato, onde a infração de alguma condição implica na invalidade do contrato.

As relações de provedor e cliente de recurso fazem referência a um serviço oferecido por uma entidade e outra que a consome (cliente). Os agentes buscam satisfazer um objetivo, e nesta busca poder entrar em contato com outros agentes

utilizando tanto reação de busca de serviços quanto subordinação para satisfazer este objetivo.

O propósito do meta-modelo de ambiente é representar o mundo em que o SMA está inserido, contendo as entidades recursos, aplicações e agentes, limitando a percepção e a atuação dos agentes. Este meta-modelo é ilustrado na figura 23.

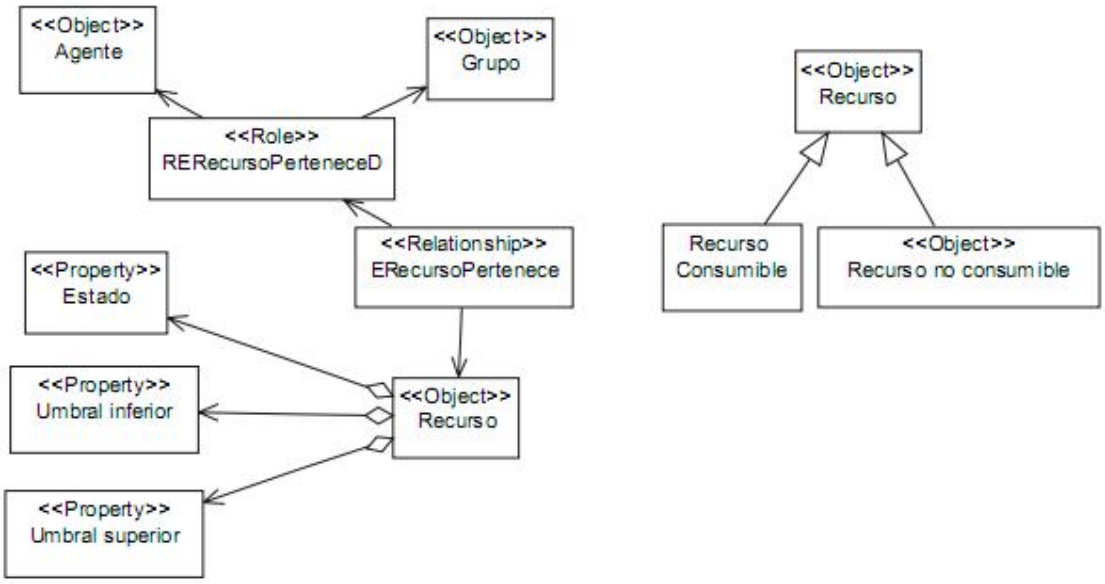


Figura 23 - Meta-modelo de Ambiente [GOM02].

As aplicações servem como atuadores e sensores dos agentes, servindo de interface para o mundo real. Os recursos podem ser utilizados ou não por um agente e são categorizados como consumíveis e não consumíveis. Além disso, os recursos são relacionados às tarefas através dos relacionamentos de consumo, produção e limitação. A relação de consumo indica o uso de um recurso e decrementa sua quantidade disponível; a relação de produção possibilita que uma tarefa ofereça um recurso; e a relação de limitação impõe pré-condições para lançar uma tarefa em função da disponibilidade de um recurso.

2.3.6 MESSAGE

O Methodology for Engineering System of Software Agents (MESSAGE) [MES08] [CAI01] é uma metodologia de engenharia de software orientada a agentes, desenvolvida para necessidades das indústrias de telecomunicação, embora abranja a maioria dos fundamentos de desenvolvimentos de SMAs, sendo genérica ao ponto de ser aplicável em outros domínios.

O MESSAGE possui uma linguagem de modelagem que estende o meta-modelo UML com conceitos de orientação a agentes. De acordo com Caire e outros [MES08], a maioria das entidades do MESSAGE podem ser agrupadas nas categorias: Entidade Concreta (*ConcreteEntity*), compostas pelas entidades agente, organização, papel e recurso; Atividade (*Activity*), compostas pelas entidades tarefas, interação e protocolo de interação; Estado Mental (*MentalStateEntity*), composto pelas entidades objetivo, entidade informacional e mensagem. A figura 24 ilustra uma visão geral de como estas entidades são relacionadas em uma visão com enfoque na entidade agente. O meta-modelo que representa todas as entidades, relacionamentos e restrições da metodologia MESSAGE está dividido em visões e pacotes, podendo ser acessado no site oficial do projeto MESSAGE da Eurescom [MES08].

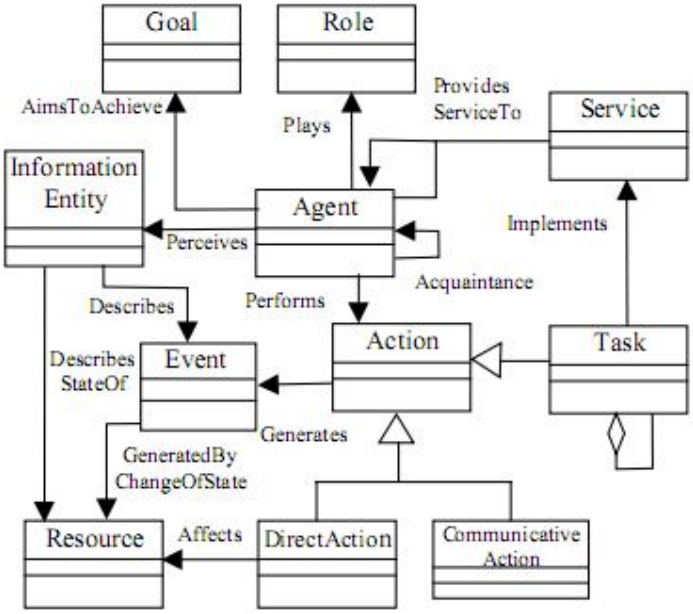


Figura 24 - Conceitos da metodologia MESSAGE sob visão de agente [EVA01].

Os conceitos aqui relacionados referem-se à metodologia de desenvolvimento de SMAs MESSAGE, de acordo com Cervenka [CER03] e Evans [EVA01].

O agente é uma entidade autônoma e atômica. Um agente pode executar papéis, prover serviços, executar tarefas, alcançar objetivos, usar recursos, ser parte de uma organização, e participar em uma interação ou protocolo de interação. Um serviço, provido por um agente, é análogo a operação de um objeto.

A organização[2] é um grupo de agentes trabalhando juntos em um propósito comum. Trata-se de uma entidade virtual, ou seja, o sistema não tem uma entidade computacional que corresponda à organização. Esta entidade atua como forma de relacionar agentes em uma relação organizacional com subordinação, controle e gerenciamento de procedimento, fluxos de tarefas e interação.

O papel exercido por um agente possibilita a separação lógica da própria identificação do agente, uma vez que um papel descreve as características externas de um agente em um contexto específico quando este o exerce. Além do agente, um papel pode ser exercido também por uma organização. Também é função de um papel prover serviços, executar tarefas, alcançar objetivos, usar recursos, ser parte de uma organização e participar em uma interação ou protocolos de interação.

O recurso é um conceito utilizado para representar entidades não autônomas como um banco de dados e pode ser utilizado por uma entidade autônoma.

Tarefa é uma unidade de atividade ligada a uma única entidade autônoma principal, sendo esta seu executor. Uma tarefa possui pré e pós-condições (restrições que definem o estado da tarefa). Quando uma pré-condição é válida espera-se que ocorra uma pós-condição associada ao término da tarefa. A tarefa pode ser composta por subtarefas, possuir estados associados, e estar relacionada com objetivos a serem alcançados.

Interação é um conceito que o MESSAGE importou da metodologia GAIA [WOO00]. A interação possui mais que um participante e um propósito que os participantes têm por objetivo alcançar coletivamente. Um protocolo de interação define o padrão de trocas de mensagens associadas com a interação.

Objetivos são intenções de entidades autônomas para alcançar um estado desejado. Alguns objetivos são intrínsecos ao agente e derivados de seus propósitos, persistindo durante toda vida do agente. Outros são transientes, que expressam o propósito em termos de funções úteis ao alcance um “bom valor” de um estado. Objetivos implicam em tarefas e podem ser decompostos em subobjetivos.

Uma entidade informacional é um objetivo que encapsula informações enquanto uma entidade mensagem é um objetivo que incorpora uma comunicação entre agentes. A transmissão de uma mensagem é realizada em um tempo finito e requer uma ação a ser executada pelo remetente, bem como pelo destinatário, possuindo um ato da fala (*speech*

[2] Em MESSAGE, um agente ou uma organização são comumente chamados de entidades autônomas

act) que categoriza a mensagem em termos de intenção do remetente, e um conteúdo composto por uma entidade informacional.

O MESSAGE introduz uma notação estendida da UML para suas entidades e relacionamentos, conforme ilustrado na figura 25.

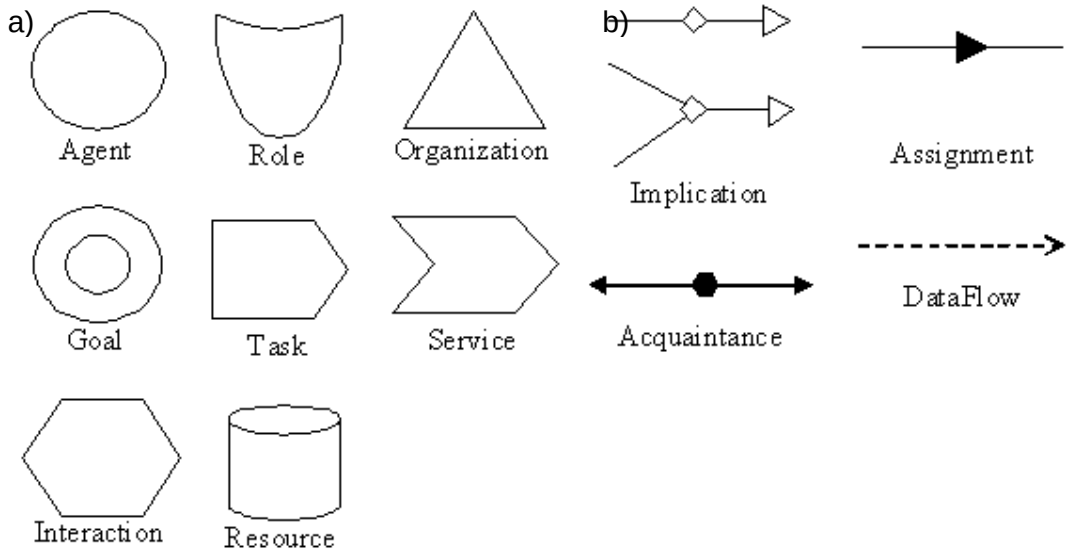


Figura 25 – Notação para a) os conceitos e b) os relacionamentos [CER03].

De acordo com Cervenka [CER03], o MESSAGE define visões ou perspectivas para diferentes aspectos do modelo. As visões são divididas em: visão de objetivo/tarefa, visão de agentes/papel, visão de interação e visão de domínio.

A visão de objetivo/tarefa mostra a dependência entre um objetivo e uma tarefa. Objetivos podem ser ligados através de uma dependência lógica com tarefas para formar gráficos que mostram como alcançar um objetivo ou conjunto de subobjetivos, e como as tarefas podem ser executadas para o alcance destes objetivos.

A visão de Agentes/papel foca um agente individualmente e seus papéis. Para cada agente/papel são usados esquemas suportados por diferentes diagramas para explicitar características, tais como: por qual objetivo o agente é responsável, que eventos são necessários, quais recursos ele controla e quais tarefas serão executadas (regras de comportamento).

A visão de interação aborda as interações, os papéis relacionados ao agente, a informação fornecida ou alcançada por agentes em uma interação, e o evento que inicia a interação.

A visão de domínio mostra o domínio específico de conceitos e relacionamentos que são relevantes para o sistema em desenvolvimento. Para esta visão, o MESSAGE utiliza os diagramas de classe da UML.

2.3.7 MAS-ML

O MAS-ML [SIL04] *(Multi-Agent System Modeling Language*) é uma linguagem de modelagem de sistema multiagentes que visa modelar todos os aspectos dinâmicos e estruturais definidos no *framework* TAO apresentado por Silva e outros em [SIL04a]. O meta-modelo do MAS-ML é definido estendendo o meta-modelo do UML de acordo com as entidades e relacionamentos definidos no TAO.

De acordo com Silva [SIL04a], o *Tamming Agent and Object* (TAO) é um *framework* desenvolvido com base em pesquisas sobre metodologias orientadas a objetos e orientadas a agentes, e sobre teorias e linguagens existentes relacionadas ao desenvolvimento de SMAs. O TAO oferece suporte a novas metodologias e linguagens que se baseiam em seus conceitos.

Os conceitos definidos pelo TAO dividem-se quanto aos aspectos estruturais e dinâmicos. Na descrição dos aspectos estruturais, as entidades que podem ser descritas são definidas juntamente com suas propriedades e relacionamentos associados. Os aspectos dinâmicos definem a criação e a destruição de entidades e também o comportamento independente de domínio. Ambos os aspectos são representados em MAS-ML através de diagramas estruturais e dinâmicos durante a fase de projeto e análise.

Para o diagrama dinâmico da MAS-ML, é utilizado o diagrama de seqüência estendido da UML com objetivo de modelar a interação entre as entidades, sua execução interna e os protocolos de interação entre os agentes. O diagrama estrutural da MAS-ML é composto pelo diagrama de classes estendido e dois novos diagramas chamados diagramas de organização e de papel.

O diagrama de classes da MAS-ML estende o diagrama de classes da UML com o objetivo de modelar os agentes, as organizações e os ambientes, utilizando os relacionamentos *inhabitat, association* e *specialization* do *framework* TAO. O diagrama de organização modela organizações, agentes, papéis de agentes, papéis de objetos e ambientes, utilizando os relacionamentos *ownership, play* e *inhabit*. O diagrama de papel modela os papéis do agente, papéis de objetos e a classe, utilizando os relacionamentos *control, dependency, association, aggregation e specialization*.

MAS-ML insere novas metaclasses para prover uma extensão de UML em que possam ser representados os agentes, as organizações, os ambientes, os papéis de agentes e os papéis de objetos como demonstrado na figura 26.

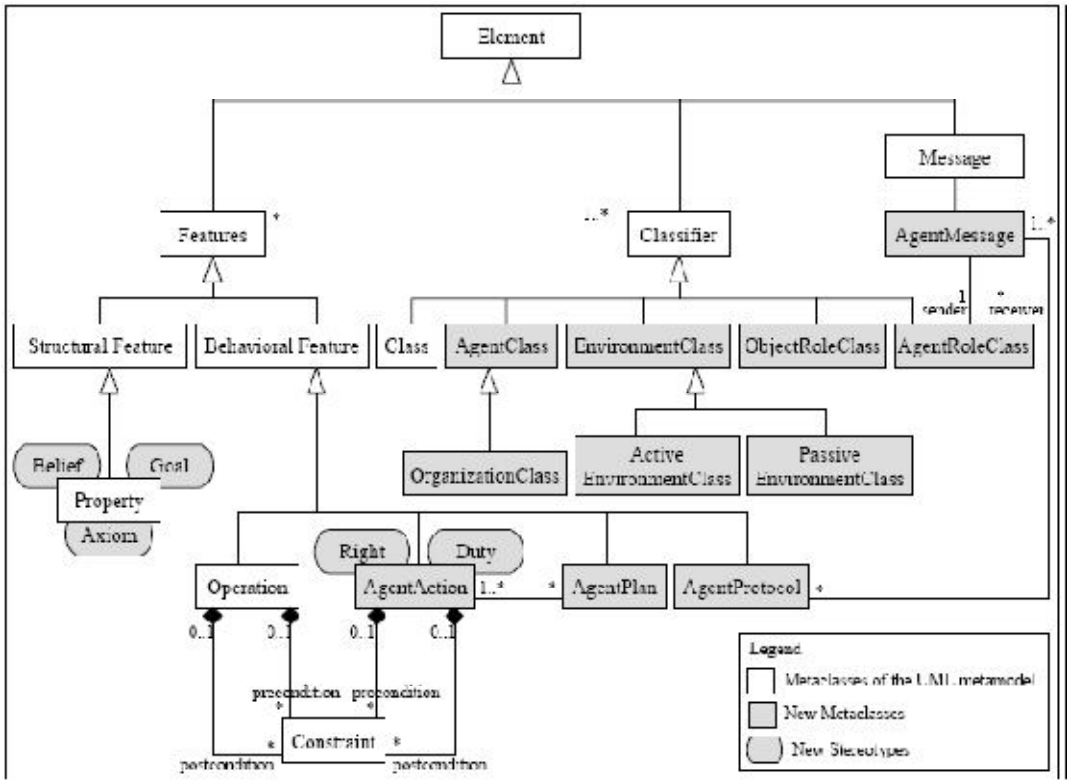


Figura 26 - Meta-modelo MAS-ML [SIL04].

Segundo Silva [SIL04], as entidades definidas no TAO são objetos, agentes, organização, papel de agente, papel de objeto, ambiente e evento. Por uma questão de similaridade entre algumas entidades, o MAS-ML define uma nova abstração chamada elemento como base de definição para a maioria das entidades. Esta entidade elemento possui estado, comportamento e relacionamentos com outros elementos. Das entidades definidas pelo TAO apenas evento não é baseada na entidade elemento. MAS-ML também apresenta uma notação para suas entidades e relacionamento, ilustradas na figura 27.

As definições seguintes são referente à linguagem de modelagem MAS-ML, e estão de acordo com Silva em [SIL04].

Os objetos são elementos que possuem: um estado que armazena informações sobre o ambiente, sobre si mesmo e sobre outros objetos; comportamentos para definir quais operações serão executadas; e relacionamentos para demonstrar como os objetos estão relacionados aos agentes, papéis e outros objetos do sistema. Um objeto pode

executar um papel definido pela organização que utiliza este objeto, e pode controlar seu estado, mas não seu comportamento, não sendo autônomo por sempre *depender* da requisição de outro objeto.

Figura 27 - Notação de entidades e relacionamento da MAS-ML [SIL04].

Agentes são elementos que definem as propriedades de estado mental e comportamento. O estado mental de um agente é expresso por crenças, objetivos, planos e ações.

A crença é o conhecimento de um agente sobre o ambiente, outro agente e de si mesmo, o que inclui questões como: o que o agente sabe, o que o agente vê, suas memórias e suas percepções sobre tudo que acontece dentro do SMA.

O objetivo consiste no estado futuro que um agente deseja alcançar através da execução de planos.

Um plano pode ser modelado utilizando uma máquina de estados, na qual os estados são compostos por: estado inicial, estado intermediário, estado final e transição entre dois estados. Os estados podem ser definidos, brevemente como: o estado inicial, que descreve as restrições para o estado mental do agente antes de executar o plano, conceito equivalente às pré-condições; o estado intermediário, que descreve as ações que um agente deve executar, sendo que esta execução pode alterar seu estado mental enquanto o agente as executa. As ações compõem o plano e quando executadas levam o agente mais próximo de seu objetivo; as transições são definidas com base nas alterações do estado mental do agente; o estado final descreve o estado mental do agente após executar o plano.

O comportamento de um agente é expresso por meio de seus planos e ações, e baseado em suas características, tais como autonomia, interação e adaptação. O relacionamento descreve como um agente relaciona-se a outro elemento.

O ambiente é o habitar de agentes, organização e objetos. O estado e comportamento de um ambiente são baseados em suas características, podendo ser passivo como um objeto ou ativo (autônomo, adaptativo e interativo) como um agente.

Os eventos são gerados por objetos, ambientes, agentes e organização. Um evento pode iniciar a execução de ações associadas a agentes ou operações associadas a objetos de acordo com a percepção deste evento.

A organização é um grupo de agentes em um SMA que define suborganizações e um conjunto de leis (axiomas ou regras estabelecidas) que os agentes desta organização devem obedecer, assim como papéis que devem ser executados. O estado de uma organização é representado por seus objetivos, crenças e axiomas, e o comportamento é representado por ações, planos e papéis que os agentes pertencentes da organização devem executar. A organização relaciona-se com as entidades crença, objetivo, ação, plano e axioma.

O papel em MAS-ML é um elemento que define um conjunto de propriedades e relacionamentos, e divide-se em papel de objeto e papel de agente. O papel de objeto guia, restringe e pode incrementar o estado e o comportamento de um objeto e de como ele é visualizado (ou acessado) por outros elementos. O papel de agente guia e restringe o comportamento de um agente. Este papel descreve o objetivo do agente para executar o papel, as crenças e ações a serem executadas, podendo também adicionar novos objetivos e crenças, e define os direitos e deveres e protocolos relacionados ao agente enquanto executa este papel.

O estado de um papel de agente é definido por sua crença gerada a partir de fatos do ambiente e de seu objetivo, formado pelo objetivo do agente ao executar este papel. Os objetivos do grupo de papéis formam o objetivo da organização.

O comportamento de um papel de agente é definido pelas regras e direitos e protocolos. As regras definem as ações atribuídas ao agente durante a execução do papel, e os direitos definem as ações que o agente pode executar durante a execução do papel. O protocolo define o conjunto de mensagens que um agente está habilitado a enviar e receber de outro agente em uma interação. Os relacionamentos de agentes são baseados nos protocolos associados ao papel.

Durante a extensão da UML novas entidades e relacionamentos foram adicionados para a MAS-ML. Os relacionamentos do MAS-ML especializam-se em associação e relacionamento direto, este último especializa-se em dependência, generalização e as novas metaclasses *inhabit, ownership, play e control* incluídas no meta-modelo do MAS-ML.

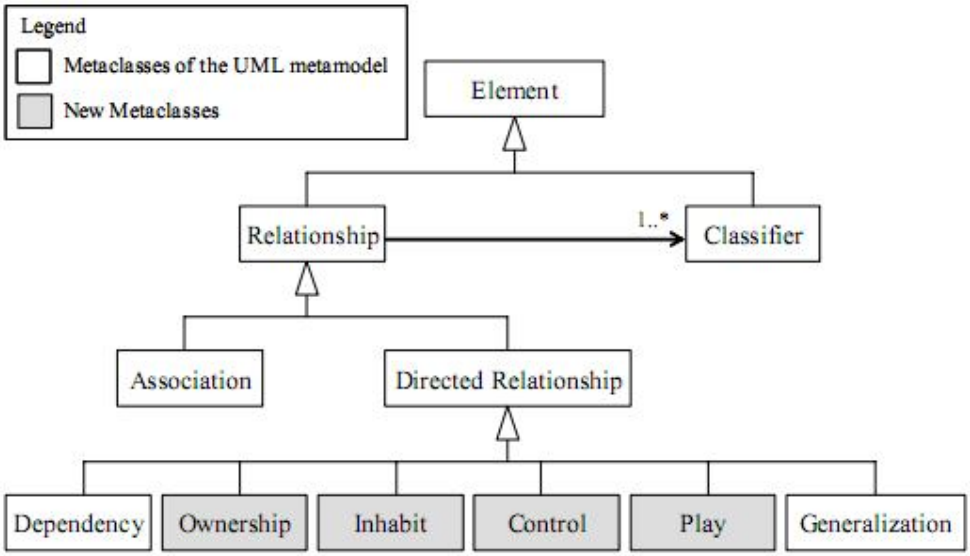


Figura 28 – Segunda parte do Meta-modelo MAS-ML [SIL04].

O relacionamento *inhabit* é uma nova metaclasse que especifica a criação e destruição de um elemento (também chamado de cidadão) em um habitat e sua entrada e saída mediante permissão deste habitat. Este relacionamento é aplicado ao ambiente, ao agente, ao objeto ou à organização.

O relacionamento *ownership* especifica que um elemento (o membro) é definido no escopo de outro elemento (o dono) e que o membro deve obedecer a um conjunto de restrições globais do dono. O membro não existe fora do escopo do dono e o dono conhece todos seus membros. Este relacionamento é aplicado ao papel (membro) e a organização (dono) que as define, desta forma a organização possui total controle sobre o papel.

O relacionamento *play* especifica que um elemento é relacionado a um papel. Quando um agente ou organização executa um papel, eles incorporam os objetivos e crenças do papel. O relacionamento que liga dois papéis define a interação entre os elementos que os executam.

O relacionamento *control* define que o elemento controlado deve fazer qualquer coisa que o elemento controlador requer. Este relacionamento pode ser usado entre dois papéis de agentes.

Os demais relacionamentos são conceitos da UML utilizados na modelagem de SMAs, conforme ilustrado na figura 28.

2.4 Visão geral sobre metodologias e linguagens de modelagem de SMAs

As Plataformas de implementação se SMAs são geralmente *frameworks* que possibilitam o desenvolvimento dos SMAs em uma linguagem de programação. Algumas plataformas são propostas como suporte a metodologias e meta-modelos específicos através de um ambiente integrado de desenvolvimento e outras apresentam maior flexibilidade de uso. Nesta seção, serão apresentadas as diferentes plataformas de implementação de SMAs estudadas. Esse estudo foi realizado com o objetivo de verificar os conceitos de SMA tratadas nas plataformas.

2.4.1 SemantiCore

O SemantiCore [BLO04] é um *framework* que surgiu a partir de uma extensão da arquitetura *Web Life* [RIB02]. A finalidade deste *framework* é prover uma camada de abstração sobre uma arquitetura distribuída para criação de SMAs na *Web* Semântica.

O desenvolvimento de SMAs com o SemantiCore é possível através de sua instanciação em um domínio específico. Este *framework* fornece uma interface para o desenvolvimento de aplicações e encapsula os detalhes de implementação, tais como protocolos de comunicação, envio e recebimento de mensagens, entre outros, como ilustrada na figura 29.

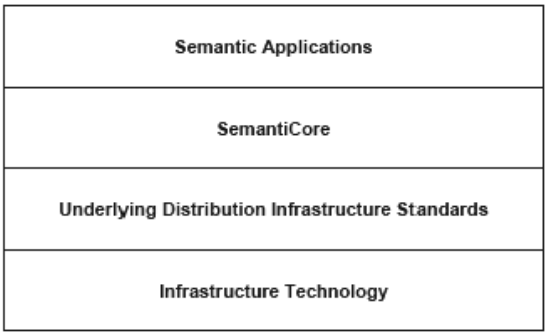


Figura 29 - Arquitetura do SemantiCore [BLO04].

O *framework* SemantiCore é dividido em dois modelos: o modelo do agente (*SemanticAgent*) e o do domínio semântico. Ambos possuem pontos de flexibilidade (*hotspots*) que permitem aos desenvolvedores adaptar diferentes padrões, protocolos e tecnologias às funcionalidades do *framework*.

O modelo de agente define todos os elementos necessários para a construção do agente, que quando implementado, é capaz de acessar seus componentes com intuito de realizar determinadas tarefas. Os agentes do SemantiCore são compostos por quatro

componentes, no qual cada componente é responsável por uma tarefa específica. Estes componentes são: Sensorial, Decisório, Executor e Efetuador, conforme ilustrado na figura 30.

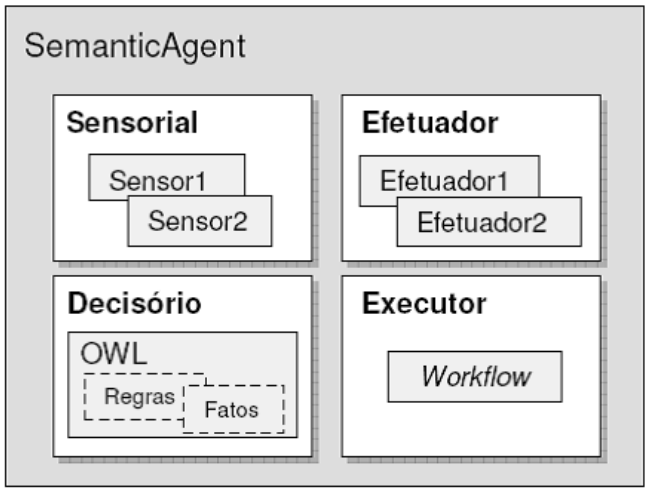


Figura 30 - Arquitetura de um agente semântico [ESC06].

O componente sensorial é responsável por perceber e capturar os recursos que trafegam no ambiente, sendo composto por uma série de sensores definidos pelo desenvolvedor. Cada sensor captura um tipo diferente de objeto no ambiente, sendo estes encaminhados para processamento em outros componentes.

O componente decisório encapsula o mecanismo de tomada de decisão do agente, sendo este um dos pontos de flexibilidade (*hotspots*) do *framework*. Na *Web* semântica é importante que o componente opere sobre ontologias escritas em OWL para a definição semântica dos dados. O escopo do componente decisório deve ser uma instância de uma ação (classe *Action*) com mapeamento das ações a serem executadas adequadamente, aplicadas tanto ao agente quanto aos elementos do domínio semântico. O desenvolvedor pode implementar suas ações estendendo a classe *Action.* Para o conceito de ação no *SemantiCore*, Escobar e outros [Esc06] consideram o conceito de processos, modelado por Ferber em [Fer99].

O componente executor contém e executa os planos de ação do agente. Este componente trabalha com o mecanismo de fluxo de trabalho (*workflow*) para controlar as transições das atividades dentro de um processo deste fluxo.

O componente efetuador é responsável pelo encapsulamento de dados em mensagens para transmiti-las no ambiente. Este componente armazena vários efetuadores, onde cada um tem o objetivo de publicar um tipo diferente de objeto no

ambiente. Esta comunicação utiliza diferentes padrões como *Web Service SOAP* [GUD06] e FIPA-ACL [FIP08].

Ainda de acordo com Escobar e outros, os componentes do agente semântico estão associados a quatro classes que representam os componentes estruturais básicos de um agente no SemantiCore, que são: o sensorial (classe *Sensor*), o efetuador (classe *Effector*), o decisório (classe *Rule*) e o executor (classe *Action*).

O SemantiCore define o ambiente onde o agente atuará através do domínio semântico, e requer um domínio *Web* para sua execução. O domínio *Web* está vinculado ao domínio semântico que é administrado por duas entidades: o controlador de domínio e o gerente do ambiente. O Controlador de Domínio (*Domain Controller*) registra os agentes no ambiente e o Gerente de Ambiente atua como uma ponte entre o domínio semântico do SemantiCore e os domínios *Web* convencionais. O modelo de domínio está ilustrado na figura 31.

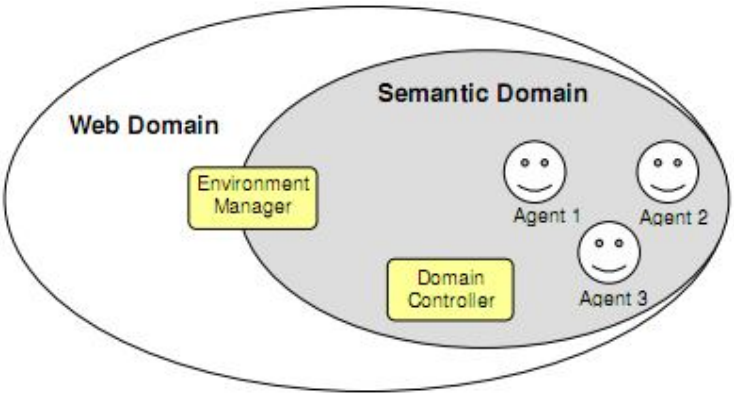


Figura 31 - Modelo semântico do SemantiCore [ESC06].

A versão mais atual do SemantiCore é a versão 2006 a qual apresenta como inovações a forma de como onde os agentes podem ser descritos usando representação ontológica, a capacidade de criação de domínios e agentes distribuídos e a separação entre o barramento de controle e o de dados.

A representação ontológica dos agentes do *SemantiCore* utiliza a linguagem OWL. A figura 32 [ESC06] ilustra esta representação, na qual se observa a classe *Fact* e suas especializações *SimpleFact*, *FuncitonBasedFact* e *ComposedFact*. Estas classes representam fatos usados como pré e pós-condição das ações e na seleção de mensagem nos sensores. Dentre elas, a classe *SimpleFact* representa um fato simples através das propriedades sujeito, predicado e objeto que formam um tripla RDF. Os fatos são utilizados principalmente na tomada de decisão do agente.

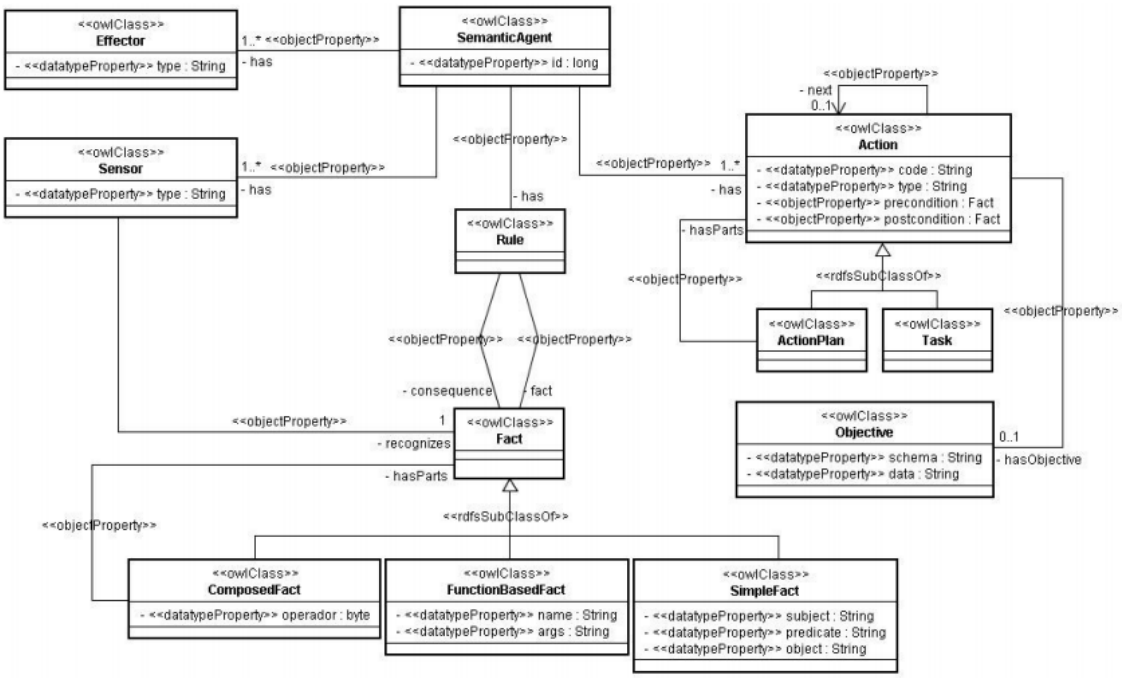


Figura 32 - Representação ontológica de um agente semântico.

A possibilidade de distribuição de um agente permite aos componentes do *SemantiCore* 2006 estarem espalhados em distintas partes do domínio e serem localizados através do componente sensorial. Uma tabela de roteamento com os endereços dos componentes é criada para determinar o caminho entre os componentes do sistema, permitindo assim ao componente transmissor, recuperar a localização do componente de destino ao solicitar um envio de informação. A distribuição de mensagens e o controle de localização dos componentes é responsabilidade do *framework SemantiCore*.

Para o processo de comunicação no *SemantiCore* é apresentado dois tipos de barramento: de controle e de dados. O barramento de controle trafega as mensagens em um formato fixo e proprietário sem acesso ao desenvolvedor, visando garantir a integridade da execução ao agente. O barramento de dados trafega as mensagens de dados entre diferentes agentes, estando ligado diretamente ao componente sensorial e efetuador para recepção e envio de mensagens de outros agentes. O formato de mensagens do barramento de dados é um dos *hostposts* do *framework*, a ser definido pelo desenvolvedor.

Estudos utilizando a plataforma de implementação *SemantiCore* são apresentados por Blois e outros [BLO07] e Escobar e outros [ESC07]. No primeiro estudo, é apresentado um caso de estudo adaptado de um clássico exemplo da *Web* Semântica, construído sobre solução baseada em agentes. O segundo estudo apresenta o

desenvolvimento do *SemantiCore Mobile*, com objetivo de prover suporte de aplicações multiagentes em dispositivos móveis e embarcados.

2.4.2 Jason

O Jason é uma plataforma de desenvolvimento de sistemas multiagentes, apresentada por Bordini e outros [BOR07], baseada em um interpretador para uma versão estendida da linguagem *AgentSpeak* [Rao96]. Segundo Hübner e outros [HUB04], além de interpretar a linguagem *AgentSpeak* original, o Jason possui os seguintes recursos: negação forte (*strong negation*), portanto tanto sistemas que consideram mundo-fechado (*closed-world*) quanto mundo-aberto (*open-world*) são possíveis; tratamento de falhas em planos; comunicação baseada em atos de fala; anotações em identificadores de planos, que podem ser utilizadas na elaboração de funções personalizadas para seleção de planos; suporte para o desenvolvimento de ambientes (no Jason o ambiente é programado em Java); possibilidade de executar o SMAs distribuidamente em uma rede com o uso do Jade [BEL07]; possibilidade de especializar as funções de seleção de planos e confiança, e toda a arquitetura do agente (percepção, revisão de crenças, comunicação e atuação); biblioteca básica de “ações internas” e extensão da biblioteca de ações internas.

A linguagem *AgentSpeak*(L) foi projetada para a programação de agentes *BDI* na forma de sistemas de planejamento reativos (*reactive planning systems*). Esta linguagem é orientada a agentes e baseada na lógica de primeira ordem, com uso de eventos e ações baseada em arquitetura *BDI*, como *Procedural Reasoning System* (PRS) [GEO86]. Embora o Jason tenha sido implementado com base na linguagem *AgentSpeak*, ele ainda possibilita uma série de extensões que são necessárias para o desenvolvimento de sistemas multiagentes.

Um agente *AgentSpeak* é definido por um conjunto de crenças que permite obter seu estado inicial através de um conjunto de fórmulas atômicas de primeira ordem e um conjunto de planos que formam sua biblioteca de planos. Para se descrever um plano, o Jason introduz as noções de objetivos e eventos de disparo.

Nesta abordagem, o objetivo de um agente é um estado do sistema que o agente deseja alcançar, sendo dividido em objetivos de alcance e objetivos de teste. No primeiro, o agente deseja atingir o estado de mundo onde a fórmula atômica associada é verdadeira. No segundo, o agente deseja testar se a fórmula atômica associada é (ou

pode ser unida com) uma de suas crenças. O objetivo é atingido através da biblioteca de planos do agente, determinada a partir de um conjunto de ações que deve ser executada.

Um plano da biblioteca de planos é constituído de cabeçalho e corpo. O cabeçalho é formado por um evento de disparo (*triggering event*), o qual define quais eventos podem iniciar a execução de um plano e um conjunto de literais representando um contexto. O corpo do plano inclui ações básicas, ou seja, ações que representam operações atômicas que o agente pode executar a fim de alterar o ambiente.

Um evento pode ser interno quando gerado pela execução de um plano, ou externo quando gerado pelas atualizações de crenças que resultam da percepção do ambiente. O contexto deve ser conseqüência lógica do conjunto de crenças do agente no momento em que o evento é selecionado pelo agente para o plano ser considerado aplicável.

O Jason é implementado em Java e está disponível em código aberto com um ambiente de desenvolvimento integrado (*IDE*) que permite a execução de programas também em modo de depuração. Uma vantagem do uso da linguagem *AgentSpeak* para o desenvolvimento de sistemas multiagentes é que ela possui semântica formal, o que possibilita a verificação formal de sistemas programados.

2.4.3 JACK

O JACK *Inteligent Agent* [HOW01] é um *framework* desenvolvido por *Agent Oriented Software* (AOS) que traz um conceito de agentes inteligentes em uma forte tendência a engenharia de software comercial com suporte ao modelo *BDI*. O JACK é a terceira geração de *frameworks* de agentes, desenhada como um conjunto de componentes leves com alto desempenho de forte tipação de dados.

Segundo Howden e outros [HOW01], os principais objetivos do JACK são: fornecer aos desenvolvedores um *framework* que provê um robusto, estável e leve produto; satisfazer aplicações práticas para uma variedade de necessidades; facilitar a transferência tecnológica dos pesquisadores para a indústria; e habilitar adição de aplicações de pesquisas.

De uma perspectiva de engenharia de software, o JACK é constituído por uma instalação independente de arquitetura, e acrescido de um conjunto de *plugins* de componentes que abordam os requisitos específicos da arquitetura de agentes. Em uma visão de programação, o JACK consiste em três principais extensões para Java: um

conjunto de adições sintáticas, um compilador e um conjunto de classes chamado *Kernel*, formando a linguagem de programação orientada a agente *Jack Agent Language* (JAL).

O conjunto de adições sintáticas de JACK se divide em: um pequeno número de palavras-chave para identificar o principal componente do agente; um conjunto de expressões para definição de atributos fortemente tipados; um conjunto de expressões para definição de relacionamentos estáticos; e em um conjunto de expressões para manipulação do estado de um agente.

O compilador do JACK converte as adições sintáticas em classes Java e em expressões que podem ser carregadas e chamadas por outro código Java. O compilador também transforma parcialmente o código de planos para obter uma correta semântica da arquitetura *BDI*.

O *kernel* do JACK provê o suporte em tempo de execução requerido na geração de código, o qual inclui gerenciamento automático de concorrência entre tarefas, comportamento padrão para agentes em reação a eventos, falhas de ação e tarefas, em uma infra-estrutura de comunicação para aplicações multiagentes. Este *kernel* suporta múltiplos agentes com um processo único, muitos processos, e uma mescla dos dois, sendo esta particularidade conveniente para economizar recursos do sistema.

Segundo Nunes [NUM07], os agentes JACK são componentes de software autônomos que têm objetivos explícitos para atingir ou eventos para tratar. O agente pode exibir um comportamento racional sob estímulos pró-ativos (direcionado a objetivos) e reativos (orientado a eventos).

Ainda segundo Nunes, cada agente possui um conjunto de crenças sobre o mundo, de eventos que ele irá responder, e objetivos que deseja atingir. O alcance de um objetivo pode ser tanto em resposta a uma requisição de um agente externo, como uma conseqüência à ocorrência de um evento, ou quando uma ou mais de suas crenças mudam. Um conjunto de planos descreve como o agente pode lidar com os objetivos e eventos que possam surgir.

Quando um agente JACK é criado no sistema, ele normalmente fica inativo até que receba um objetivo ou um evento ao qual ele deve responder. Uma vez que ele receba tal objetivo ou tal evento, o agente determina quais ações são necessárias para alcançar o objetivo ou responder ao evento.

Assim, o Jack apresenta-se como uma plataforma de desenvolvimento de SMAs, com uma linguagem de programação orientada a agentes e voltada para aplicações

industriais. O JACK possui aplicações comerciais como *Unmanned Aerial Vehicles* (UAVs) [KAR04], e continua em desenvolvimento atualmente.

2.5 Considerações

Nesse capítulo, foi apresentado todo o estudo teórico realizado para o desenvolvimento do meta-modelo, sendo que este foi uma pesquisa refeita e mais aprofundada em relação ao estudo bibliográfico apresentado em [SAN08], no intuito de incrementar a proposta deste trabalho com novas abordagens. O estudo de agentes de software possibilitou uma visão geral da área, além de permitir a identificação das características internas de agentes tratadas na literatura. Por outro lado, com o estudo das abordagens, foi possível a definição de um meta-modelo inicial. Ainda nesse capítulo, foram introduzidos os conceitos de restrições de integridade e linguagem OCL, possibilitando assim a aplicação das restrições de integridade e a posterior verificação da consistência de modelos instanciados a partir do meta-modelo. Por fim, o estudo de algumas plataformas de implementação de SMAs foi realizado com o objetivo de verificar a possibilidade de geração de código nas mesmas.

No capítulo seguinte será apresentado o processo de desenvolvimento do meta-modelo baseado no estudo teórico realizado ao longo desse capítulo.

3 TRABALHOS RELACIONADOS

3.1 Meta-modelo de representação interna de um agente

O meta-modelo para representação interna de agentes de software foi proposto por Santos [SAN08] a partir do estudo de algumas metodologias e plataformas de implementação de SMAs. O trabalho de Santos justifica-se pela deficiência na modelagem interna dos agentes de software encontrada nas abordagens pesquisadas e pela necessidade de representar a estrutura interna e o comportamento de um agente para sua posterior implementação.

Em uma análise comparativa com as metodologias MASUP, Tropos, MaSE [DEL99], MAS-ML, MAS-CommonKADS [IGL98] e Prometheus, Santos identificou algumas entidades e relacionamentos para modelagem interna de agentes, sintetizando os conceitos pesquisados. Após um processo de refinamento, Santos propôs o MRIA apresentando a definição das entidades, relacionamentos e restrições presentes no meta-modelo.

Santos define pacotes para a organização das entidades e relacionamentos do MRIA:

- Pacote *Decision*, o qual contém as entidades *Belief*, *Term*, *Sentence*, *Operator* e *Rule*. Este pacote é responsável pela tomada de decisão dos agentes.
- Pacote *Sensorial*, constituído pelas entidades *Perceptron*, *Event*, *InternalEvent* e *ExternalEvent*. Este pacote é responsável pela percepção e disparo de eventos internos e externos.
- Pacote *Communication*, contendo as entidades *Protocol*, *Message* e *Field*. Este pacote representa a interface de interação e comunicação do agente.
- Pacote *Executor*, formado pelas entidades *Action* e *Plan*. Este pacote é responsável pelas tarefas a serem executadas pelo agente.
- Pacote *Main*, o qual possui as entidades *Agent*, *Goal*, *Resource* e *Role*.
- O meta-modelo define os seguintes conceitos.
- Agente (*Agent*): é um sistema computacional inserido em um ambiente, capaz de atingir os objetivos planejados por meio de ações autônomas nesse ambiente.

- Objetivo (*Goal*): representa os desejos ou estados futuros que o agente deve atingir.
- Recursos (*Resource*): representa uma entidade física ou uma informação.
- Papel (*Role*): responsável por uma representação abstrata de uma função de agente, serviço ou identificação dentro de um grupo. Cada papel pode ter associado a si um conjunto de atribuições e restrições.
- Plano (*Plan*): formada por um conjunto de ações, de acordo com a definição de Woldridge [WOO02], a qual define o plano constituído por pré-condições, corpo e pós-condições.
- Ação (*Action*): também conhecido como tarefa, apresenta-se como parte de um trabalho que pode ser atribuída a um agente ou ser executada por este.
- Protocolo (*Protocol*): representa o protocolo da mensagem a ser utilizada pelo agente.
- Mensagem (*Message*): representa as mensagens de entrada e saída do agente.
- Campo (*Field*): representa os parâmetros que compõem determinado tipo de mensagem.
- Percepção (*Perceptron*): responsável por perceber as mensagens que vem do ambiente para o agente de acordo com um padrão pré-definido.
- Evento (*Event*): tem a função de comunicar alterações no ambiente, mensagens enviadas por outros agentes ou mesmo mensagem enviada internamente. Para cada evento espera-se que o agente dispare uma ação ou um plano.
- Evento Interno (*InternalEvent*): especializada da entidade *Event*, a qual representa uma alteração interna no comportamento do agente.
- Evento Externo (*ExternalEvent*): especializada da entidade *Event*, representa os eventos externos disparados pelas mensagens aceitas.
- Crença (*Belief*): representa as crenças de um agente expressas em expectativas sobre o estado atual do mundo e sobre a probabilidade de um curso de ação atingir determinados efeitos.
- Termo (*Term*): constituído por uma expressão lógica que se refere a um objeto.

- Sentença (*Sentence*): uma sentença enuncia fatos, sendo representado por um símbolo de predicado seguido por uma lista de termos, podendo utilizar conectivos lógicos.
- Operador (*Operator*): representa os conectivos lógicos utilizados na relação entre uma sentença e uma crença.
- Regra (*Rule*): representa os tipos de sentença que devem necessariamente possuir crenças como antecedentes e conseqüentes, em que a primeira implica na segunda.

Os relacionamentos e restrições apresentados entre as entidades do meta-modelo são:

- *Agent has Resource*: um agente usa zero ou mais recursos de determinado tipo para auxiliar no alcance de seus objetivos. Um recurso é usado por um ou mais agentes.
- *Agent starts InternalEvent*: um agente dispara zero ou mais eventos internos. Estes eventos possuem um *Clock* que pode dispará-lo quando o tempo atribuído coincidir com o tempo atual do sistema ou mesmo sem nenhuma condição associada. Um evento interno é disparado por um agente.
- *Agent has Belief*: um agente contém zero ou mais crenças que armazenam seus conhecimentos. Uma crença está relacionada com zero ou mais agentes.
- *Agent has Perceptron*: um agente contém um ou mais *Perceptrons* que percebem e avaliam as mensagens recebidas do ambiente. Um *Perceptron* está relacionado a um agente.
- *Agent plays Role*: um agente exerce um ou mais papéis relacionados a sociedades. Um papel é exercido por um ou mais agentes.
- *Role aims Goal*: um papel almeja o alcance de um ou mais objetivos. Um objetivo é almejado por um papel.
- *Role must execute Action*: um papel deve executar zero ou mais ações. Uma ação deve ser executada por zero ou um papel.
- *Goal aggregates Goal*: um objetivo agrega zero ou mais subobjetivos. Um objetivo é agregado por zero ou um objetivo.

- *Plan achieves Goal*: um plano alcança um ou mais objetivos. Um objetivo é alcançado por um ou mais planos.
- *Perceptron starts ExternalEvent*: um *perceptron* dispara um evento externo. Um evento externo é disparado por um *perceptron*.
- *Perceptron evaluates Message*: um p*erceptron* avalia uma ou mais mensagens. Uma mensagem é avaliada por um *perceptron*.
- *InternalEvent extends Event*: um evento interno especializa um evento.
- *ExternalEvent extends Event*: um evento externo especializa um evento.
- *Event generates Belief*: um evento gera uma ou mais crenças. Uma crença é gerada por zero ou um evento.
- *Plan is composed by Action*: um plano é composto por uma ou mais ações. Uma ação compõe zero ou um plano.
- *Action generates Belief*: uma ação gera uma ou mais crenças, agindo como pós-condições. Uma crença é gerada por zero ou uma ação.
- *Belief controls Action*: uma crença regula zero ou mais ações, agindo como pré-condições. Uma ação é regulada por zero ou mais crenças.
- *Action publishes Message*: uma ação publica zero ou mais mensagens no ambiente. Uma mensagem é publicada por uma ação.
- *Action follows Action*: uma ação posterior sucede zero ou mais ações. Uma ação anterior precede zero ou mais ações.
- *Plan aggregates Plan*: um plano agrega zero ou mais subplanos. Um plano é agregado por zero ou um plano.
- *Belief controls Plan*: uma crença regula zero ou mais planos, agindo como pré-condição. Um plano é regulado por zero ou mais crenças.
- *Rule has antecedent Belief*: uma regra tem uma crença como antecedente. Uma crença é antecedente de zero ou mais regras.
- *Rule has consequent Belief*: uma regra tem uma crença como conseqüente. Uma crença é conseqüente de zero ou mais regra.
- *Sentence extends Belief*: uma sentença especializa uma crença.
- *Term extends Belief*: um termo especializa uma crença.

- *Sentence Operator Belief*: uma sentença agrega zero ou mais crenças com o uso de uma classe associativa *Operator*. Uma crença é agregada por zero ou mais sentenças com o uso de uma classe associativa *Operator*.
- *Rule extends Sentence*: uma regra especializa uma sentença.
- *Protocol aggregates Message*: um protocolo agrega zero ou mais mensagens. Uma mensagem é agregada por um protocolo.
- *Message is composed by Field*: uma mensagem é composta por um ou mais campos. Um campo compõe uma mensagem.
- *Message follows Message*: uma mensagem posterior sucede zero ou mais mensagens. Uma mensagem anterior precede zero ou mais mensagens.
- *Field aggregates Field*: um campo agrega zero ou mais subcampos. Um campo é agregado por zero ou um campo.

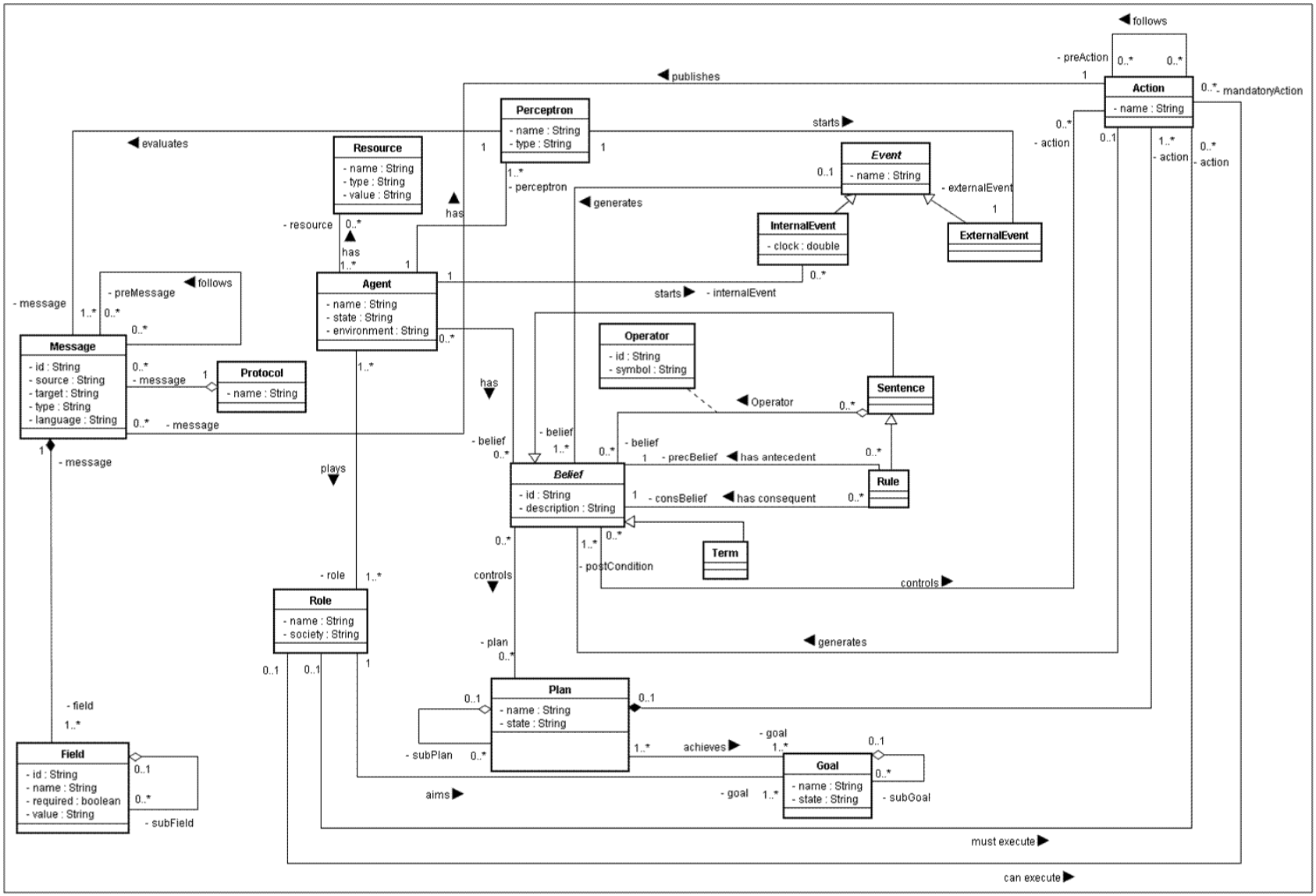


Figura 33- Meta-modelo de representação interna de uma agente [SAN08].

Estas entidades, relacionamentos e restrições formam a proposta de um meta-modelo de representação interna de um agente de software. Sua estrutura modelada está representada na figura 33.

Além do meta-modelo de representação interna de um agente, Santos apresenta um mapeamento deste meta-modelo para a plataforma de implementação *SemantiCore*, juntamente com um protótipo que aborda este mapeamento.

3.2 Meta-modelo unificado

O Meta-modelo Unificado [BER04a] [BER05] tem como objetivo a integração e interoperabilidade de abordagens metodológicas para desenvolvimento de SMAs. Esta abordagem envolve a definição de um *framework* para especificação de SMAs o qual inclui a identificação de um conjunto mínimo de conceitos e métodos.

Esta proposta procura unificar os meta-modelos de ADELFE [BER02], Ingenias, PASSI [COS05], RICA [SER05] e Tropos. É realizada uma compatibilização dos conceitos dessas abordagens através de comparações entre os meta-modelos, com objetivo de se chegar a um meta-modelo comum. Após esta compatibilização foi proposto um meta-modelo inicial que, após passar por um processo de refinamento, formou a proposta atual do Meta-modelo Unificado de Sistemas Multiagentes (*MMM – Multiagent Meta-Model*). Esta proposta abrange conceitos de agente, papel, tarefa, ambiente e organização, conforme ilustrado na figura 34. Estes conceitos são definidos, resumidamente, como:

- Agente: é uma entidade capaz de agir no ambiente onde está situado, é autônomo por controlar seu próprio comportamento baseado em estímulos interno ou externos, pode comunicar com outros agentes e é capaz de perceber seu ambiente.
- Papel: é uma abstração de uma porção de comportamentos sociais de um agente.
- Tarefa: especifica um conjunto de atividades que geram algum efeito.
- Ambiente: algo que o agente possa interagir e/ou perceber.
- Organização: pode ser definida como uma agregação de papéis, podendo também emergir através da interação entre agentes, sendo que este conceito não foi estabelecido durante a elaboração desta proposta, ficando determinado que é apenas composto por papéis [BER05].
- Comunicação: é uma propriedade essencial de um agente, a qual permite a interação de um agente para com outro.

- Conversação: é uma especialização da comunicação, a qual é definida como a comunicação entre agentes com um propósito pretendido.
- Representação: este conceito foi encontrado na metodologia Adelfe, e adicionado a esta proposta, onde um agente possui uma representação do mundo em termos de crença sobre outro agente, sobre si mesmo e sobre o ambiente onde está inserido. Esta representação determina o comportamento do agente, podendo ser compartilhada entre os agentes.
- Agente cognitivo: nesta pesquisa, o agente cognitivo é considerado como o próprio agente, sendo pró-ativo e usa uma representação de seu ambiente.

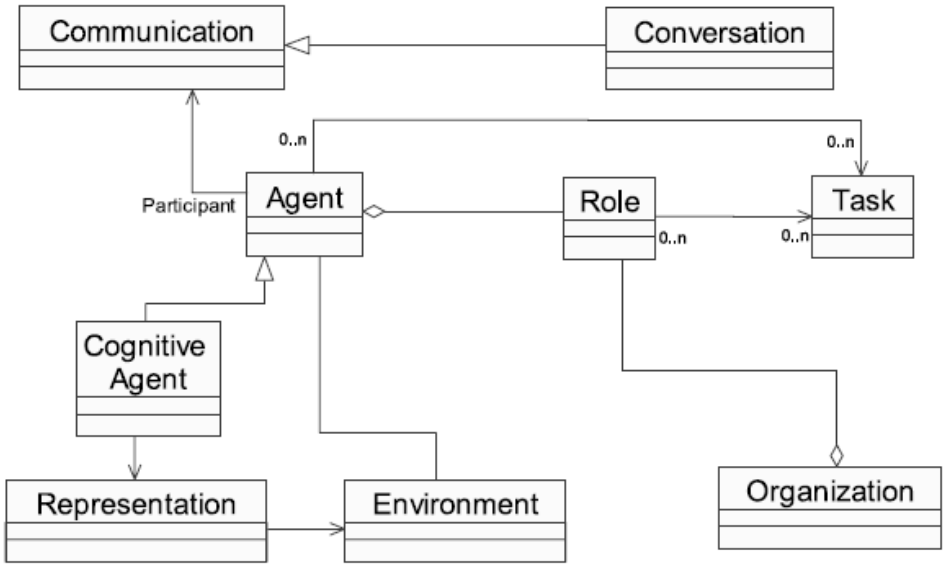


Figura 34 - Meta-modelo unificado [BER04a].

Esta proposta de meta-modelo está em andamento e será alvo de futuros refinamentos a partir da inclusão de outras metodologias.

3.3 Notação Unificada

A Notação Unificada [PAD08] é uma iniciativa que visa um nível de maturidade para modelagem de sistemas multiagentes similar ao da orientação a objetos. Para a formulação desta notação foram consideradas as metodologias GAIA [WOO00], O-MaSE [DEL05], Tropos, Prometheus e Passi.

Os desenvolvedores de cada metodologia abordada envolveram-se nesta proposta no sentido de produzir ‘uma notação comum, pois acreditam que compartilhando uma notação visual, toma-se o primeiro passo para fazer as metodologias orientadas a

agentes serem aplicáveis em consumo industrial. Esta notação será usada em cada uma das metodologias envolvidas e será integrada dentro de ferramentas que a suporte.

Os conceitos de cada metodologia foram identificados juntamente com seus símbolos para que a padronização fosse criada. Para confecção dos símbolos foram considerados os critérios propostos por Rumbaugh para desenvolvimento de notações visuais [RUM96]. De acordo com estes critérios a notação deve:

- Possuir um claro mapeamento dos conceitos para símbolos.
- Não apresentar sobrecarga de símbolos.
- Possuir mapeamento uniforme dos conceitos para símbolos.
- Ser fácil desenhar pelas mãos.
- Possuir boa aparência quando impressos.
- Usar imagem monocromática.
- Copiar bem ao enviar por fax e cópias usando imagens monocromáticas.
- Ser coerente com práticas passadas.
- Ser auto consistente.
- Possuir distinções não muito sutis.
- Possibilitar que casos comuns apareçam simples.
- Possuir detalhes suprimíveis.

A proposta sintetizou os seguintes conceitos e seus símbolos associados conforme a figura 35: Objetivo “leve” (*Softgoals*), Ator, Capacidade, Plano, Recurso, Serviço, Objetivo, Agente, Percepção, Evento, Posição, Organização, Papel, Mensagem, Conversação e Ação.

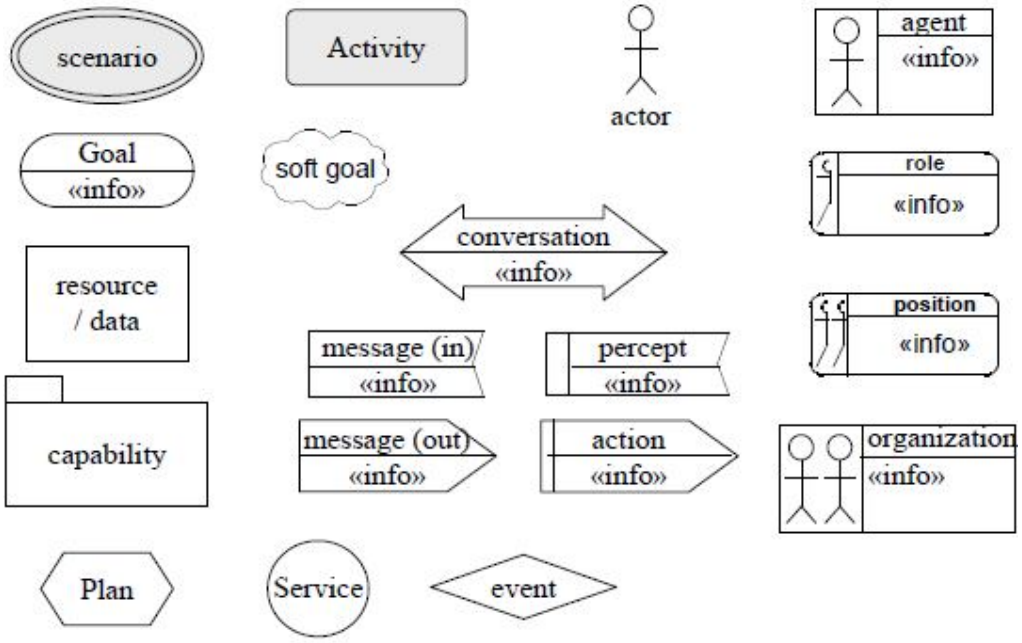


Figura 35 - Notação unificada proposta [PAD08].

Os conceitos adotados para estas entidades, resumidamente, são:

- O Objetivo leve (*Softgoal*) é utilizado para modelar requisitos não-funcionais.
- O Ator (*Actor*) é uma entidade externa que pode ser um ser humano ou um software.
- A Capacidade (*Capability*) é um módulo que contém planos, eventos, dados e sub-capacidades.
- O Plano (*Plan*) é o principal conceito da plataforma de agentes BDI, algumas vezes chamado de tarefa.
- O Recurso (*Resource*) representa informações e recursos externos a serem utilizados pelo agente, tais como um banco de dados, uma impressora, um objeto ou um conjunto de crenças.
- O Serviço (*Service*) foi adicionado nesta proposta dado o crescimento de SMAs baseados em serviço e por este conceito estar presente no PASSI, amparando assim este conceito nesta metodologia. No PASSI o conceito de serviço é um único bloco coerente de atividades no qual um agente irá desempenhar, sendo o serviço associado com o papel do agente [COS05].
- O Objetivo (*Goal*) representa a pró-atividade dos agentes.
- O Agente (*Agent*) representa uma entidade autônoma.

- A Percepção (*Percept*) percebe as informações que o agente recebe do ambiente.
- O Evento (*Event*) representa a reatividade de um agente a uma ocorrência significante.
- A Posição (*Position*) é um espaço reservado para um ou mais papéis em uma organização.
- A Organização (*Environment*) representa as formas de organizações particulares.
- O Papel (*Role*), a Mensagem (*Message*) e a conversação (*Conversation*) são utilizadas para representar os aspectos sociais do agente, assim como a posição e a organização.
- A Ação (*Action*) representa as ações executadas pelo agente, e afetam o ambiente onde o agente está situado.

Os relacionamentos entre estas entidades são representadas por links que podem apresentar rótulos de acordo com seu tipo. Exemplo: “<<*precede*>>” e “<< *inicia* >>”.

Padghan [PAD08] acredita que a remoção da incompatibilidade gratuita entre as notações encontradas nestas metodologias contribui positivamente para a visualização entre similaridades e diferenças de entidades e notações de cada abordagem, e permite a extensão para abrangência desta proposta. A pesquisa da Metodologia Unificada está em desenvolvimento em busca de maiores unificações.

3.4 Considerações

Neste capítulo, foi apresentado os trabalhos relacionados a proposta desta pesquisa. Dentre estes trabalhos, o Metamodelo de Representação Interna de um agente foi tomado como foco desta pesquisa para que seja tomado como base na construção desta proposta, incrementando e extendendo-o. A Notação Unificada para representação de SMAs e o Meta-Modelo Unificado para modelagem de SMAs, apresentam propostas que abordam a unificação de meta-modelos, mas não uma solução integrada que incorpore o mapeamento dos meta-modelos de diferentes metodologias e que seja capaz de gerar o esqueleto de código para uma plataforma de implementação, independente da metodologia ou da linguagem de modelagem utilizada.

No capítulo seguinte será apresentado o estudo para servir de base na consolidação da proposta de integração de soluções de SMAs, com uma compilação comparativa de meta-modelos e notações.

4 EM BUSCA DA INTEGRAÇÃO DE SOLUÇÕES SMAS

4.1 Estudo comparativo de meta-modelos e notações visuais

Diferentes meta-modelos e notações visuais são encontrados em soluções para o desenvolvimento de SMAs. Para compor a construção desta proposta é apresentada uma revisão destes meta-modelos, e suas respectivas notações, no intuito de identificar elementos fundamentais, seus conceitos e relacionamentos.

Além do MRIA, foram pesquisadas as linguagens de modelagem de Sistemas Multiagentes ANote e MAS-ML, as metodologias Ingenias, Prometheus, Tropos e MESSAGE. Os meta-modelos destas abordagens foram investigados no intuito de encontrar conceitos e entidades utilizados para composição de um meta-modelo que represente as características comuns e relevantes a todas as abordagens. Durante a pesquisa não foi possível encontrar o meta-modelo da metodologia Prometheus. Para levantamento de seus conceitos e relacionamentos, inspecionamos diferentes trabalhos sobre a metodologia e procuramos resolver as inconsistências apresentadas nestes trabalhos sobre o corpo conceitual da metodologia.

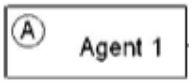
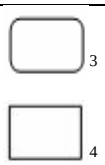
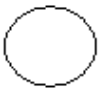
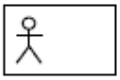
A comparação das entidades partiu das definições do MRIA, identificando os conceitos e relacionamentos que esse compartilha com as demais abordagens e outros conceitos e relacionamentos que estejam ausentes e que possam ser introduzidos no MRIA, ampliando a sua semântica. Vale lembrar que as abordagens trazem conceitos relacionados ao SMA como um todo e o foco deste trabalho foi o subconjunto dessas abordagens voltado à descrição do agente e suas partes internas. Os conceitos, notações e comparações entre as entidades encontradas na pesquisa, exceto do MRIA que não possui notação visual, são apresentadas nas subseções seguintes.

4.1.1 Agente

Durante esta pesquisa encontramos o uso dos termos ator e agente para representar um agente de software. No decorrer deste trabalho usaremos o termo agente.

Tabela 1 – Notação e conceito de Ação das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA, o agente é um sistema computacional inserido em um ambiente, capaz de atingir os	

	objetivos planejados por meio de ações autônomas nesse ambiente [SAN08].	
ANote	No ANote, o Agente é um modulo que interage com outro agente e desempenha as ações no sentido de alcançar um objetivo, e possui uma limitação de percepção do ambiente do sistema [CHO04].	
MAS-ML	No MAS-ML, os Agentes são elementos que definem um comportamento e um estado mental, o qual é formado pelas crenças, objetivos, planos e ações [SIL04]. Além do Agente, MAS-ML aborda o conceito de Objeto com características semelhantes a um agente, porém não autônomo.	 [3] [4]
Ingenias	No Ingenias, o Agente é uma entidade autônoma caracterizada por ter identidade única, propósitos, responsabilidades e capacidades [GOM02].	
MESSAGE	No MESSAGE, o Agente é uma entidade autônoma e atômica que pode executar papéis, prover serviços, executar tarefas, alcançar objetivos, usar recursos, ser parte de uma organização e participar em uma interação com outro agente [EVA01].	
Tropos	No Tropos um Ator representa um Agente, um Papel ou uma Posição. O Agente no Tropos é autônomo, reativo, pró-ativo e possui habilidade social [BRE04].	 [5]
Prometheus	No Prometheus, o Agente é um software que está situado em um ambiente, é autônomo, reativo, pró-ativo, flexível, robusto e social [PAD05].	

O conceito de Agente é apresentado em todas as abordagens pesquisadas, com exceção do Tropos que modela o Agente como um Ator. As demais abordagens apresentam definições similares, tornando assim o conceito de Agente passível de mapeamento entre estas abordagens.

A metodologia Tropos modela o agente como um Ator, por incorporar os conceitos do *framework* i*, sendo este ator uma entidade autônoma e social, similar aos conceitos

[3] Notação visual para o conceito de Agente do MAS-ML.
[4] Notação visual para o conceito de Objeto do MAS-ML.
[5] Notação para ator do Tropos

de Agentes nas demais abordagens. O Ator em Tropos pode representar, além de um Agente, um papel ou uma posição, onde o papel é uma caracterização abstrata do comportamento de um ator social dentro de um contexto específico e a posição representa um conjunto de papéis, tipicamente executados por um agente.

A linguagem de modelagem MAS-ML, além de apresentar o conceito de Agente, aborda também o Objeto com características similares ao Agente, porém sem autonomia. Segundo Silva [SIL04], o Objeto em MAS-ML é um elemento que possui informações sobre o ambiente, sobre si mesmo e sobre outros objetos, e pode executar um papel definido pela organização, podendo controlar seu estado mas não seu comportamento, não sendo autônomo por sempre depender da requisição de outro objeto.

As notações visuais para o Agente divergem entre as metodologias e linguagens de modelagem de SMA. O Ingenias e o Prometheus apresentam um símbolo com forma de bonecos que fazem alusão à forma humana, enquanto que Tropos utiliza um círculo e MESSAGE uma elipse. Já o ANote apresenta um retângulo com uma marcação “A” e o nome do Agente dentro desta forma geométrica, similar ao MAS-ML que apresenta um retângulo para o Objeto e o retângulo com bordas arredondadas para o Agente, diferenciando assim o Objeto do Agente.

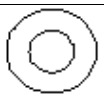
4.1.2 Objetivo

Durante a pesquisa encontramos as nomenclaturas Objetivos e “Objetivos Leves” (*SoftGoal*), sendo que esta pesquisa adota o termo Objetivo. A definição de Objetivo como um estado que um Agente deseja alcançar está presente na maioria das abordagens e será o conceito adotado nesta pesquisa.

Tabela 2 – Notação e conceito de objetivo das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
Tropos	No Tropos, o objetivo representa os interesses estratégicos dos atores. Nesta metodologia o objetivo é dividido em *Goal* para requisitos funcionais e *SoftGoal* para requisitos não funcionais [BRE04].	 [6]  [7]
ANote	No ANote, o objetivo é um conjunto de funcionalidades que um ou mais agentes devem executar para alcançá-lo. Nesta abordagem o Objetivo está estruturado em	

[6] Notação visual para o conceito de *SoftGoal* (requisitos não-funcionais) do Tropos.
[7] Notação visual para o conceito de *HardGoal* (requisitos funcionais) do Tropos.

	forma de uma árvore de objetivos e subobjetivos [CHO04].	
MAS-ML	Nestas abordagens, o objetivo é um estado que o agente deseja alcançar [GOM02] [MES08] [SAN08] [SIL04]. No MAS-ML este conceito é abordado como um atributo e modelado como estereótipo da entidade *Property* (da UML).	
Ingenias		
MESSAGE		
MRIA		
Prometheus	Em Prometheus, o objetivo implementa a pró-atividade de um agente, representando o seu propósito em existir [WIN04].	 [8]

O conceito de Objetivo é apresentado em todas as abordagens pesquisadas com conceitos semelhantes. O Tropos aborda, além do objetivo, o conceito de *SoftGoal* para modelar os requisitos não-funcionais de um SMA. Com o mesmo intuito o Ingenias apresenta o recurso para modelar os requisitos não-funcionais, porém esta pesquisa adotou apenas o conceito de objetivo. As definições apresentadas para o objetivo são equivalentes tornando o conceito de objetivo passível de mapeamento entre estas abordagens.

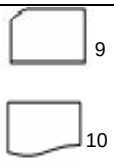
As notações visuais para este conceito coincidem entre o Tropos e o ANote, e divergem entre as demais abordagens. Tropos e ANote representam visualmente o objetivo com o símbolo de um retângulo com bordas arredondadas. Além disso, o Tropos apresenta uma nuvem para representar o *SoftGoal*, representação esta importada do *framework* i*. O Ingenias apresenta um círculo para a notação de objetivo. O MESSAGE apresenta um círculo com bordas duplas e o Prometheus apresenta uma elipse. Embora o Objetivo esteja presente entre os conceitos do MAS-ML, esta linguagem não apresenta um símbolo para este conceito.

4.1.3 Papel

Durante a pesquisa encontramos as nomenclaturas Papel, Funcionalidades e Cenário, sendo que esta pesquisa adota o termo Papel. A definição de Papel como representação abstrata de uma função ou comportamento de um agente é coerente com as abordagens estudadas e será o adotado nesta pesquisa.

[8] Notação visual para Objetivo do Prometheus apresentando por Winikoff e Padgham em [WIN04].

Tabela 3 – Notação e conceito de Papel das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA, o papel é uma representação abstrata de uma função de agente, serviço ou identificação dentro de um grupo, e pode ter associado a si um conjunto de atribuições e restrições [SAN08].	
ANote	O ANote não possui uma entidade explícita para representar o papel, ficando sob a responsabilidade do desenvolvedor definir a estrutura do agente, a partir de uma hierarquia de objetivos, utilizando as classes de agentes como papéis responsáveis por desempenhar as funções atribuídas. Além disso, o ANote apresenta o conceito de Cenário semelhante ao conceito de Papel em outras abordagens. Nesta linguagem de modelagem de SMA, o Cenário ilustra o comportamento de um agente com a execução de uma sequência de ações para o alcance de um objetivo em um determinado contexto (estado do sistema). Assim o cenário descreve o contexto no qual os agentes atuam [CHO04].	
MAS-ML	No MAS-ML o Papel atua como um guia e restringe o comportamento, podendo ser associado a um agente ou a um objeto, sendo representados pelas entidades *AgentRole* e *ObjectRole* respectivamente [SIL04].	 9 10
Ingenias	O papel no Ingenias é uma abstração de um conjunto de funções que possui estado e depende da entidade agente para desempenhá-lo [GOM02].	
MESSAGE	No MESSAGE, o conceito de Papel possibilita a separação lógica da própria identificação do agente, uma vez que um papel descreve as características externas de um agente em um contexto específico quando um agente o exerce [EVA01].	
Tropos	No Tropos, o Papel é representado junto com o	

[9] Notação visual para o conceito de Papel do Objeto do MAS-ML.
[10] Notação visual para o conceito de Papel do Agente do MAS-ML.

	conceito de Ator. Nesta metodologia o Papel é definido como uma caracterização abstrata do comportamento de um ator social dentro de um contexto específico [BRE04].	
Prometheus	O Prometheus apresenta o conceito de Funcionalidades semelhante ao conceito de Papel em outras abordagens. Nesta metodologia a Funcionalidade é a habilidade necessária para atingir os objetivos [PAD05].	 [11]

O conceito de Papel é apresentado em todas as abordagens pesquisadas com conceitos semelhantes. O MRIA aborda o Papel como uma representação abstrata do comportamento, identificador do agente dentro de um grupo e associado a atribuições e restrições. O ANote, embora aborde o conceito de Papel, não apresenta esta entidade na composição de seu meta-modelo. O MAS-ML possui o conceito papel tato associado a objeto quanto a agentes. As definições de papel nas abordagens pesquisadas são equivalentes, tornando este conceito passível de mapeamento entre estas abordagens.

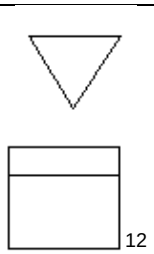
As notações visuais encontradas para este conceito divergem entre as abordagens. O MAS-ML apresenta um retângulo com um corte reto na borda superior esquerda para o papel do objeto, e um retângulo com a base com um corte curvilíneo para representar o Papel do Agente. O Ingenias apresenta um retângulo com um pequeno corte na parte superior na forma da base de um círculo como símbolo para o papel. O MESSAGE apresenta uma elipse vertical cortada próximo ao meio por uma linha curvilínea. O Prometheus apresenta um retângulo. O Tropos por apresentar o conceito de Ator como modelador de Papel, não possui uma notação visual especificamente para o Papel.

4.1.4 Recurso

Durante a pesquisa encontramos as nomenclaturas Recurso, Aplicação (Ingenias) e Dados (Prometheus), com características associadas a semântica de Recurso, mas utilizaremos o termo Recurso para representar esse conceito. A definição do Recurso como uma entidade não autônoma e/ou informacional a ser utilizada pelo Agente está coerente com todas as abordagens e será a usada neste trabalho.

[11] Notação visual para o conceito de Funcionalidade do Prometheus.

Tabela 4 – Notação e conceito de Recurso das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	O MRIA é o único que associa recurso ao agente. As demais abordagens relacionam recurso ao ambiente. Para o MRIA, o recurso representa uma entidade física ou uma informação [SAN08].	
ANote	O ANote e o MESSAGE modelam recurso como uma entidade não autônoma, como um banco de dados ou um programa externo usado pelo agente [CHO04] [CAI01].	
MESSAGE		
Ingenias	No Ingenias um recurso representa requisitos não funcionais. Esta metodologia apresenta a entidade Aplicação utilizada para modelar uma camada de aplicação para entidades de sistemas computacionais [GOM02], sendo este conceito similar ao conceito de Recurso de outras abordagens [GOM02].	 [12]
MAS-ML	O MAS-ML não apresenta este conceito.	
Tropos	No Tropos o Recurso representa uma entidade física ou informacional [BRE04].	
Prometheus	O Prometheus utiliza o conceito de Dados (*Data*), que além de representar as crenças de um agente, representa as informações disponíveis no ambiente a ser utilizada pelo agente, como no exemplo apresentado em [PAD05], na qual é utilizada a entidade *Data* para representar um Banco de Dados.	 [13]

O conceito de Recurso é apresentado no MRIA, no MESSAGE, no Ingenias, no Tropos e no Prometheus com conceitos semelhantes. As definições encontradas nas abordagens para o recurso são equivalentes entre si, tornando este conceito passível de mapeamento entre as abordagens que o apresentam.

Além das definições de recurso, o Ingenias aborda o conceito de Aplicação para modelar as aplicações computacionais que podem estar disponíveis no SMA, e o

[12] Notação visual para o conceito de Aplicação do Ingenias.
[13] Notação visual para o conceito de *Data/Belief* do Prometheus

Prometheus apresenta o conceito de Dados para representar um Recurso e também as Crenças de um Agente.

As notações visuais para este conceito coincidem entre o MESSAGE e o Prometheus e divergem entre as demais abordagens. O MESSAGE e o Prometheus representam visualmente o Recurso (Dados no Prometheus) com o símbolo de um cilindro, fazendo alusão à notação utilizada para bancos de dados. O Ingenias apresenta a notação de Recurso como um triângulo com a ponta para baixo e o Tropos como um retângulo. Além disso o Ingenias apresenta uma notação para a Aplicação como um quadrado com uma linha em seu interior na parte superior dividindo-o, notação esta semelhante ao diagrama de classes da UML. Embora o Recurso esteja presente entre os conceitos do ANote e do MAS-ML, estas linguagens não apresentam um símbolo para este conceito.

4.1.5 Evento

Durante a pesquisa encontramos a nomenclatura Evento, e este termo foi adotado nesta pesquisa. Os conceitos encontrados para Evento podem ser divididos em uma comunicação de alteração ocorrida no ambiente, ou na estrutura interna do agente, embora ambos impliquem em um comportamento do agente como resposta. Esta definição será a utilizada nesta pesquisa.

Tabela 5 – Notação e conceito de Evento das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA, o Evento é uma comunicação da alteração ocorrida no ambiente onde o agente está situado. Este meta-modelo possui entidades distintas para evento externo e interno, sendo externo disparado pelas mensagens aceitas e o interno representado pelas alterações internas no comportamento do agente [SAN08].	
Ingenias	No Ingenias, o Evento é uma comunicação da alteração ocorrida no ambiente onde o agente está situado [GOM02].	
MESSAGE	No MESSAGE, o Evento é tratado como um lançador de uma interação. Esta metodologia utiliza o conceito	

	comportamental da UML para definir o Evento [CAI01].	
ANote	No ANote, os Eventos, juntamente com as Ações, especificam um Cenário [CHO04]. Este conceito não está presente no meta-modelo desta linguagem de modelagem como uma entidade específica.	
MAS-ML	No MAS-ML o evento pode iniciar a execução de ações de um agente ou operações de um objeto de acordo com a percepção associada a este evento. Esta linguagem de modelagem de SMA utiliza a metaclasse Event da UML para representação de eventos [SIL04].	
Tropos	O Tropos aborda o conceito de Evento como um lançador de capacidades [BRE04].	
Prometheus	No Prometheus, o Evento pode ser percebido de um ambiente, de outro agente ou disparado por uma ocorrência interna [PAD05].	 [14]

O conceito de Evento é apresentado em todas as abordagens pesquisadas com conceitos semelhantes. Embora o evento esteja presente conceitualmente em todas as abordagens, o ANote e o Tropos não apresentam uma entidade Evento em seus meta-modelos. As definições apresentadas para o Evento nas demais metodologias são equivalentes tornando este conceito passível de mapeamento entre estas abordagens.

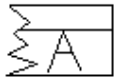
Apenas o Ingenias e o Prometheus possuem uma notação visual para o evento, e estas divergem entre si. O Ingenias apresenta um quadrado dividido em sua parte superior por uma linha e com seu lado esquerdo com fissuras, para representar o Evento, e o Prometheus o simboliza através de uma elipse.

4.1.6 Percepção

Durante a pesquisa encontramos somente a nomenclatura Percepção, sendo este termo adotado nesta pesquisa. A definição de Percepção apresentada no MRIA está coerente com as demais definições do MAS-ML, Ingenias e Prometheus, sendo esta adotada nesta pesquisa.

[14] Notação visual para o conceito de Evento em Prometheus, encontrado em [PAD02]

Tabela 6 – Notação e conceito de Percepção das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA a Percepção, representada pela entidade *Perceptron*, é responsável por perceber as mensagens vindas do ambiente para o agente de acordo com um padrão pré-definido [SAN08].	
ANote	O ANote não apresenta este conceito.	
MAS-ML	No MAS-ML a crença é responsável pela percepção do agente sobre as coisas que acontecem no SMA [SIL04].	
Ingenias	No Ingenias há dois tipos de percepção: a amostragem, que representa uma operação executada em uma frequência determinada, e a notificação, que representa uma ocorrência realizada no ambiente [GOM02].	 [15]
MESSAGE	O MESSAGE não apresenta este conceito.	
Tropos	O Tropos não apresenta este conceito.	
Prometheus	No Prometheus, a Percepção representa a interface do agente e provê informações do ambiente e das ações que o agente executa para alteração deste [PAD02].	

O conceito de Percepção é apresentado no MRIA, Ingenias, Prometheus e MAS-ML. No MAS-ML, a Percepção é realizada pela Crença não apresentando uma entidade para este conceito no seu meta-modelo. O ANote, o MESSAGE e o Tropos não apresentam este conceito em seus respectivos meta-modelos. As definições apresentadas para a Percepção nas demais abordagens são equivalentes, tornando este conceito passível de mapeamento entre si.

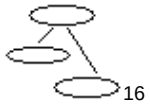
Somente o Prometheus e o Ingenias apresentam notação visual para este conceito. O Ingenias utiliza o mesmo símbolo do evento, porém adicionado a letra “A” no centro. O Prometheus utiliza uma estrela para representar a Percepção.

[15] Notação visual para o conceito de AplicationEvent.do Ingenias.

4.1.7 Plano

Esta seção aborda as definições para o Plano encontradas nas abordagens pesquisadas com suas respectivas notações. Durante a pesquisa encontramos as nomenclaturas Plano, Plano de Ação e Fluxo de Trabalho, sendo que esta pesquisa adota o termo Plano. A definição comum entre as abordagens para o Plano é uma sequência de ações para alcançar um objetivo, sendo este conceito adotado nesta pesquisa.

Tabela 7 – Notação e conceito de Plano das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA o Plano é formado por um conjunto de ações, com pré-condições, corpo e pós-condições associadas.	
ANote	No ANote, planos de ação são representados como diagramas de ações, não possuindo uma entidade que represente explicitamente o plano	
MAS-ML	No MAS-ML, o plano é uma sequência de ações executadas para alcançar um objetivo. Este conceito é modelado pela metaclasse PlanClass [SIL04].	
Ingenias	O Ingenias apresenta o conceito de Fluxo de Trabalho, semelhante ao conceito de Plano de outras abordagens. Nesta metodologia, o Fluxo de Trabalho permite contextualizar a execução de tarefas e relacioná-las independentemente de seus executadores [GOM02].	 16
MESSAGE	O Message não apresenta este conceito.	
Tropos	No Tropos o Plano representa o meio para satisfazer um objetivo, quando executado. [BRE04].	
Prometheus	O Prometheus na sua fase de Projeto Detalhado (*Detailed Design*) foca na estrutura interna de cada agente do SMA, trabalhando com agentes baseados em planos e que possuam uma biblioteca de planos definidos pelo usuário. Além disso, no Prometheus os planos otimizam o tempo de resposta aos eventos,	

[16] Notação visual para o conceito de Fluxo de Trabalho do Ingenias.

	através de uma biblioteca de planos estabelecida, que quando executados alcançam os objetivos do Agente [PAD02].	

O conceito de Plano é apresentado nas abordagens pesquisadas com definições semelhantes. O MRIA aborda o Plano como um conjunto de Ações com pré-condições, corpo e pós-condições associadas. O Ingenias apresenta o conceito de Fluxo de Trabalho para contextualizar as Tarefas no alcance de um Objetivo. O Prometheus apresenta uma biblioteca de planos para agilizar a resposta aos eventos. O ANote usa um diagrama de ações, não possuindo uma entidade para este conceito em seu meta-modelo. O MESSAGE é a única abordagem que não apresenta este conceito. As definições apresentadas para Plano no MRIA, MAS-ML, Ingenias, Tropos e Prometheus são equivalentes tornando este conceito passível de mapeamento entre estas abordagens.

Apenas Ingenias, Tropos e Prometheus apresentam notações visuais para este conceito, e as três divergem entre si. Ingenias apresenta três elipses ligadas por traços para representar o Fluxo de Trabalho. Tropos apresenta um losango para simbolizar o Plano, enquanto que o Prometheus utiliza um retângulo com bordas arredondadas.

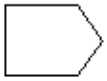
4.1.8 Ação

Durante a pesquisa encontramos as nomenclaturas Ação e Tarefa sendo que esta pesquisa adota o termo Ação. A definição de Ação como parte de um trabalho ou de um plano que causa mudança no estado do agente quando executada é coerente com as abordagens que utilizam este conceito, sendo esta definição adotada nesta pesquisa.

Tabela 8 – Notação e conceito de Ação das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA, a Ação é parte de um trabalho que pode ser atribuído a um agente.	
ANote	No ANote a ação é uma computação que resulta em alteração do estado do agente. Nesta linguagem a ação é dividida em dois tipos: ação direta e ação adaptativa, onde a primeira é usualmente executada por um agente enquanto participa de um cenário (ou contexto) para o	 [17]  [18]

[17] Notação visual para o conceito de Estado de ação do ANote
[18] Notação visual para o conceito de Estado de ação adaptativo do ANote

	alcance de um objetivo, e a segunda é executada quando o contexto requer a adaptação do agente por motivo de alguma funcionalidade [CHO04].	
MAS-ML	As ações compõem o plano e são modeladas como características do comportamento de um agente. Este conceito é representado pela entidade *AgentAction* [SIL04].	
Ingenias	O Ingenias apresenta o conceito de Tarefa, similar ao conceito de Ações nas outras abordagens. Nesta metodologia a execução de tarefas é um meio de satisfazer os objetivos [PAV03], e de representar uma atividade ou uma capacidade associada a um agente ou a um papel [GOM02].	 [19]
MESSAGE	No MESSAGE, a Tarefa é uma operação que pode causar mudança no estado do mundo do agente [CAI01].	 [20]
Tropos	Conforme apresentado no capítulo da base teórica, Tropos é uma metodologia estendida no *framework* i*. Neste *framework* é apresentado o conceito de Tarefa como uma forma particular de se fazer algo. Esse conceito, assim como a notação visual correspondente, foi absorvido por Tropos na forma de Plano. O conceito de Ação nas demais abordagens está presente em Tropos como parte da definição do Plano, e não como um entidade própria que compõe seu meta-modelo.	
Prometheus	No Prometheus, as Ações são executadas por agentes para efetuar uma mudança no ambiente [PAD05].	

O conceito de Ação é apresentado no MRIA, ANote, MAS-ML, Tropos e Prometheus. O Ingenias e o MESSAGE utilizam o termo Tarefa com conceitos semelhantes à ação de outras abordagens. O conceito de Tarefa é apresentado apenas por Ingenias e MESSAGE. No Ingenias uma tarefa é usada para representar uma atividade ou uma capacidade associada a um agente ou a um papel. Ela também é usada

[19] Notação visual para o conceito de Tarefa do Ingenias
[20] Notação visual para o conceito de Tarefa do MESSAGE

para expressar atividades realizadas em uma organização, atividades de gestão ou tomada de decisão. No MESSAGE uma tarefa é uma unidade de conhecimento. No ANote a Ação é dividida em Ação direta, quando se executa em curso normal, e Ação Adaptativa, quando uma ação toma um curso alternativo durante sua execução. O Tropos não possui uma entidade que represente a Ação em seu meta-modelo. As definições apresentadas para a Ação são equivalentes tornando este conceito passível de mapeamento entre estas abordagens.

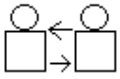
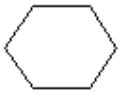
Apenas ANote, Ingenias, MESSAGE e Prometheus apresentam notações visuais para este conceito. Estas notações coincidem entre o Prometheus e o MESSAGE e divergem entre as demais abordagens. O ANote apresenta um retângulo com os lados curvilíneos, separando por um traço inclinado no canto superior esquerdo a ação direta da adaptativa. O Ingenias apresenta uma elipse para representar este conceito e MESSAGE e Prometheus apresentam um pentágono horizontal com um cumprimento maior e proporcional da linha da base e do topo.

4.1.9 Mensagem

Durante a pesquisa encontramos as nomenclaturas Mensagem e Interação, sendo que esta pesquisa adota o termo Mensagem. A definição de Mensagem como uma informação transportada de um agente para outro é coerente com as definições destas abordagens, sendo este conceito adotado nesta pesquisa.

Tabela 9 – Notação e conceito de mensagem das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA, o conceito mensagem representa as mensagens de entrada e saída dos agentes. Junto ao conceito de mensagem, este meta-modelo utiliza a entidade Campo utilizada para representar os parâmetros que compõem determinado tipo de mensagem [SAN08].	
ANote	No ANote a mensagem é uma informação transportada de um agente para outro assincronamente e possui protocolos [CHO04].	
MAS-ML	O MAS-ML apresenta o conceito de protocolo para definir o conjunto de mensagens que um agente está habilitado a enviar e receber de outro agente em uma	

	interação. Além disso, esta linguagem de modelagem de SMA estende a entidade *Message* da UML e cria a nova metaclasse *AgentMessage* para diferenciar as mensagens de agentes das mensagens de objetos [SIL04].	
Ingenias	O Ingenias apresenta o conceito de interação como a interatividade entre dois ou mais agentes ou papéis [GOM02], sendo este conceito similar ao conceito de Mensagem das outras abordagens. A interação no Ingenias é representado pela entidade *Iuiterate* e *Iuconcurrence* que representam a interação propriamente dita e a execução não determinística de um conjunto de unidades de interação.	 21
MESSAGE	O MESSAGE apresenta o conceito de Interação, semelhante ao conceito de Mensagem das outras abordagens. Nesta metodologia a Interação é o ato de troca de mensagens entre agentes, no sentido de alcançar um propósito comum [CAI01].	 22
Tropos	O Tropos apresenta o conceito de Dependência como um relacionamento onde um agente depende de outro para alcançar um objetivo, executar um plano ou prover ou consumir um recurso [BRE04]. Assim um Ator em Tropos comunica com o outro por estes propósitos, sendo este conceito semelhante ao conceito de mensagem apresentado nas outras abordagens.	
Prometheus	O Prometheus apresenta o conceito de Mensagem como parte de um protocolo de interação na comunicação de agentes [PAD05].	

O conceito de Mensagem é apresentado em todas as abordagens pesquisadas com conceitos semelhantes. O MAS-ML estende a classe Message da UML para criação de uma nova metaclasse para representar este conceito. O Ingenias e o MESSAGE utilizam a nomenclatura Interação para as trocas de mensagens entre os agentes. Tropos

21 Notação para o conceito de Interação do Ingenias
22 Notação para o conceito de Interação do MESSAGE.

não possui uma entidade que represente a mensagem em seu meta-modelo, porém este conceito está presente através do seu relacionamento de Dependência. As definições apresentadas nas abordagens para a Mensagem são equivalentes, tornando este conceito passível de mapeamento entre si.

Apenas Ingenias, MESSAGE e Prometheus apresentam notações visuais para este conceito, com símbolos que divergem entre si. O Ingenias apresenta o símbolo utilizado para uma notação visual com dois símbolos de agente interagindo com duas setas para representar a Interação entre agentes. O MESSAGE apresenta um hexágono para representar a Interação, e o Prometheus utiliza o símbolo de um envelope fazendo alusão a um correio, para representar a Mensagem.

4.1.10 Protocolo

Esta seção aborda as definições para o Protocolo encontradas nas abordagens pesquisadas com suas respectivas notações. Durante a pesquisa encontramos apenas a nomenclatura Protocolo. A definição para este conceito como o padrão de troca de mensagens associadas com uma interação entre agentes é coerente com as demais, sendo este conceito adotado nesta pesquisa.

Tabela 10 – Notação e conceito de Protocolo das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA a entidade *Protocol* representa o protocolo da mensagem a ser utilizada pelo agente [SAN08].	
ANote	O ANote define o protocolo junto com a mensagem, não possuindo uma entidade explícita para representar o protocolo.	
MAS-ML	No MAS-ML, o conceito de Protocolo é representado pela entidade *AgentProtocol*, a qual define um conjunto de mensagens que o agente está apto a enviar ou receber em uma interação [SIL04].	
Ingenias	O Ingenias não apresenta este conceito.	
MESSAGE	No MESSAGE o protocolo define um padrão de troca de mensagens associadas com uma interação.	
Tropos	O Tropos não apresenta este conceito.	
Prometheus	O Prometheus apresenta o Protocolo para a interação	

	entre agentes, sendo usado por mensagens na comunicação entre os agentes [PAD05].	

O conceito de Protocolo é apresentado no MRIA, ANote, MAS-ML, MESSAGE e Prometheus. Embora o ANote apresente este conceito, não há um entidade para o Protocolo no seu meta-modelo. O Tropos e o Ingenias não possuem uma entidade que represente o Protocolo em seus meta-modelos. As definições apresentadas para o Protocolo são equivalentes tornando este conceito passível de mapeamento entre estas abordagens.

Apenas Prometheus apresenta uma notação visual para este conceito, apresentando como símbolo de Protocolo uma seta com duas pontas, uma para a esquerda e uma para a direita, fazendo uma alusão à troca de mensagens dos Agentes.

4.1.11 Crença

Esta seção aborda as definições para a Crença encontradas nas abordagens pesquisadas com suas respectivas notações. Durante a pesquisa encontramos as nomenclaturas Crença, Estado Mental e Dados sendo que esta pesquisa adota o termo Crença. A definição de Crença como o conhecimento e expectativas de um agente sobre o estado atual do mundo é coerente com as demais, sendo este conceito adotado para a Crença nesta pesquisa.

Tabela 11 – Notação e conceito de Crença das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA a Crença representa as expectativas de um agente sobre o estado atual do mundo e sobre a probabilidade de um curso de ação atingir certos efeitos. Junto a este conceito, o MRIA, apresenta os conceitos de Termo, Sentença e Operador. O Termo é constituído por uma expressão lógica que se refere a um objeto. A Sentença enuncia fatos, sendo representado pelo símbolo de predicado seguido por uma lista de termos com conectivos lógicos. O Operador representa os conectivos lógicos utilizados na relação entre uma sentença e uma crença [SAN08].	
ANote	O ANote não apresenta este conceito.	

MAS-ML	No MAS-ML, a crença é o conhecimento de um agente sobre o ambiente, outro agente e de si mesmo, e suas percepções sobre tudo que acontece dentro do SMA [SIL04].	
Ingenias	No Ingenias a crença é um conjunto de afirmações que não são certezas, apenas expectativas. Além do conceito de Crença, esta metodologia apresenta o conceito de Entidade Mental com conceitos semelhante a Crença em outras abordagens. Nesta metodologia o estado mental é representado por várias entidades que abordam, entre outras coisas, as informações sobre o mundo, a determinação sobre o que deve ser requerido para cada entidade mental em determinados momentos, e as entidades utilizadas no processo de decisão [GOM02].	  [23]
MESSAGE	No MESSAGE a crença é uma entidade que faz parte do estado mental do agente sendo entendida como uma situação verdadeira [CAI01].	
Tropos	Em Tropos a crença representa o conhecimento de mundo do Ator [BRE04].	
Prometheus	No Prometheus, a Crença armazena informações sobre o estado do mundo do Agente. Esta metodologia utiliza a entidade *Data* para representar este conceito, além de utilizar esse mesmo símbolo para representar o Recurso. [PAD02].	

O conceito de Crença é apresentado em quase todas as abordagens, com exceção do ANote, com conceitos semelhantes. O Ingenias aborda além da Crença, o conceito de Entidade Mental para modelar, entre outras coisas, as informações sobre o mundo vista pelos Agentes, e o conceito de Fato como uma informação que o agente aceita como confiável. As demais definições apresentadas para este conceito são equivalentes tornando-a assim passível de mapeamento entre estas abordagens.

Dentre as abordagens pesquisadas, apenas Ingenias apresenta uma notação somente para Crença e apresenta uma nuvem como símbolo para este conceito. Além

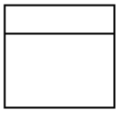
23 Notação visual para o conceito de Estado Mental do Ingenias

disso, esta metodologia apresenta também um círculo com a letra “M” no centro, para representar a Entidade Mental. O Prometheus utiliza o mesmo símbolo que representa o Recurso para representar a Crença. As demais abordagens não possuem uma notação visual para este conceito.

4.1.12 Regra

Esta seção aborda as definições para a Regra encontradas nas abordagens pesquisadas com suas respectivas notações. Durante a pesquisa encontramos as nomenclaturas Regra, Axioma e Fato sendo que esta pesquisa adota o termo Regra. A Regra é apresentada entre as abordagens, de uma forma geral, como controladora do comportamento de um agente, sendo este conceito adotado nesta pesquisa.

Tabela 12 – Notação e conceito de Regra das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	No MRIA a regra é vista como um tipo de sentença que necessariamente deve possuir crenças como antecedentes e conseqüentes, em que a primeira implica na segunda [SAN08].	
ANote	O ANote não apresenta este conceito.	
MAS-ML	O MAS-ML aborda o conceito de Axioma semelhante ao conceito de regras em outras abordagens. Nesta linguagem de modelagem de SMA, o Axioma é uma regra, principio ou lei estabelecida, a qual caracteriza as limitações globais de uma organização. Este conceito é apresentado como um atributo de um agente e modelado como estereótipos da entidade *Property* da UML [SIL04].	
Ingenias	O Ingenias apresenta o conceito de Fato, similar ao conceito de regras em outras abordagens. Nesta metodologia o Fato descreve uma informação que o Agente aceita como confiável [GOM02].	 [24]
MESSAGE	No MESSAGE, a Regra é representada por duas entidades (*BehaviorRule* e *InferenceRule*) usadas para	

[24] Notação visual para o conceito de Fato do Ingenias

	modelar as regras de comportamento e regras de inferências como uma declaração sobre o que um agente conclui de evidencias sobre o mundo [CAI01].	
Tropos	O Tropos não apresenta este conceito.	
Prometheus	O Prometheus não apresenta este conceito.	

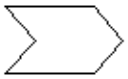
O conceito de Regra é apresentado no MRIA e no MESSAGE. O MAS-ML apresenta o conceito de Axioma semelhante ao conceito de Regra, assim como o Ingenias apresenta a entidade Fato. As demais abordagens não apresentam este conceito. As definições apresentadas para a Regra são equivalentes tornando este conceito passível de mapeamento entre estas abordagens.

Nenhuma abordagem apresenta uma notação visual para a Regra. O Ingenias apresenta a mesma notação de Recurso como notação para o Fato.

4.1.13 Capacidade

Esta seção aborda as definições para a Capacidade encontradas nas abordagens pesquisadas com suas respectivas notações. Durante a pesquisa encontramos duas nomenclaturas Capacidade e Serviço sendo que esta pesquisa adota o termo Capacidade. A definição de Capacidade é semelhante em MESSAGE (que utiliza o termo Serviço) e Prometheus, os quais a definem como a funcionalidade que o agente é capaz de desempenhar. Estas definições divergem da definição de Tropos, que apresenta a Capacidade como a habilidade do agente em escolher um plano para determinado objetivo. Esta pesquisa adota os dois conceitos para a Capacidade.

Tabela 13 – Notação e conceito de Capacidade das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	O MRIA não apresenta este conceito.	
ANote	O ANote não apresenta este conceito.	
MAS-ML	O MAS-ML não apresenta este conceito.	
Ingenias	O Ingenias não apresenta este conceito.	
MESSAGE	O MESSAGE apresenta o conceito de Serviço semelhante ao conceito de Capacidade em outras abordagens. No MESSAGE o serviço é o nível de	 [25]

[25] Notação visual para o conceito de Serviço do MESSAGE.

	conhecimento análogo a operações de objeto, representando a capacidade funcional de um agente [CAI01].	
Tropos	O Tropos apresenta o conceito de Capacidade como a habilidade de um ator de definir, escolher e executar um plano para alcançar um objetivo, dada certas condições do mundo e na presença de um evento específico. Embora não apresente uma entidade capacidade em seu meta-modelo, o Tropos utiliza o diagrama de atividades para modelar este conceito em sua metodologia [BRE04]. Na ferramenta TAOM4E, após serem modeladas as fases de Tropos é gerada uma tabela de capacidade com os Planos que alcançam determinados objetivos.	
Prometheus	No Prometheus, Capacidade é um módulo que pode conter planos, dados e eventos, além de outras capacidades em uma estrutura hierárquica. Cada agente possui capacidades para cada funcionalidade adquirida. [PAD02].	

O conceito de Capacidade é apresentado no MESSAGE, no Tropos e no Prometheus. As demais abordagens não apresentam este conceito. As definições apresentadas em MESSAGE e Prometheus são equivalentes tornando este conceito passível de mapeamento entre estas abordagens. A definição apresentada em Tropos complementa a definições de MESSAGE e Prometheus, e ao estender este conceito, permite o mapeamento entre estas três abordagens.

Apenas MESSAGE e Prometheus apresentam notações visuais para este conceito, e estes símbolos divergem entre si. Para representar este conceito, MESSAGE apresenta um hexágono com a lateral direita para dentro da figura e Prometheus apresenta um retângulo com bordas arredondadas.

4.1.14 Compromisso

Esta seção aborda a definição para o Compromisso encontrado na metodologia Ingenias, com sua respectiva notação.

Tabela 14 – Notação e conceito de Compromisso.

Abordagem	***Definição e observações***	***Notação***
MRIA	O MRIA não apresenta este conceito.	
ANote	O ANote não apresenta este conceito.	
MAS-ML	O MAS-ML não apresenta este conceito.	
Ingenias	O Compromisso do Ingenias expressa o dever de um agente em executar uma tarefa devido a um pedido feito por outro agente [GOM02].	 [26]
MESSAGE	O MESSAGE não apresenta este conceito.	
Tropos	O Tropos apresenta o relacionamento de Dependência com conceito similar ao Compromisso da metodologia Ingenias. No Tropos, a dependência é um relacionamento entre dois atores no qual um ator depende do outro por alguma razão, seja para alcançar um objetivo, executar um plano, ou entregar um recurso [BRE04].	
Prometheus	O Prometheus não apresenta este conceito.	

Apenas a metodologia Ingenias apresenta o conceito de Compromisso. Este conceito é correlato ao conceito de Dependência do Tropos, e ambos podem ser mapeados para troca de mensagem e solicitação de execução de um plano, alcance de um objetivo, fornecimento de um recurso, etc.

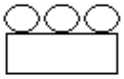
A notação do Ingenias para o Compromisso são dois círculos formando uma intersecção, fazendo alusão ao enlace de dois anéis, simbolizando o compromisso destes dois agentes.

[26] Notação visual para o conceito de Compromisso do Ingenias

4.1.15 Organização

Esta seção aborda as definições para a Organização encontradas nas abordagens pesquisadas com suas respectivas notações. A definição da Organização como um grupo de agentes com um propósito comum é coerente com as demais, sendo este conceito adotado nesta pesquisa.

Tabela 15 – Notação e conceito de Organização das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	O MRIA não possui este conceito	
ANote	No ANote a organização é uma entidade virtual que provê serviços [CHO04], não apresentando uma entidade computacional correspondente.	
MAS-ML	No MAS-ML, a organização é um grupo de agentes em um SMA que define suborganizações e um conjunto de leis (axiomas) que os agentes desta organização devem obedecer [SIL04].	
Ingenias	No Ingenias uma organização é um conjunto de agentes, papéis e recursos que estão juntos para alcançar um ou vários objetivos [GOM02].	
MESSAGE	No MESSAGE, a organização constitui-se de um grupo de agentes trabalhando juntos com um propósito em comum, podendo prover e utilizar serviços, alcançar objetivos, ser parte de outra organização, possuir grupos subordinados e participar de interações [CAI01].	
Tropos	O Tropos não apresenta este conceito.	
Prometheus	O Prometheus não apresenta este conceito.	

O conceito de Organização é apresentado no ANote, MAS-ML, Ingenias e MESSAGE.Os conceitos abordados para a Organização nestas quatros abordagens são correlatos e passíveis de mapeamento. O MRIA, o Tropos e o Prometheus não apresentam este conceito.

O ANote, o MAS-ML, o Ingenias e o MESSAGE apresentam notações visuais para este conceito que divergem entre si. O ANote apresenta como símbolo um cubo. MAS-ML

apresenta um retângulo com a linha da base e do topo curvilíneo. O Ingenias apresenta um retângulo com três círculos em cima, fazendo alusão a vários agentes juntos, e o MESSAGE apresenta um triângulo apontado para cima.

4.1.16 Ambiente

Esta seção aborda os conceitos apresentados relacionados ao Ambiente, encontrados nas abordagens pesquisadas. A definição de Ambiente como o local (ou *habitat*) onde o agente está situado e percebido por este agente é coerente com as abordagens, sendo este conceito adotado para o Objetivo nesta pesquisa.

Tabela 16 – Notação e conceito de Ambiente das abordagens pesquisadas.

Abordagem	***Definição e observações***	***Notação***
MRIA	O MRIA não possui este conceito.	
ANote	No ANote o agente está situado em um Ambiente de um SMA, e este ambiente pode ser especificado por um ontologia [CHO04].	
MAS-ML	No MAS-ML, o conceito de ambiente é estendido do TAO, e é modelado como o habitat de agentes e objetivos. Este conceito dividi-se em ambiente ativo ou passivo, onde no primeiro habitam agentes e no segundo objetos [SIL04].	
Ingenias	No Ingenias, o ambiente é definido pela percepção e atuação dos agentes. Esta metodologia apresenta uma visão de ambiente para identificar os recursos e aplicações disponíveis no ambiente com as quais um agente pode interagir [GOM02].	
MESSAGE	Segundo Caire [CAI01], o Agente no MESSAGE está situado em um ambiente, porém esta metodologia não apresenta uma entidade no seu meta-modelo para modelar o ambiente.	
Tropos	O Tropos não apresenta este conceito em seu meta-modelo.	
Prometheus	No Prometheus, o Ambiente é o local onde os Agentes estão situados [PAD05].	

O conceito de Ambiente é apresentado no ANote, MAS-ML, Ingenias, MESSAGE e Prometheus e são passíveis de mapeamento. O MRIA e o Tropos não apresentam este conceito.

Nenhuma abordagem apresenta notação visual para representar o Ambiente.

4.2 Considerações

Neste capítulo, foram apresentadas as definições e notações visuais para os conceitos encontrados nas metodologias e linguagens de modelagens de SMA pesquisadas. Foram abordados os conceito de Agente, Objetivo, Papel, Recurso, Evento, Percepção, Plano, Ação, Mensagem, Protocolo, Crença, Regra, Capacidade, Compromisso, Organização e Ambiente e suas respectivas notações nas Metodologias Ingenias, MESSAGE, Tropos e Prometheus e linguagens de modelagens de SMA ANote, MAS-ML, além do MRIA.

Estas comparações foram tabeladas, e seus conceitos foram compatibilizados a fim de permitir um mapeamento do conceito entre estas abordagens. Também foram encontradas e demonstradas as notações que são equivalentes e divergentes.

Puderam-se perceber as diferentes nomenclaturas utilizadas para conceitos similares, Sendo que um termo representativo para os conceitos foi selecionado. Além disso, algumas definições se completam, havendo pouca divergência entre os conceitos, permitindo o mapeamento entre eles sem que haja incoerência ou perda em seus significados.

As notações são extremamente divergentes entre as abordagens. Alguns símbolos são iguais para conceitos diferentes, e outros não fazem referência ao conceito que está sendo representado. Levando em consideração os princípios de Rumbaugh [RUM96], muitas dessas notações não se enquadram nestes requisitos, e percebe-se uma não-padronização entre as simbologias das abordagens pesquisadas.

5 MAS META-MODEL INTERCHANGE

Este capítulo propõe uma extensão ao MRIA para composição do MMI. São abordadas as adições de novos conceitos e alterações nas entidades e relacionamentos do meta-modelo, e, por fim, são apresentadas as considerações finais sobre a extensão proposta.

5.1 A proposta de extensão do MRIA

Esta proposta estende o MRIA para suportar o meta-modelo do Tropos com objetivo de demonstrar a cobertura aos modelos derivados das metodologias e linguagens de modelagem pesquisadas neste trabalho. Os conceitos de cada entidade e relacionamento que compõem o MRIA estão elencados e definidos na seção 3.1 do capítulo de Trabalhos Relacionados.

Como passo inicial, foi utilizada a metodologia Tropos para demonstrar como um meta-modelo pode oferecer um intercâmbio entre os diferentes modelos, a fim de mapear as entidades relevantes das metodologias de acordo com a base teórica de sistemas multiagentes.

Durante o processo de extensão do meta-modelo, a metodologia Tropos foi analisada em profundidade e a ausência de conceitos que estão presentes na literatura indicam lacunas conceituais dessa metodologia. Conseqüentemente, este meta-modelo contribui para a metodologia ou linguagem de modelagem que está sendo coberta, demonstrando a ausência de conceitos não abordados, mas presentes na literatura.

A extensão realizada para cobertura dos modelos do Tropos resultou na criação do *MMI, e* em um processo de mapeamento desta metodologia para este novo meta-modelo e posterior análise de consistência e geração de código.

O processo de mapeamento de entidades e relacionamentos do Tropos para o meta-modelo proposto considera os meta-modelos apresentados em [PER04] [BRE02], e os conceitos presentes na ferramenta TAOM4E, que é a ferramenta que dá suporte a modelagem usando a metodologia Tropos. Esta ferramenta permite ao desenvolvedor modelar o sistema multiagentes de acordo com a metodologia Tropos abordando as suas três fases especificadas na seção 3.2.3 deste trabalho.

A ferramenta TAOM4E gera um arquivo XML contendo o modelo de aplicação Tropos. Este arquivo é utilizado no protótipo proposto neste trabalho para demonstrar o

processo de mapeamento entre um modelo de Tropos e o MMI. Embora os meta-modelos de Tropos apresentem entidades que foram suprimidas no TAOM4E, foram considerados os relacionamentos e as entidades presentes neste software, pois o uso do mesmo esclarece algumas questões inconsistentes derivadas das explicações presentes nos artigos estudados sobre esta metodologia.

A seguir são apresentados os conceitos da metodologia Tropos e seu mapeamento para o MMI.

5.2 Mapeamento dos conceitos de Tropos

Durante o mapeamentos forma compatibilizados os conceitos de Tropos para a construção do MMI. Foram mapeados os conceitos de ator, objetivo, plano, recurso, além dos relacionamentos de dependência, decomposição e contribuição. Estes conceitos estão sumarizados na tabela 17. Vale lembrar que para a composição deste meta-modelo partiu-se do MRIA e assim, alguns dos seus conceitos também foram adequados neste mapeamento.

Tabela 17 – Quadro sumários dos conceitos mapeados.

Conceito	Definição	Mapeamento
Ator	Um ator no Tropos pode ser um papel, uma posição ou um agente.	Na ferramenta TAOM4E o ator em Tropos é mapeado diretamente para agentes, sendo este mapeamento adotado por este trabalho. Assim, o ator de Tropos é mapeado para agente do MMI.
Objetivo	O objetivo em Tropos é representado por *Hardgoal* e *Softgoal* sendo o primeiro equivalente ao conceito de objetivo apresentado na seção 5.4 e o segundo utilizado para representar requisitos não-funcionais	Ambas as entidades foram mapeadas para o conceito de objetivo, atribuindo ao desenvolvedor a tarefa de tornar o *softgoal* computável.
Papel	Um ator em Tropos é um agente que pode ocupar um papel.	O conceito de papel de Tropos foi mapeado para a entidade papel existente no MMI, embora a

		ferramenta TAOM4E mapeie o ator diretamente para agentes. Além disso, o atributo sociedade da entidade papel do MRIA é agora representado pelo relacionamento com a entidade organização, adicionada na extensão deste meta-modelo.
Recurso	O recurso em Tropos é uma entidade que participa dos relacionamentos de dependência no qual um ator depende de outro para obtenção de um recurso, e de um relacionamento de meios-fins no qual o recurso é um meio para um ator alcançar um objetivo	Esta entidade foi mapeada para entidade recurso do MMI.
Plano	O plano em Tropos é uma forma de se fazer algo	Esta entidade foi mapeada para entidade plano do MMI.

No Tropos há o conceito de subobjetivos através do relacionamento de decomposição. No processo de mapeamento, este conceito foi mapeado para as especializações do objetivo que são o objetivo simples (*SimpleGoal*) e o objetivo composto (*ComposedGoal*), sendo este último formado por agregações de objetivos, conforme ilustrado na figura 36.

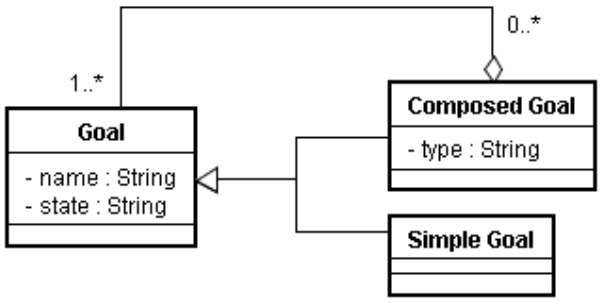


Figura 36 - Mapeamento da decomposição de objetivos.

O plano em Tropos participa do relacionamento de dependência no qual um ator pode depender de outro para realizar um plano, e do relacionamento de meios-fins, onde este plano é um meio para alcançar um objetivo. O Tropos possui o conceito de

subplanos em um relacionamento de decomposição, o qual foi mapeado através da entidade plano e de um relacionamento de especialização em planos simples ou compostos. Os planos compostos são formados pela agregação de planos compostos ou simples, conforme ilustrado na figura 37.

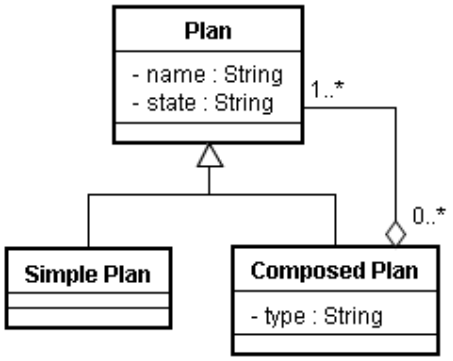


Figura 37 - Conceito de plano do MMI.

Em Tropos um ator pode ser uma posição. Nesta metodologia uma posição é um conjunto de papéis a ser ocupado por um agente. A idéia de uma posição nos remete ao conceito de organização, que é formada por agentes que podem executar papéis e são vinculadas a esta organização de tal forma que o papel pertence a uma organização e o conjunto de vários papéis forma uma posição. O agente pode alcançar um objetivo através de três formas (figura 38).

- Um agente pode alcançar um objetivo diretamente. Este relacionamento foi adicionado ao meta-modelo, pois um agente pode ter um objetivo inerente a si, independente da execução de um papel ou da ocupação de uma posição.
- Um agente pode executar um papel, e este papel alcança um objetivo. O papel é uma função que pode ser incorporada ao agente, e está vinculada a uma organização. Logo ao exercer este papel o agente incorpora este objetivo e almeja alcançá-lo.
- Um agente pode ocupar uma posição. Esta posição é composta por vários papéis e cada papel possui seu objetivo. Desta forma o agente alcança um objetivo através do papel coberto pela posição ocupada pelo agente.

Uma organização alcança um objetivo e é composta por vários papéis executados por agentes diretamente ou através de uma posição. O alcance dos objetivos de cada papel que compõe uma organização contribui para o alcance do objetivo da organização. Embora o conceito de organização não esteja presente em Tropos, esta entidade foi

adicionada ao MMI para que ele possa mapear alguns conceitos relacionados ao sistema multiagentes e não apenas a estrutura interna de cada agente. O conceito de organização esta presente na linguagem de modelagem ANote, a qual apresenta a organização como uma entidade virtual para representar um grupo de agentes trabalhando juntos para prover um serviço.

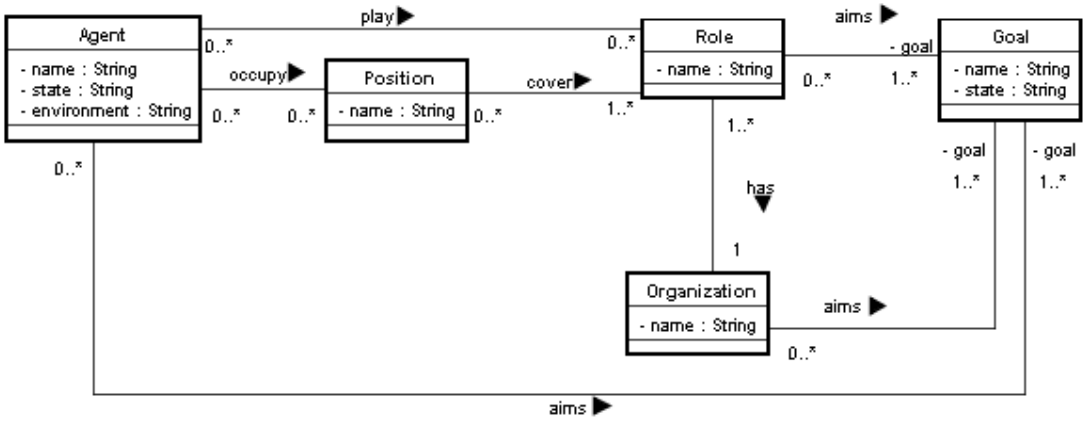


Figura 38 - Formas do agente alcançar um objetivo no MMI.

O relacionamento de dependência de Tropos foi mapeado através de troca de mensagens entre agentes. No MMI, assim como no MRIA, a comunicação entre agentes se dá através das mensagens disparadas pelas ações que compõem um plano. A mensagem esta relacionada à entidade campo e ao protocolo, com os quais é possível especificar o objeto de dependência de um agente para com o outro. As mensagens recebidas pelo agente são percebidas através da entidade *perceptron*, que podem iniciar eventos e estes eventos podem provocar o envio de mensagens de retorno.

O relacionamento de contribuição de Tropos indica que um plano, recurso ou objetivo contribuem para um objetivo de forma positiva ou negativa, adotando como métrica a contribuição positiva parcial (+), a contribuição positiva suficiente (++), a contribuição negativa parcial (-) e a contribuição negativa suficiente (--). A contribuição de Tropos foi mapeada para a entidade crença com suas especializações termos, sentença, e a classe de relacionamento operador para armazenar estas contribuições. O desenvolvedor pode estabelecer uma métrica apropriada para o uso destas informações.

O relacionamento de decomposição utilizado para objetivos e planos em Tropos foi mapeado para o MMI como objetivos ou planos simples e objetivos ou planos compostos. Conforme explicado anteriormente, o operador booleano AND ou OR mapeado do relacionamento de decomposição de Tropos é representado através do atributo tipo das respectivas entidades.

O relacionamento de meios-fins de Tropos indica que um objetivo, plano ou um recurso é um meio para se satisfazer um objetivo. O objetivo é um meio para se alcançar outro objetivo através das entidades *SimpleGoal* ou *HardGoal*; um plano é um meio para alcançar um objetivo através do relacionamento *plan achieve goal;* e um recurso é um meio para se alcançar um objetivo através do relacionamento *plan has resource* e *plan achieve goal*, onde o plano pode utilizar ou prover este recurso ao ser executado.

5.3 Novos conceitos e alterações adicionados

A extensão ao MRIA implicou em alteração de algumas entidades e seus atributos. Estas alterações, juntamente com os novos conceitos e relacionamentos adicionados formam o meta-modelo proposto nesta pesquisa.

O objetivo do MRIA foi mapeado para as especializações objetivo simples ou composto, a qual agrega outros objetivos, e em ambas foi adicionado o atributo booleano *type* para mapear o relacionamento de decomposição de Tropos. Assim como a entidade objetivo, o plano possui as novas especializações plano simples e plano composto. Para mapear a decomposição foi adicionado o atributo *type* de valor booleano. O relacionamento *Agent has Resource* foi movido para *Plan has Resource*, uma vez que recursos são utilizados ou consumidos por planos.

O relacionamento do MRIA que representa o alcance do objetivo por um agente foi estendido por novos conceitos. No MRIA, o agente pode alcançar um objetivo somente através de um papel. Nesta extensão proposta, o agente pode alcançar um objetivo que compete diretamente a ele, não necessitando de executar um papel. O conceito de posição foi adicionado ao MRIA, e esta posição cobre um ou mais papéis. Assim, um agente pode ocupar uma posição, que agrega papéis e estes alcançam objetivos.

O atributo *society* foi removido da entidade papel, e substituído por um relacionamento com a entidade organização. Esta entidade representa um novo conceito adicionado ao meta-modelo, referente ao sistema multiagentes e não a estrutura interna de um agente.

O conceito posição do Tropos foi adicionado ao meta-modelo. Este conceito foi mapeado para a nova entidade *position* e relacionada à entidade papel, formando o relacionamento *position cover role*. Além disso, foi adicionada a entidade *organization* e vinculado ao papel, pois um papel pertence a uma organização e esta organização possui um objetivo. Esta entidade está presente em ANote, Ingenias, Prometheus e MAS-ML O meta-modelo proposto é apresentado na figura 39.

A cada compilação de uma metodlogia que ainda não está mapeada, é realizada uma extensão e/ou alteração no MMI para que a suporte, bem como a verificação de consistências dos modelos gerados a partir deste meta-modelo. O MMI é um meta-modelo que será adaptado e enriquecido a cada nova metodologia suportada, absorverndo os conceitos da metodologia que estão presententes na literatura de agentes e sistemas muiltiagentes. Estes novos conceitos possibilitam o suporte de novas metodologias afim de manter um kernel de conceitos que possa suportar modelos de diferentes metodologias para sejam passíveis de mapeamento para código de diferentes plataformas de implementação. Vale ressaltar que a cada nova adoação de metodologia o MMI se altera e consequentemente deve ser realizado um novo rol de regras OCL para verificação de consistência.

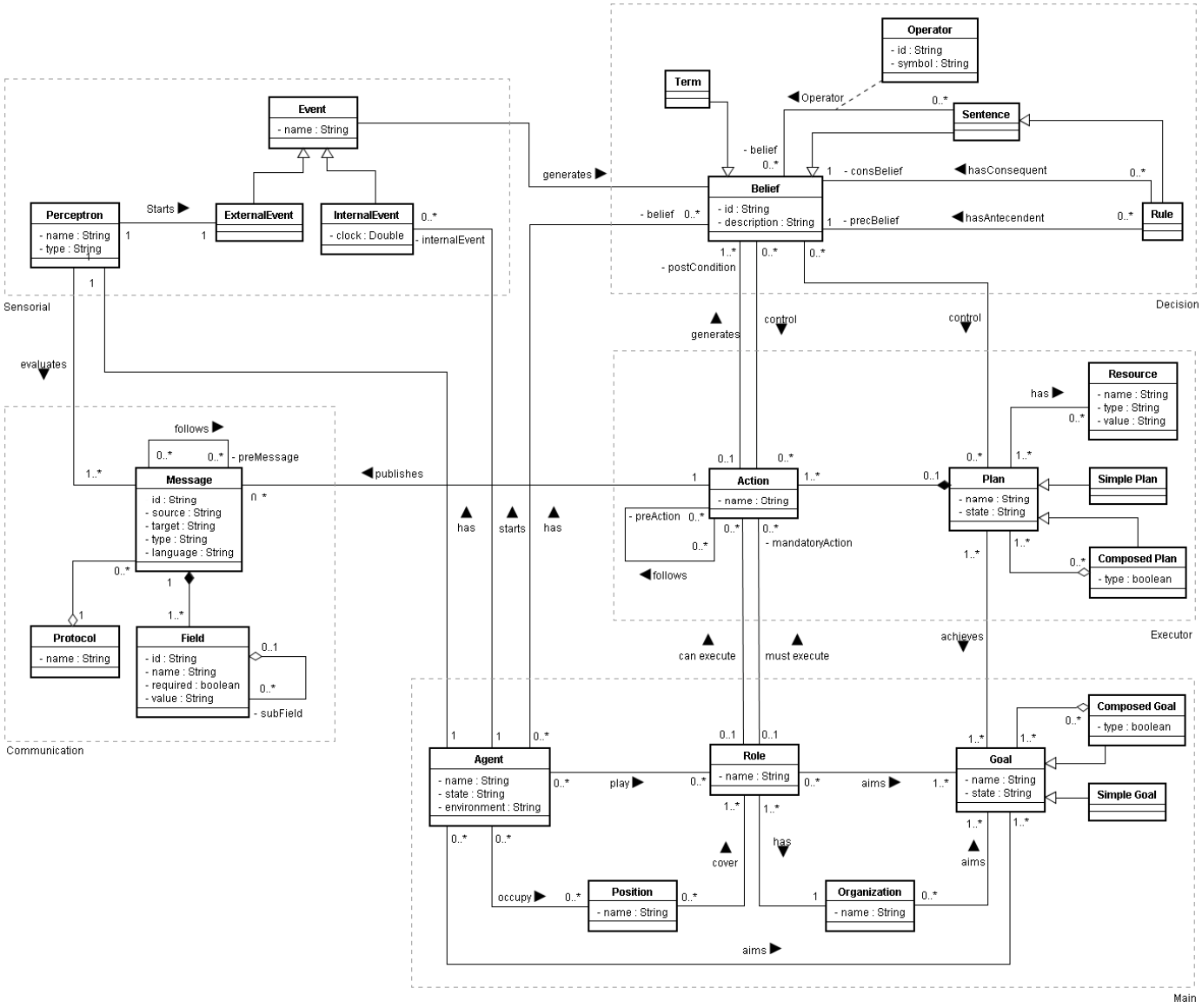


Figura 39 – Meta-modelo estendido proposto.

5.4 Conceitos do MMI

A proposta é estruturada em pacotes para facilitar a compreensão. Sendo assim, na Figura 40 é apresentada a visão geral dos pacotes do meta-modelo estendido. Esta seção apresenta também as restrições de integridades escrita na linguagem *OCL* [OCL08], pois segundo [WAR03] o uso desta linguagem permite uma restrição não-ambígua e tornam um modelo mais preciso e mais detalhado, podendo ser verificadas por ferramentas de automação para garantir que estão corretas e consistentes com outros elementos do modelo. Devemos considerar que todas as restrições aplicadas ao MMI consideram os atributos como do tipo *String*, pois esse é o único tipo de dado que será usado na classificação dos atributos de cada conceito no protótipo, com o objetivo de evitar constantes conversões de tipos de dados no mesmo.

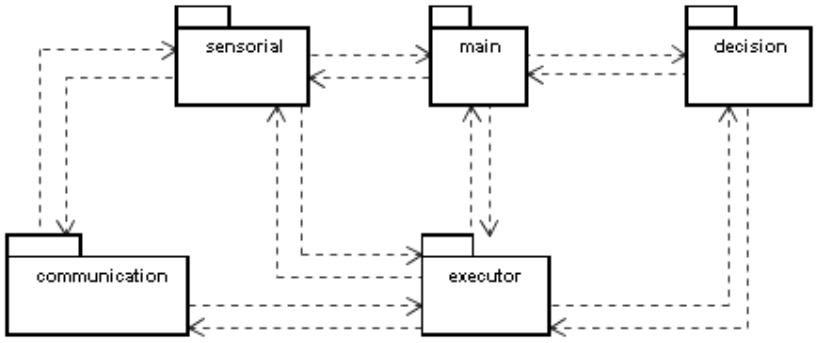


Figura 40 – Visão Geral dos Pacotes do Meta-modelo estendido.

Nas figuras 41, 42, 43, 44 e 45 são apresentados os diferentes pacotes que compõem o meta-modelo, sendo eles: Pacote *Main*, Pacote *Sensorial*, Pacote *Executor*, Pacote *Decision* e Pacote *Communication*. Estes pacotes foram importados do MRIA não havendo necessidade de alteração durante o mapeamento. Após a apresentação visual, são detalhados os atributos de cada pacote, os relacionamentos entre conceitos e as restrições de integridade aplicadas utilizando a linguagem OCL.

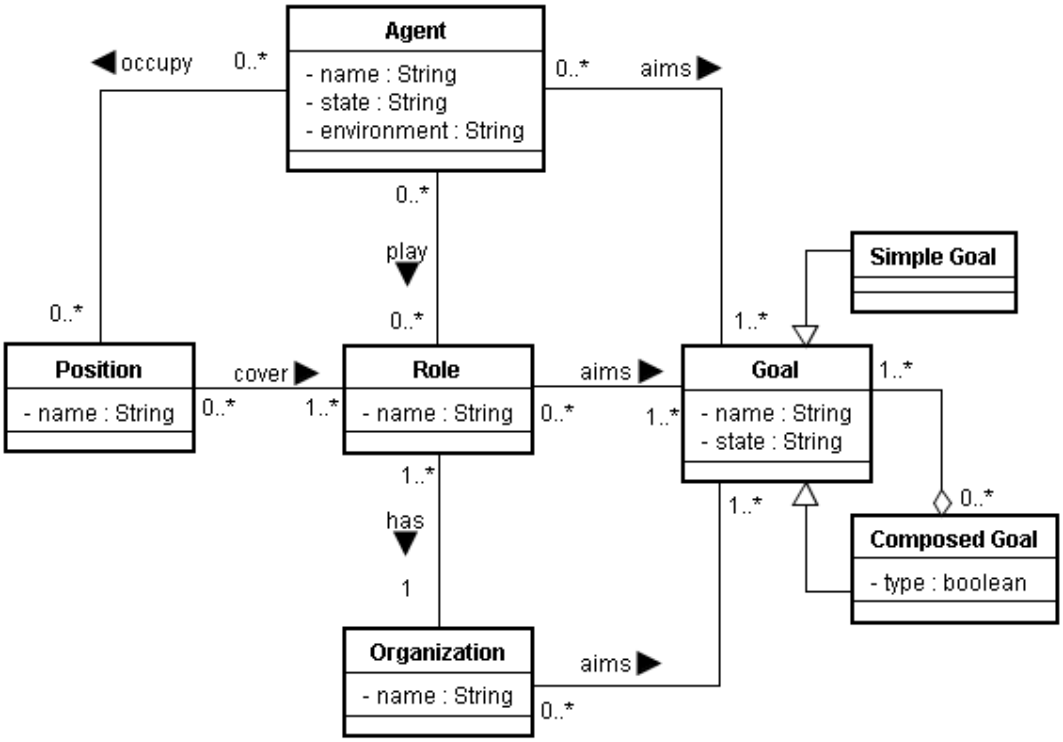


Figura 41 – Pacote *Main*.

Detalhamento do pacote *Main*:

- ***Agent*:** um agente, representado pela entidade *agent, é* um sistema computacional inserido em um ambiente, capaz de atingir os objetivos planejados por meio de ações autônomas nesse ambiente. Esta entidade possui os seguintes atributos: *name*, atributo alfanumérico que identifica um agente no ambiente; *state*, atributo alfanumérico que descreve o estado atual de um agente, podendo assumir os valores *created* (agente criado no ambiente), *execution* (agente realizando uma tarefa), *ready* (agente pronto para executar tarefa), *blocked* (agente em espera) e *finished* (agente finalizado); *environment*, atributo alfanumérico que descreve o ambiente em que um agente está localizado. Ao atributo *name* é aplicada uma restrição de obrigatoriedade, indicando que o atributo *name* deve ser informado para o conceito *Agent*, e uma restrição de unicidade, indicando que o atributo *name* não pode assumir o mesmo valor para diferentes instâncias do conceito *Agent*. Ao atributo *state* é aplicada uma restrição de obrigatoriedade e uma restrição que indica que o atributo pode assumir os seguintes valores: *created*, *execution*, *ready*, *blocked* e *finished*.

As restrições em OCL para o conceito de agente são apresentadas da seguinte forma:

context Agent inv MandatoryAgentName: self.name.size()>0

context Agent inv UniqueAgentName: Agent.allInstances ->
forAll(other|self.name = other.name implies self = other)

context Agent invAgentState: Agent.allInstances->forAll(self.state = 'created'
xor self.state= 'execution' xor self.state = 'ready' xor self.state = 'blocked' xor
self.state = 'finished')

O conceito *Agent* possui os seguintes relacionamentos:

- ***Agent starts InternalEvent***: um agente dispara zero ou mais eventos internos. Estes podem ser disparados no instante em que os *clocks* dos mesmos coincidirem com o tempo atual do sistema ou mesmo sem nenhuma condição associada. Um evento interno é disparado por um agente.
- ***Agent has Belief***: um agente contém zero ou mais crenças que armazenam seu conhecimento. Uma crença está contida em zero ou mais agentes.
- ***Agent has Perceptron***: um agente contém um ou mais *perceptrons* que avaliam as mensagens recebidas do ambiente. Um *perceptron* está contido em um agente.
- ***Agent plays Role***: um agente exerce um ou mais papéis relacionados a organização. Um papel é exercido por um ou mais agentes.
- ***Agent aims Goal***: um agente almeja o alcance de um ou mais objetivos. Um objetivo é almejado por zero ou mais agentes.
- ***Role*:** é uma representação abstrata de uma função de agente, serviço ou identificação dentro de um grupo, e pode ter associado a si um conjunto de atribuições e restrições. Esta entidade possui o atributo *name* que identifica unicamente o papel na organização. Ao atributo *name* é aplicada uma restrição de integridade e uma restrição de unicidade. Essa restrição pode ser expressa da seguinte maneira:

  context Role inv UniqueAgentName: Agent.allInstances ->

forAll(other|self.name = other.name implies self = other)

- ***Role aims Goal***: um papel almeja o alcance de um ou mais objetivos. Um objetivo é almejado por um papel.
- ***Role has Organization*****:** um papel tem uma organização. Uma organização pode conter zero ou mais papéis.
- ***Role must execute Action***: um papel deve executar zero ou mais ações. Uma ação deve ser executada por zero ou um papel.
- ***Role can execute Action***: um papel pode executar zero ou mais ações. Uma ação pode ser executada por zero ou um papel.

Nos dois últimos relacionamentos é aplicada uma restrição de integridade que indica que um papel pode ou deve executar pelo menos uma ação, conforme a seguir:

context Role inv Actions:
self.action->notEmpty() or
self.mandatoryAction->notEmpty()

Nessses relacionamentos também é aplicada uma restrição de integridade que indica que as ações de um plano que alcança um objetivo almejado por um papel devem estar dentre as ações que o papel pode ou deve executar:

context Role inv ActionsPlan:
(self.action->union(self.mandatoryAction))->includesAll(self.goal.plan.action)

- ***Goal*****:** é um estado que o agente deseja alcançar. Esta entidade possui os seguintes atributos: *name***,** atributo alfanumérico que identifica um objetivo; *state***,** atributo alfanumérico que define o estado necessário para que um plano alcance esse objetivo. Esse estado é representado pelas crenças que o agente possui. Ao atributo *state* é aplicada apenas uma restrição de obrigatoriedade O conceito *Goal* possui os seguintes relacionamentos:
  - ***Role aims Goal****.*

- ***Agent aims Goal.***
- ***Organization aims Goal.***
- ***ComposedGoal is aggregates Goal*.**
- ***SimpleGoal extends Goal*:** um objetivo simples especializa um objetivo.
- ***ComposedGoal extends Goal*:** um objetivo composto especializa um objetivo.
- ***Plan achieves Goal***: um plano alcança um ou mais objetivos. Um objetivo é alcançado por um ou mais planos.

- ***Composed Goal:*** um objetivo pode agregar outros objetivos, formando desta forma o objetivo composto para alcance do estado desejado pelo agente. Segundo [HEN05], o objetivo principal pode conter subobjetivos para seu alcance, podendo fazer parte deste objetivo principal, sendo assim este conceito é representado no MMI pelo objetivo composto. Este objetivo composto pode conter objetivos relacionandos através de uma conjução pelo operador lógico *AND*, ou disjunção pelo operador lógico OR, sendo estes valores lógicos reprentados pelo atributo *type*. Esta entidade é uma especialização da entidade objetivo (*Goal*). [HEN05a] destaca que a noção de subobjetivo é ambígua. Em certos casos, subobjetivo é conceituado como um objetivo contido no caminho do alcance do objetivo principal, outras vezes, é considerado como uma parte do objetivo principal. Para o segundo caso, conforme [HEN05a], podemos denominá-lo como objetivo parcial.
  - ***ComposedGoal is aggregates Goal***: um objetivo composto é agregado por um o ou mais objetivos. Um objetivo agrega zero ou mais objetivos compostos.
- ***Simple Goal:*** representa um objetivo simples, no qual o agente pode executá-lo sozinho para alcançar um determinado estado. Esta entidade é uma especialização da entidade objetivo (*Goal*).
  - ***SimpleGoal extends Goal.***
- ***Position:*** representa um conjunto de papéis executados por agentes. Esta entidade possui o atributo.*name* que identifica unicamente o nome da posição. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. O conceito *Position* possui os seguintes relacionamentos:
  - ***Position cover Role:*** uma posição cobre zero ou mais papéis. Um papel é coberto por zero ou mais posições.

- ***Organization:*** é um grupo de agentes desempenhando funções com um propósito comum. Esta entidade possui o atributo *name* para identificá-la. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. O conceito *Organization* possui os seguintes relacionamentos:
  - ***Role has Organization.***
  - ***Organization aims Goal:*** uma organização almeja o alcance de um ou mais objetivos. Um objetivo é almejado por zero ou mais organizações.

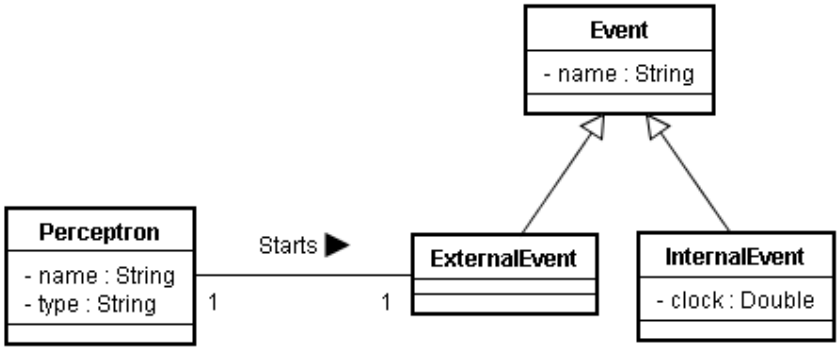


Figura 42 – Pacote *Sensorial*.

Detalhamento do pacote *Sensorial*:

- ***Perceptron*****:** representada pela entidade *Perceptron*, é responsável por perceber as mensagens vindas do ambiente para o agente de acordo com um padrão pré-definido. Esta entidade possui os seguintes atributos: *name*, atributo alfanumérico que identifica um *perceptron*; *type*, atributo alfanumérico que define o padrão de mensagens aceita por um *perceptron*. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Ao atributo *type* é aplicada apenas uma restrição de obrigatoriedade O conceito *Perceptron* possui os seguintes relacionamentos:
  - ***Agent has Perceptron.***
  - ***Perceptron starts ExternalEvent***: um *perceptron* dispara um evento externo. Um evento externo é disparado por um *perceptron*.
  - ***Perceptron evaluates Message****:* um *perceptron* avalia uma ou mais mensagens. Uma mensagem é avaliada por um *perceptron*.
- ***Event:*** é uma comunicação da alteração ocorrida no ambiente onde o agente está situado. Esta entidade possui o atributo alfanumérico *name* que identifica

unicamente um evento. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. O conceito *Event* possui os seguintes relacionamentos:

- ***InternalEvent extends Event***: um evento interno especializa um evento.
- ***ExternalEvent extends Event***: um evento externo especializa um evento.
- ***Event generates Belief****:* um evento gera uma ou mais crenças. Uma crença é gerada por zero ou um evento.

- ***InternalEvent*****:** representa as alterações internas no comportamento do agente. Esta entidade possui o atributo numérico *clock* que define o instante de tempo que um evento interno será disparado. O conceito *InternalEvent* possui os seguintes relacionamentos:
  - ***Agent starts InternalEvent.***
  - ***InternalEvent extends Event.***

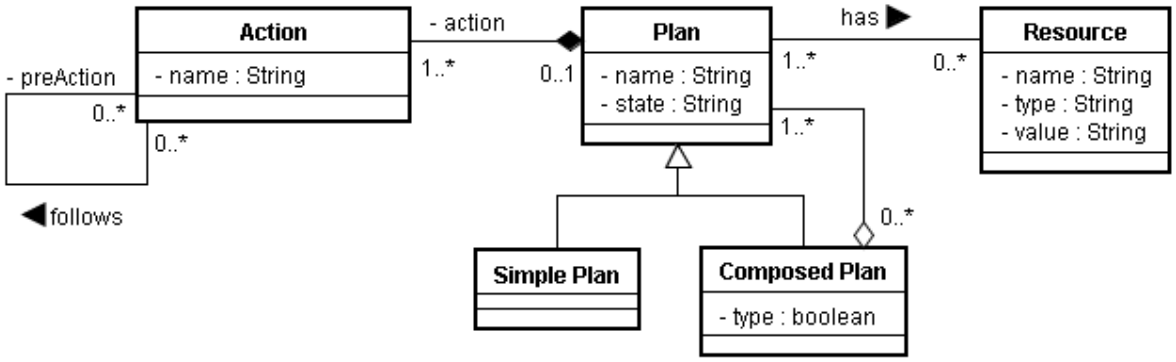


Figura 43 – Pacote *Executor*.

Detalhamento dos atributos do pacote *Executor*:

- ***Action:*** é parte de um trabalho que pode ser atribuído a um agente Esta entdiade possui o atributo alfanumérico *name* que identifica unicamente uma ação. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. O conceito *Action* possui os seguintes relacionamentos:
  - ***Role must execute Action.***
  - ***Role can execute Action.***

- ***Plan is composed by Action***: um plano é composto por uma ou mais ações. Uma ação compõe zero ou um plano.
- ***Action generates Belief***: uma ação gera uma ou mais crenças, sendo tratadas como pós-condições dessa. Uma crença é gerada por zero ou uma ação.
- ***Belief controls Action***: uma crença regula zero ou mais ações, sendo tratada como pré-condição destas. Uma ação é regulada por zero ou mais crenças.
- ***Action publishes Message****:* uma ação publica zero ou mais mensagens no ambiente. Uma mensagem é publicada por uma ação.
- ***Action follows Action***: uma ação posterior sucede zero ou mais ações. Uma ação anterior precede zero ou mais ações.

- ***Plan*****:** é formado por conjuntos de ações com pré-condições (circunstâncias no qual o plano é aplicável), corpo (sequência de ações) e pós-condições (o estado que o plano pode atingir) associadas, que um agente deve executar para atingir seus objetivos. Esta entidade possui os atributos: *name*, atributo alfanumérico que identifica um plano; *state*, atributo alfanumérico que descreve o estado atual da execução de um plano. Esse estado pode ser representado por crenças do agente ou ainda por ações que estão sendo executadas em um dado instante. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Ao atributo *state* é aplicada apenas uma restrição de obrigatoriedade. O conceito *Plan* possui os seguintes relacionamentos:
  - ***Plan achieves Goal.***
  - ***Plan is composed by Action.***
  - ***Belief controls Plan:*** uma crença regula zero ou mais planos, sendo tratada como pré-condição destes. Um plano é regulado por zero ou mais crenças.
  - ***Plan has Resource:*** um plano usa zero ou mais recursos de determinado tipo para auxiliar no alcance de seus objetivos. Um recurso é usado por um ou mais planos.
  - ***ComposedPlan aggregates Plan****.*
  - ***SimplePlan extends Plan:*** um plano simples especializa um objetivo.
  - ***ComposedPlan extends Plan:*** um plano composto especializa um objetivo.

- ***ComposedPlan:*** um plano pode agregar outros planos formando desta forma o plano composto para alcance do objetivo do agente. Este plano composto pode conter outros planos relacionando-os através de uma conjução pelo operador lógico *AND*, ou disjunção pelo operador lógico OR, sendo estes valores lógicos representados pelo atributo *type*. Esta entidade é uma especialização da entidade plano (*Plan*).
  - ***Composed Plan aggregates Plan***: um plano agrega zero ou mais subplanos, que também são planos. Um plano é agregado por zero ou um plano.
- ***SimplePlan:*** representa um plano simples que o agente executa para alcançar seu objetivo. Esta entidade é uma especialização da entidade plan (*Plan*).
  - ***SimplePlan extends Plan.***
- ***Resource***: representa uma entidade física ou uma informação a ser utilizada por um agente durante a execução de um plano. Esta entidade possui os atributos: *name*, atributo alfanumérico que identifica um recurso; *type*, atributo alfanumérico que descreve o tipo de um recurso; *value*, atributo alfanumérico que define o valor de um recurso. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Ao atributo *type* é aplicada apenas uma restrição de obrigatoriedade. O conceito *Resource* possui os seguintes relacionamentos:
  - ***Plan has Resource****.*

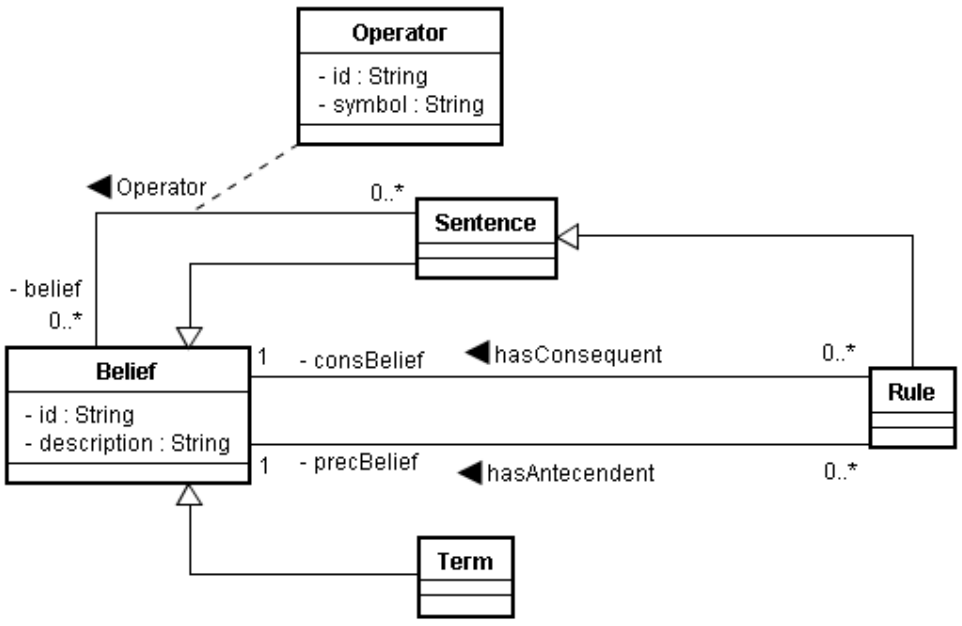


Figura 44 – Pacote *Decision*.

Detalhamento do pacote *Decision*:

- ***Belief:*** representa as expectativas de um agente sobre o estado atual do mundo e sobre a probabilidade de um curso de ação atingir certos efeitos. Esta entidade possui os atributos: *id*, atributo alfanumérico que identifica uma crença; *description*, atributo alfanumérico que descreve uma crença. Ao atributo *id* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Ao atributo *description* é aplicada apenas uma restrição de obrigatoriedade. O conceito *Belief* possui os seguintes relacionamentos:
  - ***Agent has Belief.***
  - ***Event generates Belief.***
  - ***Action generates Belief.***
  - ***Belief controls Action.***
  - ***Belief controls Plan.***
  - ***Sentence extends Belief***: uma sentença especializa uma crença.
  - ***Sentence Operator Belief***: uma sentença agrega zero ou mais crenças com o uso de uma classe associativa *Operator*. Uma crença é agregada por zero ou mais sentenças com o uso de uma classe associativa *Operator*.
  - ***Rule extends Sentence*****:** uma regra especializa uma sentença. *Rule* é um tipo de sentença que necessariamente deve possuir crenças como antecedentes e conseqüentes, em que a primeira implica na segunda.
  - ***Rule has antecedent Belief***: uma regra tem uma crença como antecedente. Uma crença é antecedente de zero ou mais regras.
  - ***Rule has consequent Belief***: uma regra tem uma crença como conseqüente. Uma crença é conseqüente de zero ou mais regras.
  - ***Term extends Belief****:* um termo especializa uma crença.
- ***Operator*****:** *id***,** atributo alfanumérico que identifica um operador; *symbol*, atributo alfanumérico que representa o conetivo lógico associado ao conceito *Operator***,** podendo assumir os valores $\neg$, $\wedge$, $\vee$ e $\Rightarrow$. Ao atributo *id* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Enquanto que ao atributo *symbol*

é aplicada apenas uma restrição de obrigatoriedade. O conceito *Operator* possui os seguintes relacionamentos:

- ***Sentence Operator Belief.***

O pacote *Decision* possui o relacionamento detalhado a seguir:

- ***Rule extends Sentence****:* é um tipo de sentença que necessariamente deve possuir crenças como antecedentes e conseqüentes, em que a primeira implica na segunda. Uma regra especializa uma sentença.

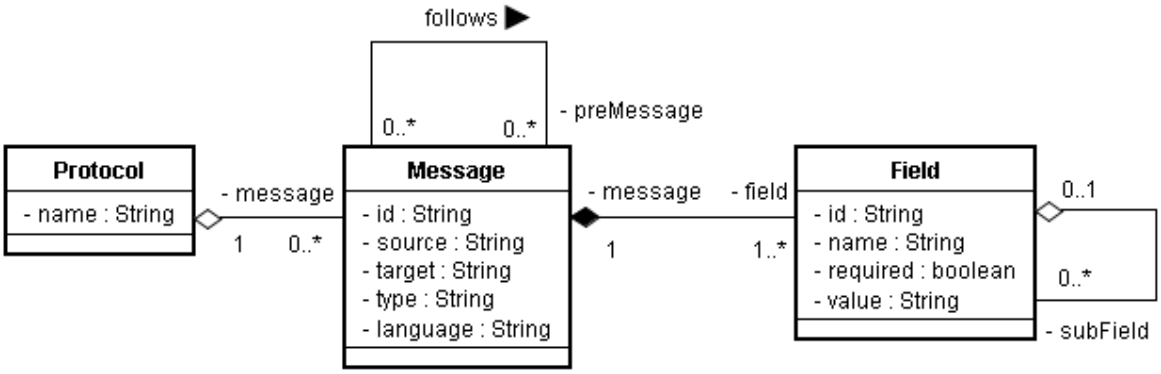


Figura 45 – Pacote *Communication*.

Detalhamento do pacote *Communication*:

- ***Protocol*****:** representa o protocolo da mensagem a ser utilizada pelo agente. Esta entidade possui o atributo alfanumérico *name* que identifica o nome de um protocolo de comunicação usado pelo agente. Ao atributo *name* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. O conceito *Protocol* possui os seguintes relacionamentos:
  - ***Protocol aggregates Message***: um protocolo agrega zero ou mais mensagens. Uma mensagem é agregada por um protocolo.

- ***Message*****:** representa as mensagens de entrada e saída dos agentes*.* Esta entidade possui os atributos: *id***,** atributo alfanumérico usado como identificador de uma mensagem; source, atributo alfanumérico usado para identificar o agente emissor de uma mensagem; *target***,** atributo alfanumérico usado para identificar o

agente receptor de uma mensagem; *type*, atributo alfanumérico usado para identificar o tipo de mensagem ou performativo de determinado protocolo correspondente a uma mensagem; *language*, atributo alfanumérico usado para identificar a linguagem que está sendo utilizada para a representação de uma mensagem. Ao atributo *id* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Aos atributos *source*, *target*, *type* e *language* é aplicada apenas uma restrição de obrigatoriedade. O conceito *Message* possui os seguintes relacionamentos:

- ***Perceptron evaluates Message****.*
- ***Action publishes Message****.*
- ***Protocol aggregates Message***.
- ***Message is composed by Field***: uma mensagem é composta por um ou mais campos. Um campo compõe uma mensagem.
- ***Message follows*****:** Message: uma mensagem posterior sucede zero ou mais mensagens. Uma mensagem anterior precede zero ou mais mensagens.

- ***Field*****:** representa os diferentes parâmetros que compõem um determinado tipo de mensagem.. Esta entidade possui os atributos: *id,* atributo alfanumérico que identifica um campo; *name***,** atributo alfanumérico que descreve o nome de um campo; *required***,** atributo booleano que define se um campo é obrigatório ou não para determinado tipo de mensagem; *value***,** atributo alfanumérico que define o valor de um campo. Ao atributo *id* é aplicada uma restrição de obrigatoriedade e uma restrição de unicidade. Ao atributos *name* é aplicada apenas uma restrição de obrigatoriedade. Ao atributo *required* é aplicada uma restrição de obrigatoriedade e uma restrição que indica que esse atributo pode assumir o valor ‘*True’* ou *‘False’.* O conceito *Field* possui os seguintes relacionamentos:

  - ***Message is composed by Field****.*
  - ***Field aggregates Field:*** um campo agrega zero ou mais subcampos, que também são campos. Um campo é agregado por zero ou um campo.

5.5 Considerações

Neste capítulo, foi apresentado o meta-modelo gerado a partir da compilação dos conceitos do Tropos e do MRIA. As entidades, atributos e relacionamentos do meta-modelo foram detalhados. Em seguida foi apresentado o meta-modelo estendido juntamente com seus conceitos e detalhamento das entidades, dos atributos e dos relacionamentos que compõe este meta-modelo estendido.

O meta-modelo apresentado tem sua base principal no MRIA. Além disso, recebeu contribuição de conceitos da metodologia Tropos como: dependência entre agentes em um sistema multiagentes para alcance cooperativo de um objetivo, a decomposição de planos em subplanos e objetivo em subobjetivos e o conceito de posição o qual agrupa papéis. Além disso, foi identificada uma lacuna conceitual na metodologia Tropos com relação aos conceitos de crença, percepção e eventos que estão presentes na literatura da área.

Os aspectos macro do SMA não estão plenamente incorporados, pois este meta-modelo partiu do MRIA que abrange especificamente a estrutura interna de um agente e incorporou os conceitos da metodologia Tropos para desse suporte aos seus modelos. Assim, o foco do MMI está na estrutura interna de um agente, embora seja estendido para suportar novas metodologias, neste será incorporado os aspectos macros que suportes estes novos modelos.

No próximo capítulo será apresentado todo o processo de construção do protótipo para o uso do meta-modelo estendido. Em sequência, será apresentado um exemplo de uso que ilustre toda a abordagem.

6 IMPLEMENTAÇÃO E EXEMPLO DE USO

Neste capítulo, serão apresentadas as ferramentas utilizadas para a construção do protótipo MMI4E (*MAS Meta-model For Eclipse),* a partir do protótipo de Santos [SAN08], permitindo o mapeamento dos modelos Tropos para os modelos do MMI através de um assistente (*Wizard*) e a criação e validação de modelos com base no MMI. Após, são apresentados também o processo de consistência dos modelos de aplicação e a geração de código, seguido do mapeamento dos conceitos e dos relacionamentos do MMI para a plataforma de implementação *SemantiCore*. Neste capítulo também é detalhado o desenvolvimento do protótipo, como o mesmo pode ser estendido e são descritas quais restrições de integridade já estão cobertas no mesmo. Além disso, é explicado o padrão de representação dos modelos em *XML* da ferramenta TAOM4E e do MMI4E, e as funcionalidades do protótipo por meio da visualização de suas interfaces. Por fim, é apresentado um exemplo de uso que ilustre toda a abordagem.

6.1 Ferramentas Utilizadas

Para o desenvolvimento do protótipo foram utilizadas a linguagem de programação Java 6, a ferramenta de verificação de restrição de integridade *USE* e a ferramenta de auxilio de geração de código *Velocity*.

A ferramenta USE *(UML-based Specification Environment)* foi escolhida por preencher os requisitos básicos necessários para o protótipo, tendo como principal contribuição a possibilidade de. compilação das restrições *OCL* e a possibilidade de verificação de consistência entre o metamodelo e os modelos. A versão utilizada é a *USE 2.3.1,* e está disponível em [USE09], podendo ser livremente distribuído sob licença *LGPL*.

Para a geração de código, foi escolhida a ferramenta *Velocity* [APA09b] da *Apache Software Foundation* sob licença de software Apache [APA09a]. O *Velocity* oferece alguns projetos, dos quais este protótipo utiliza o *Velocity Engine,* sendo escrito em JAVA e pode ser facilmente integrado em aplicações Web e *frameworks* [APA09c].

O MMI4E abrange um processo de consistência de modelos e geração de códigos através da entrada de um modelo de aplicação. O protótipo recebe como entrada três arquivos: um arquivo XMI representando o MMI, sendo gerado pelo software *Argo/UML*, um arquivo *OCL* com as restrições aplicadas ao metamodelo, e um arquivo XML representando o modelo de aplicação, sendo este um modelo Tropos gerado pela

ferramenta TAOM4E e mapeado para o MMI através de um assistente de importação (*Wizard*). Este processo pode ser visualizado através da figura 46.

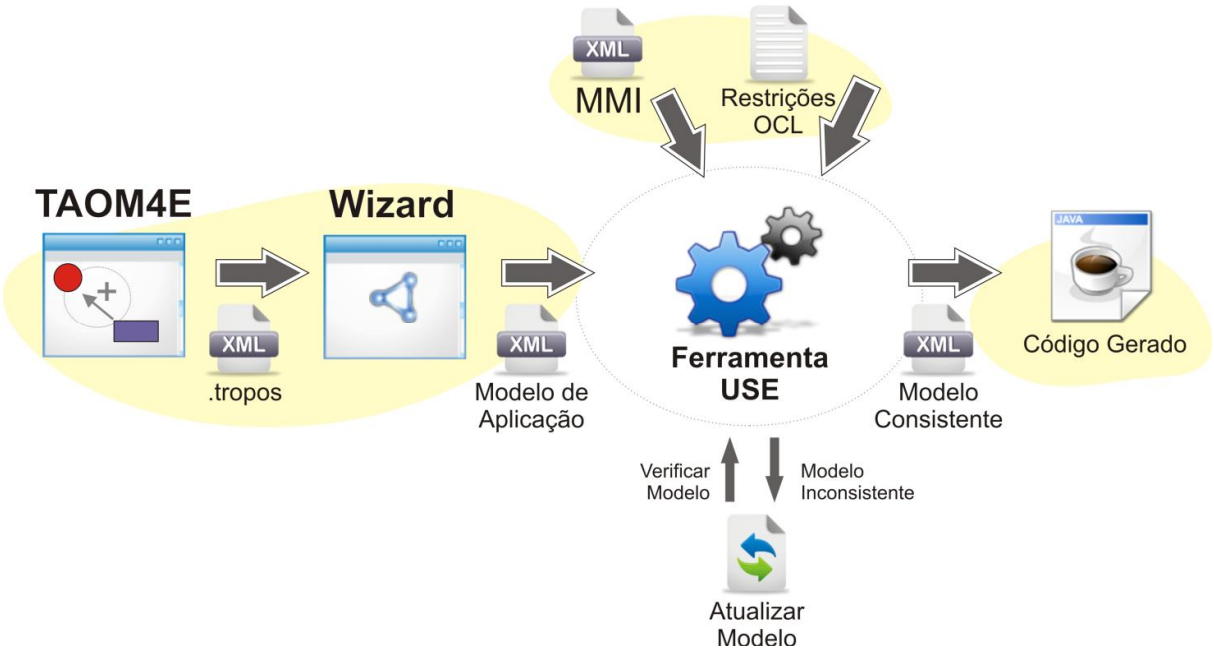


Figura 46 - Processo de Mapeamento, Consistência e Geração de Código.

O processo do MMI4E divide-se em quatro passos: entrada dos arquivos do meta-modelo e regras OCL, importação e mapeamento do modelo de entrada, verificação das restrições *OCL*, verificação da consistência entre o modelo de aplicação e as restrições aplicadas ao meta-modelo, e por fim a geração de código a partir do modelo de aplicação.

No primeiro passo, o protótipo recebe como entradas o arquivo *XMI* do meta-modelo e o arquivo *OCL* das restrições de integridade no desenvolvimento do software, antes mesmo de executá-lo. Esses dois arquivos são transformados por um *parser* do protótipo em um arquivo *USE*. Com isso, o arquivo *USE* gerado poderá ser usado como entrada da ferramenta *USE* que verificará se as restrições aplicadas ao meta-modelo estão escritas de maneira correta.

No segundo passo, o protótipo recebe um modelo Tropos modelado na ferramenta TAOM4E e mapeia-o para um modelo MMI utilizando um assistente de importação (Wizard*).* Este *Wizard* realiza os mapeamentos automáticos quando possível e interage com o usuário para tomadas de decisões e preenchimento de informações não disponíveis no modelo original. Após o mapeamento é gerado um arquivo XML para representar o modelo de aplicação utilizando os conceitos do MMI.

Uma vez realizado o primeiro passo, pode-se verificar se o modelo de aplicação gerado pelo segundo passo está consistente com o MMI Para isso, o protótipo usa o modelo de aplicação representado por um arquivo XML em um formato proprietário. Esse arquivo será transformado em um arquivo do CMD que será utilizado como entrada da ferramenta USE, instanciando o meta-modelo com o modelo de uma aplicação. Com isso, a ferramenta USE terá a representação do meta-modelo e suas restrições (por meio do arquivo USE gerado no primeiro passo) e a representação de um modelo de aplicação (por meio do arquivo CMD). Desta forma, a ferramenta USE pode verificar se um modelo de aplicação está consistente com o meta-modelo definido. Caso esteja consistente, o protótipo permitirá a geração de código do modelo em uma dada plataforma de implementação. Em caso negativo, o modelo deve ser atualizado até que esteja de acordo com o MMI e suas restrições.

O quarto passo consiste na geração de código para uma plataforma de implementação como o SemantiCore. Basicamente, este passo recebe como entrada o arquivo XML e o transforma em código fonte proprietário de uma plataforma de implementação por meio de um parser definido pelo usuário.

Assim, o modelo de entrada é o elemento mapeado e será validado para geração de código, e este deve respeitar as restrições vinculadas ao MMI. O MMI4E realiza um mapeamento entre o metamodelo de Tropos com base no MMI e posteriormente verifica a consistência e gera código voltados à plataforma *SemantiCore*.

6.2 Mapeamento do MMI para o SemantiCore

Nesta seção, será apresentado o mapeamento criado entre os conceitos e relacionamentos do MMI e os elementos da plataforma para a correta geração de código a partir do uso de um modelo de aplicação. O mapeamento dos conceitos é descrito a seguir:

- ***Agent***: um conceito *Agent* foi mapeado para uma extensão da classe *SemanticAgent*. O atributo *name* foi mapeado para o nome da extensão; o atributo *state* não foi mapeado diretamente, pois o *SemantiCore* trata o estado do agente internamente; o atributo *environment* foi mapeado para o atributo *environment* passado como argumento de um *SemanticAgent* em um arquivo *semanticoreconfig.xml* que instancia os agentes na plataforma. A figura 47 exemplifica parte do mapeamento:

```
public class eCultureSystem extends SemanticAgent
```

Figura 47 – Mapeamento do agente.

- ***Goal***: um conceito *Goal com suas especializações SimpleGoal* e *ComposedGoal* foram mapeados para uma classe *Goal*. O atributo *name* foi mapeado para o nome de uma instância da classe; os atributos *state* e *type* (do *ComposedGoal*) não foram mapeados, sendo que o atributo *state* não foi mapeado diretamente pois a plataforma trata o estado dos objetivos internamente. A figura 48 exemplifica parte do mapeamento, onde o primeiro parâmetro indica o agente e o segundo indica o modelo de ontologia associado ao objetivo:

```
Goal getCulturalInformation = new Goal(this.getOwner(),null);
```

Figura 48 – Mapeamento de objetivo.

- ***Resource***: um conceito *Resource* foi mapeado para um atributo de uma extensão da classe *ActionPlan*. Os atributos *name*, *type* e *value* foram mapeados respectivamente para o nome, o tipo e o valor do atributo que representa o conceito *Resource*. A figura 49 seguir exemplifica parte do mapeamento:

```
private String queryResults = "Results of Query";
```

Figura 49 – Mapeamento do recurso.

- ***Perceptron***: um conceito *Perceptron* foi mapeado para uma extensão da classe *Sensor*. O atributo *name* foi concatenado com a palavra *Sensor* e mapeado para o nome da extensão; o atributo *type* foi mapeado para o argumento do tipo *Object* passado no método *evaluate* da extensão. A figura 50 exemplifica parte do mapeamento:

```
public class DependencySensor extends Sensor;
```

Figura 50 – Mapeamento da percepção.

- ***Plan***: um conceito *Plan* com suas especializações *SimplePlan* e *ComposedPlan* foram mapeados para uma extensão da classe *ActionPlan*. O atributo *name* foi mapeado para o nome da extensão; os atributos *state* e *type* (do *ComposedPlan*) não foram mapeados, sendo que o atributo *state* não foi mapeado diretamente para o *SemantiCore* pois o estado de um plano é tratado internamente na plataforma. A figura 51 exemplifica parte do mapeamento:

```
public class SynthsizeResults extends ActionPlan;
```

Figura 51 – Mapeamento do plano.

- ***Action***: um conceito *Action* foi mapeado para uma extensão da classe *Action*. O atributo *name* foi mapeado para o nome da extensão. A figura 52 exemplifica parte do mapeamento:

```
public class ActionGetInfoOnArea extends Action;
```

Figura 52 – Mapeamento da ação.

- ***Term***: um conceito *Term* foi mapeado para uma classe *SimpleFact*. O atributo *id* foi mapeado para o nome de uma instância da classe e o atributo *description* foi mapeado para os atributos sujeito, predicado e objeto passados como argumentos da instância. A figura 53 a seguir exemplifica parte do mapeamento:

```
SimpleFact contribution1 = new SimpleFact("simpleGoal
Portable","Contributes +","SimpleGoal Available eCulture System");
```

Figura 53 – Mapeamento do termo.

- ***Sentence***: um conceito *Sentence* pode ser mapeado para uma classe *ComposedFact* ou para duas instâncias, cada uma pode ser das classes *SimpleFact*, *ComposedFact* ou *Rule*. Explicações adicionais sobre o mapeamento de um conceito *Sentence* são apresentadas no detalhamento do relacionamento *Sentence Operator Belief*. Quando o conceito *Sentence* é mapeado para uma classe *ComposedFact*, o atributo *id* é mapeado para o nome de uma instância da classe e o atributo *description* não é mapeado diretamente para o *SemantiCore*, pois não existe um atributo que permita a descrição de uma sentença. A figura 54 exemplifica parte de um possível mapeamento, onde *contributions* representa o nome da instância, e *a* e *b* representam *SimpleFacts* instanciados:

```
ComposedFact contributions = new ComposedFact(contribution1,
                                              contribution2);
```

Figura 54 – Mapeamento da sentença.

- ***Rule***: um conceito *Rule* foi mapeado para uma classe *Rule*. O atributo *id* foi mapeado para o nome de uma instância da classe e o atributo *description* foi mapeado para o atributo *name* da instância. A figura 55 exemplifica parte do mapeamento, onde *regra* representa o nome da instância, *DecisaoCompra* representa o atributo *name*, enquanto *u* e *v* representam respectivamente as crenças antecedente e conseqüente da regra:

```
Rule regra = new Rule("DecisaoCompra",u,v);
```

Figura 55 – Mapeamento da regra.

- ***Message***: um conceito *Message* foi mapeado para uma extensão da classe *SemanticMessage*. O atributo *id* foi mapeado para o nome de uma instância da extensão; os atributos *source* e *target* foram mapeados respectivamente para os atributos *from* e *to*, já existentes na *SemanticMessage*; o atributo *type* foi concatenado com a palavra *Message* e mapeado para o nome da extensão; o atributo *language* foi mapeado para o atributo *language* que foi incluído na extensão. A figura 56 exemplifica parte do mapeamento:

```
public class DependencyMessage extends SemanticMessage
```

Figura 56 – Mapeamento da regra.

- ***Protocol***: um conceito *Protocol* foi mapeado para um atributo que foi incluído nas extensões da classe *SemanticMessage* (conceitos *Message* associados a um conceito *Protocol*). O atributo *name* foi mapeado para o valor desse atributo. O trecho a seguir exemplifica parte do mapeamento:

```
private String protocol = “ContractNet”;
```

Figura 57 – Mapeamento do protocolo.

- ***Field***: um conceito *Field* foi mapeado para um atributo que foi incluído nas extensões da classe *SemanticMessage* (conceito *Message* associado a um conceito *Field*). O atributo *id* não foi mapeado diretamente para o *SemantiCore*, pois o mesmo não necessita ser gerado em código visto que sua função é apenas permitir a associação com um conceito *Message*; o atributo *name* foi mapeado para o nome do atributo incluído; o atributo *value* foi mapeado para o valor do atributo incluído; o atributo *required* não foi mapeado diretamente, pois esse apenas indica se o valor de um campo deve ou não ser informado. A figura 58 exemplifica parte do mapeamento:

```
private String RequestDependency1= “The Plan Find info sources request of
  the eCulture System request Info about source Resource of the Museum”;
```

Figura 58 – Mapeamento do campo da mensagem.

Detalhado o mapeamento dos conceitos do MMI para a plataforma *SemantiCore*, o mapeamento dos relacionamentos entre esses conceitos é descrito a seguir:

- ***Plan has Resource***: como explicado no detalhamento do conceito *Resource*, esse conceito foi mapeado para um atributo de uma extensão da classe *ActionPlan*. A figura 59 exemplifica parte do mapeamento:

```
public class SynthesizeResults extends ActionPlan {
      private String queryResults;
}
```

Figura 59 – Mapeamento do relacionamento *Plan has Resource*.

- ***Agent has Belief***: um conceito *Belief* foi mapeado para uma classe *SimpleFact*, *ComposedFact* ou *Rule* criada e adicionada no método *setup* de uma extensão da classe *SemanticAgent*. A figura 60 exemplifica parte do mapeamento:

```
public class eCultureSystem extends SemanticAgent {
      protected void setup ( ) {
            SimpleFact contribution1 = new SimpleFact("simpleGoal Portable",
"Contribrutes +", "SimpleGoal Available eCulture System");
            addFact(contribution1);
      }
}
```

Figura 60 – Mapeamento do relacionamento *Agent has Belief.*

- ***Agent has Perceptron***: esse relacionamento foi mapeado para uma chamada do método *addSensor* dentro do método *setup* de uma extensão da classe *SemanticAgent*. A figura 61 exemplifica parte do mapeamento:

```
public class eCultureSystem extends SemanticAgent {
       protected void setup ( ) {
            addSensor ( new DependencySensor ("Percebe as Dependências") );
      }
}
```

Figura 61 – Mapeamento do relacionamento *Agent has Perceptron*.

- ***Plan achieves Goal***: um conceito *Plan* foi mapeado para o atributo *plan* da classe *Goal*. O plano é representado no terceiro argumento apresentado no trecho a seguir:

```
Goal searchInformation = new Goal(this.getOwner(),null,plan,null);
```

Figura 62 – Mapeamento do relacionamento *Plan achieves Goal*.

- ***ComposedPlan* aggregates *Plan***: um conceito *Plan* foi mapeado para uma extensão da classe *ActionPlan*. Uma instância do tipo *ActionPlan* pode ser incluída em uma outra instância do mesmo tipo pelo uso do método *addAction*. A figura 63 exemplifica parte do mapeamento:

```
ComposedPlan composedPlan = new ComposedPlan ("plano Composto");
composedPlan.addAction((ActionPlan)new SimplePlan(“Plano Simples”));
```

Figura 63 – Mapeamento do relacionamento *ComposedPlan* aggregates *Plan*.

- ***Plan is composed by Action***: um conceito *Action* foi mapeado para uma extensão da classe *Action*. Uma instância do tipo *Action* pode ser incluída em uma instância do tipo *ActionPlan* pelo uso do método *addAction*. A figura 64 exemplifica parte do mapeamento:

```
GetInfoOnArea getInfoOnArea = new GetInfoOnArea ("Get Info On Area");
getInfoOnArea.addAction((ActionPlan) new ActionGetInfoOnArea());
```

Figura 64 – Mapeamento do relacionamento *Plan is composed by Action*.

- ***Belief controls Plan***: um conceito *Belief* foi mapeado para o atributo *preCondition* de uma extensão da classe *Action* que inicia um *ActionPlan*. A figura 65 exemplifica parte do mapeamento, onde *preCondition* representa a crença e *AcaoInicial* representa a ação que inicia um plano:

```
SimpleFact preCondition = new SimpleFact("simpleGoal Portable","Contribrutes
+","SimpleGoal Available eCulture System");
SimpleFact postCondition = new SimpleFact("simpleGoal eCulture System ","
Contribrutes +"," Usable eCulture System");
new AcaoInicial (“AcaoInicial”, preCondition,postCondition);
```

Figura 65 – Mapeamento do relacionamento *Belief controls Plan*.

- ***Action generates Belief***: um conceito *Belief* foi mapeado para o atributo *postCondition* de uma extensão da classe *Action*. A figura 66 parte do mapeamento, onde *Contributions* representa a ação e *postCondition* representa a crença:

```
SimpleFact preCondition = new SimpleFact("simpleGoal Portable","Contribrutes
+","SimpleGoal Available eCulture System");
SimpleFact postCondition = new SimpleFact("simpleGoal eCulture System ","
Contribrutes +"," Usable eCulture System");
new Contributions (“Contributions”, preCondition,postCondition);
```

Figura 66 – Mapeamento do relacionamento *Action generates Belief*.

- ***Belief controls Action***: um conceito *Belief* foi mapeado para o atributo *preCondition* de uma extensão da classe *Action*. A figura 67 exemplifica parte do mapeamento, onde *EnviarProposta* representa a ação e *preCondition* representa a crença:

```
SimpleFact preCondition = new SimpleFact("simpleGoal Portable","Contribrutes
+","SimpleGoal Available eCulture System");
SimpleFact postCondition = new SimpleFact("simpleGoal eCulture System ","
Contribrutes +"," Usable eCulture System");
new Contributions (“Contributions”, preCondition,postCondition);
```

Figura 67 – Mapeamento do relacionamento *Belief controls Action*.

- ***Action publishes Message***: um conceito *Message* foi mapeado para uma extensão da classe *SemanticMessage*. A instância dessa extensão é passada como argumento do método *transmit* da extensão da classe *Action*. A figura 68 exemplifica parte do mapeamento:

```
DependencyMessage mensagem = new DependencyMessage(from, to, content);
transmit ( mensagem);
```

Figura 68 – Mapeamento do relacionamento *Action publishes Message*.

- ***Rule has antecedent Belief***: a instância de uma classe *SimpleFact*, *ComposedFact* ou *Rule* é mapeada para o atributo *fact* da instância da classe *Rule*. O trecho de código para esse relacionamento pode ser visto no mapeamento do conceito *Rule*.
- ***Rule has consequent Belief***: a instância de uma classe *SimpleFact*, *ComposedFact* ou *Rule* é mapeada para o atributo *consequence* da instância da classe *Rule*. O trecho de código para esse relacionamento pode ser visto no mapeamento do conceito *Rule*.
- ***Sentence Operator Belief***: caso o atributo *symbol* de *Operator* seja igual à $\wedge$, a *Sentence* é mapeada para uma instância da classe *ComposedFact*. Caso o atributo *symbol* tenha o valor $\vee$, a *Sentence* é mapeada para duas instâncias, cada uma pode ser das classes *SimpleFact*, *ComposedFact* ou *Rule*. Por fim, caso o valor de *symbol* seja $\neg$, a crença que a *Sentence* agrega é negada e é mapeada para uma instância da classe *SimpleFact*, *ComposedFact* ou *Rule*. O trecho de código para esse relacionamento pode ser visto no mapeamento do conceito *Sentence*.
- ***Perceptron evaluates Message***: esse relacionamento foi mapeado para o método *evaluate* de uma extensão da classe *Sensor*. A figura 69 exemplifica parte do mapeamento:

```
public Object evaluate(Object arg0){
      if (arg0 instanceof DependencyMessage) {
      }
}
```

Figura 69 – Mapeamento do relacionamento *evaluates Message*.

- ***Protocol aggregates Message***: esse relacionamento foi mapeado para o atributo *protocol* que foi incluído nas extensões da classe *SemanticMessage*. A figura 70 exemplifica parte do mapeamento:

```
public class DependencyMessage extends SemanticMessage {
      private String protocol;
}
```

Figura 70 – Mapeamento do relacionamento *Protocol aggregates Message*.

- ***Message is composed by Field***: esse relacionamento foi mapeado para atributos que foram incluídos nas extensões da classe *SemanticMessage*. A figura 71 exemplifica parte do mapeamento:

```
public class DependencyMessage extends SemanticMessage {
      private String RequestDependency1;
}
```

Figura 71 – Mapeamento do relacionamento *Message is composed by Field*.

- ***Action follows Action***: o valor do atributo *postCondition* de uma extensão da classe *Action* definida previamente deve ter o mesmo valor do atributo *preCondition* de uma extensão da classe *Action* definida na seqüência. A figura 72 exemplifica parte do mapeamento:

```
super ( "previousAction", beliefAnt, beliefCons );
super ( "postAction", beliefCons, belief );
```

Figura 72 – Mapeamento do relacionamento *Message is composed by Field*.

- ***Field aggregates Field***: o atributo representando o subcampo no relacionamento deve ser mapeado para o atributo de um campo representado por uma classe que é um atributo de uma extensão da classe *SemanticMessage*. A figura 73 exemplifica parte do mapeamento:

```
public class Field {
      private String subField;
}
public class Message extends SemanticMessage {
      private Field field;
}
```

Figura 73 – Mapeamento do relacionamento *Message Field aggregates Field*.

Os conceitos *Role*, *Position*, *Organization*, *InternalEvent*, *ExternalEvent* e os relacionamentos *Agent plays Role, Agent starts InternalEvent, Role aims Goal, Role must execute Action, Role can execute Action, Role has Organization, ComposedGoal*

aggregates Goal, Perceptron starts ExternalEvent, Message follows Message, Event generates Belief não foram mapeados diretamente para o *SemantiCore,* pois a plataforma não trata esses conceitos. O conceito *Operator* não foi mapeado diretamente para o *SemantiCore, p*orém, foi considerado no mapeamento do relacionamento *Sentence Operator Belief*.

6.3 Desenvolvimento do Protótipo

Nesta seção será apresentada a arquitetura do protótipo criado para o uso do MMI. Os principais objetivos desse protótipo são mapear os modelos de Tropos, facilitar a entrada de dados de modelos de aplicação, verificar a consistência desses modelos com o meta-modelo e possibilitar a geração de código em uma plataforma de implementação, neste caso, no *SemantiCore*. O protótipo é dividido nos seguintes pacotes principais: *application*, *constraints*, *concepts*, *gui*, *metamodel*, *parser*, *relationships*, *support*, *use, velocity e wizard*. Na apresentação de cada pacote, serão suprimidos atributos e métodos secundários para facilitar a leitura. A Figura 74 apresenta o Diagrama de Pacotes do protótipo desenvolvido e as próximas seções detalham cada pacote assim como as classes e arquivos que compõem cada um desses.

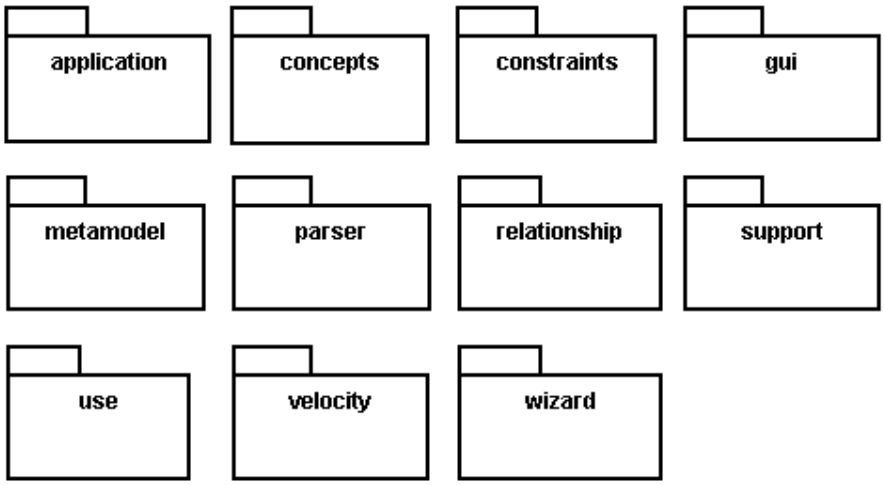


Figura 74 - Diagrama de Pacotes do MMI4E

Sucintamente, estes pacotes podem ser descritos como:

- ***Application***: armazena as classes geradas que representam os agentes de um modelo de aplicação na plataforma SemantiCore

- ***Concepts***: este pacote armazena as classes utilizadas para guardar temporariamente os valores dos conceitos do MMI durante o processo de geração de código com a plataforma de implementação. Dentre as classes que compõem esse pacote, apenas *Sentence* e *Rule* possuem uma estrutura que se difere das demais. A primeira armazena os atributos do conceito Sentence, além dos identificadores das crenças que essa agrega e o valor do atributo *symbol* do conceito Operator participante da relação de agregação. Por outro lado, a segunda armazena os atributos do conceito *Rule*, juntamente com os identificadores das crenças antecedente e conseqüente que se relacionam com a mesma. A Figura 75 apresenta o Diagrama de Classes UML desse pacote.

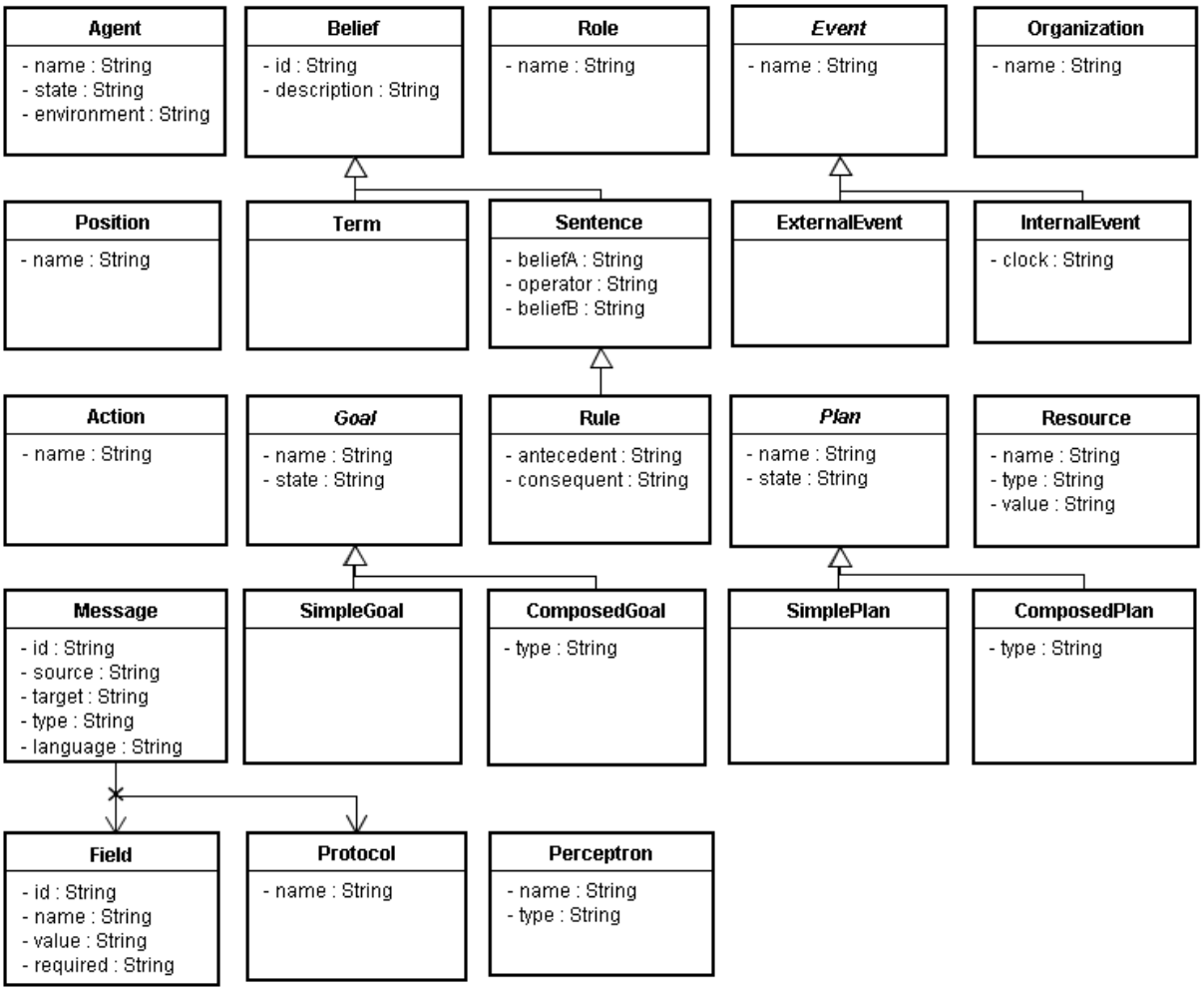


Figura 75 – Diagrama de Classes UML do pacote concepts.

Estas classes representam os conceitos do MMI e todos os atributos, independente do tipo, foram mapeados para atributos do tipo *String* com o objetivo de facilitar a implementação.

- ***Constraints***: armazena o arquivo *constraints.ocl,* contendo todas as restrições aplicáveis ao metamodelo.
- ***Gui***: este pacote armazena todas as interfaces gráficas do protótipo. Além dos subpacotes *gui.consult* e *gui.register*, o pacote é composto pelas classes *CreateModel*, *LoadModel*, *MainGui* e *UseLog*. A Figura 76 apresenta a estrutura geral do mesmo.

A classe *MainGui* representa a interface principal da aplicação. Os principais atributos dessa classe são: *ArrayList model*, *Vector conceptsList* e *Vector relationshipsList*. O primeiro atributo representa todos os dados de um modelo de aplicação corrente, o segundo representa uma lista dos conceitos criados para esse modelo e o último representa uma lista dos relacionamentos entre esses conceitos. As classes *CreateModel*, *LoadModel* e *ImportModel* possuem uma referência para *MainGui*. A primeira é responsável pelo armazenamento de um modelo em um arquivo *XML* no padrão de representação de modelos do protótipo. A segunda permite o carregamento de um modelo representado por um arquivo desse mesmo tipo. A terceira permite importar um modelo gerado por outras ferramentas, no caso deste trabalho do modelo Tropos construído na ferramenta TAOM4E, para o assistente de importação (*Wizard*). Por fim, a classe *UseLog* representa a interface gráfica onde são apresentados os resultados da checagem de modelos de aplicação. O pacote *gui* ainda é constituído pelos subpacotes *gui.register* e *gui.consult.*

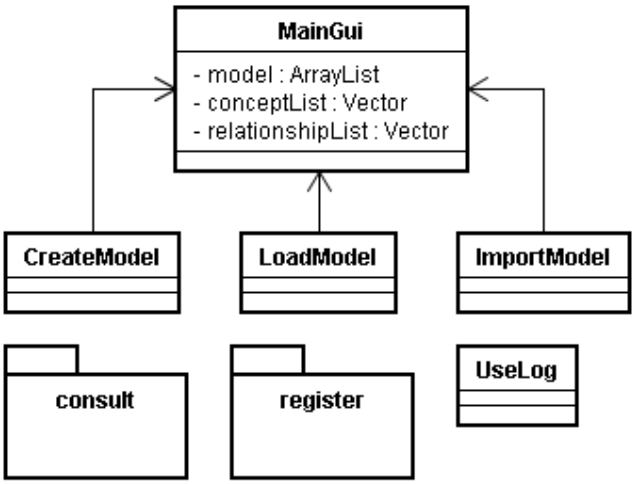


Figura 76 – Estrutura geral do pacote *gui*

O pacote *gui.consult* armazena todas as interfaces gráficas das consultas de conceitos do protótipo. Estas interfaces apresentam o conteúdo de cada conceito que é consultado. Cada interface possui uma referência para *MainGui* e outra para a classe que

armazena o conceito (por exemplo, *ConsultAction* possui uma referência para *MainGui* e para *Action*).

O pacote *gui.register* armazena todas as interfaces gráficas dos cadastros da aplicação. Além dos subpacotes *gui.register.concepts* e *gui.register.relationships*, o pacote é composto pelas classes *CreateConcept* e *CreateRelationship*. A primeira consiste em uma interface gráfica onde é feita a escolha do tipo de conceito que será criado e a segunda consiste em uma interface gráfica onde é feita a escolha do tipo de relacionamento entre conceitos que será criado. Ambas possuem uma referência para a classe *MainGui*. Este pacote possui um subpacote *gui.register.relationship.*

O pacote *gui.register.relationship* armazena todas as interfaces gráficas dos cadastros de relacionamentos da aplicação. Assim, em cada interface devem ser informados os dados do relacionamento a ser cadastrado. Cada uma das classes desse pacote possui uma referência para *MainGui* e uma lista para cada conceito que participa de um tipo de relacionamento. Por exemplo, a classe *ActionBelief* possui duas listas: uma de ações (*Vector actionsList*) e outra de crenças (*Vector beliefsList*).

- ***Metamodel***: armazena o arquivo *metamodel.xmi*, para representação do Diagrama de Classes UML do MMI Esse arquivo foi gerado com a ferramenta *Argo/UML* [ARG09].
- ***Parser***: armazena classes responsáveis pelas traduções do protótipo. Este pacote é composto pelas classes *MetamodelToUseParser*, *ModelToObjectParser*, *ObjectToUseParser*, *CodeParser* e *ObjectToSemantiCoreParser,* conforme ilustrado na figura 77.

A classe *MetamodelToUseParser*: traduz do arquivo *XMI* que representa o meta-modelo juntamente com o arquivo de restrições de integridade *OCL* para um arquivo *USE* usado como entrada da ferramenta *USE*. Esta classe possui os atributos *xmiPathname*, *oclPathname* e *usePathname,* que representam respectivamente os caminhos dos arquivos *metamodel.xmi*, *constraints.ocl* e o caminho onde será gerado o arquivo *Metamodel.use*.

A classe *ModelToObjectParser* traduz do arquivo que representa o modelo da aplicação para um objeto da classe *Metamodel*. Esta classe possui o atributo *xmlPathname* que indica o caminho do arquivo *XML* referente ao modelo da aplicação. Além do método construtor, possui três métodos: *readXML*, *readConcepts* e *readRelationships*. O primeiro retorna um objeto da classe *Metamodel* contendo todos os conceitos e relacionamentos do modelo, o segundo retorna uma lista com todos os

conceitos do modelo enquanto o último retorna uma lista com todos os relacionamentos do modelo.

A classe *ObjectToUseParser*: traduz de um objeto da classe *Metamodel* para um arquivo *CMD* usado como entrada da ferramenta *USE*. Esta classe possui os atributos *usePathname* e *metamodel*. O primeiro indica o caminho onde será gerado o arquivo *Model.cmd* e o segundo representa um objeto da classe *Metamodel* que contém o modelo da aplicação. Possui o método *convertModelToUse* que traduz o modelo da aplicação em um arquivo *Model.cmd*.

A classe abstrata *CodeParser*: é estendida para a construção de *parsers* entre modelos de aplicação e código fonte da plataforma de implementação de SMA. Além do método construtor, possui os métodos *initVelocity()*, responsável pela inicialização da ferramenta *Velocity*, e *parseMetamodel(Metamodel metamodel)*, responsável pela tradução do modelo em código. Este último consiste em um método abstrato que deverá ser implementado na classe filha, conforme a plataforma de implementação escolhida.

A classe *ObjectToSemantiCoreParser*: traduz de um objeto da classe *Metamodel* para código fonte da plataforma *SemantiCore*. Para isso, deve estender a classe *CodeParser* e implementar o método *parseMetamodel(Metamodel metamodel)*. Nesse método são feitas as chamadas para os demais métodos do tipo *parser*, os quais são: *parseAction*, *parseAgent*, *parseDecision*, *parseMessage*, *parseMetamodel*, *parsePerceptron*, *parsePlan*, *parseSemantiCoreConfig* e *parseSemantiCoreInstantiation*. Cada um desses métodos recebe um objeto da classe *Metamodel* como argumento e efetua a criação de um tipo de arquivo no *SemantiCore*. Dentro de cada método ainda existe uma chamada para um método do tipo *createContext* que recebe como argumentos apenas os atributos do objeto da classe *Metamodel* relevantes para o contexto do arquivo que está sendo criado. Por exemplo, o método *parseAction* efetua a chamada do método *createActionContext (String actionName)* e o valor de *actionName* é atribuído a uma variável de contexto *Velocity*. Uma vez atribuídos os valores às variáveis de contexto *Velocity*, as mesmas podem ser referenciadas por arquivos do tipo *VTL* que servirão como *templates* para os arquivos gerados.

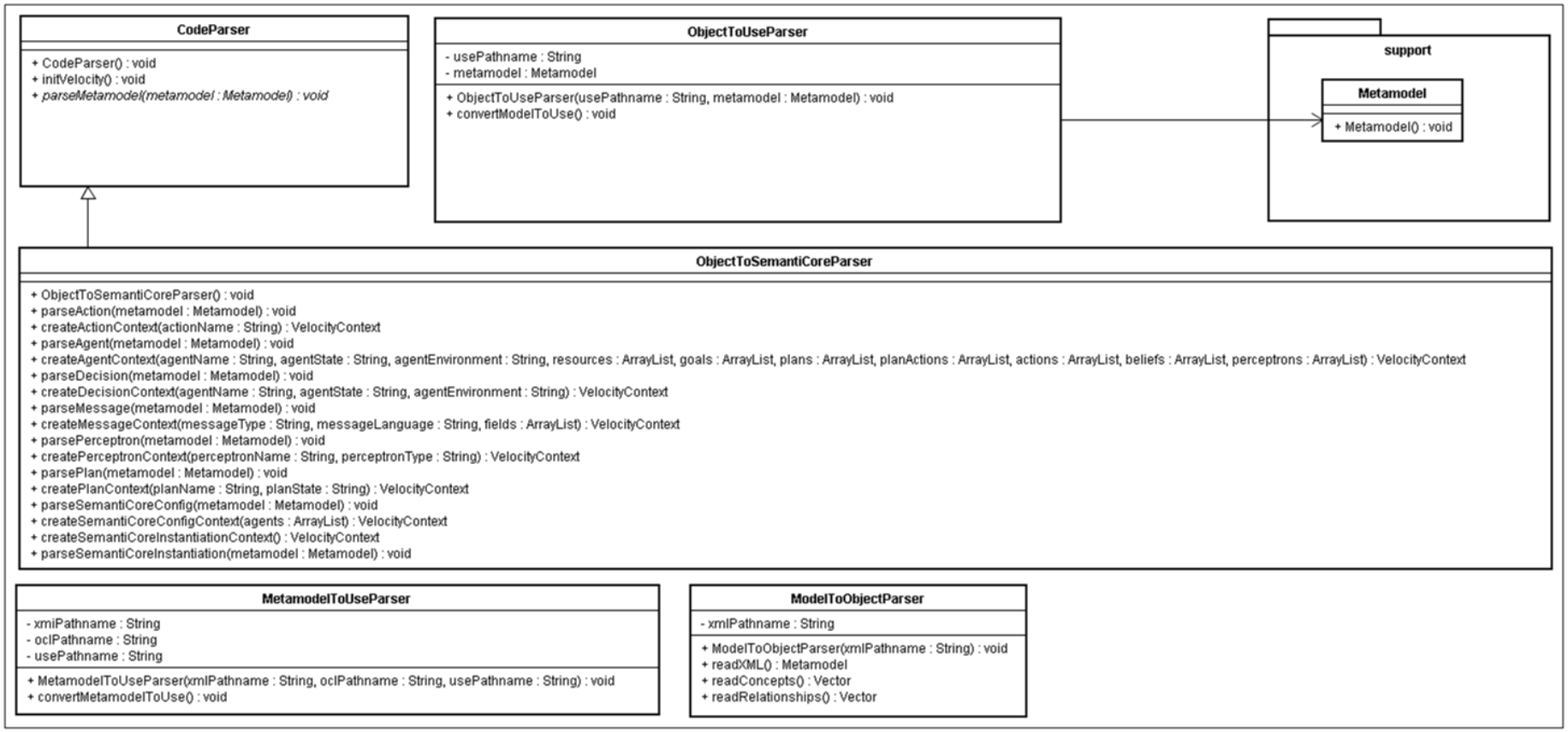


Figura 77 - Diagrama de Classes UML do pacote *parser.*

- ***Relationship***: armazena classes utilizadas para guardar temporariamente os valores dos relacionamentos do meta-modelo durante o processo de geração de código com a plataforma de implementação. Todas as classes desse pacote estendem a classe *Relationship*, que é composta por três atributos, são eles: *name*, definindo o nome do relacionamento, *idA*, definindo o nome do conceito que inicia o relacionamento, e *idB*, definindo o nome do conceito que finaliza o relacionamento.
- ***Support***: armazena as classes *Metamodel*, *ClassAux*, *FormatPlan* e *FormatResource* para auxiliar o protótipo.

A classe *Metamodel* é responsável por armazenar todos os conceitos e relacionamentos de um modelo de aplicação. Esta classe possui uma lista (*ArrayList*) de cada conceito e de cada relacionamento do meta-modelo.

A classe *ClassAux* é utilizada na classe *MetamodelToUseParser* para armazenar temporariamente os dados de conceitos do meta-modelo. Esta classe possui um atributo *id* do tipo *String* e um atributo *description* do tipo *String*.

A classe *FormatPlan.*é responsável pela formatação em caixa baixa ou caixa alta do atributo *name* do plano a ser gerado no código fonte da plataforma de implementação. Esta classe possui os atributos *name*, *nameUpper*, *nameLower* e *state*, todos do tipo *String*.

A classe *FormatResource é* responsável pela formatação em caixa baixa ou caixa alta do atributo *name* do recurso a ser gerado no código fonte da plataforma de implementação. Esta classe possui os atributos *name*, *nameUpper*, *nameLower*, *type* e *value*, todos do tipo *String*.

Este pacote possui o subpacote *support.semanticore* Neste pacote são armazenadas as classes auxiliares *MessageAux* e *TermAux* usadas para a geração de código para a plataforma *SemantiCore.* A classe *MessageAux*: é responsável por armazenar os dados de um conceito *Message* e dos conceitos *Protocol* e *Field* que se relacionam com o mesmo. E a classe *TermAux*: é responsável por armazenar o atributo *id* de um termo e armazenar partes do atributo *description* desse termo nos atributos *subject, predicate e object* que serão usados no *SemantiCore* para a representação de um *SimpleFact.*

- ***Use***: armazena os subpacotes *use.source* e *use.output*, além dos arquivos *Metamodel.use* e *Model*.cmd, utilizados como entrada da ferramenta USE, sendo

que o primeiro contém as definições dos conceitos, dos relacionamentos e das restrições aplicadas ao meta-modelo, enquanto o segundo representa o modelo de uma aplicação instanciada do meta-modelo.

O pacote *use.source* armazena as classes que foram criadas com o objetivo de integrar a ferramenta *USE* com o protótipo desenvolvido. Sendo assim, foram criadas as seguintes classes: *MyMain*, *MyModelToGraph*, *MyOptions*, *MySession* e *MyShell*. Essas têm como base algumas classes do código fonte da ferramenta *USE* com pequenas modificações, possibilitando assim seu uso com o protótipo.

A classe *MyMain* tem como origem a classe *org.tzi.use.main.Main*. Ela difere da classe original na referência para as novas classes *MyOptions*, *MySession* e *MyShell*, ao invés de ter referência para as classes *Options*, *Session* e *Shell*. Além disso, esta classe, mapeia as saídas da ferramenta *USE* para os arquivos *logUse.txt* e *logErr.txt.*

A classe *MyModelToGraph* tem como origem a classe *org.tzi.use.main.shell.ModelToGraph.* Ela difere da classe original na alteração do escopo *default* da classe para público, permitindo assim o acesso pela classe *MyShell*.

A classe *MyOptions* tem como origem a classe *org.tzi.use.config.Options*. Ela difere da classe original nos valores dos atributos *specFileName* e *cmdFileName*, representando respectivamente o arquivo *USE* do meta-modelo e o arquivo *CMD* do modelo de aplicação. Dessa maneira, tornaram-se possíveis várias checagens de consistência do modelo para uma mesma instância da aplicação *USE*.

A classe *MySession* tem como origem a classe *org.tzi.use.main.Session*. Ela difere da classe original na inclusão dos métodos *getFSystem()* e *setFSystem(MSystem system)*, possibilitando o retorno e a atribuição do estado atual do sistema.

A classe *MyShell* tem como origem a classe *org.tzi.use.main.shell.Shell*. Difere da classe original na atribuição do valor *false* à variável *fFinished*, possibilitando assim novas checagens de consistência para uma mesma instância da aplicação *USE*, e no método *cmdExit()*, onde ao invés do encerramento da aplicação com o comando *System.exit()* é utilizada a linha de comando *fSession.getFSystem().reset()*, reiniciando o estado atual do sistema.

O pacote *use.output* armazena os arquivos que gravam as saídas que seriam impressas no console da ferramenta *USE*, são eles: *logUse.txt* e *logErr.txt*. Assim, o primeiro arquivo armazena erros na estrutura do modelo criado e erros de consistência do modelo com o meta-modelo e suas restrições de integridade, enquanto que o segundo armazena apenas os erros de construção dos arquivos de entrada da ferramenta *USE* (caso a modelagem seja feita pelo protótipo, não ocorrem erros de construção).

- ***Velocity***: armazena todos os arquivos do tipo de template VTL usados na geração de código.

Foram criados os *templates action.vm, agent.vm, decision.vm, message.vm, perceptron.vm, plan.vm, semanticoreconfig.vm* e *semanticoreinstatiation*.vm para a geração de código para a plataforma de implementação SemantiCore. O *template action.vm* é utilizado para gerar código de uma classe *Action*, o *agent.vm* de uma classe *SemantiAgent*, o *decision.vm* de uma classe *DecisonEngine*, o *message.vm* de uma classe *SemanticMessage,* o *perceptron.vm* de uma classe *Sensor*, o *plan.vm* de uma classe *ActionPlan*, o *semanticoreconfig.vm* para a geração do arquivo *semanticoreconfig.xmlx e o semanticoreconfig.vm* para a geração do arquivo *semanticoreconfig.xml.*

Este pacote possui o subpacote *velocity.conf*, o qual armazena os arquivos de configuração e de manutenção de *logs* da ferramenta *Velocity*, são eles: *velocity.properties e velocity.log*.

- ***Wizard:*** armazena as classes responsáveis por iniciar um assistente de importação de metodologias ou linguagens de modelagem de SMA para o MMI. Este pacote armazena também o arquivo *model.xml*, sendo este o modelo de aplicação resultante do mapeamento, a ser verificado a consistência e gerado o código.

O pacote *wizard* contém as classes *Wizard, WizardForm, ConceptForm, RelationshipForm, Menu, ConceptIndex , RelationshipIndex, Welcome, ModelMapper, Model, Metamodel, ImportProcessor,* e *XMLHandler.* Estas classes e seus relacionamentos estão ilustradas no diagrama de classe UML apresentado na figura 78.

A classe *Wizard* é responsável por desenhar a janela principal do assistente de importação assim como os elementos gráficos que são comum em todas as telas. Para a composição do ambiente visual, este assistente utiliza o java.swing e java.awt..

A classe *WizardForm* é uma classe abstrata que utilizada por todos os formulários do assistente, e implementa um painel permitindo a navegabilidade entre as telas, utilizando para isso os objetos *previous, current* e *next*.

A classe *ConceptForm* é uma classe abstrata que estende a classe *WizardForm* e implementa a classe *ActionListener* do java.awt.event. Esta classe implementa a estrutura de um formulário para todos os conceitos a serem importados, interagindo com o usuário

através de campos de textos e botões para preenchimento de informações necessários para o mapeamento dos modelos Tropos para o MMI.

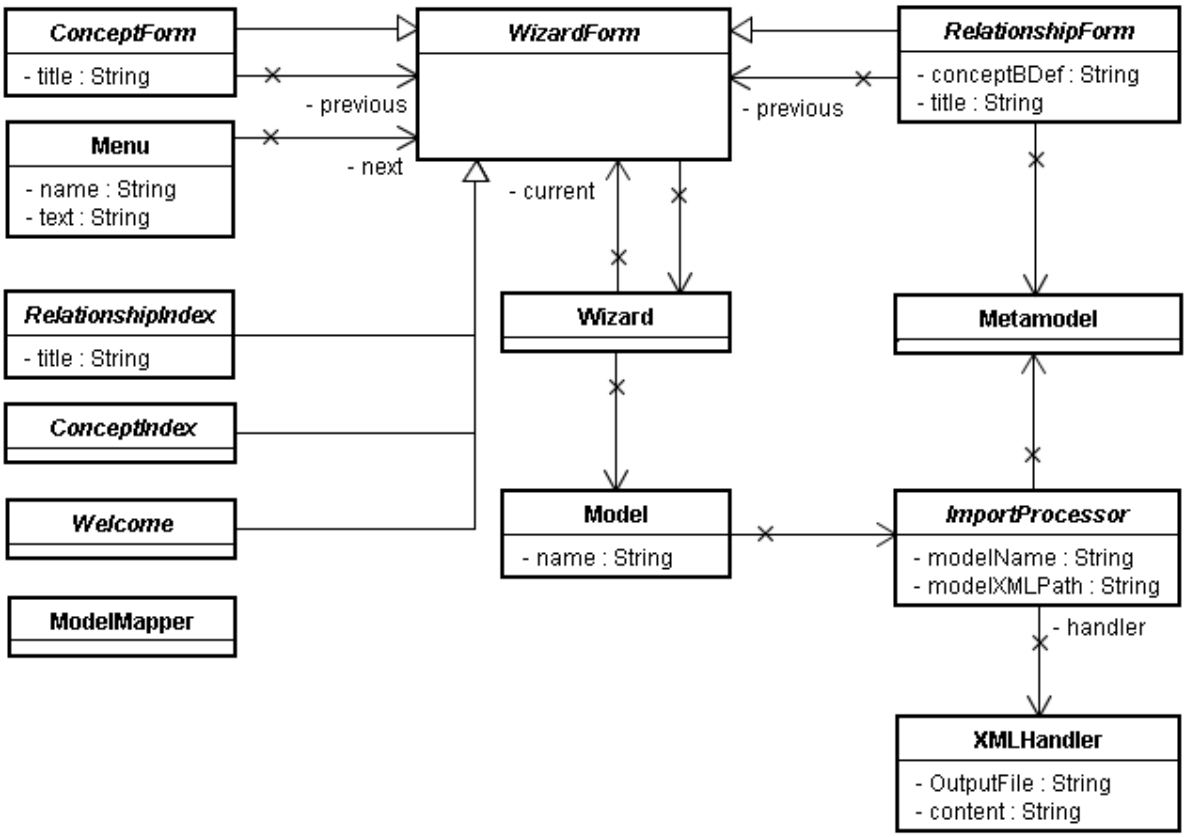


Figura 78 - Diagrama de Classes UML do pacote *wizard.*

A classe *RelationshipForm* é uma classe abstrata que estende a classe *WizardForm*. Assim como a classe *ConceptForm* está para os conceitos, esta classe implementa a estrutura de um formulário para todos os relacionamentos a serem importados através de interatividade com o usuário.

A classe *Menu* implementa um tipo abstrato de dados para construção dos menus do assistente.

A classe *ConceptIndex* é uma classe abstrata que estende a classe *WizardForm.* Esta classe implementa uma tela principal para todos os conceitos a serem mapeados. A partir desta tela o usuário pode selecionar para qual instância dos conceitos deseja importar e pode partir para os próximos conceitos.

A classe *RelationshipIndex é uma classe abstrata que* estende a classe *WizardForm.* Esta classe implementa uma tela principal para todos os relacionamentos a serem mapeados. Após a seleção do conceito, através da tela do *ConceptIndex,* é aberta a tela implementada pela classe *RelationshipIndex* na qual o usuário pode selecionar para

qual instância do relacionamento deseja importar e pode partir para os próximos conceitos.

A classe *Welcome* é uma classe abstrata que estende a classe *WizardForm*, que quando implementada apresenta uma tela inicial de boas vindas e uma breve explicação e dicas de acordo com a abordagem a ser importada.[

A classe *ModelMapper* recebe o arquivo a ser importado pelo método *ImportModel* da tela principal do protótipo, antes de iniciar o assistente, verifica a metodologia ou linguagem de modelagem que será importada. Após selecionado a abordagem, esta classe instancia o processador de importação adequado, carrega os conceitos e relacionamentos a serem mapeados, cria o modelo de importação e inicia o assistente. Este procedimento está ilustrado na figura 79.

```
public void run(ImportModel loader) {
            Document docReader = readFile();
            String modelName = this.getModelName();

            if (modelName.equalsIgnoreCase("masup")) {
                  // TODO MASUP
            } else if (modelName.equalsIgnoreCase("tropos")) {
                  ImportProcessor importProcessor = new TroposImport(docReader
                              .getElementsByTagName("TroposClasses"), docReader
                              .getElementsByTagName("TroposRelations"));

                  Wizard wizard = new Wizard(new Model("Tropos",
importProcessor),loader);
            }
}
```

Figura 79 – trecho de código da importação do modelo.

A classe *Model* representa o modelo a ser mapeado, contendo o nome do modelo e o processador de importação utilizado.

A classe Metamodel possui uma lista encadeada (<*LinkedList*>) para todos os conceitos e todos os relacionamentos do MMI, e permite a adição de instâncias destes conceitos ou relacionamentos em suas respectivas listas. Estas listas serão posteriormente mapeadase processadas para ser gerado o novo arquivo com o modelo de aplicação coerente ao MMI.

A classe *ImportProcessor* é um classe abstrata que implementa o processamento adequado do modelo. Esta classe possui os atributos *modelName* do tipo *String* para o nome do modelo, *handler* do tipo *XMLHandler* contendo o arquivo XML importado a ser manipulado, *metamodelo* do tipo *Metamodel* contendo listas com todos os conceitos e

relacionamentos do MMI e *modelXMLPath* do tipo String contendo o caminho do modelo de aplicação pós-mapeamento.

A classe *XMLHandler* utiliza a biblioteca java.io para manipular e escrever o arquivo novo a ser gerado com os conceitos mapeados.

Além destas classes, o pacote *Wizard* pode possuir subpacotes para cada metodologia ou linguagem de modelagem a serem mapeadas, e como o foco deste trabalho é o mapeamento da metodologia Tropos, foi criado um subpacote *wizard.tropos* o qual estende as classses abstratas deste assistente para permitir a correta importação dos modelos Tropos para o MMI.

O pacote *wizard.tropos* possui a classe *TroposWelcome,* o qual implementa a tela de boas-vindas para a importação da metodologia Tropos e dicas que guiam o usuário ao preenchimento dos formulários, e a classe *TroposImport* sendo esta uma extensão da classe abstrata *ImportProcessor*, contendo os métodos que mapeiam os conceitos e relacinoamento de Tropos para o MMI.

Os principais métodos para o processo de mapeamento da classe *TroposImport* são o *mapConcepts e o mapRelationships,* sendo estes indexadores para os métodos *mapActor, mapResource, mapGoal, mapPlan, mapDependency, mapDecomposition, mapContribution, mapContribution* e *mapMeansEnd.* A figura 80 mostra um trecho de código deste processo de mapeamento.

```
private void mapConcepts() {
      String classe = null;
      Element element = null;
      LinkedList<String> attributes = null;
      for (int i = 0; i < classes.getLength(); i++) {
            element = (Element) classes.item(i);
            classe = element.getAttribute("xsi:type").split(":")[1]
                        .substring(1).replace(" ", "").trim();
            attributes = new LinkedList<String>();
            attributes.add(element.getAttribute("xmi:id"));
            attributes.add(element.getAttribute("name"));
            if (classe.equalsIgnoreCase("actor"))
                  mapActor(attributes);
            else {
                  String owner = getNameById(classes, element
                              .getAttribute("Actor"));
                  if (classe.equalsIgnoreCase("resource"))
                        mapResource(attributes, owner);
                  else if (classe.equalsIgnoreCase("hardgoal"))
                        mapGoal(attributes, owner);
                  else if (classe.equalsIgnoreCase("softgoal"))
                        mapGoal(attributes, owner);
                  else if (classe.equalsIgnoreCase("plan"))
```

```
                        mapPlan(attributes, owner);
                }
        }
}

private void mapRelationships() {
        String classe = null;
        Element element = null;
        for (int i = 0; i < relations.getLength(); i++) {
                element = (Element) relations.item(i);
                classe = element.getAttribute("xsi:type").split(":")[1].replace(
                                " ", "").trim();
                if (classe.equalsIgnoreCase("FDependency"))
                        mapDependency(element);
                else if (classe.equalsIgnoreCase("BooleanDecomposition"))
                        mapDecomposition(element);
                else if (classe.equalsIgnoreCase("FContribution"))
                        mapContribution(element);
                else if (classe.equalsIgnoreCase("MeansEnd"))
                        mapMeansEnd(element);
        }
}
```

Figura 80 – trecho de código do mapeamento do modelo.

Este pacote possui, também, um subpacote para cada conceito, os quais são: *wizard.tropos.action, wizard.tropos.agent, wizard.tropos.belief, wizard.tropos.event, wizard.tropos.field, wizard.tropos.goal, wizard.tropos.message, wizard.tropos.organization, wizard.tropos.perceptron, wizard.tropos.plan, wizard.tropos.position, wizard.tropos.resource* e *wizard.tropos.role.* Cada subpacote deste estende as classes abstratas *ConceptForm, ConceptIndex, RelationshipForm, RelationshipIndex* para criar as telas de interação com o usuário para permitir a interação no processo de mapeamento, além de usarem a classe *TroposImport* para mapear estes conceitos com os valores atribuídos no assistente. Além disso, nestes subpacote estão as classes dos relacionamentos a qual cada conceito participa permitindo o usuário interagir quando o processo de mapeamento não puder ser automático.

Este assistente é flexível o suficiente para suportar mapeamento de novas abordagens, para posterior verificação de consistência e geração de código.

6.4 Especialização do Protótipo

Uma importante característica do protótipo desenvolvido é a possibilidade de extensão do mesmo de forma que possa ser usado para mapear modelos de novas abordagens e gerar código em outras plataformas de implementação existentes. Para isso, os seguintes passos são necessários:

- Mapeamento dos conceitos e relacionamentos da metodologia ou linguagem de modelagel para o MMI estendendo-o quando necessário.
- Criação de subpacotes do pacote *Wizard* do assistente de importação, e extensão da classe *ImportProcessor*, *Welcome, ConceptForm, ConceptIndex, RelationshipForm* e *RelationshipIndex.*
- Mapeamento dos conceitos e relacionamentos do meta-modelo para a plataforma de implementação escolhida.
- Criação de uma classe que estende a classe *CodeParser* e implementa o método *parseMetamodel(Metamodel metamodel)*. Nesse método, os valores dos conceitos e dos relacionamentos de um modelo devem ser atribuídos a diferentes variáveis inclusas em contextos da ferramenta *Velocity*. Com isso, os valores podem ser recuperados por arquivos *VTL* que servirão como *templates* para os arquivos gerados.
- Criação dos arquivos *VTL* usados na geração de código para determinada plataforma de implementação.
- Conforme a plataforma de implementação usada, pode ser necessária a criação de classes que auxiliem na geração de código.

6.5 Padrão de Representação de modelos Tropos e modelos MMI

O padrão de representação dos modelos em *XML* utilizado pelo protótipo foi criado com o objetivo de facilitar a integração do mesmo com diferentes ferramentas. Assim, o arquivo *XML* gerado pode ser traduzido para uma entrada da ferramenta *USE* (representada pelo arquivo *Model.cmd*), assim como pode ser traduzido para código fonte de uma plataforma de implementação, como *SemantiCore* utilzada neste trabalho. Na figura 81 apresentado um exemplo de um possível trecho de um modelo de aplicação representado no padrão utilizado pelo protótipo estendido.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<metamodel>
<concept def='Agent' name='Cliente' state='created' environment='ComponentesEnvironment'/>
<concept def='SimpleGoal' name='Comprar'>
<relationship def='Agent_SimpleGoal' idA='Cliente' idB='Comprar'/>
</metamodel>
```

Figura 81 - Padrão de Representação de Modelo

Este protótipo estendido, utiliza os mesmos conceitos desenvolvido por Santos em seu trabalho. Desta forma, na figura 81, a primeira linha representa apenas a inicialização do arquivo *XML*. As *tags* <*metamodel*> e <*/metamodel*> indicam respectivamente o início e o fim do modelo de aplicação. Existem dois tipos de elementos no padrão: *concept* e *relationship*. O elemento *concept* é composto pelo atributo *def*, descrevendo o conceito do metamodelo, e pelos atributos relacionados ao conceito. Porém, existem duas exceções, essas ocorrem quando o atributo *def* for igual à *Sentence* ou à *Rule*. Na primeira, além de *def* e dos atributos relacionados ao conceito, o elemento *concept* possui os atributos *beliefA*, *beliefB* e *operator*, descrevendo o relacionamento de agregação entre o conceito *Sentence* e o conceito *Belief* com o uso do atributo *symbol* da classe associativa *Operator*. Na segunda, o elemento *concept* além de *def* e dos atributos relacionados ao conceito, também é composto pelos atributos *antecedent* e *consequent*, descrevendo os relacionamentos entre um conceito *Rule* e dois conceitos *Belief*. No exemplo, é apresentado um *concept* com os atributos *def* igual à *Agent*, *name* igual à *Cliente*, *state* igual à *created* e *environment* igual à *ComponentesEnvironment*. Além disso, é apresentado um *concept* com o atributo *def* igual à *SimpleGoal*, atributo *name* igual à *Comprar*. De maneira semelhante, o elemento *relationship* sempre possui um atributo *def*, descrevendo o relacionamento do metamodelo, e dois atributos, *idA* e *idB*, identificando os conceitos participantes do relacionamento. Assim, no exemplo é apresentado um *relationship* com os atributos *def* igual à *Agent_SimpleGoal*, *idA* igual à *Cliente* e *idB* igual à *Comprar*

De maneira similar, o padrão de representação do arquivo gerado pelo software TAOM4E para modelos Tropos utiliza a linguagem XML para estruturar seus modelos e cada arquivo é criado ao se salvar um modelo desenhado visualmente por esta ferramenta. A figura 82, retirada de um trecho do arquivo gerado pelo TAOM4E, ilustra este padrão.

```
<?xml version="1.0" encoding="ASCII"?>
<it.itc.sra.taom4e.model.core.informalcore.formalcore:FormalBusinessModel
   xmi:id="_dy4dInB8Ed6W3cR8Dm7eaw">
<TroposClasses xsi:type="it.itc.sra.taom4e.model.core.informalcore.formalcore:FActor"
        xmi:id="_14aSsHB8Ed6W3cR8Dm7eaw" name="Cliente"
        ownedElements="_5VjH8HB8Ed6W3cR8Dm7eaw"/>
<TroposClasses xsi:type="it.itc.sra.taom4e.model.core.informalcore.formalcore:FHardGoal"
        xmi:id="_5VjH8HB8Ed6W3cR8Dm7eaw" name="Comprar"
        Actor="_14aSsHB8Ed6W3cR8Dm7eaw"/>
</it.itc.sra.taom4e.model.core.informalcore.formalcore:FormalBusinessModel>
</xmi:XMI>
```

Figura 82 - Padrão de representação de modelos Tropos do software TAOM4E.

Na figura 82, a primeira *tag* apresenta a inicialização do arquivo XML, a versão utilizada e os tipos de caracteres aceitos. As tags <it.itc.sra.taom4e.model.core.informalcore.formalcore:FormalBusinessModel> e </it.itc.sra.taom4e.model.core.informalcore.formalcore:FormalBusinessModel> marcam o conteúdo modelado no software, identificados por um identificador gerado pelo TAOM4E. Para demonstrar as entidades, é utilizado a *tag* <*TroposClasses* />, com um tipo, um identificador e o identificador do dono do elemento. No exemplo apresentado, o *HardGoal* Comprar pertence ao *Actor* Cliente, vinculados através de seu *Id*. Além da *tag* <*TroposClasses*> é utilizado a tag <*TroposRelations*> para os relacionamentos Tropos, tais como dependência, meios-fins e contribuição. Um exemplo do uso da *tag* <*TroposRelations*> está apresentada na figura 50 retirada de um trecho de código gerado pelo software TAOM4E.

```
<TroposRelations xsi:type="it.itc.sra.taom4e.model.core.informalcore:MeansEnd"
xmi:id="_f2wmoRAXEd6eO5v6VLoETQ" targets="_cFWxkBAXEd6eO5v6VLoETQ"
sources="_dzUvQBAXEd6eO5v6VLoETQ" Actor="_qyHZ0Q97Ed6hcLTJZOz2FA">
        <indexedSourcesAndTargets xmi:id="_f2wmohAXEd6eO5v6VLoETQ" markup="Unique
source" contents="_dzUvQBAXEd6eO5v6VLoETQ"/>
         <indexedSourcesAndTargets xmi:id="_f2wmoxAXEd6eO5v6VLoETQ" markup="Unique
        target" contents="_cFWxkBAXEd6eO5v6VLoETQ"/>
</TroposRelations>
 <TroposRelations xsi:type="it.itc.sra.taom4e.model.core.informalcore.formalcore:FDependency"
xmi:id="__TJ0Ri3gEd6MOrhLHDbNFA" name="Dependency 1"
targets="_4aONkS3gEd6MOrhLHDbNFA __TJ0QC3gEd6MOrhLHDbNFA"
sources="_qyHZ0Q97Ed6hcLTJZOz2FA _dzUvQBAXEd6eO5v6VLoETQ">
        <indexedSourcesAndTargets xmi:id="__TJ0Ry3gEd6MOrhLHDbNFA"
markup="Dependee" contents="_4aONkS3gEd6MOrhLHDbNFA"/>
        <indexedSourcesAndTargets xmi:id="__TJ0SC3gEd6MOrhLHDbNFA"
markup="Depender" contents="_qyHZ0Q97Ed6hcLTJZOz2FA"/>
        <indexedSourcesAndTargets xmi:id="__TJ0SS3gEd6MOrhLHDbNFA"
markup="Dependum" contents="__TJ0QC3gEd6MOrhLHDbNFA"/>
         <indexedSourcesAndTargets xmi:id="__TJ0Si3gEd6MOrhLHDbNFA" markup="Why"
contents="_dzUvQBAXEd6eO5v6VLoETQ"/>
</TroposRelations>
```

Figura 83 - Padrão de Representação de Relacionamentos em Tropos

Na figura 83 é demonstrado um exemplo de representação XML de um relacionamento meios-fins (*Means-ends*) e dependência (*Dependency*). No exemplo acima, o atributo *type* da classe *TroposRelations* determina qual tipo de relacionamento utilizado, o atributo *id* é o identificador deste relacionamento e o *name* traz o nome do relacionamento. No primeiro relacionamento, de meios-fins, os atributos *targets* e *sources*, informa qual entidade é o meio e qual entidade o fim, e o *Actor* identifica a qual ator pertence esta entidade. No segundo relacionamento, de dependência, além das entidades já citadas, há *tag <indexSourcesAndTargets>*, com o atributo *id* que a identifica. Esta *tag* possui os atributos *markupe* que atua como um rótulo, o qual mostra qual entidade é o *dependee, depender e dependum* neste relacionamento, bem como o motivo *why* desta relação, e o atributo *contents* que traz o identificador da entidade relacionada.

6.6 Utilização do Protótipo

Nesta seção será apresentado o detalhamento da estrutura geral do protótipo e serão mostradas as suas funcionalidades específicas por meio da visualização de suas interfaces.

6.6.1 Tela inicial

A tela principal do protótipo é composta por quatro menus principais, são eles: *File*, *Model*, *Code* e *Help*. A Figura apresenta a tela principal do protótipo.

O menu *File* é composto por seis itens, são eles: *New*, *Load*, *Import*, *Close*, *Save* e *Exit*. O primeiro possibilita a criação de um novo modelo de aplicação, o segundo permite o carregamento de um modelo contido em um arquivo *XML*, o terceiro permite importar modelos de outras abordagens a serem mapeados para o MMI, o quarto viabiliza o fechamento de um modelo carregado no protótipo, o quinto permite o armazenamento do modelo em uso no protótipo em um arquivo *XML*, e o último possibilita o fechamento do protótipo

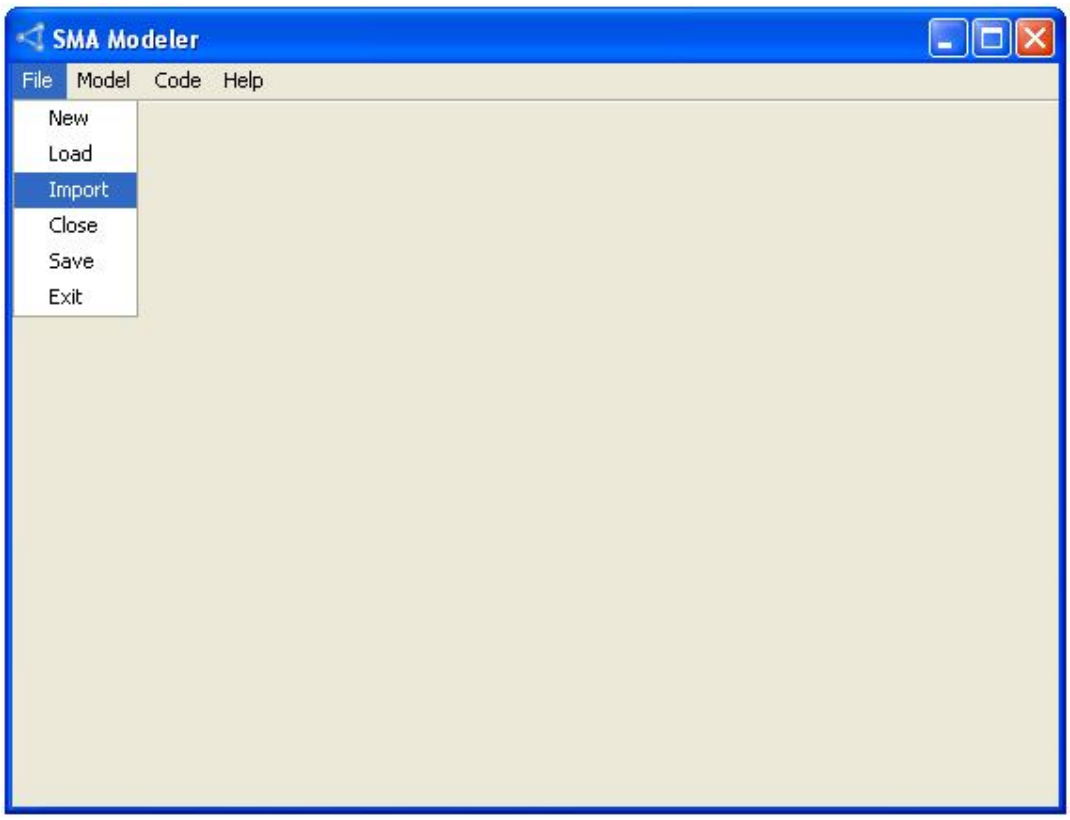


Figura 84 - Tela inicial do protótipo.

O menu *Model* possibilita a checagem de modelos com base no meta-modelo e suas restrições de integridade. Para isso, existe o item *Check Model* que quando acionado dispara o processo de checagem retornando para o usuário possíveis erros de consistência entre o modelo de aplicação e o meta-modelo proposto e suas restrições.

O menu *Code* permite a geração de código caso o modelo esteja consistente com o meta-modelo e suas restrições. No escopo deste trabalho, foi feita a geração de código na plataforma *SemantiCore*. Sendo assim, quando o item *Generate SemantiCore code* é acionado será gerado o código fonte da plataforma respeitando a modelagem realizada.

O menu *Help* apresenta apenas informações sobre a versão do protótipo disponibilizada no item *About*.

6.6.2 Importando Modelos Tropos com o Assistente de Importação

A importação de um modelo de outra abordagem é realizada ao selecionar a opção *Import* do menu principal, e após será aberto uma janela de seleção para escolha do menu, conforme ilustrado na figura 85. Embora este protótipo seja extensível para diferentes abordagens, apenas os modelos tropos são suportados para mapeamento, pois este é o objetivo deste trabalho.

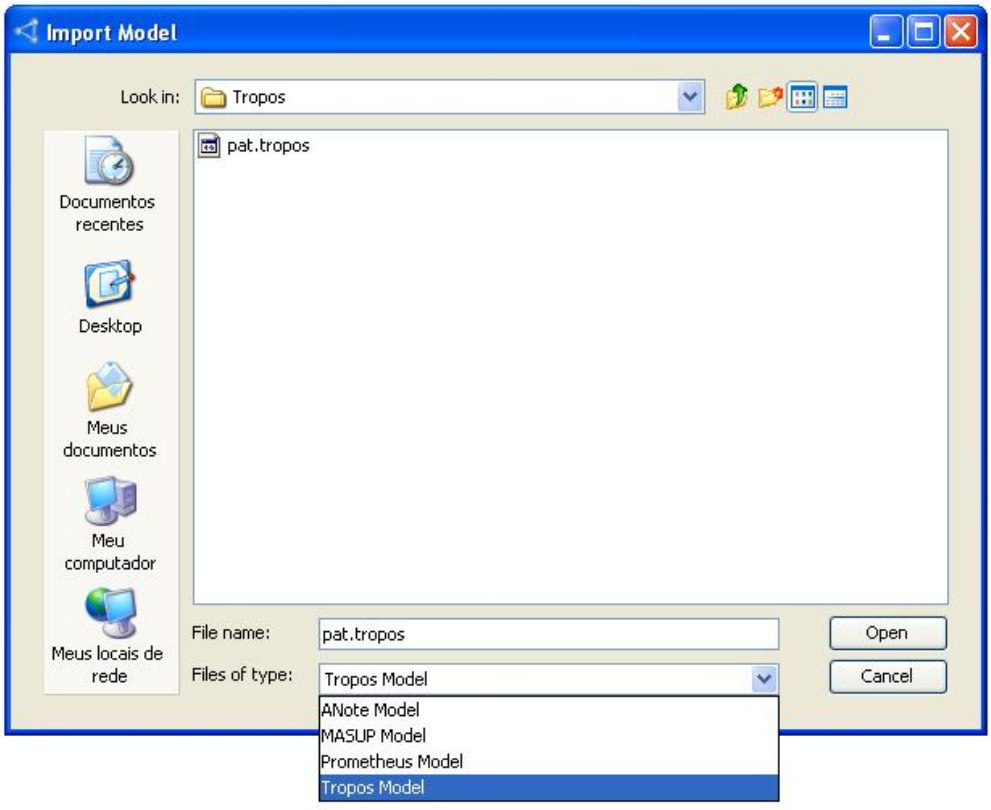


Figura 85 – Escolha do modelo a ser importado.

Após a escolha do modelo, é iniciado o Assistente de Importação (*Wizard*) para a metodologia do modelo. Junto com a tela é carregado o modelo e o objeto que processa o mapeamento do Tropos para o MMI. Nesta fase, todos os mapeamentos que puderam ser automáticos já foram realizados na instância do processador de importação. Ao iniciar o *Wizard*, é apresentado uma tela de boas vindas com informações sobre o assistente e dicas que guiam o usuário no processo de mapeamento, conforme ilustrado na figura .

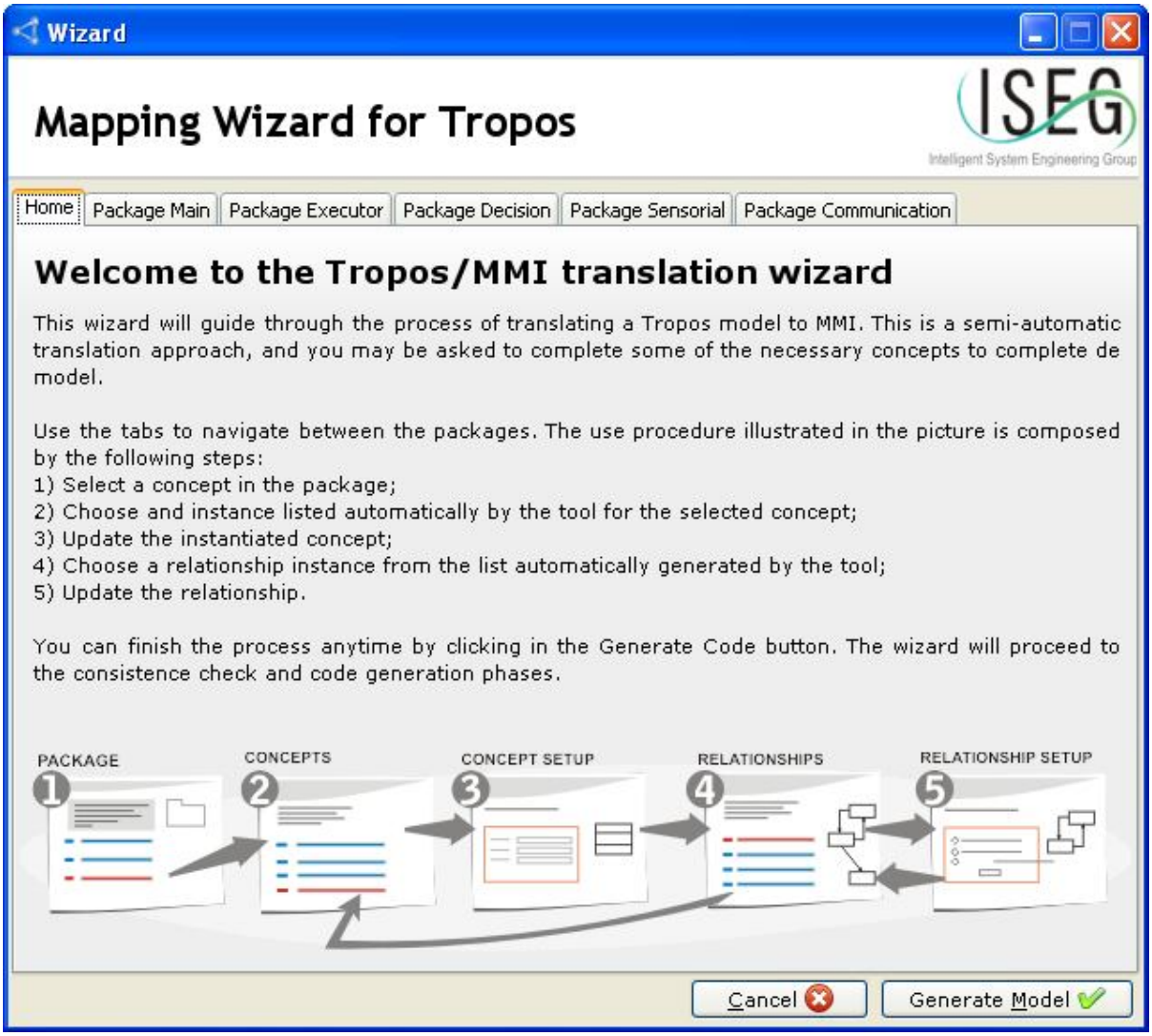


Figura 86 – Tela inicial de boas vindas ao assistente de importação.

Após a tela de boas vindas, é apresentada a tela inicial do conceito, sendo o agente o primeiro conceito, o qual lista todas as instâncias deste conceito encontradas na importação, conforme ilustrado na figura 87. Cada item pode ser selecionado para que o mapeamento seja completado. Esta interação é necessária quando não foi possível

mapear o conceito automaticamente, pois há conceitos que não estão presentes nas metodologias inseridas e este assistente completa-as para que se adéqüe ao MMI.

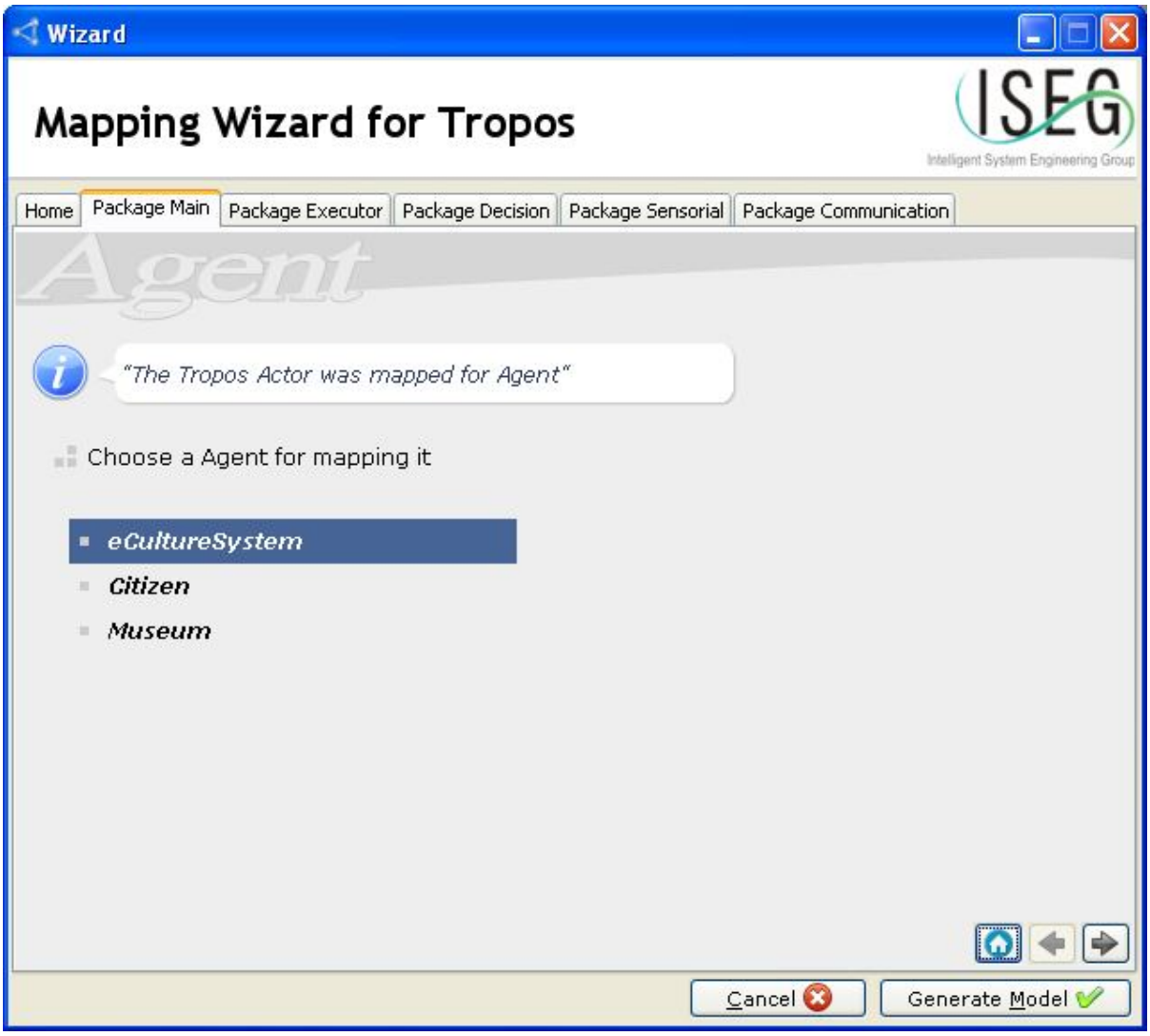


Figura 87 – Tela de índice do primeiro conceito a ser mapeado.

Uma vez que uma instância de um conceito foi selecionado, é aberto um formulário para que seja completado informações restantes, conforme ilustrado na figura 88. No exemplo da figura, foi selecionado o agente *Citizen*, e o assistente espera que seja selecionado o estado do agente e seja atribuído o ambiente no qual este agente está inserido. Para facilitar este processo, o *Wizard* possui ilustrações em suas telas das entidades que estão sendo mapeadas, para auxiliar a compreensão do usuário nas tomadas de decisões ou atribuições de informações.

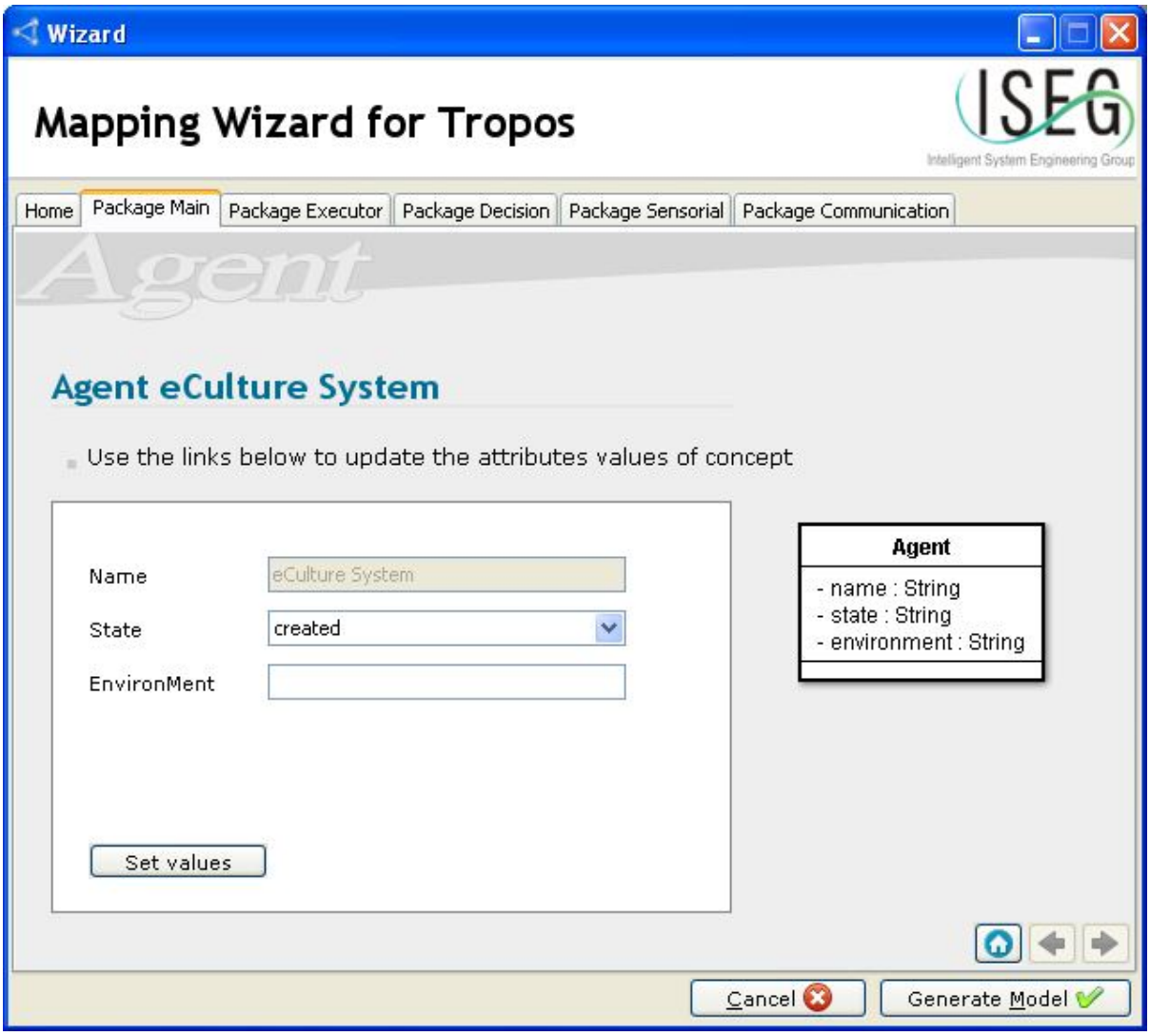


Figura 88 – Tela do formulário do conceito a ser mapeado

Após preenchido o formulário do conceito selecionado, é exibido uma tela dos relacionamentos em que este conceito está inserido, conforme ilustrado na figura 89. Este passo é de fundamental importância para o processo de importação, pois demonstra os mapeamentos já realizados automaticamente e espera a decisão para os relacionamentos que possuem mais de uma opção de mapeamento.

Nesta fase, lista-se todas os relacionamentos e estes são menus clicáveis, sendo que cada escolha leva a próxima tela de opção de preenchimento do relacionamento escolhido. É possível voltar pra esta tela para seleção de todos os itens, porém não são obrigatórios, e o usuário pode prosseguir para o próximo passo a qualquer momento.

Mesmo com este processo, há a possibilidade de o modelo não estar consistente pois o usuário não está obrigado a preencher todos os campos, sendo assim a verificação de consistência validará o modelo, e permitirá futuras atualizações de adequação.

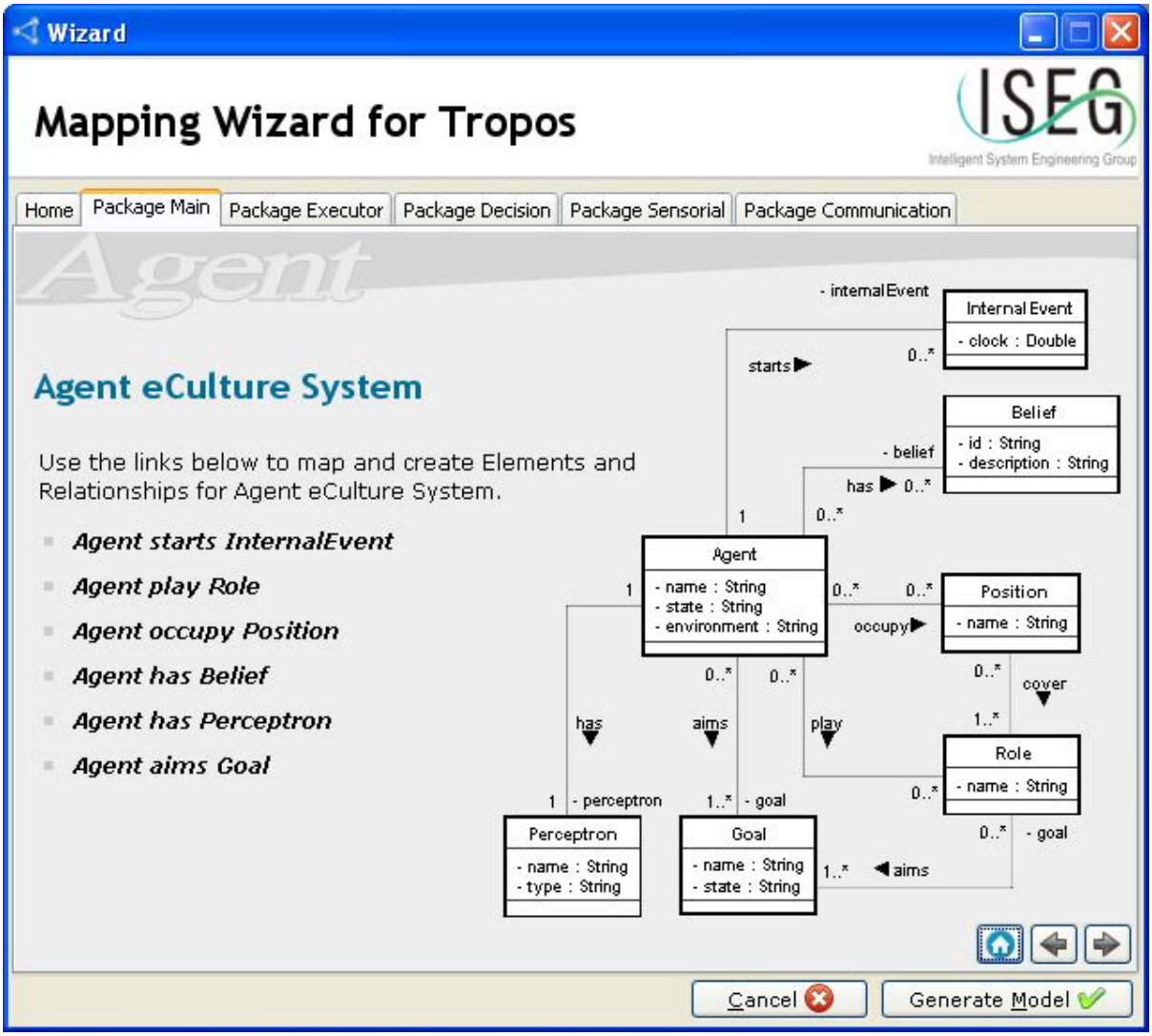


Figura 89 – Tela dos relacionamentos para determinado conceito.

A seleção de um relacionamento, conforme exposto anteriormente, exibe a tela para tomada de decisão de como este relacionamento será mapeado. A figura 90 traz um exemplo deste mapeamento, no qual o agente pode alcançar os objetivos listados de três formas: através de uma posição, de um papel ou relacionar o agente diretamente ao objetivo.

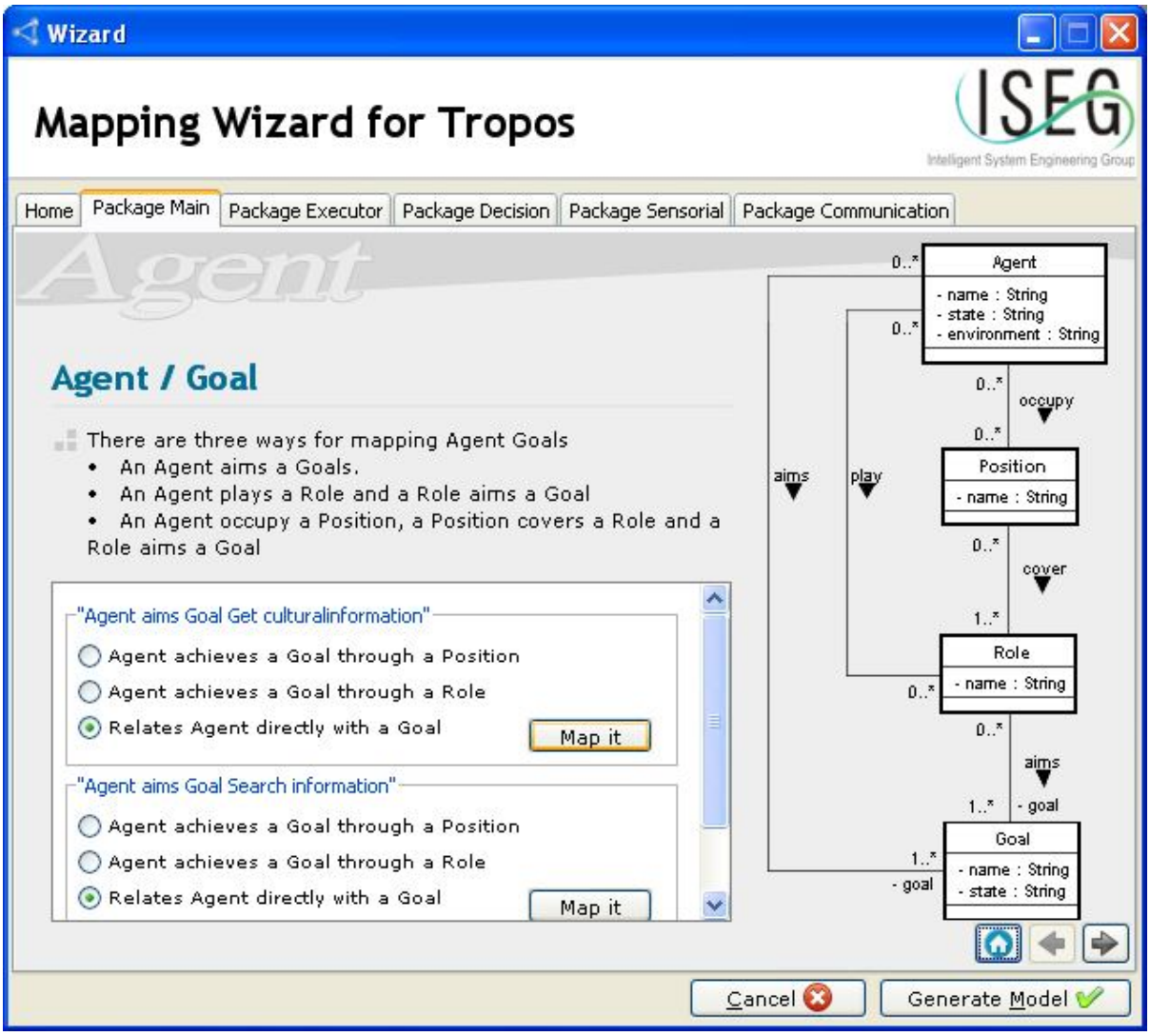


Figura 90 – Tela do mapeamento com interatividade do usuário.

Após realizar o mapeamento, será apresentado a tela da próxima entidade. Assim como no exemplo foi exibido o conceito de agente, em seguida será o conceito de objetivo, seguidos por papel, organização, posição, enfim, por todos os conceitos presentes no MMI.

6.6.3 Verificação de consistência e geração de código

Uma vez mapeado, o modelo de aplicação é gerado de acordo com o MMI e este é enviado à tela inicial do protótipo, conforme ilustrado na figura 91. Nesta tela é possível alterar qualquer entrada, com modificação dos conceitos apenas clicando em cima do conceito ou relacionamento escolhido. Além da alteração, é possível criar ou remover o conceito ou relacionamento.

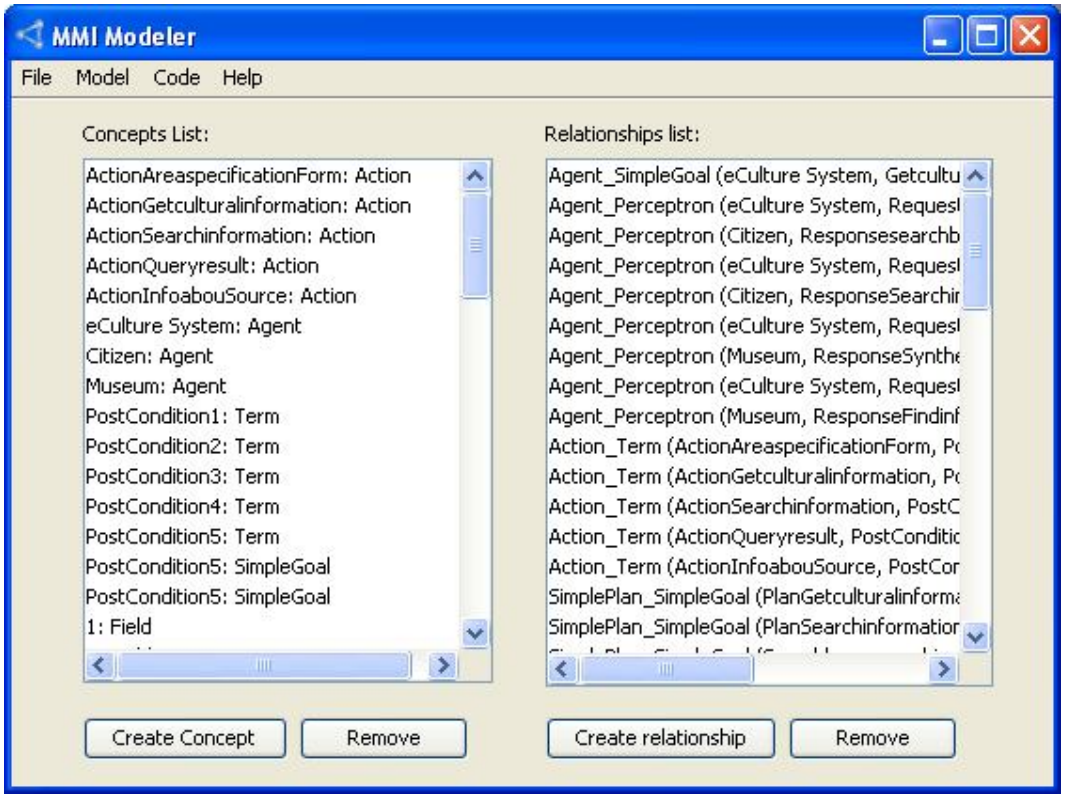


Figura 91 – Tela com o modelo mapeado e carregado.

Uma vez que o protótipo está com o modelo carregado e completo, esta ferramenta permite a checagem de consistência através da opção *model* e depois *check model*. Nessa checagem, são gerados dois arquivos dentro do subpacote *use.output*, são eles: *logErr.txt* e *logUse.txt*. O primeiro foi utilizado com maior freqüência durante o desenvolvimento do protótipo, pois apresenta erros na construção dos arquivos de entrada da ferramenta *USE*. O segundo arquivo apresenta erros na estrutura do modelo criado e erros de consistência do modelo com MMI e suas restrições de integridade. Nesse caso, podem ocorrer erros na construção do modelo e o conteúdo do arquivo é apresentado em uma interface do protótipo. Caso não sejam encontrados erros estruturais, o modelo está consistente e é exibido uma janela com todas as invariantes verificadas e o número de falhas, conforme ilustrado na figura 92.

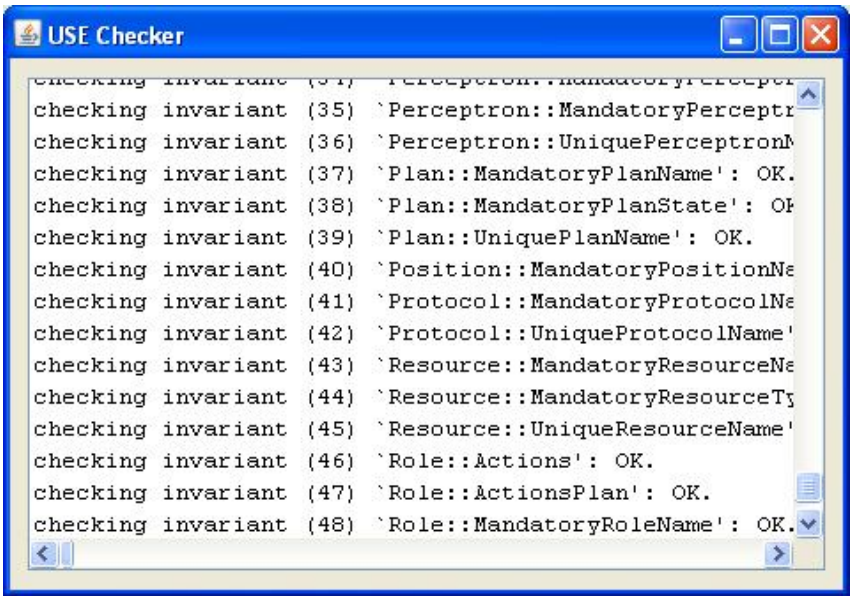


Figura 92 – Modelo Estruturalmente Consistente.

Tomando-se como base o modelo consistente com o meta-modelo criado, para a continuidade da apresentação, prossegue-se para o processo de geração de código para a plataforma *SemantiCore* por meio do menu *Code – Generate SemantiCore code,* e após gerado o código é apresentado uma mensagem de sucesso, conforme ilustrado na figura 93.

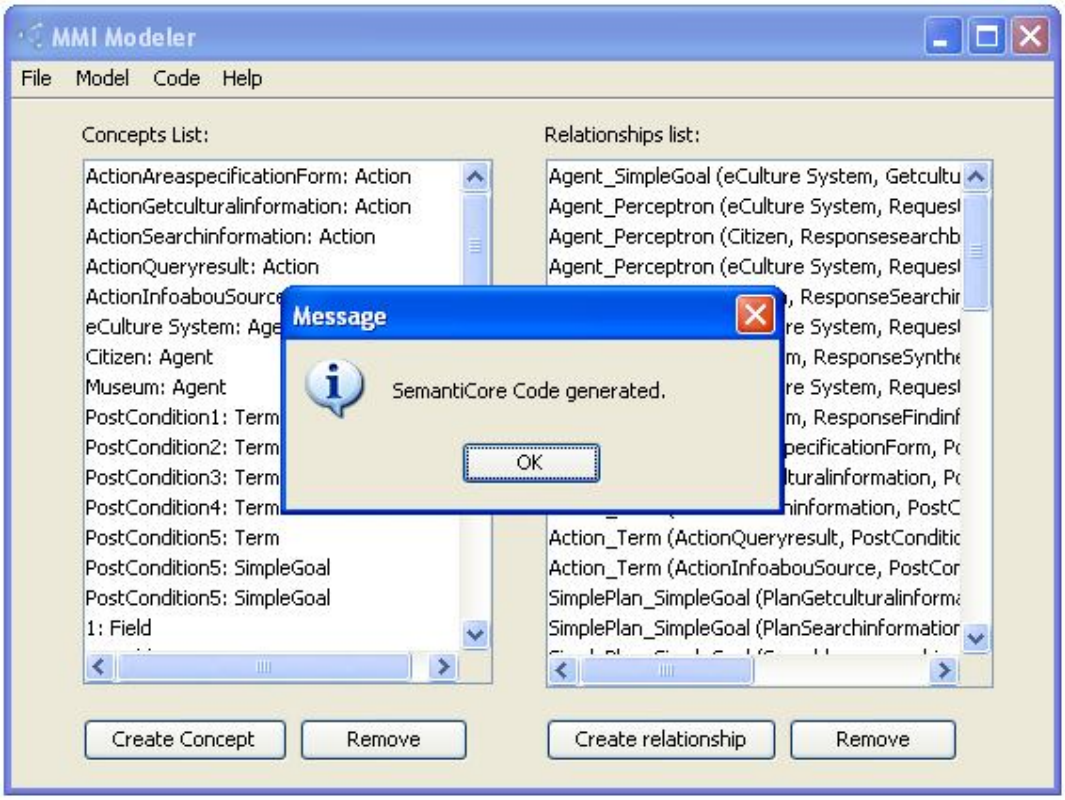


Figura 93 – Tela com o código gerado para a plataforma SemantiCore

Após gerado os esqueleto de código para a plataforma *SemantiCore* por meio do menu *Code – Generate SemantiCore code,* pode-se utilizar o menu *Code – Launch SemantiCore* para iniciar a plataforma com os agentes instanciados e sendo executados, pronto para utilizar os recursos disponíveis desta plataforma de implementação de SMA, conforme ilustrado na figura 94.

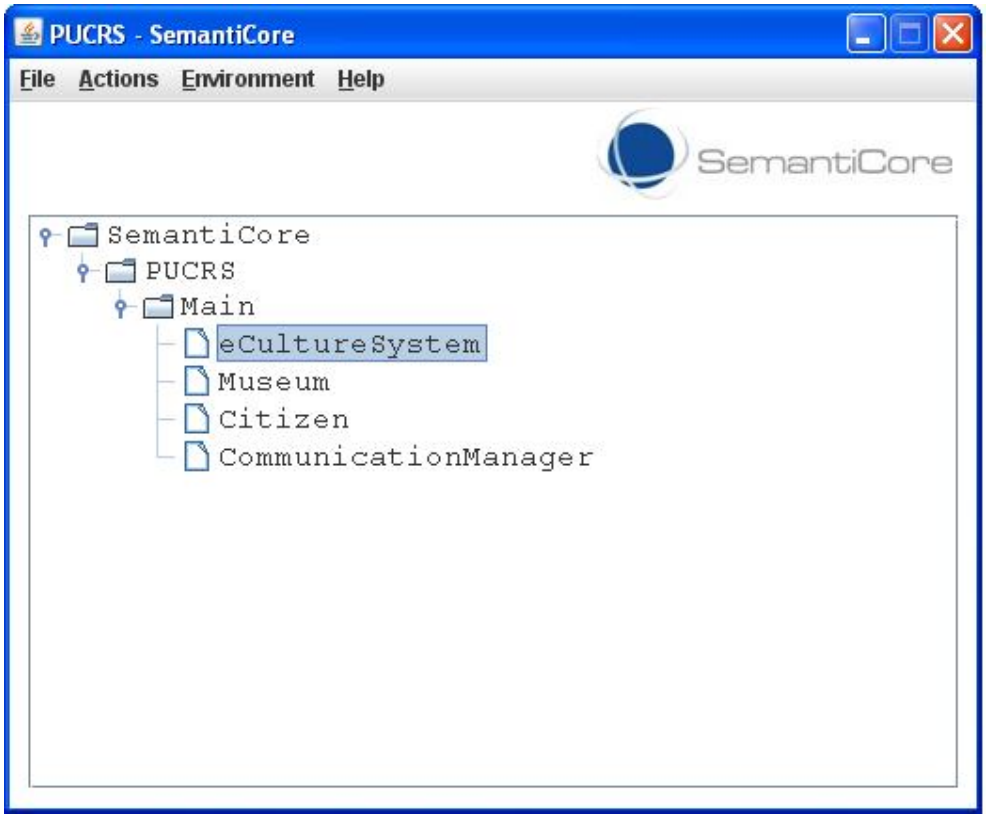


Figura 94 – Tela com do SemantiCore com os agentes mapeados.

6.7 Exemplo de Aplicação

Para demonstração da proposta desta pesquisa, utilizamos um exemplo apresentado em [BRE04] para demonstrar a metodologia Tropos. Este exemplo é um fragmento de uma aplicação real desenvolvida para o governo de Trentino (Província Autônoma de Trento, ou *PAT*). Este sistema, chamado de *eCulture* system, é um "*broker*" de informações culturais e serviços para a *PAT*, incluindo informações obtidas de museus, exibições e outras organizações culturais e eventos, o qual pode ser utilizado por diversos usuários, incluindo cidadãos de Trento e turistas procurando o que fazer, ou estudantes escolares pesquisando por conteúdos relevantes a seus estudos.

Na fase de requisitos iniciais, Tropos identifica os *stakeholders* e suas intenções. Para o exemplo do *eCulture* System, foram identificados os seguintes *stakeholders*:

- Província Autônoma de Trento (*PAT*), que é a agência do governo, com objetivo de incluir informações e serviços públicos, incrementar o turismo através de novas informações de serviços;
- Museu, que é o maior fornecedor de informações culturais, e possibilita uma interface de seu sistema para outros sistemas culturais ou de serviços;
- Visitante, que quer acessar as informações culturais antes ou durante sua visita a Trento;
- Cidadãos, os quais desejam acessar facilmente informações;

Durante o processo de mapeamento do modelo de Tropos gerado no modelador TAOM4E [PER04] cada ator é mapeado para um agente, possuindo este mapeamento baseado na utilização deste software que durante a modelagem, para a fase de arquitetura, é criado uma tabela de capacidades onde é convertido cada ator para agentes, sendo este processo adotado nesta pesquisa durante a fase de mapeamento de modelo Tropos para o metamodelo estendido proposto.

Na fase de requisitos finais, Tropos foca no sistema, neste caso no sistema *eCulture* System, juntamente com suas funções e qualidades. O sistema é representado por um ator e suas dependências com outros atores que juntos definem os requisitos funcionais e não funcionais. Na figura 95 [BRE04], Bresciane entre outros demonstra um conjunto de objetivos e *softgoals* que o ator *PAT* delega para o ator *eCulture* System, e também o diagrama de objetivos de *eCulture* System com as decomposições de objetivos e a forma de contribuição entre objetivos e/ou *softgoals.* Os objetivos e *softgoals* de Tropos são mapeados para o metamodelo estendido proposto com objetivo simples ou objetivo composto, ficando a cargo do desenvolvedor implementar o *softgoal* como objetivo, pois trata-se um requisito não-funcional.

O relacionamento de contribuição do Tropos para com o objetivo é mapeado para crença, no qual durante o processo de importação esta contribuição é armazenada em sua base de crenças, contendo quais objetivos contribuem e de que forma para outros objetivos.

Ainda nesta fase, há uma revisão do diagrama de ator após a introdução do ator na dependência. A figura 96 apresenta como a dependência é analisada dentro do diagrama de objetivo do sistema. Desta análise pode se observar: o objetivo “busca de informação” (*search information*) pode ser alcançado por diferentes planos: busca por área, busca por área geográfica, busca por palavras chaves e busca por período; é demonstrado a decomposição em subplanos para os quatro tipos de pesquisas, no qual o subplano “encontrar fontes de informação” depende do Museu para a descrição das informações a

serem providas e o subplano “sintetizar resultados” depende do Museu para consulta do resultado, e por fim, ao pesquisar informações sobre uma determinada área temática, o cidadão é obrigado a fornecer informações, utilizando um formulário de especificação de área.

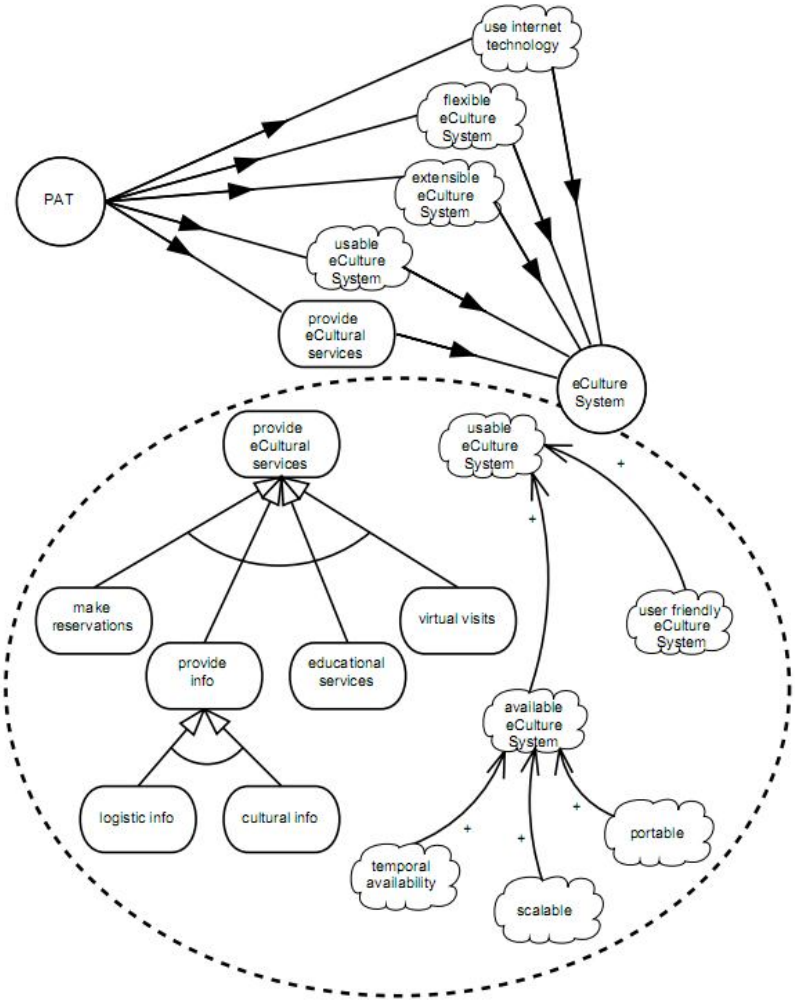


Figura 95 - Diagrama de ator e de objetivos do *eCulture* System [BRE04].

No processo de mapeamento para o metamodelo estendido proposto, o recurso área *especification form* é mapeado para recurso a ser relacionado com um ou mais planos. Para a realização dos relacionamentos, no momento da importação o protótipo proposto possibilita a associação com entidades já existentes ou criação de novas entidades. Assim, o protótipo lista os planos existentes ou possibilita a criação de novo plano para ser vinculado com este recurso, e conseqüentemente realizar a associação deste plano com uma ou mais ações e um objetivo.

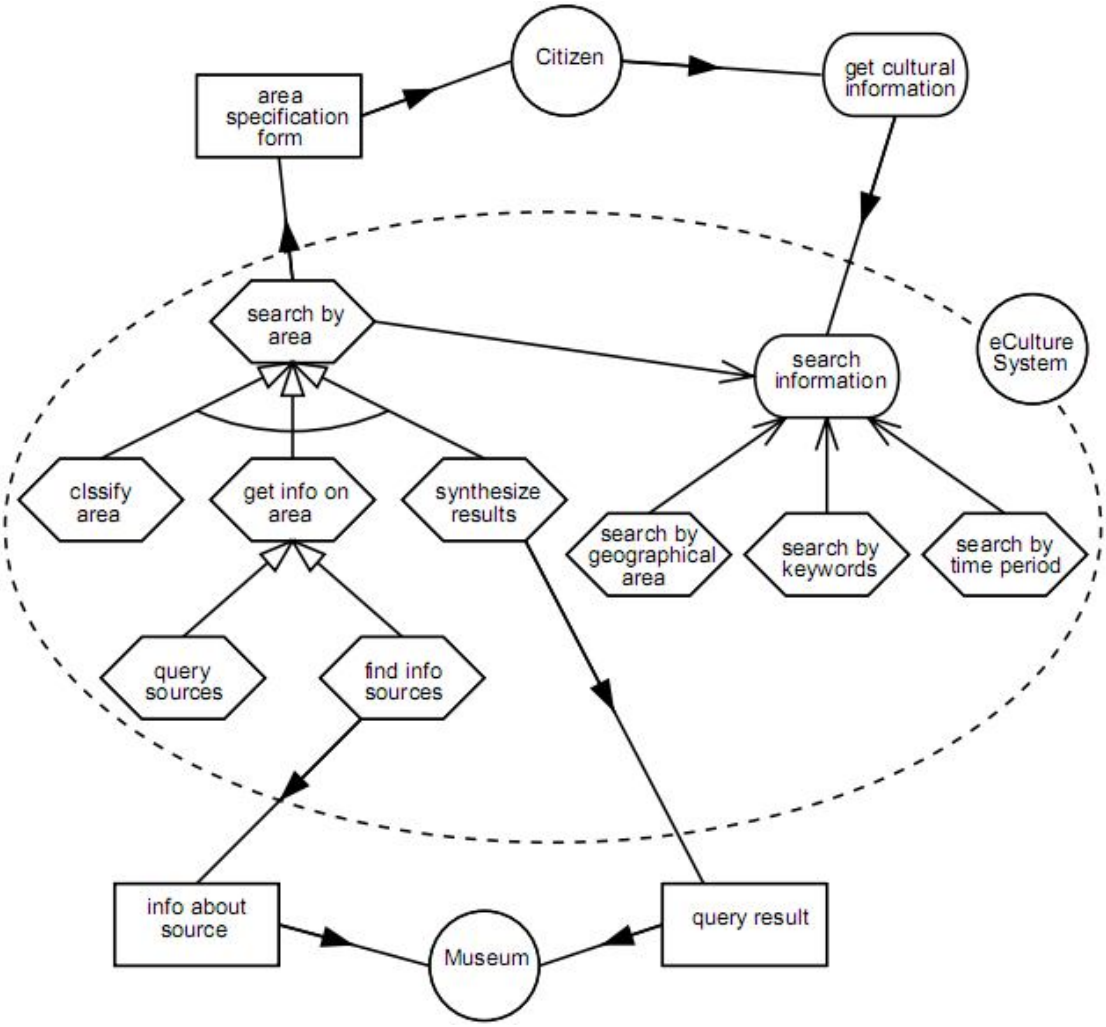


Figura 96 – Diagrama de objetivos e dependência entre atores [BRE04].

Como utilizamos a mensagem para o mapeamento de dependência de Tropos para o metamodelo estendido proposto, a relação de dependência entre o ator *eCulture* system e *Citizen*, o mapeamento trata este relacionamento como uma mensagem a ser disparada por uma ação, neste caso a mensagem é disparada por uma ação que compõe o plano search by área contendo como conteúdo da mensagem o recurso área *especification form*. Assim como anteriormente, a dependência do *depender Citizen* para o *dependee eCulture System* é mapeada utilizando mensagem e o *dependum* objetivo *get cultural information* é mapeado para o conteúdo da mensagem a ser disparada por uma ação presente na estrutura interna do agente remetente. Esta ação é criada ou selecionada e devidamente associada através do protótipo durante o processo de importação. Para este último mapeamento de dependência, pode se considerar o conteúdo da menagem como "O agente *Citizen* solicita que o agente *eCulture* System alcance seu objetivo *get cultural information*", pois o *dependum* está vinculado a estrutura interna do *dependee*, podendo

constatar esta afirmação ao se utilizar o modelador TAOM4E [PER04] para criação de dependência.

O relacionamento de meios-fins é mapeado para uma associação entre entidade. Assim o relacionamento do plano *search by área* para o objetivo *search information* é mapeado para o relacionamento *plan achieve goal*, bem como os planos *search by geografical area, search by keywords* e *search by time period.*

O relacionamento de decomposição *AND* ou *OR* é mapeado para plano simples ou plano composto e objetivos simples ou objetivo composto, e tanto o plano quanto objetivo possui um atributo *type* que identifica se esta decomposição é do tipo *AND* ou *OR*. Assim, o plano *search by area* é mapeado para um plano composto do tipo *AND* pelos planos simples *classify area, e synthesize results* e pelo plano composto do tipo *OR get info on area*, que por sua vez é composto pelos planos simples *query sources* e *find info sources*.

Os relacionamentos de dependência entre o ator *eCulture System* e *Museum*, pelos recursos info about source e *query result* são mapeados para envio de mensagens, através de ação ligados no primeiro relacionamento ao plano *find info sources* e no segundo ao plano *syntesize results* da estrutura interna do agente *eCulture System* contendo como conteúdo da mensagem consecutivamente: o agente *eCulture System* solicita o recurso *info about source* do agente *Museum* e o agente *eCulture System* solicita o recurso *query result* do agente *Museum*.

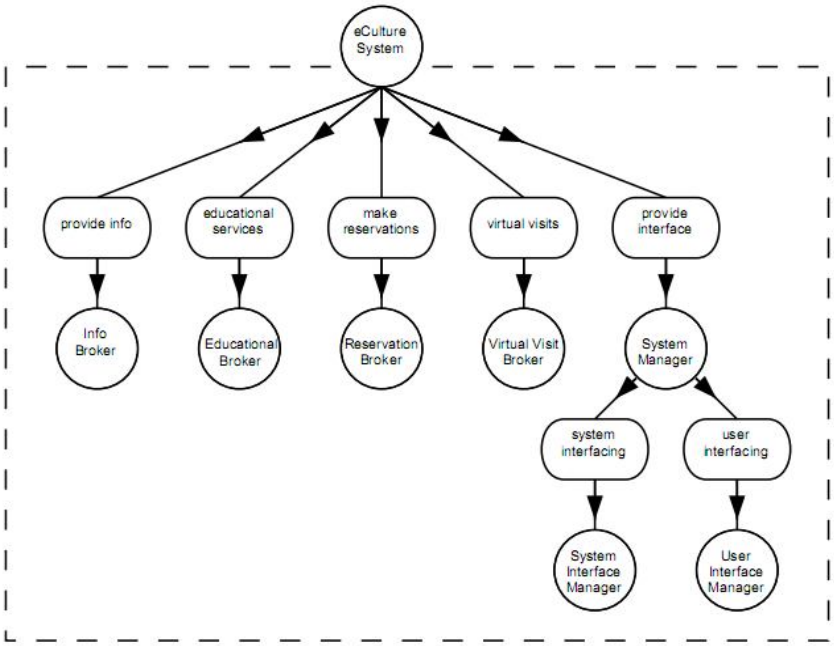


Figura 97 - Diagrama de Ator para *eCulture* system [BRE04].

A fase arquitetural define a arquitetura global do sistema em termos de subsistemas, representado por atores, interconectados entre si por um sistema de dependência, demonstrados na figura 97. Estes relacionamentos de dependência entre atores para alcance de objetivos formam a estrutura multiagentes, onde há a comunicação e cooperação entre diferentes agentes para alcance de objetivos particulares pertencentes à estrutura interna de cada agente. A dependência é mapeada através de mensagens ligadas a ação de cada agente remetente (tratado em Tropos como *depender*).

Desta forma, é realizada a importação de um modelo Tropos para o MMI intermediado pelo protótipo. Este modelo Tropos, na importação com o protótipo, foi gerado pelo software TAOM4E através de uma modelagem visual e salvo em um arquivo *XML* com descrição das entidades e relacionamentos modelados. Este arquivo é lido e através de um assistente de importação é mapeado como apresentado nesta seção, passo a passo.

6.7.1 Código gerado

Para ilustrar o escopo do código da plataforma SemantiCore gerado pelo protótipo, a partir da modelagem do exemplo exposto, abordaremos os arquivos gerados: *semanticoreconfig.xml* gerado na raiz do projeto, contendo os dados dos agentes que serão instanciados na plataforma *SemantiCore*; um arquivo *semanticoreinstantiation.xml* gerado na raiz do projeto, contendo as informações dos *hotspots* da plataforma; a classe *DependencySensor* do tipo *Sensor* gerada no pacote *semanticore.agent.sensorial.hotspots*; a classe *DependencyMessage* do tipo *SemanticMessage* geradas no pacote *semanticore.domain.model.hotspots*; a classe *eCultureSystem* do tipo *SemanticAgent* geradas no pacote *application*; a classes *eCultureSystemDecision* do tipo *DecisionEngine* geradas no pacote *semanticore.agent.decision.hotspots*; a classe *SynthesizeResults* do tipo *ActionPlan* geradas no pacote *semanticore.domain.actions.lib*; e as classes *ActionSynthesizeResults* do tipo *Action* geradas no pacote *semanticore.domain.actions.lib*.

Os arquivos *semanticoreconfig.xml* e *semanticoreinstantiation.xml* e as classes *DependencySensor*, *DependencyMessage* e *SynthesizeResults* puderam ser geradas automaticamente em sua totalidade.

Para a classe *eCultureSystem* foram gerados automaticamente os termos, as sentenças e as regras associadas ao agente. Assim como foram criadas as associações

com as classes *Sensor*, *ActionPlan* e a associação entre as classes *ActionPlan* e *Action.* Além disso, no Semanticore, cada ação deve estar associada à pelo menos um plano, e no MMI, uma ação pode ser independente de planos. Caso isso ocorra, o trecho de código com a criação e o uso de uma classe *ActionPlan* não é gerado automaticamente. Um pequeno trecho do código desta classe ilustra este mapeamento na figura 98.

```
public class eCultureSystem extends SemanticAgent{
      public eCultureSystem ( Environment env, String agentName, String arg){
            super (env, agentName, arg);
      }
      protected void setup(){
             addSensor ( new DependencySensor ("Percebe as Dependências") );
             addFact ( new SimpleFact("simpleGoal Portable", "Contribrutes +",
"SimpleGoal Available eCulture System") );
            ActionPlan synthesizeResult = new ActionPlan(“Synthesize Results”);
            synthesizeResult.addAction(new ActionSynthesizeResult());
      }
}
```

Figura 98 – Trecho de código da classe *eCultureSystem* gerada.

Para a classe *eCultureSystemDecision* foi gerada apenas a estrutura geral da classe, pois as demais estruturas do mecanismo decisório serão dependentes de cada aplicação modelada. Dentre o código gerado, está inclusa a assinatura do método *decide*, responsável por avaliar os fatos que a classe *SemanticAgent* recebe do ambiente. Um trecho abreviado do código desta classe está demonstrado na figura 99.

```
public class eCultureSystemDecision extends DecisionEngine{
      public Vector<Object> decide (Object facts){
            Vector<Object> vector = new Vector<Object> ();
            if( facts instanceof SemanticMessage){

                  if( ((SemanticMessage) facts).getContent() instanceof
ComposedFact ){
                        SimpleFact simpleFactA = (SimpleFact) ( (ComposedFact)
( (SemanticMessage) facts ). getContent())).getTerm1();
                        SimpleFact simpleFactB = (SimpleFact) ( (ComposedFact)
( (SemanticMessage) facts ). getContent())).getTerm2();
                  }
                  if ( simpleFactA.getpredicate().equalsIgnoreCase("Constributes
+")){
                        SimpleFact m = new SimpleFact("simpleGoal
Portable","Contributes +","SimpleGoal Available eCulture System");
                        ( (eCultureSystem) this.getOwner() ).addFact(m);
                  }
            }
      }
```

```
}
```

Figura 99 – Trecho de código da classe *eCultureSystemDecision* gerada.

Para a classe *ActionSynthesizeResults* foram geradas automaticamente as mensagens associadas à ação assim como as crenças geradas pela mesma. O campo *content* (conteúdo da mensagem) associado a uma mensagem (extensão da classe *SemanticMessage*) não foi gerado, pois esse terá um valor diferente para cada execução da aplicação. Um trecho do código desta classe está demonstrado na figura 100.

```
public class ActionSynthesizeResults extends Action{
      public ActionSynthesizeResults (){
            super ("Synthesize Results", null, null);

      }
      public void exec(){
            SimpleFact fact = new SimpleFact("SimplePlan Synthesize
Result","Dependency","Resouce Query Result");
            DependencyMessage message = new DependencyMessage(eCultureSystem,
Museum, fact, "fipa-sl", null, null);
            transmit(message);
      }
}
```

Figura 100 – Trecho do código da classe *ActionSynthesizeResults* gerada.

6.8 Considerações

Nesse capítulo foi detalhado a implementação e exemplo de uso, apresentado com as ferramentas utilizadas, o mapeamento do MMI para o SemantiCore, o desenvolvimento do protótipo e sua especialização, o padrão de representação dos modelos do Tropos e do MMI, a utilização do protótipo. e um exemplo de aplicação.

Para a composição do protótipo, foram utilizadas as ferramentas USE para compilação das restrições OCL, verificando assim a consistência do modelo e *Velocity* para geração de código integrado ao protótipo. Além destas ferramentas foi desenvolvido um *Wizard* para importação de modelos de outras abordagens para o MMI, que neste trabalho foram mapeados os modelos de Tropos gerados pela ferramenta TOM4E. Após foi apresentado os quatros passos utilizados para o uso do protótipo, chamado de MMI4E.

Após, foram apresentados o mapeamento do MMI para o Semanticore. Neste mapeamento as entidades, atributos e relacionamentos do MMI foram mapeados para

código da plataforma SemantiCore. Entre os conceitos mapeados estão o agente, objetivo, recurso, percepção, plano, ação, termo, sentença, regra, mensagem, protocolo, campo e os relacionamentos plano tem recurso, agente possui crença, agente possui percepção, plano alcança objetivo, plano composto agrega plano, plano é composto por ação, crença controla plano, ação gera crença, crença controla ação, ação publica mensagem, regra tem crença antecedente, regra tem crença conseqüente, sentença opera crença, percepção avalia mensagem, protocolo agrega mensagem, mensagem é composta por campo, ação segue ação e campo agrega campo.

Em seguida, foi apresentado o desenvolvimento do protótipo, abordando os seus conceitos e estrutura geral através dos pacotes, diagramas de classes e trechos de código. Além disso, foi apresentado os passos a serem tomadas para a especialização deste protótipo possibilitando cobertura para novos modelos de diferentes abordagens e geração de códigos para outras plataformas de implementação.

Este capítulo apresenta, também, o padrão de representação dos modelos de Tropos gerados pelo TAOM4E e do modelo do MMI. É descrito os elementos que compõem estes padrões de representação de modelo, permitindo assim o mapeamento de um modelo para o outro.

A utilização do protótipo é apresentada em detalhes através da visualização de suas interfaces. Inicialmente é apresentada a tela inicial, os menus e suas funcionalidades. Após, é apresentado os passos para a importação dos modelos e o assistente de importação. O assistente de importação é demonstrado através de suas telas, passando por índice do conceito, formulário do conceito, índice do relacionamento e formulário do relacionamento. E por fim é apresentado o processo de verificação de consistência e geração do código para a plataforma SemantiCore.

Um exemplo de aplicação é apresentado para ilustrar a abordagem desta proposta. Como exemplo foi utilizado um fragmento de uma aplicação real desenvolvida para o governo da Província Autônoma de Trento, sendo este um exemplo de uso do Tropos publicado na literatura. Após o exemplo, é demonstrado o código gerado pelo protótipo para a plataforma de implementação SemantiCore.

Como principais contribuições têm-se a possibilidade de inserir novos conceitos não abordados na metodologia Tropos, embora presente na literatura, durante o processo de importação. O assistente mapeia os conceitos e possibilita a criação de novas entidades e relacionamentos tais como crenças, pós e pré-condições para planos e ações, adição de ações que compõe uma ação, mensagens, percepções, entre outros.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A pesquisa realizada busca possibilitar uma integração de soluções que abrangem metodologias, linguagens de modelagem e plataformas de desenvolvimento de SMAs, bem como os meta-modelos inerentes destas abordagens, junto ao MRIA, apresentado no capítulo 3. Além disso, é proposta uma extensão ao protótipo de Santos, para a geração do esqueleto de código de aplicações para a plataforma de implementação de SMAs, desconsiderando aspectos específicos de cada abordagem pesquisada.

Neste trabalho, foram pesquisadas as metodologias MASUP, Tropos, Prometheus e Ingenias, e as linguagens de modelagem de agentes ANote, MAS-ML e Message, além das plataformas de implementação *SemantiCore*, Jason e Jack. Dentre os estudos realizados, apresentou-se uma análise comparativa dos conceitos das metodologias e linguagens de modelagem, com enfoque nas entidades e relacionamentos que formam o metamodelo de cada abordagem, bem como a utilização de notações visuais, quando presente.

No MMI foram mesclados os conceitos de sistemas multiagentes aos conceitos da estrutura interna de um agente para suportar a metodologia Tropos. Foram adicionados os conceitos de Posição e Organização, e foram alterados os conceitos de objetivo, plano e o relacionamento entre objetivo e agente. Este metamodelo suporta as entidades e relacionamentos da metodologia Tropos, através de um mapeamento apresentado no capítulo 5.

Para a proposta do metamodelo estendido, são consideradas as características tratadas na literatura e nas metodologias estudadas. Assim, torna-se possível o mapeamento destas metodologias estudadas para o metamodelo proposto. Neste processo, este metamodelo absorve os conceitos presentes nas metodologias quando estendido. Após, busca-se um mapeamento direto para a geração de código fonte para plataformas de sistemas multiagentes.

Além disso, também foram estudadas duas soluções que buscam unificar conceitos de entidade e relacionamentos de algumas metodologias através de um Metamodelo Unificado [PAD08], e padronizar símbolos de notações visuais através da Notação Unificada [BER05]. Segundo Padghan [PAD08], uma notação visual única Constitui-se em um primeiro passo para elevar o nível de maturidade de desenvolvimento de SMAs. Este nível de maturidade já é constatado na programação orientada a objetos, e um metamodelo unificado torna possível abordar diferentes metodologias, focando-se no processo de desenvolvimento, uma vez que as entidades e relacionamentos são comuns.

Estas propostas buscam integrar soluções ao desenvolvimento de SMAs, porém não apresentam um mapeamento direto para geração de código fonte para plataformas de sistema multiagentes através de um processo automático de importação, verificação e geração de código.

Do estudo realizado, as maiores contribuições são: a síntese das abordagens das metodologias e linguagens de modelagem estudadas, com enfoque no metamodelo de cada abordagem; o mapeamento dos conceitos de cada entidade e relacionamento de todas as metodologias e linguagens de modelagem pesquisadas, bem como a tabulação dos diferentes símbolos utilizados na notação visual de cada abordagem, o que torna possível o mapeamento entre elas e a interoperabilidade entre os diferentes meta-modelos; a constatação e demonstração da possibilidade de extensões ao metamodelo que possam dar cobertura a todos estes conceitos, trazendo independência entre o processo de desenvolvimento do software para plataformas de implementação; a extensão ao protótipo proposto por Santos para suporte ao MMI, permitindo a importação de modelos do ambientes de modelagem TAOM4E para o MMI, verificação de consistência através de novas adições de restrições de integridades e novos mapeamentos de adequação ao metamodelo estendido para geração de esqueleto de código para a plataforma *SemantiCore*.

Contudo, surgem diversas possibilidades de trabalhos futuros para esta pesquisa. A principal continuação deste trabalho é a adoção das outras metodologias e linguagens de modelagem já mapeadas, resultando em novas extensões deste metamodelo, com intuito de suportar um número maior de soluções voltadas ao desenvolvimento de SMAs. Uma vez que os conceitos das entidades e relacionamentos destas soluções já estão mapeados, a extensão desta proposta para abranger as demais linguagens e metodologias é facilitada, contribuindo para uma interoperabilidade entre os meta-modelos destas abordagens evidenciando os conceitos que não são abrangidos, porém estão presentes na literatura.

Uma vez que os símbolos das notações diagramáticas estão mapeados, torna-se mais fácil a construção de um ambiente visual que permita a criação dos modelos com notação diagramática para o metamodelo, auxiliando a construção e a visualização de diferentes modelos pelo uso de diagramas com importação, mapeamento e comparação destes símbolos, a fim de se alcançar um ambiente visual com símbolos que estejam coerentes com as notações já utilizadas nas abordagens existentes.

Por fim, o protótipo estendido é adaptável para receber novas importações de diferentes metodologias e estender as regras de restrições para verificação de seus

modelos. Esta adaptação é possível pois há flexibilidade nos arquivos para geração de código, sendo este um *template,* o qual permite gerar código para a plataforma de implementação desejada.

REFERÊNCIAS

[APA09a] Apache License. “Licensa de uso para os Projetos Apache”. Capturado em: http://www.apache.org/licenses/LICENSE-2.0, Julho 2009.

[APA09b] Apache Software Foundation. “The Apache Velocity Project". Capturado em: http://velocity.apache.org/engine/, Julho 2009.

[APA09c] Apache Software Foundation. “The Apache Velocity Tools Project”. Capturado em: http://velocity.apache.org/tools/releases/1.4/, Julho 2009.

[ARG09] ARGO/UML. “Ferramenta livre para modelagem UML”. Capturado em: http://argouml.tigris.org/, Julho 2009.

[BAK97] Baker, A.; Chauhan, D. “A java-based agent framework for multiagent systems. Development and Implementation”. In: International Conference on Autonomous Agents, 1998, pp. 100-107.

[BAS05] Bastos, R. M.; Ribeiro, M. B. “MASUP: An Agent-Oriented Modeling Process for Information Systems”. In: Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications, 2005, pp. 19-35.

[BEL07] Bellifemine, F. L.; Caire, G.; Greenwood, D. “Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)”. Wiley, 2007, 300p.

[BER02] Bernon, C.; Gleizes, M. P.; Picard, G.; Glize, P. “The ADELFE methodology for an intranet system design.” In: P. Giorgini, Y. Lespérance, G. Wagner, & E. Yu (Eds.), Agent-Oriented Information Systems, 2002, pp. 1-15.

[BER04] Bergenti, F.; Gleizes, M.; Zambonelli, F. “Methodologies and Software Engineering For Agent Systems – The Agent-Oriented Software Engineering Handbook”. Kluwer Academic Publishers, 2004, 505p.

[BER04a] Bernon, C.; Cossentino, M.; Gleizes, M.; Turci, P.; Zambonelli, F. “A Study of some Multi-Agent Meta-Models”. In: 5th International Workshop (AOSE), 2004, pp. 62-77.

[BER05] Bernon, C.; Conssentino, M.; Pavón, J. “Agent-oriented software engineering: The Knowledge Engineering Review”. Cambridge University Press, 2005, pp. 99–116.

[BLO04] Blois, M.; Lucena, C. “Multi-agent Systems and the Semantic Web - The SemanticCore Agent-based Abstraction Layer” In: ICEIS – International Conference on Enterprise Information Systems, 2004, pp. 263 – 270.

[BLO07] Blois, M.; Escobar, M.; Choren, R.; “Using agents and ontologies for application development on the semantic web”. Journal of the Brazilian Computer Society, vol. 13, Jun 2007, pp. 35-45.

[BOR07] Bordini, R. H.; Hübner, J. F.; Wooldridge, M. “Programming multi-agent system in Agent Speak using Jason”. Wiley Series in Agent Technology, 2007, 292p.

[BRE04] Bresciane, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. “Tropos: An Agent-Oriented Software Development Methodology”. In: Autonomous Agents and Multi-Agent Sytems, 2004, pp. 203–236.

[CAI01] Caire, G.; Chainho, F.; Evans, R. "Agent-oriented analysis using Message/UML" In: Agent-Oriented Software Engineering, Wooldridge, M., Weiss, G. and Ciancarini, P., Eds., Second International Workshop, AOSE 2001, LNCS 2222 Springer, 2001, pp. 119-135.

[CER03] Cervenka, R. “Modeling Notation Source - MESSAGE (Methodology for Engineering Systems of Software Agents)”. Relatório Técnico, Foundation for Intelligent Physcial Agents (FIPA), 2003, 4p.

[CHO04] Choren, R.; Lucena, C. “Modeling Multi-agent systems with ANote”. Software and Systems Modeling, Vol. 4, May 2005, pp. 199-208.

[COS05] Cossentino, M. “From Requirements to Code with the PASSI Methodology”. Idea Group Publishing, 2005, pp. 79-106.

[DEL99] Deloach, S. A. “Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems”. In: Agent-Oriented Information Systems '99 (AOIS'99), 1999, 9p.

[DEL05] DeLoach, S. A. “Engineering organization-based multiagent systems”. In: Garcia, A.F., Choren, R., de Lucena, C.J.P., Giorgini, P., Holvoet, T., Romanovsky, A.B., eds.: SELMAS. Volume 3914 of Lecture Notes in Computer Science., Springer, 2005, pp. 109–125.

[ESC06] Escobar, M.; Lemke, A. P.; Ribeiro, M. B. “SemantiCore 2006: Permitindo o Desenvolvimento de Aplicações baseadas em Agentes na Web Semântica”.

In: Workshop on Software Engineering for Agent-Oriented Systems (SEAS), 2006, pp. 72-82.

[ESC07] Escobar, M.; Ries, L. H.; Lemke, A. P.; Ribeiro, M. B. “SemantiCore Mobile - Permitindo o Desenvolvimento de Aplicações”. In: I Workshop on Pervasive and Ubiquitous Computing - WPUC, 2007, 6p.

[EVA01] Evans, R.; Kearney, P.; Caire, G.; Garijo, F. J.; Gomez-Sanz, J. J.; Pavon, J.; Leal, F.; Chainho, P.; Massonet, P. “MESSAGE: Methodology for Engineering System of Software Agents”. Informação Técnica, Eurescom, 2001, 75p.

[FER99] Ferber, J. “Multi-agent systems: an introduction to distributed artificial Intelligence”. Addison-Wesley, 1999, 528p.

[FIN94] Finin, T.; Fritzson, R.; Mckay, D.; Mcentire, R. “KQML as an Agent Communication Language”. In: Proceedings of the 3rd International Conference on Information and Knowledge Management, 1994, pp. 456 - 463.

[FIP08] FIPA ACL. “Foundation for Intelligent Physical Agents”. Capturado em: http://www.fipa.org, Novembro 2008.

[GEO86] Georgeff, M. P.; Lansky, A. L. “Procedural Knowledge”. In: Proceedings of the IEEE Special Issue on Knowledge Representation, 1986, pp. 1383-1398.

[GOM02] Gomez-Sanz, J. J. “Modelado de Sistemas Multi-Agentes”. Tese de Doutorado, Universidead Complutense de Madrid, 2002, 255p.

[GOM04] Gomez-Sanz, J.; Fuentes, R. “Agent Oriented Software Engineering with INGENIAS” International Journal of Web Engineering and Technology, vol. 1, Jun 2004, pp. 47.

[GUD06] Gudgin, M.; Hadley, M.; Mandelsohn, N.; Moreau, J.; Nielsen, H. “SOAP Version 1.2, 2006”. Capturado em: http://www.w3c.org/2000/xp/Group/2/06/LC/soap12-part1.html, Novembro 2008.

[HOW01] Howden, N.; Rönnquist, R.; Hodgson, A.; Lucas, A. “JACK Intelligent Agents – Summary of an Agent Infrastructure”. In: 5th International Conference on Autonomous Agents, 2001, 6p.

[HUB04] Hübner, J. F.; Bordini, R. H.; Vieira, R. “Introdução ao Desenvolvimento de Sistemas Multiagentes com Jason”. In: XII Escola de Informática da SBC - Paraná, 2004, pp. 51-89.

[IGL98] Iglesias, C. A.; Garijo, M.; Centeno-Gonzalez, J.; Velasco, J. R. “Analysis and Design of Multiagent Systems using MAS-CommonKADS”. In: Intelligent Agents IV Agent Theories, Architectures, and Languages, 1998, pp. 313-327.

[JUC01] Juchem, M.; Bastos, R. M. “Engenharia de Sistemas Multiagentes: Uma Investigação sobre o Estado da Arte”. Relatório Técnico, Faculdade de Ciências Exatas da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), 2001, 45p.

[KAR04] Karim, S.; Heinze, C.; Dunn, S. "Agent-based mission management for a UAV", In: Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004, pp. 481-486.

[KRU04] Kruchten, P. “The Rational Unified Process: An Introduction”. Addison-Wesley Longman, 2004, 320p.

[MES08] MESSAGE Website. “Methodology for Engineering Systems of Software AGEnts”. Capturado em: http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm, Novembro 2008.

[MUL96] Müller, J. P. “The design of Intelligent Agents: A Layered Approach”. Springer-Verlag, 1996, 227p.

[NUM07] Nunes, I. O. “Implementação do Modelo e da Arquitetura BDI”. Monografias em Ciência da Computação, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2007, 17p.

[NWA96] Nwana, H. S. “Software Agents: An overview. The knowledge Engineering Review”. Cambridge University Press, 1996, pp. 1-40.

[OCL08] OCL Website. “UML 2.0 OCL SPECIFICATION”. Capturado em: http://www.omg.org/docs/ptc/05-06-06.pdf, Dezembro 2008.

[ODE00] Odell, J.; Parunak, H.; Bauer, B. "Extending UML for Agents". In: Wagner, G., Lesperance, Y. and Yu, E. Proceedings of the Agent-Oriented Information Systems Workshop, 2000, pp. 3-17.

[OMG08] OMG. “Object Management Group: UML Resource Page”. Capturado em: http://www.uml.org/, Novembro 2008.

[PAD02] Padgham, L.; Winikoff, M. “Prometheus: A Methodology for Developing Intelligent Agent”. In: 3th International Workshop on Agent Oriented Software Engineering, at Autonomous Agents and Multi-Agent Systems (AAMAS), 2002, 12p.

[PAD02a] Padgham, L; Winikoff, M. “Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents”. In: Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, 2002, pp. 97-108.

[PAD05] Padgham, L.; Winikoff, M. “Prometheus: A Practical Agent-Oriented Methodology”. Idea Group Publishing, 2005, pp. 107-135.

[PAD08] Padgham, L.; Winikoff, M.; DeLoach, S.; Cossentino, M. “A Unified Graphical Notation for AOSE”. In: Proc. of the Ninth International Workshop on Agen-Oriented Software Engineering (AOSE-2008) at The Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2008, pp. 116-130.

[PAV03] Pavón, J.; Gomez-Sanz, J. “Agent Oriented Software Engineering with INGENIAS”. In: Marik V, Müller J, and Pechoucek M (Eds.) Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS. Lecture Notes in Artificial Intelligence 2691, 2003, pp. 394-403.

[PDT08] PDT. “Prometheus Development Tool”. Capturado em: http://www.cs.rmit.edu.au/agents/pdt/, Novembro 2008.

[PER04] Perini, A.; Susi, A. “Developing Tools for Agent-Oriented Visual Modeling”. In: Proceedings of the Second German Conference, 2004, pp. 169–182.

[RIB02] Ribeiro, M. B. “Web Life: Uma arquitetura para a implementação de sistemas multi-agentes para a Web”. Dissertação (Mestrado em Informática), Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2002, 204p.

[RUM96] Rumbaugh, J. “Notation notes: Principles for choosing notation”. Journal of Object Oriented Programming vol. 9, May 1996, pp. 11–14.

[RUS04] Russel, S.; Norvig, P. “Inteligência Artificial (tradução da segunda edição do original)”. Campus/Elsevier, 2004,1040p.

[SAN08] Santos, D. R. “Um Meta-modelo para Representação Interna de Agentes de Software”. Dissertação de Mestrado da Faculdade de Ciências Exatas, Pontifícia, Universidade Católica do Rio Grande do Sul – PUCRS, 2008, 153p.

[SER05] Serrano, J. M.; Ossowski, S.; Saugar, S. “The RICA metamodel: an organizational stance on agent communication languages”. In: Agentlink III AOSE TFG 2, 2005, 6p.

[SIL04] Silva, V. T.; Choren, R.; Lucena, C. “Using the MAS-ML to Model a Multi-Agent System”. In: Software Engineering for Multi-Agent Systems II : Springer-Verlag, 2004, pp. 129-148.

[SIL04a] Silva, V.; Choren, R.; Lucena, C. "A UML Based Approach for Modeling and Implementing Multi-Agent Systems" In: Proceeding of the third International Conference on Autonomous Agents and Multi-Agents Systems, 2004, pp. 914-921.

[SIL08] Silva, V. T; Choren, R.; Lucena, C. J. P. “MAS-ML: A Multi-Agent System Modelling Language”. In: International Journal of Agent-Oriented Software Engineering, 2008, pp. 382 - 421.

[SIL08a] Silva, M. J. “U-TROPOS: uma proposta de processo unificado para apoiar o desenvolvimneto de software orientado a agentes”. Dissertação de Mestrado, Universidade Federal do Pernambuco, 2008, 191p.

[SUS05] Susi, A; Perini, A; Mylopoulos, J. “The Tropos Metamodel and its Use”. Informatica,. Vol. 29, Nov 2005, pp 401-408.

[USE09] USE. “A UML-based Specification Environment”. Capturado em: http://www.db.informatik.uni-bremen.de/projects/USE/, Julho 2009.

[WAR03] Warmer, J.; Kleppe, A. “The Object Constraint Language: Getting Your Models Ready for MDA”. Addison-Wesley, 2003, 240p.

[WEI01] Weiss, G. “Agent Orientation in Software Engineering. Knowledge Engineering Review”. Cambridge University Press, Vol. 16, Aug 2001, pp 349–373.

[WIN04] Winikoff, M.; Padgham, L. “The Prometheus Methodology”. Kluwer Publishing, 2004, pp. 217-234.

[WOO00] Wooldridge, M.; Jennings, N. R.; Kinny, D. “The Gaia methodology for agent-oriented analysis and design”. Journal of Autonomous Agent and Multi-Agent Systems, Vol. 3, Set 2000, pp. 285-312.

[WOO02] Wooldridge, M. “An Introduction to MultiAgent Systems”. John Wiley & Sons Ltd., 2002, 366p.

[YU95] Yu, E. “Modelling Strategic Relationships for Process Reengineering”. Tese de Doutorado, Universidade de Toronto, Departamento de Ciência da Computação, 1995, 181p.