UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA DE CIÊNCIA DA COMPUTAÇÃO

Mauro Henrique Jansen Pereira

Uma metodologia e uma ferramenta para o reuso gerativo na Engenharia de Domínio Multiagente

São Luís-MA
2006

MAURO HENRIQUE JANSEN PEREIRA

UMA METODOLOGIA E UMA FERRAMENTA PARA O REUSO GERATIVO NA ENGENHARIA DE DOMÍNIO MULTIAGENTE

Dissertação de Mestrado apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Orientadora*:* Profa. Dra. Rosario Girardi

São Luís-MA

2006

MAURO HENRIQUE JANSEN PEREIRA

UMA METODOLOGIA E UMA FERRAMENTA PARA O REUSO GERATIVO NA ENGENHARIA DE DOMÍNIO MULTIAGENTE

Dissertação de Mestrado apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Dissertação aprovada em ___ /___ /_____

BANCA EXAMINADORA:

_____

Profª. Drª. Maria del Rosario Girardi (PPGEE/UFMA)
Orientadora

_____

Prof. Dr. Zair Abdelouahab (PPGEE/UFMA)
Examinador Interno

_____

Profª. Drª. Rossana Maria de Castro Andrade (DC/UFC)
Examinadora Externa

À minha esposa, Elis Gerlaine Silveira Quadros

AGRADECIMENTOS

A Deus, que nos permite sempre vencer grandes desafios, nos dando a força necessária para enfrentá-los.

À minha esposa, que é quem acompanhou na maior parte do tempo esta jornada e compreendeu os sacrifícios inerentes à realização deste trabalho.

A meus pais, que sempre me deram seu exemplo, apoio e subsídios para a minha evolução pessoal e profissional.

À professora Rosario Girardi, pelos ensinamentos e orientação, sem os quais não seria possível a concretização deste trabalho.

A todos os colegas do mestrado e do grupo de pesquisa, que juntos compartilhamos momentos de aprendizado e dificuldades, em especial a Alisson Lindoso, Ivo Serra, Carla Faria, Leandro Balby, Lucas Drumond, Raimundo Osvaldo, Francislene, André Santos.

A toda a equipe docente e administrativa do mestrado, cujo trabalho, conselhos e ensinamentos são de suma importância para todos os que ingressam na pós-graduação.

Aos colegas de trabalho e direção da Unimed São Luis, pela colaboração e compreensão nesta fase difícil de jornada dupla de trabalho e estudo, em que, por vezes, a ausência e o cansaço se tornaram visíveis e inevitáveis.

E a todos aqueles que, direta ou indiretamente, participaram desta empreita, seja cedendo um espaço ou recursos para trabalho, compreendendo a minha ausência, ou mesmo dando uma palavra de incentivo nas horas difíceis.

“Vocês estão certos de que todos esses fogos de artifício, todas as facilidades maravilhosas das chamadas linguagens poderosas da programação pertencem ao campo das soluções e não ao dos problemas?”

Edsger W. Dijkstra

RESUMO

A abordagem gerativa é um dos meios mais produtivos para promover o reuso automático em linhas de produção de software, aliado a técnicas e metodologias da Engenharia de Domínio. O paradigma multiagente visa fornecer soluções para abordar a crescente complexidade dos softwares que devem operar em ambientes não predizíveis ou sujeitos a mudanças rápidas. Para obtermos os benefícios do reuso automático em famílias de sistemas multiagente, precisamos de métodos adequados ao paradigma multiagente, que são objetos de estudo da Engenharia de Domínio Multiagente. Este trabalho propõe a GENMADEM, uma metodologia baseada em ontologias para o reuso gerativo na Engenharia de Domínio Multiagente cujos principais produtos são modelos de domínio baseados em ontologias, Linguagens Específicas de Domínio (LEDs) e geradores de aplicação. Ele também contribui com a ONTOGENMADEM, uma ferramenta composta por uma ontologia e um *plugin* para o editor de ontologias Protégé que suporta a aplicação da GENMADEM, auxiliando na análise, projeto e implementação de LEDS. Um estudo de caso que consiste no desenvolvimento de uma LED e projeto do gerador para o domínio da recuperação e filtragem de informação também é apresentado com o objetivo de avaliar a metodologia.

Palavras-chave: Reuso Gerativo de Software. Ontologias. Engenharia de Domínio. Linguagens Específicas de Domínio. Geradores. Sistemas Multiagente.

ABSTRACT

The generative approach is one of the most productive ways to promote automatic reuse in software product lines, associated with Domain Engineering techniques and methodologies. The multi-agent paradigm aims to provide solutions to approach the growing complexity of software that should operate in non-predictable environments or exposed to fast changes. To obtain the benefits of automatic reuse in multi-agent system families, we need appropriate methods for the multi-agent paradigm, main study object of Multi-agent Domain Engineering. This work proposes GENMADEM, an ontology-based methodology for generative reuse in Multi-agent Domain Engineering whose main products are ontology-based domain models, Domain Specific Languages (DSLs) and application generators. It also contributes with ONTOGENMADEM, an ontology composed by an ontology and a plug-in to the ontology editor Protégé that supports the application of GENMADEM, aiding the analysis, design and implementation of DSLs. A case study that consists of the development of a DSL and a generator design for the domain of information filtering and retrieval is also presented with the goal of evaluation of the methodology.

Keywords: Generative Software Reuse. Ontologies. Domain Engineering. Domain-Specific Languages. Software Generators. Multi-agent Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

API	Application Programming Interface
ASA	Árvore de Sintaxe Abstrata
AST	Abstract Syntax Tree
BNF	Backus-Naur Form
COTS	Commercial Off-The-Shelf
DC	Diagrama de Característica
DDEMAS	Domain Design technique for Multi-Agent Systems
DEMRAL	Domain Engineering Method for Reusable Algorithmic Libraries
DIMAS	Domain Implementation technique for Multi-Agent Systems
DSL	Domain-Specific Language
DSSA	Domain Specific Software Architecture
EJB	Enterprise JavaBeans
FAST	Family-oriented Abstraction Specification and Translation
FBC	Filtragem Baseada no Conteúdo
FC	Filtragem Colaborativa
FH	Filtragem Híbrida
FI	Filtragem de Informação
FODA	Feature-Oriented Domain Analysis
FWS	Floating Weather Stations
GAL	Graphics Adaptor Language
GENMADEM	Generative Multi-agent Domain Engineering Methodology
GESEC	Grupo de pesquisa em Engenharia de Software e Engenharia do Conhecimento
GMCL	Generative Matrix Computation Library
GRAMO	Generic Requiremente Analysis Method based on Ontologies
GUI	Graphical User Interface
HTML	Hypertext Marckup Language
IDE	Integrated Development Environments
JADE	Java Agent Development Framework
JTS	Jakarta Tool Suite
L4G	Linguagem de 4ª geração

LaLa	Language Laboratory
LED	Linguagem Específica de Domínio
LESRF	Linguagem de Especificação de Sistemas para Recuperação e Filtragem de informação
LIFO	Last-in-first-out
LPG	Linguagem de Propósito Geral
MaAE	Multi-agent Application Engineering (Projeto do grupo GESEC)
MADEM	Mult-agent Domain Engeneering Methodology
MSL	Message Specification Language
ODM	Organization Domain Model
ONTOGENMADEM	Ontologia que suporta a aplicação da GENMADEM
ONTOINFO	Ontologia para o domínio da recuperação de filtragem de informação
ONTOLED	Ontologia que suporta a aplicação da GENMADEM
ONTOMADEM	Ontologia que suporta a aplicação da MADEM
PLAN-P	Programming Language for Active Networks and Protocols
RDF	Resource Description Framework
RI	Representação Intermediária
SDA	Software Design Automation
SDL	Specification and Description Language
SQL	Structured Query Language
TAD	Tipo Abstrato de dados
TF-IDF	Total Frequency-Inverse Document Frequency
TOD-LED	Técnica baseada em Ontologias para o Desenvolvimento de LED
UML	Unified Modeling Language
VHDL	Very High-speed integrated circuit Hardware Description Language
XML	Extensible Marckup Language

SUMÁRIO

1. INTRODUÇÃO

1.1 Contexto e problemática

A dissertação de mestrado ora apresentada enfoca a Engenharia de Domínio Multiagente, especificamente a definição e abordagem dos requisitos necessários para termos o reuso gerativo durante o processo da Engenharia de Aplicações.

A Engenharia de Domínio e a Engenharia de Aplicações são dois processos complementares no âmbito das linhas de produção de software. O primeiro trata da criação de artefatos de software reutilizáveis e de meios ou ferramentas para a reutilização dos mesmos. O segundo trata da criação de membros distintos da família de sistemas reutilizando os artefatos e meios produzidos na Engenharia de Domínio.

As Linguagens Específicas de Domínio (LEDs) são linguagens de programação ou especificação com alto nível de abstração, vocabulário próximo ao utilizado pelos especialistas do domínio e força expressiva focada num domínio particular. Após uma análise das técnicas existentes para o desenvolvimento de LEDs, observa-se que não há ainda uma terminologia consensual para se tratar do tema. As fases do desenvolvimento de uma LED nas metodologias existentes, apesar de estarem em geral bem definidas, não são facilmente mapeadas para as fases clássicas de análise, projeto e implementação, ficando então uma necessidade de critérios para delimitar esses grupos de tarefas. As técnicas existentes apresentam pontos controversos e carências, ficando claro que ainda é necessário esforço de pesquisa para tornar o Desenvolvimento de LEDs uma tarefa com processos claros e bem distintos.

Na Engenharia de Domínio Multiagente, notando-se a pouca quantidade de técnicas ou metodologias genuinamente elaboradas para a construção de famílias de sistemas multiagente, pretende-se mostrar uma iniciativa em uma metodologia para suprir essa carência.

1.2 Relevância e motivação

A abordagem gerativa tem demonstrado ser de grande eficácia nas linhas de produção de software, ao permitir o desenvolvimento com reuso de forma rápida e sistematizada, facilitando a manutenção e reduzindo custos, mas ainda não é um tópico de pesquisa esgotado. Muitas são as técnicas e processos existentes, mas na sua maioria, usam conceitos e abstrações não muito adequados aos utilizados no paradigma multiagente, forçando ao uso de mapeamento ou adaptação dos mesmos aos conceitos do paradigma multiagente.

As ontologias (GRUBER, 1995), representam uma solução eficaz para representar abstrações de software de alto nível, como modelos de domínio e arquiteturas reutilizáveis, por serem formas de representação de conhecimento que oferecerem terminologia livre de ambigüidades, permitirem inferências e generalizações, representando um domínio a partir de seus conceitos abstratos e seus relacionamentos, de modo formal, explícito e compartilhado.

No âmbito do grupo de pesquisa GESEC (2005), a Engenharia de Domínio é um tópico de importância, em relação ao qual já foram sistematizados, em trabalhos anteriores, técnicas para análise e projeto de domínio e uma técnica para especificação de LEDs, que justificam o aprimoramento e integração das mesmas em uma metodologia para a Engenharia de Domínio Multiagente gerativa.

Estas, portanto, são as principais motivações para a realização deste trabalho.

1.3 Objetivos

Este trabalho tem como objetivo geral definir uma metodologia para o reuso gerativo na Engenharia de Domínio Multiagente, estendendo o que foi definido na técnica TOD-LED [SERRA, 2004] para a fase de especificação de LEDs e reutilizando os produtos da metodologia MADEM [GIRARDI, LINDOSO, 2005] para análise e projeto de domínio. Os objetivos específicos são:

- Estender a técnica TOD-LED [SERRA, 2004], detalhando as fases de projeto e implementação;
- Propor uma metodologia para o reuso gerativo na Engenharia de Domínio Multiagente, estendendo e integrando a MADEM e a TOD-LED;

- Criar uma ferramenta para a análise, projeto e implementação de LEDs;
- Avaliar a metodologia proposta através do desenvolvimento de um estudo de caso;

1.4 Estrutura da Dissertação

O capítulo 2 apresenta os conceitos relacionados à Engenharia de Domínio e linhas de produção de software, apresentando a modelagem de características como técnica para capturar a variabilidade e as iniciativas em técnicas para análise e projeto de domínio do grupo GESEC (2005) que serão reutilizados neste trabalho.

O capítulo 3 apresenta os elementos da Engenharia de Domínio que possibilitam o reuso gerativo, descrevendo as Linguagens Específicas de Domínio com seus conceitos, fases genéricas de desenvolvimento e algumas técnicas para o desenvolvimento; e também os geradores de aplicações, que são necessários para a aplicação das LEDs, fazendo a seleção, adaptação e composição automática dos elementos da família de sistemas, discorrendo sobre os conceitos, técnicas de construção e abordagens gerativas conhecidas.

O capítulo 4, sendo a parte central desta dissertação, apresenta a metodologia gerativa para a Engenharia de Domínio Multiagente por nós criada, que é denominada GENMADEM, descrevendo a extensão feita nos produtos que são reutilizados pela mesma, seu processo e fases. Como a metodologia é baseada em ontologias, também é apresentada a ONTOGENMADEM, que é uma ferramenta dirigida por ontologia que dá suporte à aplicação da metodologia.

O capítulo 5 apresenta um estudo de caso com a finalidade de avaliar a metodologia com o desenvolvimento de uma LED e projeto de gerador para o domínio da recuperação e filtragem de informação.

Por fim há o capítulo 6 que expõe nossas conclusões sobre o trabalho realizado, enfatizando as contribuições, limitações, resultados alcançados e tópicos abertos para trabalhos futuros e o APÊNDICE A, apresenta a ONTOINFO, que é modelo de domínio baseado em ontologias para a recuperação e filtragem de informação que foi estendido e utilizado no estudo de caso apresentado no capítulo 5; o APÊNDICE B apresenta o código do plugin ONTOGENMADEM.

2. A ENGENHARIA DE DOMÍNIO

2.1 Conceitos

A Engenharia e Domínio e a Engenharia de Aplicações são dois processos de desenvolvimento dentro da engenharia de famílias de sistemas ou linhas de produção de software. Uma *família de produtos de software* é um conjunto de produtos de software com características suficientemente similares para permitir a definição de uma infra-estrutura comum de estruturação dos itens que compõem os produtos e a parametrização das diferenças entre os produtos [PARNAS, 1976].

Antes de descrever a Engenharia de Domínio, convém definir o que é domínio. Domínio recebe diferentes definições nas diversas disciplinas (ex: Lingüística, Inteligência Artificial, Orientação a Objetos e Reuso de Software) e todas tendem para uma das duas visões a seguir: domínio como o "mundo real" (uma área ou campo de especialização onde o conhecimento humano é usado) e domínio como um conjunto de sistemas (família de sistemas de software similares). Uma definição bem adequada é dada por [CZARNECKI, 1998]: *Domínio* é uma área de conhecimento delimitada para satisfazer os requisitos dos interessados e que inclui um conjunto de conceitos e terminologia entendida pelos praticantes da área e um conhecimento de como fazer sistemas ou partes de sistemas de softwares nessa área.

Engenharia de Domínio é a atividade de colecionar, organizar e armazenar experiência passada na construção de sistemas ou partes de sistemas num domínio particular, na forma de artefatos reutilizáveis, bem como provendo um meio adequado para reutilizar esses artefatos (ex: recuperação, qualificação, seleção, adaptação, integração, montagem, etc.) quando construindo novos sistemas [CZARNECKI, 1998].

A Engenharia de domínio, portanto, é desenvolvimento *para* reutilização e trata do desenvolvimento de recursos reutilizáveis como componentes, geradores, LEDs, documentação, etc. As seções a seguir fornecerão maiores detalhes sobre esses conceitos.

2.2 Linhas de produção de software

Linhas de produção de software referem-se a técnicas de engenharia para a criação de um conjunto de sistemas similares (família de produtos de software) a partir de um conjunto compartilhado de artefatos de software e usando meios comuns de produção [SOFTWARE PRODUCT LINES, 2006]. Portanto, as linhas de produção de software tratam da criação de famílias de sistemas. Usando técnicas de linhas de produção de software, empresas como Nokia, HP, Lucent, Philips e Cummins reduziram tempo de produção, custos de engenharia e índice de erros por um fator de 3 a 50. Os principais conceitos de uma linha de produção de software (Figura 1) são:

- *Artefatos de software de entrada*

Uma coleção de artefatos de software que podem ser configurados e montados em diferentes formas para criar todos os produtos numa linha de produção. Cada um dos artefatos tem um papel bem definido dentro de uma arquitetura comum para a linha de produção e, para permitir a variação entre os produtos, alguns artefatos podem ser opcionais ou ter pontos de variação internos que podem ser configurados de diferentes formas para prover comportamentos diferentes. Podem vir na forma de binários ou códigos fonte alteráveis, adaptáveis ou extensíveis durante a produção e não estão limitados a código, podendo incluir requisitos, arquiteturas, descrições de projeto, casos de teste ou outros artefatos;

- *Modelo de decisão e decisões de produto*

O modelo de decisão descreve requisitos opcionais e variáveis para os produtos da linha de produção. As decisões de produtos são escolhas para cada um dos requisitos opcionais e variáveis do modelo de decisão que definem cada um dos produtos únicos na linha de produção. Os formatos possíveis para representação das decisões de produto variam desde descrições textuais informais até linguagens formais e interfaces gráficas. O papel de tomador de decisões de produto pode ser exercido por um engenheiro (de aplicações ou de domínio) ou usuário especialista no produto;

- *Processo e mecanismo de produção*

São os meios para montagem e configuração dos produtos a partir dos artefatos de software de entrada. Aqui decisões de produto são usadas para

determinar que artefatos de software de entrada serão usados e como configurar os *pontos de variação* desses artefatos. A operação de produção pode criar produtos de saída parcial ou totalmente instanciados e é o conceito que discrimina as diferentes abordagens para as linhas de produção de software de acordo com suas características de:

- *Automação:* se a produção é manual, parcial ou totalmente automatizada, como no caso dos geradores de aplicação, onde as decisões de produto provêem informações suficientes para gerar automaticamente os produtos de saída;
- *Periodicidade:* se a produção ocorre em horas, como numa que use automação, ou até anos como numa produção manual em cascata;
- *Papéis:* algumas abordagens definem papéis humanos distintos para a atividade de produção e para a atividade de engenharia dos artefatos de software e outras não.

Pontos de variação representam opções em aberto sobre como o software irá se comportar. Durante o processo de produção, as decisões de produto são usadas para selecionar opções entre os pontos de variação, especificando então o comportamento desse ponto de variação no produto final. O momento em que essas decisões sobre os pontos de variação são feitas é chamado de momento de ligação, e alguns exemplos são: tempo de reuso de código, tempo de desenvolvimento, tempo de instanciação do código estático, tempo de compilação, tempo de montagem, customização pelo cliente, tempo de instalação, tempo de carga do sistema e tempo de execução.

- *Produtos de software de saída*

Produtos de software de saída são todos os produtos que podem ser produzidos pela linha de produção, e que definem o escopo da mesma. Podem ser obtidos na forma binária ou código fonte, alterável ou não.

Esses conceitos mostram os objetivos principais das linhas de produção de software que são aproveitar a parte comum e gerenciar as variações, para reduzir o tempo, esforço, custo e complexidade na criação e manutenção de uma linha de produtos de sistemas similares. O aproveitamento da parte comum é feito através da consolidação e compartilhamento no conjunto dos artefatos de software, com isso evitando duplicidade e divergência. O gerenciamento das variações é feito definindo-

se claramente os *pontos de variação* e o modelo de decisão, e assim tornando explícitas a localização, justificativa e dependências para variações.

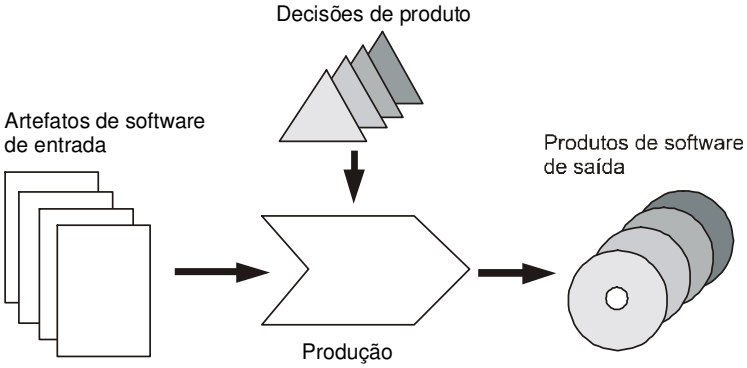

Figura 1 Conceitos básicos das linhas de produção de software [SOFTWARE PRODUCT LINES, 2006]

2.3 O Processo da Engenharia de Domínio

A maioria dos métodos da Engenharia de software conhecidos visa o desenvolvimento de um sistema específico para um cliente e contextos também específicos. A Engenharia de Domínio, por sua vez, visa o desenvolvimento de software ou artefatos de software reutilizáveis.

A Engenharia de Domínio também tem as fases de análise, projeto e implementação, como na Engenharia de software convencional, só que as mesmas estão focadas numa classe de sistemas em vez de um sistema único e seus produtos são abstrações de mais alto nível dentro do domínio em que estão inseridos, visando a sua posterior seleção para reutilização na construção de sistemas dentro desse domínio. Essas fases, com seus propósitos e produtos estão resumidos na Tabela 1 e ilustrados na Figura 2.

A *Análise de Domínio* tem por objetivo selecionar e definir o foco do domínio e coletar informações relevantes sobre o domínio, integrando-as em um *modelo de domínio* coerente. Dentre as fontes de informação sobre o domínio que são exploradas na análise destacam-se sistemas já existentes, especialistas do domínio, manuais, livros ou requisitos já conhecidos de outros sistemas.

O Modelo de domínio é uma representação explícita das propriedades comuns e variáveis dos sistemas num domínio e as dependências entre as propriedades variáveis e é composto das seguintes partes:

- Definição do domínio: define o escopo do domínio e caracteriza seu conteúdo, dando exemplos de sistemas no domínio e regras para identificar integrantes do domínio;

- Vocabulário do domínio: define o vocabulário do domínio;

- Modelos conceituais: descrevem os conceitos do domínio usando algum formalismo apropriado como diagramas de objetos, interação, transição de estado, entidade-relacionamento ou fluxo de dados;

- Modelos de características ("*feature models*"): definem um conjunto de requisitos reutilizáveis e configuráveis para a especificação de sistemas num domínio, que são geralmente referenciados como características ("*features*"), representando o aspecto de configuração do software reutilizável, pois determina quais combinações de características fazem sentido.

A análise de domínio envolve as seguintes atividades:

- Planejamento, identificação e delimitação do domínio
- Modelagem do domínio

O *Projeto de Domínio* visa desenvolver uma arquitetura para os sistemas no domínio, que é uma descrição dos subsistemas e componentes de um sistema de software e os relacionamentos entre eles, padrões que guiam sua composição e restrições desses padrões. A arquitetura deve servir como um mapa para os desenvolvedores, revelando como os sistemas do domínio são construídos e onde funções ou conceitos específicos estão localizados.

A *Implementação do Domínio* consiste em aplicar tecnologias apropriadas para implementar componentes, geradores para reutilização automática dos componentes, infra-estrutura de reuso e processos de produção de aplicação.

Fase	**Propósitos e produtos**
Análise do domínio	Definir o modelo de domínio representando um conjunto de requisitos reutilizáveis para os sistemas no domínio
Projeto do domínio	Estabelecer uma arquitetura de software comum para os sistemas no domínio
Implementação do domínio	Implementação dos recursos reutilizáveis. Ex: componentes reutilizáveis, linguagens específicas de domínio, geradores e infra-estrutura de reuso.

Tabela 1 As três fases da Engenharia de Domínio

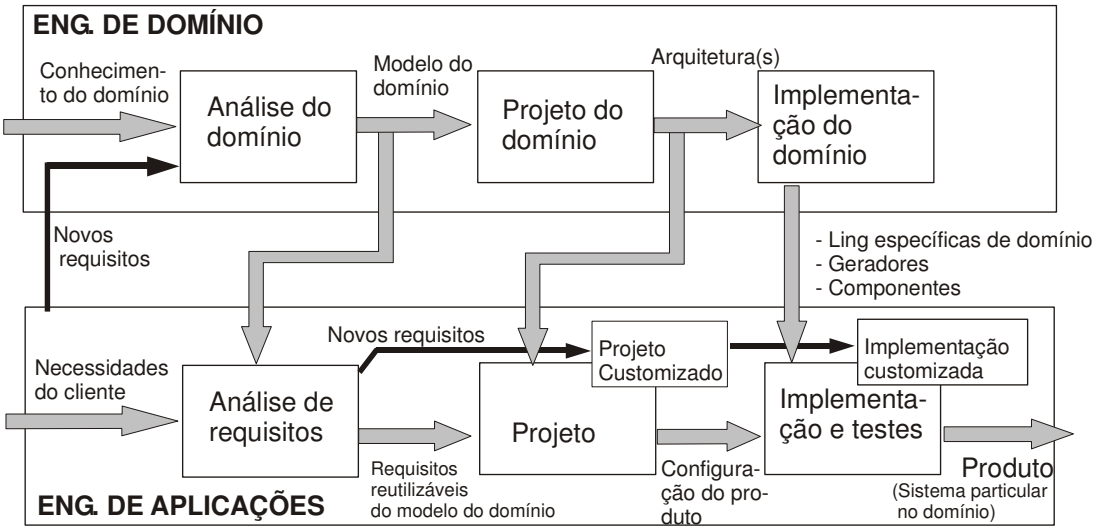

Figura 2 Desenvolvimento de software baseado na Engenharia de Domínio (traduzido de [CZARNECKI, 1998])

2.4 O Processo da Engenharia de Aplicações

A reutilização dos artefatos obtidos na Engenharia de Domínio é feita na Engenharia de Aplicações, que é o processo de construir um sistema particular num domínio. A Engenharia de Aplicações então é desenvolvimento *com* reutilização e o seu processo também realimenta o da Engenharia de Domínio, fornecendo novos requisitos, ainda não previstos nos recursos existentes.

A Engenharia de Aplicações tem as seguintes fases (Figura 2): a *análise de requisitos* identifica as necessidades do cliente num sistema particular e confronta com as características já conhecidas do domínio; as similares serão reutilizadas no projeto, juntamente com os novos requisitos encontrados, que guiarão, na fase de *projeto*, a escolha da arquitetura e configuração escolhida para o produto;

A fase de *implementação* aproveita os recursos reutilizáveis da Engenharia de Domínio para a montagem do produto (um sistema particular no domínio), e esse trabalho pode ser manual (reutilização composicional) ou automatizado (reutilização gerativa) com o uso de um gerador de aplicação.

2.5 Modelagem de variabilidade

A modelagem de variabilidade é uma atividade que visa modelar a variabilidade de sistemas numa linha de produção de software, identificando a parte *comum* e *variável* da família de sistemas. A parte comum é uma lista estruturada de

suposições que são verdadeiras para todos membros da família e a parte variável é uma lista estruturada de suposições sobre como membros de uma família diferem e que é passível de parametrização durante a produção de cada elemento da família.

As abordagens para a modelagem de variabilidade normalmente lidam com os conceitos do domínio e suas características, procurando classificá-los em um desses dois grupos (comum ou variável), identificando então os pontos de variação.

2.5.1 Modelagem conceitual

Descobrir qual a linguagem da mente é uma questão fundamental para os cientistas de diversas áreas como psicologia, filosofia e ciências cognitivas. Descrever um conceito em palavras é bem difícil na maioria das vezes e podemos comprovar isso observando a diferença que há entre ter um pensamento e colocá-lo em palavras. Isso suporta a visão de que usamos uma representação interna na mente que não é a linguagem natural. Os psicólogos cognitivos chamam essa representação de representação proposicional, que consiste na representação mental daquilo que normalmente expressamos em linguagem natural.

Smith e Medin (1981) observam que “Se nós percebêssemos cada entidade como única, poderíamos ser sobrecarregados pela absoluta diversidade daquilo que experimentamos e incapazes de relembrar mais que uma fração de minuto daquilo que encontramos. E se cada entidade individual precisasse de um nome distinto, nossa linguagem seria esmagadoramente complexa e a comunicação virtualmente impossível”. É por isso que freqüentemente reconhecemos novos objetos como instâncias de conceitos que já conhecemos, onde um conceito significa o conhecimento sobre objetos que têm certas propriedades. Essa atividade é chamada de categorização ou classificação e, nesse contexto, os conceitos são chamados de categorias ou classes. O estudo de conceitos permite aprender como o conhecimento é representado e processado na mente.

A importância do assunto *conceitos* para a programação e desenvolvimento se justifica pelo fato de que a programação nada mais é que uma modelagem conceitual, onde traduzimos modelos conceituais criados na mente para resolver problemas em modelos que possam rodar numa máquina.

Os conceitos podem ser matemáticos que são conceitos que têm uma definição precisa, como números, figuras geométricas, matrizes, etc.; e conceitos naturais, que são conceitos usados na comunicação em linguagem natural, como cachorro, mesa, carro, etc.

Conceitos são descritos listando suas propriedades. Segundo Smith e Medin (1981), existem três tipos principais de propriedades de conceitos:

- *Características ("features"):* são usadas para descrever características qualitativas dos conceitos. Por exemplo, possíveis características do conceito *carro* são: branco, popular, passeio, ano 2005;
- *Dimensões:* usadas para expressar propriedades quantitativas como tamanho ou peso. Sua faixa de valores pode ser contínua (ex: um número real) ou discreta (ex: pequeno, médio ou grande para a dimensão "tamanho");
- *Propriedades holísticas:* são propriedades únicas que podem ser vistas como um modelo do próprio conceito (i.e. é como "tudo a respeito de alguma coisa"), em oposição às características e dimensões, que são chamadas de propriedades componentes, visto que sozinhas não definem o conceito por completo. A palavra holística vem da palavra grega *holon* que significa entidade. Por exemplo, o sub-campo redes neurais da Inteligência Artificial, baseia-se na afirmação de que conexões e respostas entre nós simples arranjados em rede podem dar origem a um comportamento similar ao comportamento inteligente ou cognitivo.

Conceitos são objetos de estudo desde a antiga Grécia em múltiplas disciplinas como Filosofia, Lingüística, Psicologia, Ciência cognitiva e Ciência da computação. Das pesquisas realizadas, três visões sobre a natureza dos conceitos se destacam:

- *A visão clássica*: define um conceito listando um número de propriedades necessárias e suficientes as quais um objeto deve possuir para ser uma instância desse conceito. Exemplo: o conceito de *quadrado* pode ser definido usando quatro características (1-figura fechada, 2-quatro lados, 3-lados de tamanho igual e 4-ângulos iguais).

Essa visão tem problemas como dificuldade para definir conceitos naturais;

- *A visão probabilística*: cada conceito é descrito por uma lista de propriedades como na visão clássica, porém cada característica tem um número associado, que pode ser uma probabilidade ou peso, cujo valor pode ser calculado de acordo com fatores como a probabilidade da característica ser verdadeira para uma instância do conceito, o grau pelo qual a característica distingue o conceito de outros conceitos e a usabilidade passada ou freqüência com que a característica foi percebida. Exemplo: podemos ter a probabilidade 0.9 associada à característica *voa* do conceito *ave* para indicar que a maioria das aves voa (mas não todas);
- *A visão exemplar:* o conceito é definido por seus exemplares (instâncias representativas do conceito ou subconjuntos) em vez de um resumo abstrato. Por exemplo, a representação do conceito *cachorro* pode incluir os subconjuntos *poodle* e *pastor alemão* e a instância específica "*Bobby*".

2.5.2 Abstrações de software reutilizáveis

Uma abstração de software é uma descrição concisa de um artefato de software que descarta os detalhes irrelevantes para o desenvolvedor e enfatiza só a informação que é importante para ele [KRUEGER, 1992]. Como exemplos de abstrações de software temos as linguagens de alto nível e as classes. Todas as técnicas de reutilização de software utilizam algum tipo de abstração para representar os artefatos de software reutilizáveis.

O software basicamente consiste de vários níveis de abstração acima do hardware. A abstração de software de mais baixo nível é o código objeto ou código de máquina. A linguagem Assembler é uma camada de abstração sobre o nível do código de máquina. Uma linguagem de programação (como o Java) é uma camada de abstração acima da linguagem Assembler. Portanto, cada abstração de software possui dois níveis: um nível mais alto e genérico, chamado de *especificação*; e um nível mais baixo e detalhado, chamado de *realização* (Figura 3a). Quando temos

uma hierarquia de abstrações (Figura 3b), a especificação de uma camada é a realização de outra. Em outras palavras, a camada de especificação define O QUÊ o software faz e a camada de realização revela COMO o faz, ou seja, aumenta o nível de detalhes e conseqüentemente, sua complexidade.

Outra característica dessas abstrações é que elas têm uma *parte oculta,* uma *parte variável* e uma *parte fixa* (Figura 4a). A parte oculta consiste de detalhes na realização da abstração que não são visíveis na especificação da abstração. As partes fixa e variável são visíveis na especificação. A parte variável representa as características variantes na realização da abstração, enquanto a parte fixa representa as características invariantes. Por exemplo, numa abstração para pilhas (Figura 4b), a parte fixa da abstração expressa as características invariantes para todas as realizações de pilha, como a semântica LIFO ("*Last-in-first-out*"). A parte variável pode ser o tipo de dado armazenado na pilha. Então diferentes tipos de dados correspondem a diferentes realizações.

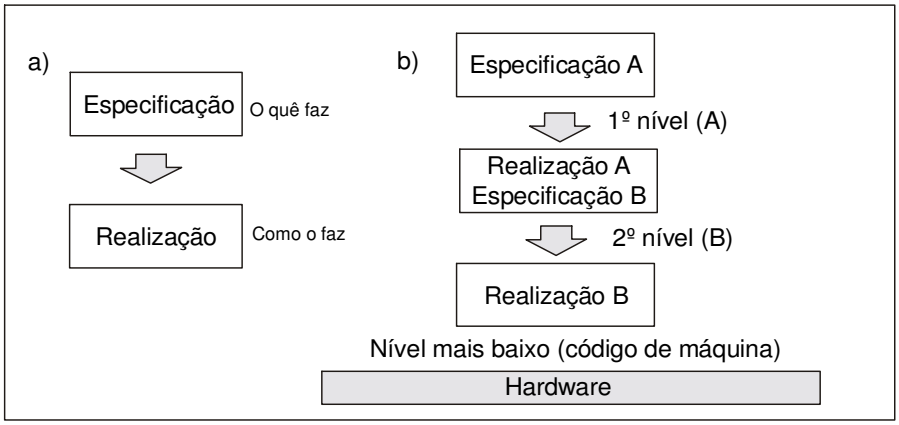

Figura 3 Níveis de abstração e hierarquia de abstrações de software

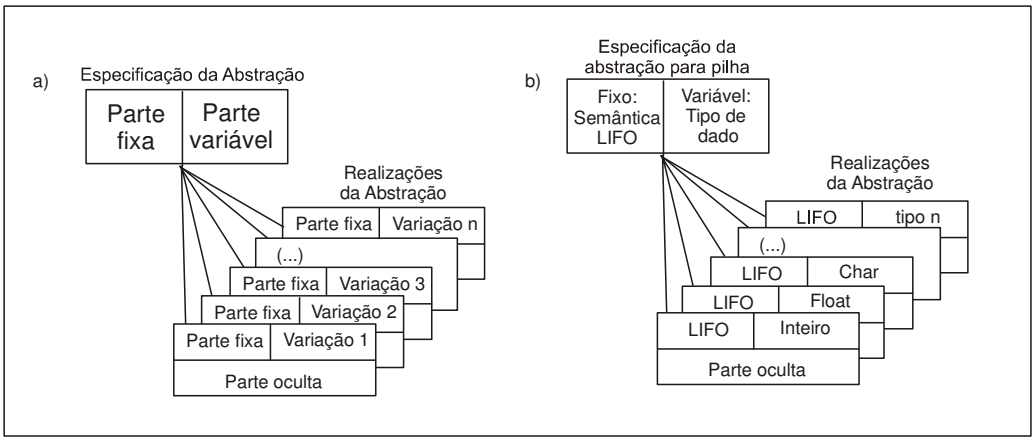

Figura 4 Partes de uma abstração de software e exemplo [KRUEGER, 1992]

Esses conceitos têm influência direta nos princípios do reuso de software: a *seleção* requer abstrações concisas, a *adaptação* corresponde à seleção de uma realização da abstração da parte variável de uma especificação e a *integração* dos artefatos de software requer o entendimento da interface dos mesmos, que é uma abstração na qual os detalhes internos estão ocultos.

2.5.3 Modelagem de características

O software reutilizável contém inerentemente mais variabilidade que aplicações concretas, o que requer técnicas apropriadas para capturar a parte fixa e variável no estudo de um domínio. Uma dessas técnicas é a modelagem de características. A modelagem de características [CZARNECKI, 1998] [LEE, KANG, LEE, 2002] é a atividade de modelar as propriedades comuns e variáveis de conceitos e suas interdependências e organizá-los em um modelo coerente chamado de modelo de domínio.

Os modelos de características produzidos durante a modelagem de características proporcionam uma representação abstrata (visto que é independente de implementação), concisa e explícita da variabilidade existente no software. Portanto, modelos de características representam os aspectos de configurabilidade de software reutilizável em um nível abstrato, sem referência a nenhuma solução específica. A seção a seguir descreve os conceitos usados na modelagem de características, alguns deles já abordados na seção 2.5.1.

2.5.3.1 Conceitos da modelagem de características

Conceitos: São elementos e estruturas no domínio de interesse. A noção de conceito se confunde com a de classes no paradigma da orientação a objeto, visto que ambos representam a descrição genérica de um conjunto de instâncias. A diferença básica é que as instâncias de classes têm propriedades semânticas (como estado, comportamento e identidade única) que os conceitos não têm (Figura 5).

Características ("features"): são propriedades importantes de um conceito. Elas possibilitam expressar semelhanças e diferenças entre instâncias de um conceito e formular descrições concisas de conceitos com grandes graus de variação entre suas instâncias. O método FODA (*Feature-Oriented Domain Analysis*)

[KANG et al, 1990], um método para a análise de domínio desenvolvido pelo SEI *(Software Engineering Institute)*, define *característica* como aspecto, qualidade ou característica de um sistema de software que é visível ao usuário. Uma característica deve ter um nome conciso e descritivo, que enriquece o vocabulário para descrever conceitos e instâncias num domínio.

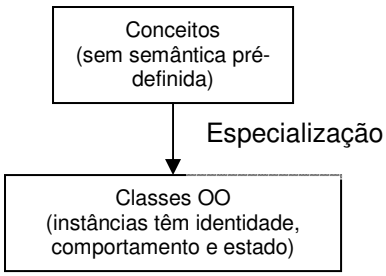

Figura 5 Diferença entre conceito e classe [CZARNECKI, 1998]

Modelos de características: São os produtos da modelagem de características. Representam as características comuns e variáveis de instâncias de conceitos e as dependências entre características variáveis. Um modelo de características representa a intenção de um conceito, enquanto o conjunto de instâncias que ele descreve é chamado de extensão do conceito. Os componentes de um modelo de características são:

- Diagramas de características;
- Descrições semânticas: descrições curtas da semântica das características. Pode-se usar outros formalismos para representar essas descrições, como diagramas de interações, pseudocódigo, equações, etc.;
- Justificativas: notas que explicam porque a característica foi incluída no modelo. No caso de uma característica variável, pode conter também condições ou recomendações de quando a mesma deve ser usada numa aplicação;
- Interessados (usuários) e programas clientes: usuários, clientes, desenvolvedores, gerentes ou programas (no caso de um componente), que precisem ou tenham interesse na característica;
- Sistemas-exemplo: exemplos de sistemas que implementam essa característica;

- Restrições: registro de dependências exigidas entre características variáveis ou até entre múltiplos diagramas de características. Exemplos de restrições:
    i. Exclusão mútua: combinações ilegais entre características variáveis. Exemplo: uma lista não pode ser desordenada e indexada ao mesmo tempo;
    ii. Elegibilidade: que características exigem a presença de outras características? Ex.: uma lista indexada deve ser ordenada;
- Regras de dependência padrão: regras para sugerir valores padrões para parâmetros não especificados, baseado em outros parâmetros;
    i. Regras horizontais ou regras de dependência padrão: estabelece relação entre itens da mesma camada de abstração. Por exemplo, para o conceito "estrela", cujo conjunto de características é {raio interno, raio externo, número de braços}, temos a regra horizontal que diz que o raio interno deve ser menor que o raio externo e maior que zero (raio externo > raio interno > 0);
    ii. Regras verticais: estabelecem relações entre camadas de abstração diferentes. Por exemplo, uma estrela é composta dos componentes gerativos círculo {raio}, e braço {ângulo, raio interno, raio externo}. No mapeamento entre o conceito estrela e os componentes gerativos temos a regra vertical *ângulo do braço = 360° / número de braços da estrela*, que estabelece uma relação entre elementos de camadas diferentes de abstração;
- Pontos de disponibilidade, pontos de ligação e modos de ligação;
- Atributo aberto/fechado: identifica o ponto de variação como aberto ou fechado, conforme ele aceite ou não novas características além das citadas;
- Prioridades: indicam a relevância da característica no projeto.

Conjuntos iniciadores de características: são conjuntos de perspectivas de modelagem para modelar conceitos, normalmente voltados a categorias de domínio,

contendo exemplos de características, fontes de características e qualquer outro item que auxilie na identificação de características, ajudando a iniciar a análise.

2.5.3.2 Diagramas de características

Um diagrama de características (DC) consiste de um conjunto de nós, um conjunto de linhas direcionadas e um conjunto de adornos de linhas formando uma árvore. Os adornos de linhas são arcos conectando subconjuntos de linhas ou todas linhas originárias do mesmo nó, dividindo os sub-nós em subconjuntos distintos.

A raiz de um diagrama de características é um conceito e os nós restantes são características. Um nó-característica pode se originar de um nó-conceito ou de outro nó-característica. O nó imediatamente abaixo de outro é uma característica ou sub-característica direta. Os que estão mais abaixo são características ou sub-características indiretas. A partir dos diagramas, a descrição de uma instância individual de um conceito pode ser descrita com conjuntos de características.

2.5.3.2.1 Tipos de nós e critérios de inclusão de uma característica

Características mandatórias: são características que estarão sempre presentes em qualquer instância do conceito, desde que o nó-pai esteja incluído na instância de conceito, visto que podemos ter características mandatórias como subcaracterísticas subordinadas a características opcionais. São caracterizadas por estarem ligadas por uma linha simples com um círculo fechado no lado da característica mandatória (Figura 6). No modelo da figura, cada instância do conceito pode ser descrita com o conjunto {C,f1,f2,f3,f4};

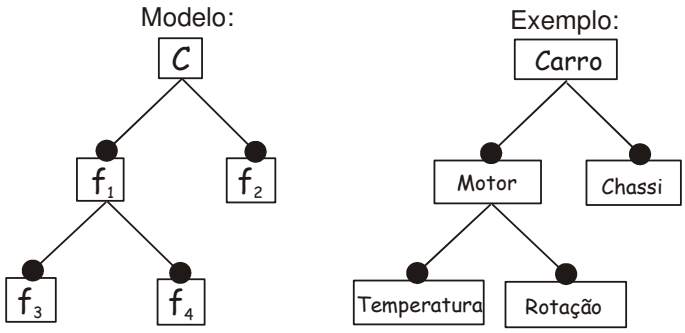

Figura 6 Representação de características mandatórias em um DC

Características opcionais: são características que podem ou não estar incluídas na descrição de uma instância do conceito, se e somente se, o nó-pai estiver incluído também. São apontadas por uma linha simples terminando com um círculo aberto (Figura 7). No modelo da figura, uma instância do conceito C pode ter as seguintes descrições {C}, {C,f1}, {C,f1,f3}, {C,f2}, {C,f1,f2}, {C,f1,f2,f3};

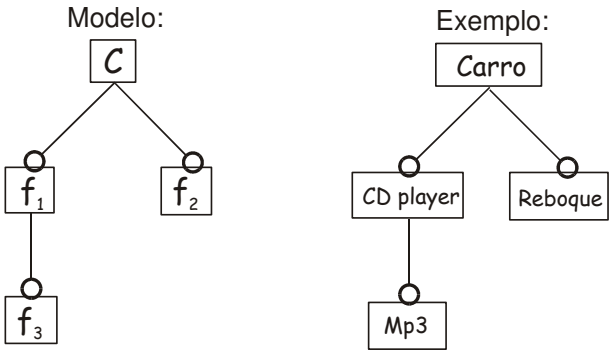

Figura 7 Representação de características opcionais em um DC

Características alternativas: São características que pertencem a um conjunto onde, se o nó pai está presente, exatamente uma característica do conjunto deve ser incluída na descrição de uma instância do conceito. Caso contrário, nenhuma sub-característica alternativa do mesmo pode ser incluída. O conjunto de características alternativas é caracterizado no gráfico através de um arco ligando as linhas das características (Figura 8).

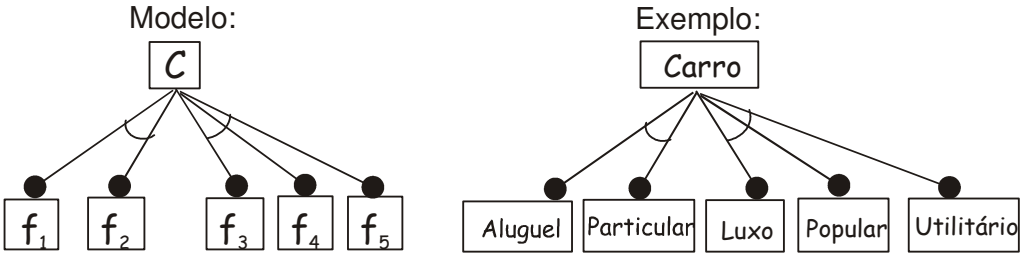

Figura 8 Representação de características alternativas em um DC

Características "ou": São características pertencentes a um conjunto onde, se o nó-pai estiver presente, qualquer subconjunto não vazio do mesmo pode ser incluído na descrição de uma instância do conceito. Caso contrário, nenhuma sub-característica do mesmo é incluída. As características integrantes de um grupo são representadas por um arco preenchido interligando as mesmas (Figura 9).

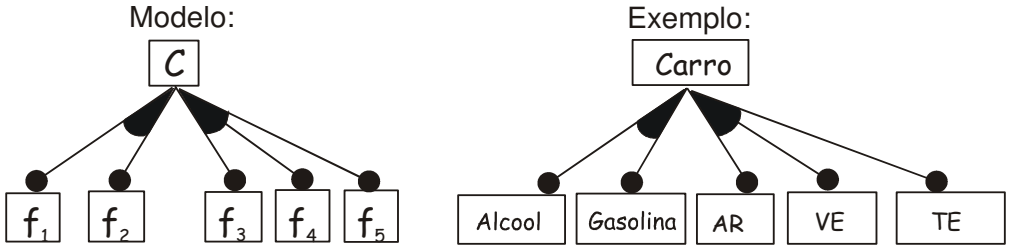

Figura 9 Representação de características "ou" em um DC

2.5.3.3 Processo da modelagem de características

Os passos básicos para a modelagem de características são:

1) Registrar as similaridades ou aspectos comuns entre instâncias. Ex: todas contas bancárias têm um número identificador;
2) Registrar as diferenças entre instâncias. Ex: existem contas correntes e contas poupança;
3) Organizar as características em diagramas de características;
4) Analisar combinações e interações entre características. Existem combinações inválidas (restrição de exclusão mútua), dependências entre características (restrição de exigibilidade) e também o relacionamento entre duas características pode servir para deduzir outras características (regras de dependência padrão);
5) Registrar as demais informações sobre características como descrições semânticas, justificativas, exemplos de sistemas com uma característica, restrições, prioridades, etc.

Dentre as estratégias para a modelagem convém destacar:

- Procurar a terminologia importante do domínio que implique variabilidade (ex: conta corrente x poupança). Lembrar que tudo que o cliente pode querer controlar sobre um conceito é uma característica;
- Usar *conjuntos iniciadores de características* (vide seção 2.5.3.1).

Os diagramas de características permitem representar conceitos de forma que torne explícitas as similaridades e diferenças entre suas instâncias.

Uma característica é comum se for mandatória e existir um caminho de características mandatórias ligando a característica e o conceito. A variabilidade é representada pelas características opcionais, alternativas e características "ou" e o

nó ao qual uma característica variável está ligada é chamado de ponto de variação. Além das características mandatórias e variáveis, existem as dependências entre conceitos variáveis, que são expressas com restrições e regras de dependência padrões. Restrições especificam combinações de características válidas e inválidas e as regras de dependência padrão sugerem valores padrões para parâmetros não especificados, baseado em outros parâmetros, permitindo assim a implementação de configuração automática.

2.6 A Engenharia de Domínio Multiagente

As experiências e pesquisas em desenvolvimento de software têm mostrado que os paradigmas computacionais existentes, como o Estruturado e o Orientado a Objetos nem sempre são eficientes no desenvolvimento de sistemas complexos. Por outro lado, o Desenvolvimento de Software Baseado em Agentes mostra grandes vantagens na criação de sistemas de alta complexidade e que necessitam operar em ambientes imprevisíveis, que mudam rapidamente. E a criação de metodologias para o desenvolvimento baseado em agentes é um tópico de pesquisa constante.

Um agente de software é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo utilizando-se dos executores. Na sua construção, deve-se decidir como fazer o mapeamento de percepções a ações [RUSSEL, NORVIG, 2004]. A autonomia é a característica principal dos agentes.

A Engenharia de Domínio Multiagente [GIRARDI, 2004] caracteriza uma tentativa de introdução das vantagens decorrentes da reutilização na Engenharia de Software Multiagente, sendo, portanto, um processo para a construção de artefatos de software reutilizáveis baseados em agentes, como modelos de domínio representando os requisitos de uma família de sistemas multiagente e arcabouços representando uma solução reutilizável baseada em agentes para esses requisitos.

O projeto MaAE (*Multi-agent Aplication Engineering*) é um projeto do grupo GESEC (2005) que aborda a complexidade e a produtividade do software através da pesquisa e criação de técnicas e ferramentas que promovem o reuso na Engenharia de Software Multiagente. Nessa reutilização, duas técnicas distintas

podem ser aplicadas: a *composicional*, onde os próprios desenvolvedores constroem a aplicação através da seleção, adaptação e integração de componentes de software recuperados a partir de repositórios [GIRARDI, 2001]; e a *gerativa*, na qual é feita a especificação da aplicação em um alto nível de abstração, a qual é, então, submetida a um gerador de aplicações, que procede ao reuso automaticamente, resultando no produto esperado [GIRARDI, 1997].

A Figura 11 mostra as três fases da Engenharia de Domínio Multiagente, que são a Análise, o Projeto e a Implementação de Domínio, bem como os fluxos de entrada e saída, insumos e produtos. Observe que todas as fases são guiadas por uma dada técnica e sofrem influência de padrões decorrentes de experiências de desenvolvimento passadas, no âmbito do projeto MaAE.

Dois dos principais produtos do projeto MaAE na área da Engenharia de Domínio são reutilizados neste trabalho, a metodologia MADEM para a análise e projeto de domínio multiagente e a técnica TOD-LED para a especificação de linguagens específicas de domínio. A MADEM é descrita na seção a seguir e a TOD-LED, na seção 3.1.4.1.

2.6.1 MADEM - Metodologia para a Engenharia de Domínio Multiagente

MADEM (*Multi-Agent Domain Engineering Methodology*) [GIRARDI, LINDOSO, 2005] é uma metodologia baseada em ontologias para análise e projeto de domínio de sistemas multiagente na Engenharia de Domínio Multiagente, definindo um roteiro para a realização das mesmas desde a captura e especificação de requisitos até o projeto de uma família de sistemas em um domínio de aplicação. MADEM estende e integra as técnicas GRAMO [FARIA, 2004] e DDEMAS [FERREIRA, 2004] para análise e projeto de domínio respectivamente, e é suportada pela ONTOMADEM, que é uma ferramenta baseada em conhecimento para guiar as tarefas de modelagem, captura e representação dos produtos da MADEM.

2.6.1.1 Conceitos da MADEM

Para especificar o domínio de problema a ser resolvido, a MADEM visa a modelagem de objetivos, papéis, atividades e interações entre entidades de uma organização. A Figura 10 mostra esses conceitos de modelagem e alguns de seus

relacionamentos, conforme definidos na ontologia ONTOMADEM. Os *objetivos* são as metas que uma empresa ou organização quer atingir e podem ser *gerais* ou *específicos.* Objetivos específicos são alcançados com o cumprimento de *responsabilidades* que indivíduos têm exercendo *papéis* com um certo grau de autonomia. Responsabilidades podem ser decompostas em *atividades* e papéis podem ter *destrezas*, que são o conhecimento de uma ou mais técnicas ou regras que suportam a execução de uma atividade ou responsabilidade. Podemos ter também *pré-condições* ou *pós-condições* que devem ser satisfeitas antes ou depois da execução de uma responsabilidade e *conhecimento* pode ser usado ou produzido na execução de uma responsabilidade. *Entidades internas* (papéis ou agentes) exercendo um papel podem precisar se comunicar com outras entidades internas ou *entidades externas* (usuários ou outros sistemas). Na fase de projeto, responsabilidades de papéis são designadas para *agentes*.

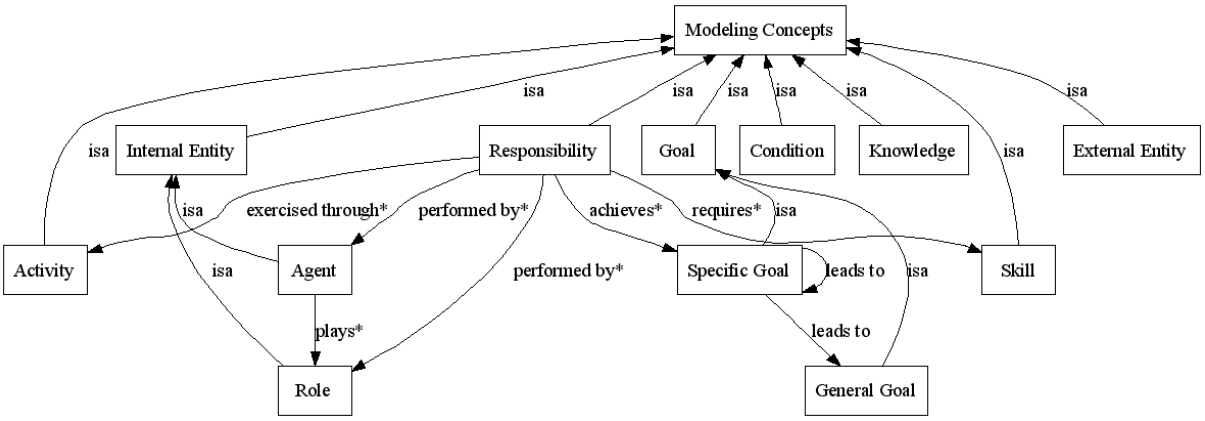

Figura 10 Conceitos da MADEM e alguns de seus relacionamentos

2.6.1.2 Tarefas e produtos da MADEM

A Figura 11 ilustra as fases da MADEM no contexto de um processo de Engenharia de Domínio Multiagente e a Tabela 2 resume as fases, tarefas, atividades e produtos da metodologia. A análise de domínio, suportada pela técnica GRAMO, produz, a partir da análise de aplicações específicas, conhecimento do domínio e requisitos da família de sistemas, um modelo de domínio. A fase de projeto do domínio, suportada pela técnica DDEMAS, consiste no projeto arquitetural e detalhado de modelos de arcabouços multiagente, oferecendo uma solução para os requisitos da família de sistemas multiagente especificada no modelo de domínio.

A técnica DIMAS que suporta a fase de implementação de domínio está em processo de desenvolvimento.

Os conceitos, tarefas e produtos da MADEM e seus inter-relacionamentos são representados na ontologia ONTOMADEM, permitindo a instanciação desses conceitos no editor de ontologias Protégé, que dessa forma será usado como uma ferramenta que permite que o trabalho de análise e projeto do domínio seja efetuado dentro do mesmo a partir da instanciação das classes presentes na ontologia. Portanto, os artefatos de software produzidos pela MADEM têm conceitos semanticamente relacionados e inferências podem ser feitas, facilitando o entendimento e reuso das características comuns e variáveis tanto das soluções de requisitos como de projeto de uma família de aplicações multiagente.

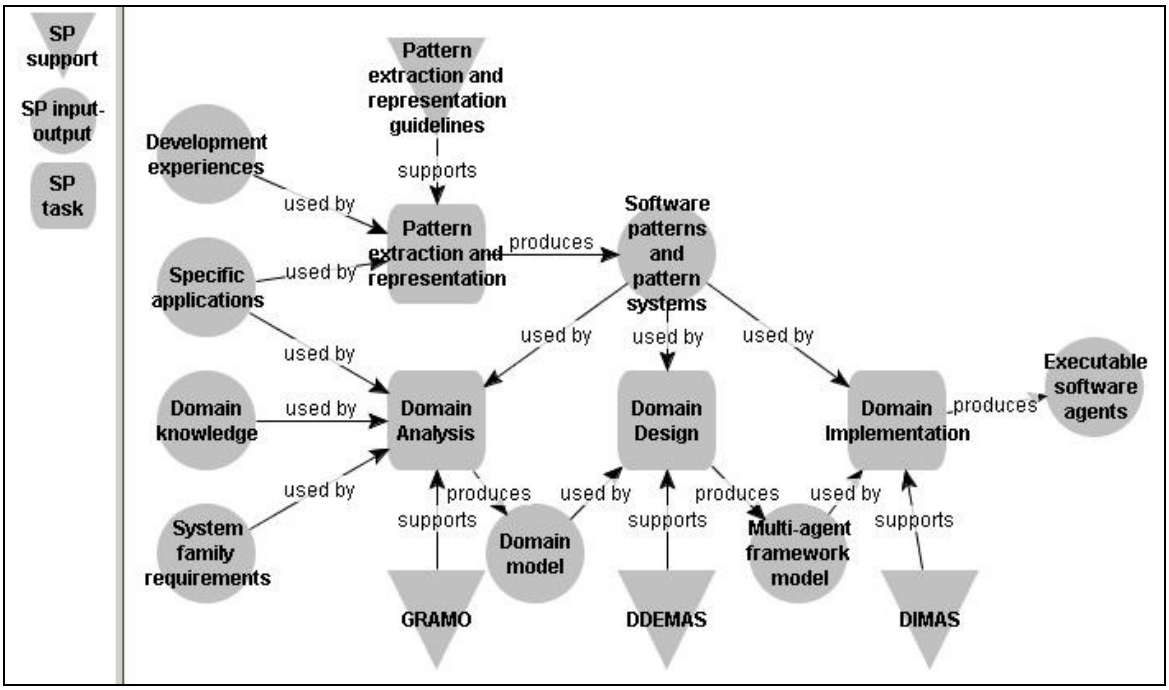

Figura 11 O processo da Engenharia de Domínio Multiagente

As tarefas de análise da ONTOMADEM são modelagem conceitual, modelagem de objetivos, modelagem de papéis, modelagem de variabilidade e modelagem de interações entre papéis. O principal produto da fase de análise é um modelo de domínio. As tarefas da fase de projeto são a modelagem da sociedade multiagente, modelagem de interações entre agentes, modelagem de cooperação e coordenação entre agentes e modelagem do conhecimento da sociedade

multiagente. O principal produto da fase de projeto é o modelo do arcabouço multiagente. Na fase de extração e representação de padrões, a MADEM fornece regras para a extração de padrões e sistemas de padrões das aplicações de software disponíveis, considerando as experiências bem-sucedidas de desenvolvimento.

<table>
<tr><th>Fases</th><th colspan="2">Tarefas</th><th colspan="2">Produtos</th></tr>
<tr><td rowspan="4">Análise de Domínio</td><td colspan="2">Modelagem conceitual</td><td>Modelo conceitual</td><td rowspan="4">Modelo de Domínio</td></tr>
<tr><td rowspan="2">Modelagem de variabilidade</td><td>Modelagem de Objetivos</td><td>Modelo de objetivos</td></tr>
<tr><td>Modelagem de Papéis</td><td>Modelo de Papéis</td></tr>
<tr><td colspan="2">Modelagem de Interações entre papéis</td><td>Modelo de Interações entre papéis</td></tr>
<tr><td rowspan="7">Projeto de Domínio</td><td rowspan="3">Projeto Arquitetural</td><td>Modelagem da sociedade agente</td><td>Modelo da Sociedade Multiagente</td><td rowspan="3">Modelo Arquitetural</td><td rowspan="6">Modelo do arcabouço multiagente</td></tr>
<tr><td>Modelagem de interações entre agentes</td><td>Modelo de Projeto do Framework</td></tr>
<tr><td>Modelagem de cooperação e coordenação</td><td>Modelo de Agentes Refinado</td></tr>
<tr><td colspan="2" rowspan="2">Modelagem de agentes</td><td>Modelos de atividade e conhecimento dos agentes</td><td rowspan="2">Modelos de agentes</td></tr>
<tr><td>Modelos de estado de agentes</td></tr>
<tr><td colspan="2">Modelagem do conhecimento da sociedade multiagente</td><td colspan="2">Modelo do conhecimento da sociedade multiagente</td></tr>
<tr><td colspan="2">Extração e representação de padrões</td><td colspan="2">Padrões de software e sistemas de padrões</td></tr>
</table>

Tabela 2 Resumo da metodologia MADEM [GIRARDI, LINDOSO, 2005]

2.7 Considerações finais

Esse capítulo apresentou diversos aspectos relacionados à produção de software em série, iniciando com os conceitos das linhas de produção de software necessários à customização dos elementos de uma família de sistemas.

A Engenharia de Domínio foi apresentada para termos o entendimento do processo de produção dos artefatos de software que são reutilizados nas linhas de produção de software. A análise da variabilidade também foi abordada apresentando como o tratamento dos conceitos e as abstrações de software são úteis no reuso de software e também apresentando a modelagem de características como técnica para capturar a variabilidade de uma família de sistemas em um domínio específico. Na Engenharia de Domínio Multiagente, vimos que temos um processo e produtos similares ao da Engenharia de Domínio convencional e que a abordagem baseada

em ontologias, como a adotada na metodologia MADEM, é uma boa solução para representar as abstrações de software e favorecer o reuso. Em ambos os casos foram apresentados apenas elementos do processo que permitem favorecer o reuso composicional

Para termos o reuso gerativo, algumas técnicas e abordagens adicionais são necessárias. O próximo capítulo apresentará conceitos e técnicas relacionados à criação da infra-estrutura necessária para prover a reutilização gerativa na construção de famílias de sistemas.

3. O REUSO GERATIVO

Como já foi mencionado anteriormente, o reuso composicional caracteriza-se pela seleção, adaptação e composição ser feita manualmente pelo desenvolvedor na Engenharia de Aplicações, enquanto no reuso gerativo essas tarefas são automatizadas por um artefato de software que favoreça a reutilização automática dos artefatos criados.

O capítulo anterior apresentou a Engenharia de Domínio, seu processo, abstrações e abordagens que são a base para promover o reuso composicional de artefatos de software, incluindo uma metodologia para a Engenharia de Domínio Multiagente, que será utilizada neste trabalho. Este capítulo irá apresentar algumas abordagens que permitem o reuso gerativo na Engenharia de Domínio, que servirão como embasamento teórico, referência e comparativo com para o que vai ser proposto neste trabalho nos capítulos seguintes.

3.1 As Linguagens Específicas de Domínio

Uma das abordagens da reutilização gerativa para especificação de aplicações se dá por meio das Linguagens Específicas de Domínio (LEDs), que são linguagens de programação restritas a domínios de aplicação particulares [THIBAULT, 1998]. Também chamadas de linguagens pequenas ou micro-linguagens, se diferenciam das linguagens de propósito geral (LPGs) por terem conjuntos de notações e abstrações menores, serem mais declarativas que imperativas e terem força expressiva focada num domínio específico [COMPOSE PROJECT, 2004] [DEURSEN, KLINT, VISSER, 2000].

As LEDs não são assunto novo. Desde 1957 já existiam linguagens que, apesar de serem desenvolvidas com enfoque diferente do atual, já apresentavam características de LEDs, como estarem voltadas a um domínio ou finalidade específica e terem vocabulário restrito a esse domínio. A Tabela 3 cita alguns exemplos de LEDs.

Ao contrário das LPGs, as LEDs permitem que o próprio usuário execute a programação de maneira simplificada, pois estas têm alto nível de abstração, ou seja, são focadas no *que* deve ser feito e não em *como* fazê-lo. Outras vantagens

são: o reuso sistemático, através da captura de experiência do domínio de forma implícita ou explícita; e o alto nível de abstração, que serve para evitar possíveis erros de programação, escondendo detalhes de implementação, e viabiliza especificações mais concisas, encurtando o tempo de desenvolvimento de novos produtos bem como sua manutenção.

LED	Domínio de aplicação
BNF	Especificação de sintaxe
Ling.de macros do Excel	Planilhas de cálculo
HTML	Páginas de internet em hipertexto
LATEX	Edição de textos
Make	Compilação / linkedição de software
SQL	Consultas a banco de dados
VHDL	Projeto de hardware

Tabela 3 Alguns exemplos de LEDs [MERNIK, HEERING, SLOANE, 2003]

Como desvantagens, tem-se o custo de projetar, implementar e manter uma LED, impondo que elas sejam desenvolvidas para especificar sistemas pertencentes a uma família; e a potencial perda de eficiência em relação ao código escrito em LPG, posto que estes são escritos sob medida para cada situação, enquanto que naquelas a geração é automática.

3.1.1 LEDs e Bibliotecas de aplicação

Um possível questionamento que pode surgir é por que usar LEDs em vez de bibliotecas de aplicação aliadas a LPGs? Evidentemente muitas LEDs passam antes pelo estágio de biblioteca de aplicação, e as bibliotecas muitas vezes constituem a solução mais viável (quando o custo de criação da LED é maior que o benefício obtido com o desenvolvimento de aplicações na mesma), mas considerando que a LED é a solução mais viável, esta leva vantagem justificada nos seguintes pontos:

- LEDs estabelecem notações específicas de domínio, que estão além do que é possível criar nas LPGs na forma de notações e operações definidas pelo usuário, e as notações da LED são nativas, existindo desde o surgimento da mesma, sendo a base do ganho de produtividade obtido com as LEDs;

- Uma LPG com biblioteca de aplicação só consegue expressar as construções e abstrações específicas do domínio indiretamente ou de forma inadequada e repetitiva, enquanto a LED incorpora isso de forma nativa;

3.1.2 Conceitos relacionados a uma LED

Os conceitos presentes no âmbito das LEDs são similares aos das LPGs, porém podem ser usados ou enfatizados de forma diferente. Alguns desses conceitos, muitos deles encontrados em maiores detalhes em [SLONNEGER, KENETH, 1995], [SCHMIDT, 1986] e [AHO, RAVI, ULLMAN, 1995], serão descritos resumidamente a seguir.

Toda notação usada para dar instruções pode ser considerada uma linguagem de programação. Partindo desse ponto de vista, podemos enumerar como linguagens de programação:

- A notação aritmética, para o matemático;
- A linguagem Pascal para um programador;
- Comandos de entrada, para uma pessoa que usa um programa;

No âmbito computacional, todas as linguagens ou pontos de vista convergem para os aspectos físicos da máquina, onde o código vai ser convertido em ação e todas têm as seguintes características:

- *Sintaxe*: refere-se à aparência e estrutura das sentenças, ou seja, os símbolos usados para construir termos, a estrutura dos termos e modo como os símbolos podem ser combinados para criar sentenças (ou programas) corretas. Existem duas definições de sintaxe, que estão ilustradas na Figura 12 e descritas a seguir:
    - *Sintaxe concreta:* descreve a estrutura do programa na forma visível e utilizável pelo desenvolvedor, normalmente em termos de expressões representativas na forma de seqüência de caracteres ou símbolos. A sintaxe concreta de linguagens textuais é definida com o uso de gramáticas;
    - *Sintaxe abstrata:* descreve a estrutura das árvores de sintaxe abstrata usada na representação interna do programa pelo

compilador (ou outro processador de linguagem). É independente da sintaxe concreta e de plataforma de hardware, apesar de poder ser bem similar.

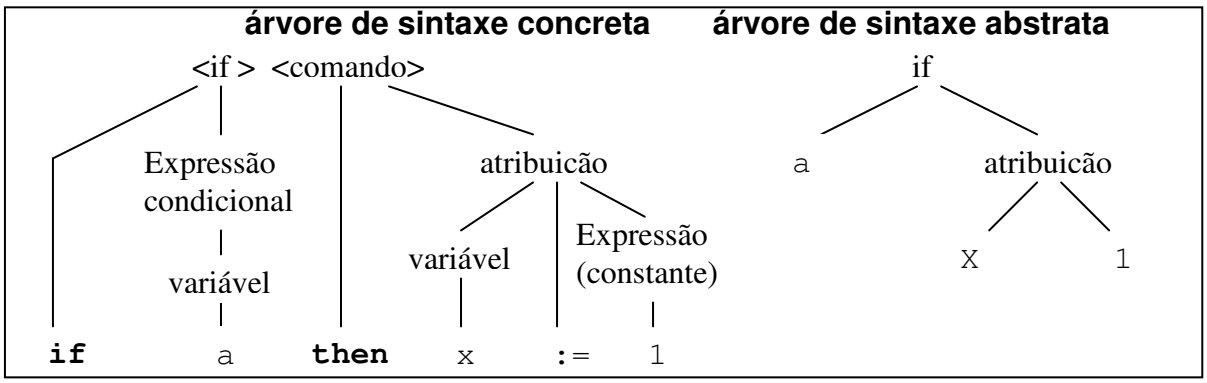

Figura 12 Árvore de sintaxe concreta e abstrata para o comando if a then x:=1

- *Semântica:* Em linhas gerais, consiste na atribuição de significados às sentenças. Na linguagem natural, a semântica significa correlacionar as sentenças e frases com os objetos, pensamentos e sentimentos já experimentados. Como exemplo de significados podemos citar números e funções (para os matemáticos), ações da máquina (para o programador), tons audíveis (para os músicos) [SCHMIDT, 1986]. Nas linguagens de programação a semântica descreve o comportamento que o computador segue ao executar um programa na linguagem, que deve ser expresso descrevendo o relacionamento entre a entrada e a saída de um programa ou uma explicação passo a passo de como o programa irá executar numa máquina real ou abstrata [SLONNEGER, KURTZ, 1995]. Existem duas definições de semântica:
  - *Semântica estática:* Corresponde às ações tomadas pelo compilador, que irão transformar o programa-fonte em uma outra forma de programa executável pela máquina e que dependem do programa compilado e do ambiente de execução;
  - *Semântica dinâmica:* são ações em tempo de execução, ou seja, computações que dependem dos dados de entrada do programa e produzem a resposta relativa ao problema tratado.

- *Pragmática:* Especifica a usabilidade da linguagem, em relação às possíveis áreas de aplicação da linguagem, sua facilidade de implementação e uso, e sucesso da linguagem no alcance dos objetivos da mesma.

O *processador* da linguagem de programação é o dispositivo que vai receber uma descrição na linguagem e processá-la (ex: um compilador ou interpretador Pascal). Tem duas partes principais:

- Módulo de validação da entrada (*parser*): lê a entrada e verifica se a sintaxe está correta;
- Módulo de avaliação: avalia a entrada para sua saída correspondente, expondo a semântica da entrada.

Parsing é o processo de tradução da sintaxe concreta para a sintaxe abstrata e um *parser* é a parte de um processador de linguagem que faz a análise sintática da seqüência de entrada (linha de programa na linguagem de programação), determina sua estrutura gramatical e transforma a seqüência linear de *tokens* em uma árvore de sintaxe abstrata (ASA ou AST de *Abstract Syntax Tree*). Um *token* é um bloco de texto categorizado, indivisível e reconhecido como um termo válido da linguagem. *Tokens* geralmente são definidos com expressões regulares.

Uma *expressão regular* (abreviada como "*regexp*" ou "*regex*"), também chamada de *pattern* (padrão ou modelo), é uma expressão que descreve ou equipara um conjunto de *strings*, de acordo com certas regras de sintaxe e permite dar uma concisa descrição de um conjunto sem ter que listar todos seus elementos (ex.: o conjunto de *strings* {Handel, Händel, Haendel} pode ser descrito com o pattern "H(ä|ae?)ndel". São usadas em qualquer área que envolva processamento de texto como compiladores, processadores de linguagem, editores de texto (ex: sed) e utilitários (ex: grep), para procurar e manipular partes de texto baseados em certos padrões. Algumas LPGs suportam expressões regulares para manipulação de strings (ex.: Perl, Tcl, Java).

3.1.3 Fases genéricas do desenvolvimento de LEDs

Mernik, Heering e Sloane (2003) realizam uma distinção adequada das fases do desenvolvimento de LEDs, que se torna oportuna neste trabalho pois cita padrões de projeto e implementação de LEDs e serve de referência na classificação das atividades das metodologias existentes, cuja maioria não classifica suas atividades dentro dessas grandes fases. As fases são:

- Decisão
- Análise;
- Projeto
- Implementação;
- Distribuição;

Essas fases estão descritas sucintamente a seguir, com os padrões conhecidos em cada uma.

3.1.3.1 Decisão

Consiste em avaliar, após uma análise preliminar do problema, a viabilidade de se criar uma LED, pois o investimento no desenvolvimento da LED deve ser recuperado com a redução de custos no desenvolvimento e manutenção de software. Os padrões influenciadores da decisão estão relacionados a itens como a redução de custos, habilitação do usuário final à programação ou especificação e habilitação de análise, verificação, otimização e / ou transformação específica de domínio. Os padrões de decisão incluem necessidades de mudança de notação, automação de tarefas de codificação repetitiva que seguem um mesmo padrão, representação de estruturas de dados, interface específica de domínio para manipular a adaptação e configuração do sistema e criação de interfaces diferenciadas com o usuário (como a linguagem de macros do Excel).

3.1.3.2 Análise

Nesta fase deve-se identificar e delimitar o domínio do problema e adquirir conhecimento sobre o mesmo. Isso pode ser feito de três formas, que são os três padrões de análise: *informalmente*; *formalmente*, com o uso de metodologias de

análise de domínio como FODA (*Feature Oriented Domain Analysis*) [KANG *et al.*, 1990], DSSA (*Domain Specific Software Architecture*) [TAYLOR, TRACZ, COGLIANESE, 1995] e ODM (*Organization Domain Model*) [SIMOS, ANTHONY, 1998]; ou através da *extração* (*mining*) de código já existente em alguma LPG. Os métodos oriundos da Engenharia do Conhecimento, baseados na captura e representação do conhecimento e ontologias, como o GRAMO (*Generic Requerement Analysis Method based on Ontologies*) [FARIA, 2004], são de grande aplicabilidade nesta fase, por oferecerem uma maneira natural de capturar os conceitos do domínio. Os requisitos de entrada desta fase são os especialistas do domínio, documentos ou códigos onde o conhecimento do domínio pode ser obtido. O principal produto desta fase é uma representação do conhecimento obtido do domínio, normalmente em um modelo de domínio que pode usar representações baseadas em diagramas de características, ontologias ou outras formas de representação.

3.1.3.3 Projeto

Nesta fase o foco é a sintaxe e semântica da linguagem. Para esta fase são identificados quatro padrões (Tabela 4). Os dois primeiros classificam o projeto da LED de acordo com o relacionamento que a LED tem com linguagens existentes:

- *Projeto baseado em linguagens existentes (aproveitamento de linguagem):* É a forma mais fácil de projetar uma LED, mas prejudica a pragmática, pois favorece principalmente usuários programadores, que são conhecedores da linguagem existente. Existem três casos especiais neste item:
    - *Superposição (Piggyback):* consiste em agregar características específicas de domínio à linguagem existente;
    - *Especialização:* implica em restringir a linguagem existente para prover especialização centrada no domínio do problema;
    - *Extensão:* consiste em estender uma linguagem existente com novas características que referenciam conceitos de domínio. Na maioria dos casos, os recursos da linguagem continuam

disponíveis. A dificuldade neste caso é integrar os recursos específicos de domínio com o resto da linguagem;

- *Projeto por invenção:* é o projeto de LED sem relacionamento com linguagens já existentes. Neste caso o esforço é bem maior e deve-se cuidar para manter um equilíbrio entre as propriedades específicas das LEDs e a simplicidade, visto que os usuários da LED não necessariamente serão programadores. Duas dicas são evitar aprimorar as notações do domínio e controlar a tendência de embelezar ou generalizar demais a linguagem;

Os outros dois padrões classificam a LED de acordo com a formalidade da descrição do projeto:

- *Projeto informal:* a especificação está geralmente em alguma forma de linguagem natural e incluindo um conjunto de programas-exemplo na LED. Evidentemente é a abordagem mais fácil para a maioria das pessoas porém, como dão maior foco à sintaxe que à semântica, podem conter imprecisões que podem causar problemas na fase de implementação;
- *Projeto formal:* consiste em uma especificação definida usando um dos métodos de definição semântica conhecidos, que incluem expressões regulares e gramáticas para definição de sintaxe e gramática de atributos, sistemas de reescrita e máquinas abstratas para definição da semântica. O padrão formal, por impor disciplina na especificação da sintaxe e semântica, permite descobrir possíveis problemas antes de passar à fase de implementação;

Padrão	**Descrição**
Aproveitamento de linguagem	A LED está baseada em uma linguagem existente. Casos especiais: Superposição (*Piggyback*): linguagem existente é parcialmente usada Especialização: Linguagem existente restrita Extensão: Linguagem existente é estendida
Invenção	A LED é projetada do zero sem influência de linguagens existentes
Informal	A LED é descrita informalmente
Formal	A LED é descrita formalmente usando um método de definição semântica existente

Tabela 4 Padrões do projeto de LEDs [MERNIK, HEERING, SLOANE, 2003]

3.1.3.4 Implementação

A implementação de uma LED consiste na concepção de uma solução para o processamento das especificações (programas) feitos na mesma. Essa solução depende do padrão escolhido para o projeto da linguagem e de outros fatores como o tipo de código que será gerado (linguagem alvo), performance desejada, recursos e nível de conhecimento sobre compiladores e geradores.

Em [SPINELLIS, 2001] considera-se que a implementação de uma LED difere da implementação de uma LPG. O escopo reduzido das LEDs permite e requer estratégias diferentes. Por exemplo, a simplicidade léxica, sintática e semântica das LEDs permite omitir alguns elementos que seriam necessários num compilador de uma LPG, como um *parser* primário, já que as implementações das LEDs freqüentemente processam a linguagem fonte usando expressões regulares. A Tabela 5 relaciona os padrões identificados para esta fase, cuja seleção é influenciada pelos fatores citados no parágrafo anterior:

- *Interpretador:* é adequado quando temos linguagens de caráter dinâmico e quando velocidade de execução não é tão importante. É um padrão que oferece controle sobre o ambiente de execução e facilidade de extensão;
- *Compilador / gerador de aplicações:* caracteriza-se pelo uso de um gerador de aplicações como “compilador” da LED, oferecendo potencialmente maior velocidade de execução do que com o uso de um interpretador, visto que o a especificação feita na LED é traduzida para estruturas na linguagem alvo e chamadas a bibliotecas, que serão compiladas para originar um código executável;
- *Pré-processador*: consiste em usar o recurso de pré-processamento existente em algumas LPGs para traduzir a notação da LED para estruturas da linguagem-alvo previamente definidas. É um recurso um pouco limitado porque nem todas as LPGs dão suporte ao mesmo e porque limita a análise estática e checagem de erros às checagens feitas pelo pré-processador da linguagem alvo, porém é uma solução prática e de fácil implementação;

- *Inserção (embedding):* consiste em embutir as estruturas da LED em uma LPG existente (chamada de linguagem hospedeira), extendendo a mesma com o uso de novos tipos abstratos e operadores com bibliotecas de aplicação ou meta-programação (*template metaprogramming*);
- *Compilador / interpretador extensível:* é a extensão de uma implementação de linguagem já existente. Para uma inclusão apropriada das características de uma LED, deve-se usar pré-processadores ou compiladores que permitam a adição de regras de otimização ou geração de código específico de domínio;
- *Commercial-Off-The-Shelf (COTS):* consiste em usar ferramentas e notações existentes (produtos de prateleira nem sempre destinados à implementação de LEDs) na implementação da LEDs. Exemplos: uso do PowerPoint como ferramenta específica para diagramação, definição de LEDs baseadas em XML;
- *Híbrida:* consiste em usar combinações dos padrões anteriores;

Padrão	**Descrição**
Interpretador	O programa na LED é reconhecido e interpretado usando um ciclo padrão ler – decodificar e executar
Compilador / gerador de aplicações	O programa na LED é traduzido para estruturas em uma linguagem-alvo e chamadas a biblioteca, permitindo uma análise estática completa no mesmo.
Pré-processador	As estruturas da LED são traduzidas para estruturas da linguagem-alvo e a análise estática fica limitada à feita pelo processador da linguagem-alvo.
Inserção (*embedding*)	Estruturas da LED são embutidas em uma LPG existente pela definição de novos tipos abstratos e operadores. Ex: bibliotecas de aplicação.
Compilador ou interpretador extensível	Um compilador ou interpretador de uma LPG é estendido com regras de otimização específicas de domínio
Commercial Off-The-Shelf (COTS)	Ferramentas e / ou notações existentes são aplicadas a um domínio específico.
Híbrida	Uma combinação dos padrões anteriores

Tabela 5 Padrões de implementação de LEDs [MERNIK, HEERING, SLOANE, 2003]

O padrão interpretador e compilador são os que podem oferecer notações mais próximas às usadas pelos especialistas do domínio e melhor detecção de

erros, porém exigem um alto esforço de desenvolvimento, que pode ser reduzido com o uso de ferramentas para criação de linguagens. O COTS também pode oferecer boas abstrações, mas tem fica limitado ao que o produto oferece. Os outros padrões implicam no conhecimento da linguagem-alvo, o que consiste uma barreira para usuários não programadores, sendo que o padrão pré-processador ainda depende da existência desse recurso na linguagem-alvo escolhida.

3.1.4 Técnicas para o desenvolvimento de LEDs

Apesar de já existirem LEDs há muito tempo, algumas com bastante sucesso e reconhecimento, somente nos últimos anos se iniciaram estudos direcionados à criação de metodologias para seu desenvolvimento sistemático, não meramente artesanal [THIBAULT, 1998].

As iniciativas existentes no sentido de se obter metodologias para o desenvolvimento de LEDs diferem entre si, por exemplo, com relação aos formalismos que utilizam e aos artefatos que produzem.

A seguir serão mostradas algumas técnicas e metodologias para construção de LEDs, incluindo a técnica TOD-LED [SERRA, 2004], que será objeto de extensão nesta pesquisa, tentando-se estruturá-las da maneira mais similar possível, de modo a facilitar a comparação entre elas.

3.1.4.1 A Técnica TOD-LED

A TOD-LED [SERRA, 2004] é uma técnica que guia o processo de especificação de LED’s a partir de um modelo de domínio resultante da aplicação da técnica GRAMO [FARIA, 2004] para a captura de requisitos na Engenharia de Domínio Multiagente. A TOD-LED define as tarefas a serem realizadas na especificação de LED's através da instanciação da ONTOLED, tendo como subsídio esse modelo de domínio. A seguir são apresentadas as principais características desta técnica.

- *Especificações baseadas em papéis*: Isso significa que o programa consiste na declaração de um conjunto de papéis que farão parte da solução de um problema. Papéis são entidades ativas que realizam um conjunto de atividades e que juntos cooperam para o alcance de um objetivo específico. Os papéis podem

possuir propriedades, que servem para ajustar alguns aspectos de seu funcionamento e podem ainda ser agrupados em pacotes, que são entidades compostas de papéis relacionados. O desenvolvedor não precisa especificar como o papel deve realizar suas atividades, e sim apenas incluir um papel em um programa para que esse cumpra sua responsabilidade de acordo com o que está especificado no modelo de domínio;

- *Puramente declarativa:* de forma similar à maioria das LED's já desenvolvidas, a TOD-LED especifica o que deve ser feito em vez de como. Programas codificados segundo essas LED's consistem basicamente da declaração de papéis e de propriedades a eles relacionadas;

- *Voltada para a Engenharia de Domínio Multiagente*: constitui-se em uma iniciativa pioneira para o desenvolvimento de LED's na Engenharia de Domínio Multiagente, que fornece, junto com a GRAMO [FARIA, 2004], um roteiro desde a análise de domínio até a especificação de LED's;

- *Sintaxe, semântica e exemplo de aplicação*: Usa a gramática BNF para definição da sintaxe. A sua definição da semântica usa uma abordagem baseada em ontologias, a partir do modelo de domínio originado com o uso da técnica GRAMO. Tem como exemplo de aplicação o desenvolvimento da LESRF, que é uma LED para a especificação de sistemas para o domínio do acesso a informação dinâmica e não estruturada.

A TOD-LED é constituída de quatro tarefas: Especificação da Sintaxe Concreta, Especificação da Sintaxe Abstrata, Especificação da Semântica e Especificação da Pragmática. Essas tarefas e os produtos obtidos em cada uma delas estão resumidos na Tabela 6.

<table>
<tr><th colspan="2">Tarefas</th><th>Produtos</th></tr>
<tr><td rowspan="2">Especificação da Sintaxe Concreta</td><td>Especificação dos papéis e propriedades</td><td rowspan="2">Vocabulário da LED
(Lista de Papéis e Propriedades da LED e Lista de Pacotes da LED)</td></tr>
<tr><td>Especificação dos pacotes</td></tr>
<tr><td colspan="2">Especificação da Sintaxe Abstrata</td><td>Gramática da LED</td></tr>
<tr><td colspan="2">Especificação da Semântica</td><td>Máquina abstrata da LED</td></tr>
<tr><td colspan="2">Especificação da Pragmática</td><td>Documentação do usuário</td></tr>
</table>

Tabela 6 Tarefas e produtos da TOD-LED

3.1.4.2 Sprint

Sprint [CONSEL, MARLET, 1998] é uma metodologia para o projeto e implementação de linguagens específicas de domínio baseada no framework proposto por Thibault e Consel (1997) e avaliada por Thibault (1998) no desenvolvimento da GAL – Graphics Adaptor Language, uma LED para a descrição de drivers de dispositivos de vídeo, e da PLAN-P, uma outra para o domínio de protocolos de aplicação em redes ativas (redes com roteadores programáveis). A Tabela 7 resume as fases e produtos dessa técnica.

Fases	**Atividades**	**Produtos**
Análise do domínio da linguagem – similar a uma análise de problema convencional, só que focando aspectos da linguagem	Análise de requisitos da linguagem	Termos do domínio
	Descrição de objetos e operações	- Descrição dos objetos e operações necessários para expressar soluções para a família de problemas
	Elementos de projeto	Escolha do paradigma (declarativo ou imperativo) - Terminologia e notação da linguagem
Definição da Interface	Definição da álgebra semântica	Álgebra semântica (domínios semânticos e operações)
	Definição da sintaxe	Sintaxe
Semântica particionada	Divisão da semântica entre ações de tempo de compilação e de tempo de execução	Ações / verificações estáticas e dinâmicas
Definição formal	Definição de funções de avaliação	Funções de avaliação
Máquina abstrata	Definição de um modelo computacional que suporte a implementação dos sistemas da família	Modelo da máquina abstrata ex: biblioteca
Implementação	Implementação da máquina abstrata	- Implementação da máquina abstrata - Interpretador ou compilador
Avaliação parcial [CONSEL, DANVY, 1993]	Transformar automaticamente um programa LED em um programa compilado a partir do interpretador	Código compilado

Tabela 7 Resumo da técnica Sprint

3.1.4.3 SDA

O método SDA ("*Software Design Automation*") [WIDEN, 1995] [WIDEN, HOOK, 1998] para automação do projeto de software é uma iniciativa do Pacific Software Research Center (PacSoft) que consiste em uma abordagem matemática

para resolver problemas e analisar soluções usando conceitos como modelos matemáticos (que tornam possível expressar os requisitos do domínio do problema formalmente), *monads*, projeto de linguagem baseado em semântica, e tecnologias de compilação funcional.

Para ilustrar a aplicação da metodologia foram usados exemplos referentes ao desenvolvimento de uma LED para a especificação de sistemas de controle de bóias meteorológicas FWS (*Floating Weather Stations*), que constituem uma família de sistemas que provêem dados sobre as condições climáticas para orientar o tráfego marítimo e aéreo. As bóias coletam esses dados através de vários sensores. Cada bóia tem um transmissor de rádio para enviar informações meteorológicas a satélites que as repassam para aviões e navios. A Tabela 8 resume as fases, atividades e produtos dessa técnica.

Fases	**Atividades**	**Produtos**
Analise do domínio	Capturar uma definição escrita	Definição do domínio
	Formular o modelo de domínio formal	Modelo formal do domínio
	Definir o modelo de solução	Modelo de solução
	Capturar a interface do sistema legado	Documento de interface e restrições do ambiente
	Validar modelos	
Definição da linguagem	Capturar a definição inicial da linguagem	- Definição inicial da linguagem - Semântica da linguagem
Implementação do gerador	- Projetar o gerador e os produtos de suporte - Desenvolver incrementalmen-te o gerador e os produtos de suporte	- Projetos e planos para a construção do gerador e dos produtos de suporte - Gerador e produtos de suporte

Tabela 8 Resumo da técnica SDA

3.1.4.4 FAST

A FAST ("Family-oriented Abstraction Specification and Translation") [WIDEN 1995] [WIESS, 1995], proveniente da *Lucent Technology Inc*. (Lucent), é uma metodologia para a construção de sistemas como instâncias de uma família de sistemas que têm descrições similares. Derivou da metodologia *Synthesis*, que incorpora conceitos como processos iterativos e reuso baseado em abstração e integra como suas atividades a Engenharia de Aplicações e a Engenharia de Domínio.

O método para Abstração, Especificação e Tradução orientado por Famílias permite analisar famílias de softwares e desenvolver LEDs para elas. Ele inclui um prático e documentado sub-processo de análise de semelhanças para coletar, analisar e capturar informações do domínio. A evolução de uma análise de semelhanças para uma ferramenta de Engenharia de Aplicação e ambiente é que não são bem definidos nesse método. Suas fases e produtos estão resumidas na Tabela 9.

Fases	**Ações**	**Produtos**
Analisar a família	- Qualificar o domínio - Analisar semelhanças - Definir modelo de decisão - Projetar linguagem de modelagem de aplicação - Definir o mapeamento composicional	- Modelo de domínio - Projeto do conjunto de ferramentas
Implementar a família	- Projetar o ambiente de engenharia de aplicação - Criar um processo de engenharia de aplicação padrão - Implementar o ambiente de engenharia de aplicação - Documentar o ambiente de engenharia de aplicação	- Processo de engenharia de aplicação padrão - Biblioteca - Ferramentas de geração - Ferramentas de análise - Documentação

Tabela 9 Resumo da técnica FAST

3.1.4.5 DEMRAL

DEMRAL (*Domain Engineering Method for Reusable Algorithmic Libraries*) [CZARNECKI, 1998] é um método de Engenharia de Domínio para o desenvolvimento de bibliotecas algorítmicas (como bibliotecas numéricas, biblioteca de contêineres de dados, processamento ou reconhecimento de imagem, reconhecimento de voz e computação gráfica) que têm as seguintes características:

- Conceitos principais podem ser capturados como tipos abstratos de dados (TAD) e algoritmos que operam nesses TADs;
- Os TADs reconhecidos frequentemente têm propriedades de containeres como matrizes, imagens, gráficos, etc;
- Estão fundamentadas em uma bem desenvolvida teoria matemática já existente, como álgebra linear, álgebra de imagens ou teoria de grafos;
- Os TADs e algoritmos vêm em grande variedade (tipos);

Desenvolvida como uma especialização do método ODM (Organization Domain Modeling), que é um método para a Engenharia de Domínio, o DEMRAL combina idéias de Engenharia de Domínio, Geradores e Metaprogramação, Programação Orientada a Aspectos e Desenvolvimento de Software Orientado a Objetos. As atividades do DEMRAL serão descritas a seguir.

O processo de desenvolvimento no DEMRAL é iterativo e incremental podendo ter atividades que acontecem ou se repetem em ordem arbitrária. A Figura 13 mostra um resumo do método DEMRAL.

As atividades do DEMRAL derivam das fases e atividades do ODM, tendo as seguintes diferenças: adota a divisão mais largamente aceita da Engenharia de Domínio (Análise de Domínio, Projeto de Domínio e Implementação de Domínio), para as quais as fases do ODM são facilmente mapeadas; tem foco maior em questões técnicas mas pode ser estendida para questões organizacionais; cobre apenas um subconjunto das atividades da ODM.

A especialização da ODM feita no DEMRAL inclui o foco nas categorias de conceitos *TADs* e *algoritmos,* oferece *conjuntos iniciadores de características* (conjunto de recursos que auxiliam a descoberta de caraterísticas - vide seção 2.5.3.1) especializados para essas duas categorias, adota a modelagem de características como tarefa de modelagem e foca o desenvolvimento de LEDs.

1. Análise de Domínio
    1.1. Definição do Domínio
        1.1.1. Análise de objetivos e interessados no projeto ("*stakeholders*")
        1.1.2. Delimitação do domínio e análise de contexto
            1.1.2.1. Análise de áreas de aplicação e sistemas existentes
            1.1.2.2. Identificação de características do domínio
            1.1.2.3. Identificação de relacionamentos com outros domínios
    1.2. Modelagem do domínio
        1.2.1. Identificação de conceitos-chave
        1.2.2. Modelagem de características dos conceitos-chave (identificação de similaridades e variabilidades e dependências/interações entre características)
2. Projeto de Domínio
    2.1. Identificação e especificação de Linguagens Específicas de Domínio
    2.2 Identificação da arquitetura de implementação
3. Implementação do Domínio (implementação da linguagem específica de domínio, tradutor da linguagem e componentes de implementação)

Figura 13 Resumo do método DEMRAL [CZARNECKI 1998]

As atividades do DEMRAL serão descritas resumidamente a seguir:

1) Análise de Domínio: Composta das sub-atividades definição do domínio e modelagem do domínio;

1.1) Definição do Domínio:

1.1.1) Análise de objetivos e usuários: Inicialmente identificam-se os objetivos e interessados no projeto ("*stakeholders*"), resultando numa lista priorizada e cruzada de objetivos e interessados. A complexidade desta tarefa depende do tamanho e contexto do projeto ou domínio em questão;

1.1.2) Delimitação do domínio e análise do contexto: Nesta atividade deve-se determinar os limites do domínio e caracterizar seu conteúdo. Isso é feito através das atividades:

1.1.2.1) Análise de áreas de aplicação e sistemas existentes: faz-se uma análise de áreas de aplicação de sistemas no domínio e análise de sistemas já existentes nesse domínio. O produto deste passo é um diagrama de características do domínio que descreve quais características são ou podem fazer parte do domínio, denotando também as prioridades das características. A Figura 14 mostra o diagrama de características da fase de definição do domínio de bibliotecas de cálculo de matrizes. As prioridades permitem a qualificação da importância da característica nas áreas de aplicação, nos sistemas-exemplo analisados e ajudam a definir que partes do domínio implementar primeiro. Durante esta atividade é recomendado estabelecer um dicionário do domínio e um registro das fontes de conhecimento sobre o domínio.

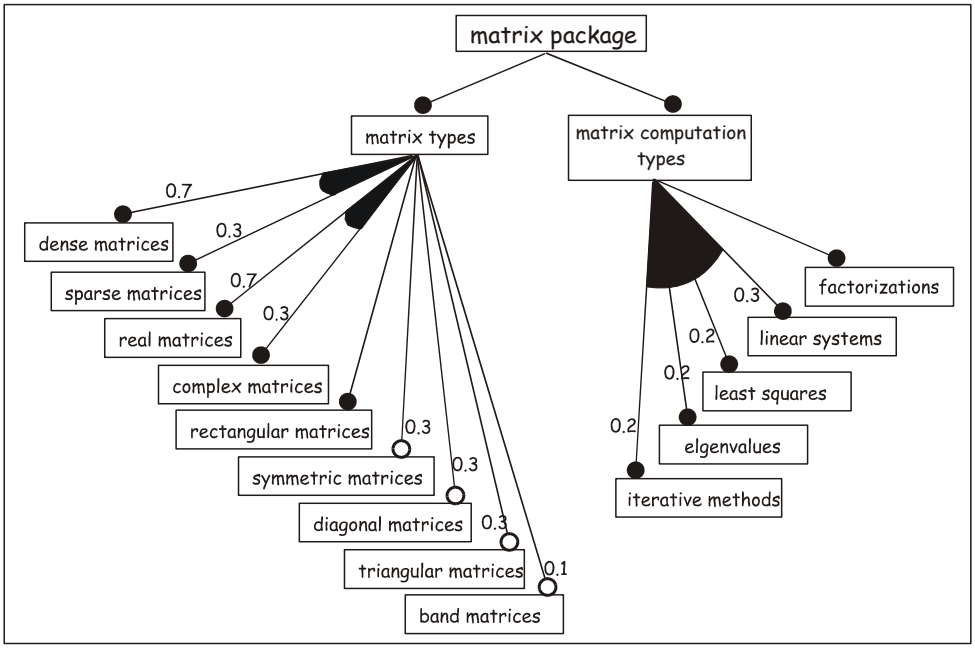

Figura 14 Diagrama de características do domínio de bibliotecas de cálculo de matrizes [CZARNECKI, 1998]

1.1.2.2) Identificação de relacionamentos com outros domínios: A última sub-atividade da análise de domínio visa identificar relacionamentos do domínio em estudo com outros domínios que possam contribuir no entendimento como, por exemplo, domínios-analogia (domínios que têm similaridades significantes com o domínio em estudo) e domínios-suporte (domínios que podem ser usados para expressar alguns aspectos do domínio em análise);

1.2) Modelagem do domínio:

1.2.1) Identificação dos conceitos-chave: Por definição, o DEMRAL foca domínios cujos principais categorias de conceitos são TADs (Tipos Abstratos de Dados) e algoritmos e a identificação dos TADs chaves é simples. Exemplo: os TADs chaves nas bibliotecas de cálculo de matrizes são matrizes e vetores e os TADs chaves no processamento de imagens são vários tipos de imagens.

1.2.2) Modelagem de características: Visa desenvolver modelos de características dos conceitos no domínio, que definem as características comuns e variáveis de instâncias de conceitos e dependências entre as características. Para isso a metodologia oferece conjuntos iniciadores de características para TADs (ex: atributos, estruturas de dados, operações, detecção / resposta / tratamento de erros, gerenciamento de memória, sincronização, persistência, perspectivas e subjetividade) e algoritmos (exemplos: aspecto computacional, acesso a dados, otimizações, detecção / resposta / tratamento de erros, gerenciamento de memória).

2) Projeto de domínio: No DEMRAL o projeto do domínio visa desenvolver, a partir dos resultados da modelagem do domínio, uma arquitetura de biblioteca que consiste de uma decomposição em pacotes e a especificação das LEDs usuárias, que incluem a sintaxe abstrata e especificações de implementação, numa forma que sirva de base para a implementação usando o padrão de extensão de linguagem. O projeto de domínio tem as seguintes sub-atividades:

2.1) Identificação da arquitetura de implementação: É a identificação de pacotes. A biblioteca é dividida para ser desenvolvida em um número de pacotes que possibilitam: definição dos módulos de alto nível da biblioteca, minimizando dependências e facilitando o entendimento; desenvolvimento simultâneo de pacotes diferentes por diferentes programadores; importação somente dos pacotes que a aplicação requer. A estratégia básica no DEMRAL é colocar cada TAD e cada

família de algoritmo em um pacote separado. Diagramas de pacotes UML podem ser usados para representar a visão de pacotes na biblioteca em desenvolvimento;

2.2) Identificação e especificação de LEDs: Envolve as atividades do projeto do domínio relacionadas à definição das LEDs:

2.2.1) Identificação de LEDs usuárias: LEDs usuárias são LEDs oferecidas pela biblioteca aos seus usuários como interfaces de programação (APIs). No caso do DEMRAL podem ser LEDs de configuração (usadas para configurar TADs e algoritmos) ou LEDs de expressão (LEDs usadas para escrever expressões envolvendo TADs e operações com elas).

2.2.2) Identificação de interações entre LEDs: consiste em identificar que tipos de informações devem ser trocadas entre as implementações das LEDs. Por exemplo: a implementação da LED de expressão de matrizes precisará acessar diferentes propriedades de matrizes (ex: tipo de elemento, padrão, formato), os quais são descritos pela LED de configuração de matrizes. Essa informação é necessária para implementar as operações contidas numa expressão com os algoritmos mais adequados de acordo com o tipo de argumentos.

2.2.3) Delimitação de LEDs: é selecionar as características dos modelos de características que serão cobertos pela implementação da LED baseando-se nas prioridades das características, objetivos do projeto e recursos.

2.2.4) Especificação de LEDs: consistem em especificar a sintaxe e semântica, atendo-se apenas à sintaxe abstrata (estrutura) de cada LED e deixando a sintaxe concreta para a Implementação do Domínio. A sintaxe abstrata é especificada na forma de gramática abstrata e a sintaxe concreta com o Formalismo *Backus-Naur* (BNF).

3) Implementação do Domínio: TADs, operações e algoritmos podem ser implementados usando funções parametrizadas, classes parametrizadas e camadas mistas.

3.1.4.6 Resumo comparativo das técnicas

A Tabela 10 faz um resumo comparativo das técnicas. Podemos notar que a maioria das técnicas, por serem textuais, adota especificação da sintaxe com a gramática BNF. Por esse mesmo motivo a semântica denotacional também é o

formalismo comumente usado para a especificação da semântica. Nem todas as técnicas abordam todas as fases citadas na seção 3.1.3 e na maioria delas, com exceção do DEMRAL, não fica claro o mapeamento das tarefas da técnica para as fases genéricas da Engenharia de Domínio e desenvolvimento de LEDs. Quanto à técnica de análise de domínio a maioria usa técnica própria ou adaptação de alguma técnica existente.

	Especifi-cação da sintaxe	**Especifi-cação da semântica**	**LED's desenvol-vidas**	**Fases abordadas**	**Técnica de Análise de Domínio**	**Implemen-tação**	**Documen-tação**
TOD-LED	BNF	Ontologias	LESRF	Análise	GRAMO	-	Sim
Sprint	Gramática BNF	Semântica denota-cional	GAL, PLAN-P [THIBAULT, 1998]	Análise, projeto e implementação	Não especifica do	Interpretador e avaliação parcial	Não
SDA	Não especifica-do	Semântica denota-cional	MSL [WIDEN, HOOK, 1998]	Projeto e implementação	Técnica própria	Interpretador	Não
FAST	Gramática BNF	Não especificado	Auditdraw [WEISS, 1995]	Análise, projeto e implementa-ção	Não especifica do	Não especificado	não
DEMRAL	Árvore abstrata e BNF	Informal	GMCL [CZARNEC KI, 1998]	Análise, projeto e implementação	Própria	Inserção	não

Tabela 10 Resumo comparativo das técnicas para o desenvolvimento de LED's

3.1.5 Ferramentas para o desenvolvimento de LEDs

O desenvolvimento de LEDs não é uma tarefa trivial, requerendo conhecimento tanto do domínio em questão como de desenvolvimento de linguagens. Para auxiliar no trabalho de desenvolvimento de LEDs existem ferramentas que podem auxiliar em diferentes fases, gerando elementos de suporte ao desenvolvimento de LEDs a partir de descrições de linguagem. Essas ferramentas podem criar desde analisadores de consistência e interpretadores até um ambiente integrado de desenvolvimento. Alguns desses sistemas dão suporte a uma metodologia específica de desenvolvimento de LEDs enquanto outras são independentes de metodologia. A entrada para esses sistemas é uma descrição de vários aspectos da LED a ser desenvolvida, em termos de meta-linguagens especializadas que, em si mesma, é uma LED para esses aspectos particulares. Esses aspectos incluem sintaxe, *prettyprinting*, checagem de consistência, execução, tradução, transformação e depuração.

As ferramentas existentes dão maior suporte à fase de implementação das LEDs. O suporte à decisão e análise não foi encontrado nas ferramentas analisadas e o suporte ao projeto é bem fraco. As sub-seções a seguir fazem um breve resumo de algumas dessas ferramentas.

3.1.5.1 JTS

O *Jakarta Tool Suite* (JTS) [BATORY, LOFASO, SMARAGDAKIS, 1998] é um conjunto de ferramentas do tipo pré-compiladoras para estender linguagens de programação populares como o Java com construções específicas de domínio, permitindo reduzir substancialmente os custos com o desenvolvimento de LEDs e geradores baseados em componentes chamados geradores GenVoca. Portanto o JTS usa projeto do tipo extensão de linguagem e implementação do tipo inserção (*embedding*).

JTS é composto de duas ferramentas: *Jak* e *Bali*. A linguagem *Jak* é um superconjunto extensível do Java que suporta meta-programação (características que permitem a um programa Java escrever outros programas Java). *Bali* é uma ferramenta para compor gramáticas. O próprio JTS é um gerador GenVoca. Linguagens e extensões de linguagens são encapsuladas como componentes reutilizáveis. Um componente JTS consiste de um arquivo de gramática Bali, que define a sintaxe da extensão de linguagem e de um conjunto de arquivos Jak, que definem a semântica da extensão como transformações de sintaxe. Diferentes combinações entre esses componentes originam diferentes variantes de linguagem e Bali e Jak trabalham em conjunto para converter automaticamente uma composição de componentes que definem uma variante de linguagem em um pré-processador para essa variante.

3.1.5.2 LaLa

LaLa (“Language Laboratory”) [PFAHLER, KASTENS, 1997] é um sistema de projeto de linguagem composto por uma interface de usuário, conhecimento específico do domínio e uma base de especificação. A base de conhecimento contém um conjunto de decisões e regras e questões relacionadas às mesmas, que certificam sua integridade e consistência. A base de especificação consiste de

fragmentos de especificação parametrizados para o sistema *Eli*, um ambiente de construção de compiladores que é usado como a máquina de implementação de linguagem do sistema. O projetista da linguagem interage com o sistema selecionando propriedades e elementos da linguagem e o sistema adiciona implicações, faz checagem de consistência e integridade, informando o usuário sobre o estado do processo e, após solicitação, produz um conjunto de especificações que serão a entrada para o Eli gerar o processador da linguagem que foi projetada.

Em resumo, o processo do sistema consiste em restringir o domínio do problema, produzir um conjunto de fragmentos especificações de linguagem e regras que controlam a composição dos mesmos e selecionar as propriedades que deseja incluir numa instância específica de linguagem cujo processador será gerado. O protótipo do sistema foi testado na geração de um pequeno sub-sistema da linguagem Pascal ("*MicroImperative*") e em linguagens de geração de relatórios ("*RepGen*").

3.1.5.3 Stratego

Stratego (2006) é uma linguagem para transformação de programas baseada no paradigma de estratégias de reescrita, que consiste em usar regras de reescrita (um formalismo natural para expressar transformações simples em programas) com estratégias específicas. Em Stratego, os programas são representados por meio de termos e a sintaxe abstrata de uma linguagem de programação é descrita por meio de uma assinatura. Os passos básicos de transformação são especificados por meio de regras de re-escrita condicionais. Uma regra reconhece um sub-termo a ser transformado por equiparação de padrão e substitui o mesmo por uma instância de padrão. A separação de estratégias de regras de transformação é oferecida para permitir um cuidadoso controle da aplicação dessas regras. O compilador Stratego transforma as especificações de entrada em código C. O sistema inclui uma coleção de módulos com regras reutilizáveis e estratégias (*Stratego Libra*) e está sendo aplicado em vários projetos, como o CobolX (ferramenta de melhoria de software para Cobol), no compilador

Tiger e uma ferramenta de geração de documentação para o SDL ("*Specification and Description Language*"),.

Stratego é também distribuído como parte do Stratego/XT, um pacote que combina a linguagem Stratego com um conjunto de ferramentas de transformação que suportam a implementação de transformações de programas. As áreas de aplicação do pacote incluem meta-programação, programação gerativa, compilação, implementação de LEDs, extensão de linguagem e geração de documentação.

3.1.5.4 Resumo comparativo das ferramentas

Conforme o resumo da Tabela 11, podemos constatar que nenhuma das ferramentas analisadas contempla a fase de análise; o projeto é tratado parcialmente e todas têm o foco maior na implementação. Dentre as ferramentas estudadas nenhuma dá suporte a uma metodologia de Engenharia de Domínio.

Ferramenta	Metodologia que suporta	Tecnologia base	Análise	Projeto	Implemen-tação
JTS	-	Genvoca / pré-compilação	-	Sim	Sim
LaLa	-	Knowledge base/ seleção	-	Sim	Sim
Stratego	-	Regras de transformação	-	Parcial	Sim

Tabela 11 Comparativo das ferramentas para desenvolvimento de LEDs

Os produtos gerados pelas ferramentas para o desenvolvimento de LEDs que dão suporte à fase de implementação são equivalentes aos geradores de aplicação, que serão abordados nas seções a seguir. Os geradores de aplicação servem como processadores para uma LED, permitindo gerar uma aplicação específica de uma família de sistemas a partir de um programa feito em uma LED. Portanto, algumas das ferramentas analisadas podem ser vistas como "geradores de geradores".

3.2 Geradores de aplicação

Os geradores de aplicação são as aplicações de software necessárias à efetivação das LEDs como ferramentas de reuso gerativo e cuja abordagem se torna

indispensável nesta pesquisa. Esta seção apresenta as características, arquitetura e técnicas para construção de geradores de aplicação, fazendo também uma descrição do relacionamento que estes têm com as LEDs.

3.2.1 Definição e características

Geradores de aplicação são ferramentas que geram automaticamente um componente ou aplicação em alguma linguagem (chamada linguagem alvo) a partir de uma especificação feita com uma Linguagem Específica de Domínio, que pode ser na forma de diálogo interativo por menus, gráfica ou textual [CLEAVELAND, 1988].

Em [LUKER, 1986] define-se um gerador como um item de software que produz um programa em alguma linguagem-alvo que é feito de modo a atender um conjunto específico de requisitos.

Outra definição de um gerador de programa é dada em [LIM, 1998]: “um gerador é um construtor automático de alto nível que esconde a interconexão manual entre os componentes usando uma linguagem orientada a problema, *template* ou filtro de opções ou ambiente de programação visual. Um gerador possibilita a especificação concisa da aplicação desejada (ou parte dela), e então gera código apropriado e / ou chamadas a procedimentos em alguma linguagem”. Um *template* ou “esqueleto” é um produto de software com parâmetros em aberto ou conexões e que pode ser usado na geração de um produto completo.

Podemos concluir então que um gerador favorece o reuso automático de artefatos de software (reutilização gerativa), figurando como um compilador para uma linguagem específica de domínio.

A utilização de geradores de aplicação pode proporcionar boas vantagens como:

- Redução do tempo de desenvolvimento e erros de programação: o usuário se concentrará na correção de possíveis erros de especificação;
- Os desenvolvedores não precisam ser especialistas em linguagens, mas especialistas no domínio;

- Facilidade para manutenção, uma vez que as alterações podem ser feitas diretamente nas especificações de entrada, sendo o código gerado automaticamente.

Os geradores de aplicação também apresentam limitações, justificadas pelas próprias características dos mesmos:

- Um gerador de aplicação tem seu uso restrito a uma pequena gama de aplicações, que depende da sua cobertura de domínio [CLEAVELAND, 1988];
- A sua construção é difícil e de alto custo: requer linguagens de especificação e interfaces cuidadosamente projetadas, bem como conhecimento avançado do domínio de aplicação.

3.2.2 Geradores de aplicação e LEDs

Nesta seção é analisada a relação que há entre geradores de aplicações e LEDs e como eles se complementam, dando uma visão do processo de reuso gerativo usando essas duas tecnologias.

As linguagens específicas de domínio constituem uma das formas de especificação para um gerador de aplicação. São linguagens com alto nível de abstração que possuem características específicas para um domínio, permitindo aos desenvolvedores programarem de forma mais conveniente. Além disso, essas linguagens são muito mais concisas que as linguagens de propósito geral.

As linguagens específicas de domínio permitem aos desenvolvedores especificar "o que" a aplicação deve fazer, deixando "o como" como tarefa para o gerador de aplicação.

A visão do mapeamento entre espaço do problema e espaço da solução (Figura 15a), também conhecida como modelo de domínio gerativo [CZARNECKI, EISENECKER, 1999], [CZARNECKI, 2004], um conceito chave no desenvolvimento gerativo de software, dá uma boa visão do relacionamento entre LEDs e geradores de aplicações. O espaço do problema é um conjunto de abstrações específicas de domínio que podem ser usadas para especificar um determinado membro de uma família de sistemas. O espaço da solução consiste de abstrações orientadas a implementação, as quais podem ser instanciadas para criar implementações das

especificações escritas usando as abstrações específicas do domínio do espaço do problema. O espaço do problema contém uma estrutura que permite a expressão de problemas de forma direta e intencional.

Nessa visão, a LED está no espaço do problema e o gerador, no mapeamento entre os dois espaços, tendo como entrada uma especificação e, como saída, a correspondente implementação. A Figura 15b mostra os conceitos presentes nesses espaços na Engenharia de Domínio Multiagente.

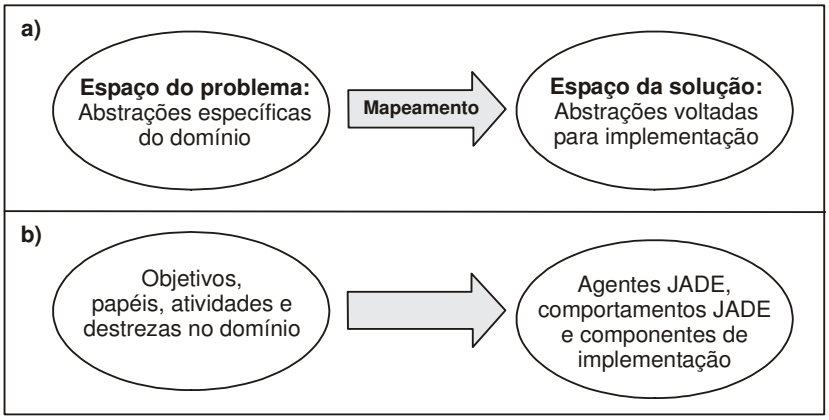

Figura 15 Mapeamento entre espaço do problema e espaço da solução

Podemos concluir que a LED é especificação de entrada para o gerador e que o gerador é o "processador" da LED, conforme ilustrado na Figura 16. A forma textual não é a única existente para a especificação do domínio para o gerador. Esta também pode ser feita com uma GUI (*Graphic User´s Interface*), ou interface gráfica de usuário; pode ser uma interface interativa por menu ou combinação das mesmas. Portanto podemos chamar todas essas formas de especificação de linguagens específicas de domínio. A Figura 16 ilustra o processo de interação entre a LED e o gerador [DELTA SOFTWARE, 2005]: o usuário (desenvolvedor) irá traduzir os conceitos do domínio em uma especificação (que pode ser numa das formas descritas anteriormente), que será submetida ao gerador. O gerador então irá automatizar o processo de implementação, traduzindo a especificação para um programa na linguagem-alvo, que será aplicado ao compilador específico da mesma, gerando finalmente a aplicação executável. A linha que liga a LED ao gerador na Figura 16 mostra que uma LED, para ter sua aplicabilidade completa, precisa de um gerador, mas existem LEDs que nem sempre visam execução, podendo ser usadas somente para especificação, como a BNF.

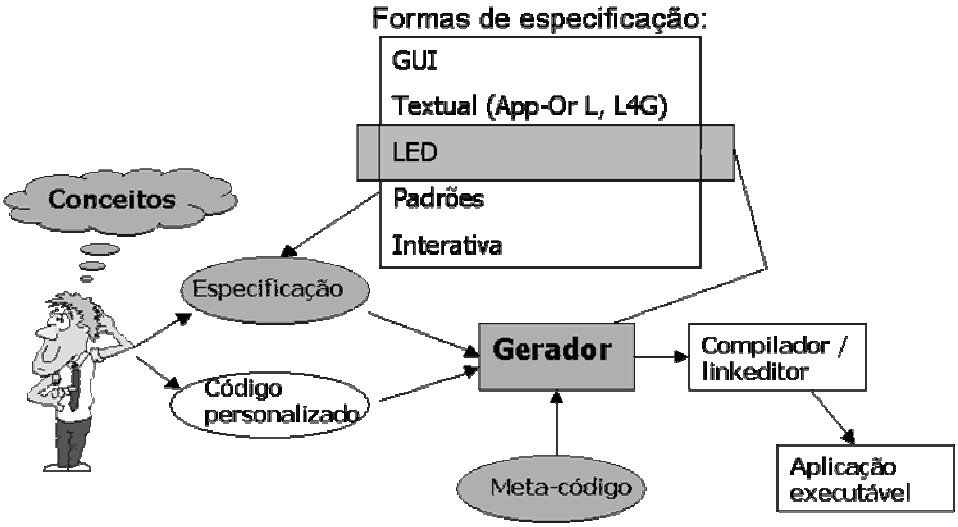

Figura 16 Gerador de aplicação e sua relação com uma LED (adaptado de [DELTA SOFTWARE, 2005])

3.2.3 Construção de Geradores de Aplicação

Uma vez que se tem bom conhecimento e experiência a respeito do desenvolvimento de software em determinado domínio, é desejável que se automatizem as tarefas de desenvolvimento de tais aplicações, reutilizando esse conhecimento e experiência na construção de geradores de aplicação.

Entretanto, para que sejam viáveis a construção e utilização de um gerador de aplicação, certos aspectos têm que ser observados, como: um perfeito entendimento do domínio do problema e que esse domínio esteja bem especificado; é necessário também que o domínio seja bem delimitado e "maduro" ou seja, não esteja sujeito a muitas alterações; faz-se necessário ainda conhecimento a respeito de linguagens de programação e tecnologia de compiladores.

Segundo Czarnecki (1998), existem duas técnicas principais usadas durante a geração: a composição e a transformação.

A composição ou modelo *composicional* consiste em agrupar um determinado número de componentes, que para maior efetividade devem ser componentes gerativos em vez de componentes concretos, ou seja, componentes na forma de uma descrição abstrata de uma instância concreta que são gerados de acordo com a descrição. Por exemplo, para construir uma estrela precisaríamos de dois componentes gerativos: um círculo que teria o raio como seu parâmetro, e o

braço, que teria os parâmetros o raio interno, raio externo e ângulo de posicionamento.

Uma *transformação* é uma modificação automatizada e semanticamente correta de um programa. As regras de reescrita ou regras de transformação constituem a categoria mais conhecida de transformação.

Cleaveland (1998) apresenta um processo para a construção de geradores de aplicação que tem sete passos:

1) *Reconhecendo domínios:* Consiste em reconhecer quando um domínio é propício para a construção de geradores de aplicação. A regra básica é reconhecer padrões tanto no nível do código (rotinas similares, código repetitivo, etc.) como em alto nível (programas similares, arquiteturas, projetos, etc.). Domínios propícios são aqueles com teoria bem entendida, que têm informação dispersa que pode ser consolidada e padrões reconhecíveis no nível do código.
2) *Definindo os limites do domínio:* Este passo irá definir a cobertura de domínio do gerador. Aqui devemos decidir quais aspectos do domínio serão incluídos no gerador. Uma cobertura grande torna o gerador mais abrangente, mas provavelmente menos eficiente ou mais difícil de usar. Uma cobertura pequena traz eficiência, porém para uma limitada gama de problemas.
3) *Definindo um modelo básico:* Consiste em prover um modelo matemático que irá tornar o gerador mais apto a ser compreensível, consistente e completo. Esse modelo torna-se a base para definir a semântica do domínio, como referência para as características, a informação de especificação e o projeto do produto. Como exemplos de modelos temos conjuntos, grafos dirigidos, sistemas de lógica formal e modelos computacionais como as máquinas de estado finito.
4) *Definindo as partes variante e invariante:* Aqui decide-se quais aspectos da geração poderão ou não ser parametrizadas pelo usuário. A parte variante é a que pode ser modificada pelo desenvolvedor. A parte invariante ou fixa é a que não pode ser mudada;.

5) *Definindo a especificação de entrada:* Consiste em definir o modo, dentre os já mencionados (textual, gráfico, interativo, etc.) pelo qual o usuário irá especificar cada instância de programa a ser gerado.
6) *Definindo produtos:* Neste passo decide-se que produto(s) o gerador irá produzir (exemplos: programas, documentação, dados de teste, etc.), que abordagem de geração de código será usada (exemplos: baseadas em código, baseadas em tabelas), linguagem-alvo (no caso de geração de código) e se haverá algum recurso de otimização para reduzir requisitos de tempo e espaço.
7) *Implementando o gerador:* Consiste em desenvolver um programa que traduz a especificação de entrada no(s) produto(s) desejado(s).

3.2.4 Arquitetura básica de um gerador de aplicações

Para construirmos um gerador, precisamos conhecer a arquitetura genérica dos mesmos. Geradores de aplicações têm grande semelhança com compiladores [AHO, RAVI, ULLMAN, 1995] que, descritos de forma simples, são programas que lêem um programa escrito numa linguagem (a linguagem-fonte) e traduzem-no para um programa equivalente numa outra linguagem (a linguagem-alvo), relatando também ao usuário a presença de possíveis erros no programa fonte. No caso do gerador de aplicações a linguagem-fonte é a Linguagem Específica de Domínio.

Portanto, a arquitetura básica de um gerador é similar a de um compilador (Figura 17), que é estruturada em três elementos principais:

- Módulo primário (*front-end*), que é responsável principalmente pela análise léxica, fazendo um mapeamento da forma de entrada original para uma representação interna mais conveniente, como grafos de fluxo e árvores sintáticas;
- O Mecanismo de tradução: é a principal parte de um gerador, que implementa transformações na representação intermediária (RI), produzindo um programa, mas ainda representado como grafos de fluxo ou árvores sintáticas;

- Módulo secundário (back-end): faz o mapeamento da RI do programa gerado para o texto do programa na linguagem-alvo;

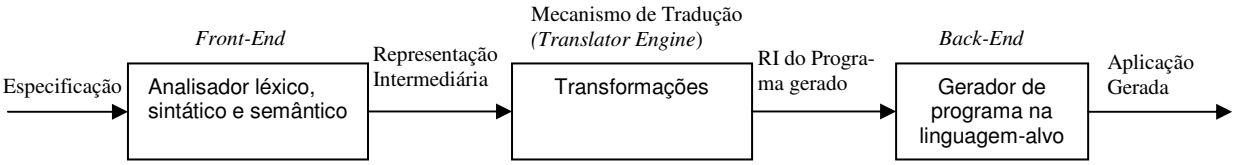

Figura 17 Arquitetura básica de um gerador

O que difere um gerador de aplicação de um compilador é que suas especificações de entrada são de mais alto nível, normalmente são abstrações de um domínio de aplicação limitado; e sua expansão de código (razão do tamanho na entrada para o da saída) é bem maior que no caso dos compiladores [KRUEGER, 1992]. Outra diferença notável é que os compiladores produzem diretamente instruções executáveis, enquanto os geradores, na maioria dos casos, produzem somente uma transformação em formato diferente (não diretamente executável).

3.2.5 Formas de geração de código

Os geradores de código se dividem em duas categorias de alto nível: geradores ativos e passivos [HERRINGTON, 2003].

Os geradores passivos constroem código que fica disponível para ser editado ou alterado pelos desenvolvedores, não mantendo nenhuma responsabilidade pelo código gerado, nem a curto nem a longo prazo. Os geradores passivos são usados apenas uma vez para criar algum artefato. Concluída a geração, o desenvolvedor assume o item gerado e melhora ou completa o mesmo. Um exemplo de geradores passivos são os "*wizards*" existentes em Ambientes Integrados de Desenvolvimento.

Os geradores ativos mantém responsabilidade pelo código gerado, permitindo diversas gerações da saída, a partir de mudanças na sua entrada durante a evolução do projeto, tornando-se portanto parte do processo de desenvolvimento. Eles tomam um modelo como entrada e transformam-no num artefato diferente. Se o modelo muda, o processo de geração deve ser repetido. Normalmente, não é permitido ao desenvolvedor alterar o código gerado, ou é utilizado algum sistema

eficiente para distinguir o código gerado do código escrito à mão. A seguir serão apresentadas algumas formas de geração ativa de código.

3.2.5.1 "Code Munging"

Uma das formas mais comuns de geradores de código é a técnica conhecida como "*code munging*" consiste em "moldar" a forma de entrada para obter a saída desejada. A O termo "*munging*" nessa nomenclatura é uma gíria para torcer e moldar. O processo dessa geração (Figura 18) é bem simples e consiste em tomar um código fonte como entrada, geralmente usando expressões regulares ou *parsing* de código, e construir os arquivos de saída, usando *templates* internos ou externos. Esse tipo de geração pode ser usada para criar documentação ou ler constantes ou protótipos de funções a partir de um arquivo. Um exemplos é o gerador de documentação de programas Java (JavaDoc), que lê os comentários nos códigos fonte Java e gera uma documentação HTML a partir dos comentários, usando alguns templates; outro exemplo é o XDoclet, que gera classes EJB e interfaces necessárias para apoiar o bean que contém os comentários.

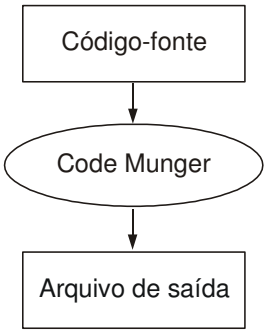

Figura 18 Processo de um gerador tipo "*code munging*"

3.2.5.2 "Inline code expander"

O "*Inline Code Expander*" ou expansor de código em linha, tem como entrada código fonte contendo marcações especiais que o expansor troca pelo código de produção para criar o código de saída (Figura 19). São geralmente usados para embutir comandos SQL em um código fonte, com marcas especiais no código SQL que serão reconhecidas pelo expansor que, ao ler o código, insere o código fonte que implementa a consulta ou comando SQL no ponto onde encontra as

marcações, a exemplo do Pro*C e SQLJ. Outros usos desse tipo de gerador incluem a inserção de seções de montagem de performance crítica e inserção de equações matemáticas que são implementadas pelo gerador. A principal vantagem é ser uma solução fácil para a simplificação do código-fonte e a desvantagem é a dificuldade de depurar o código de saída, que será bem maior que o código original.

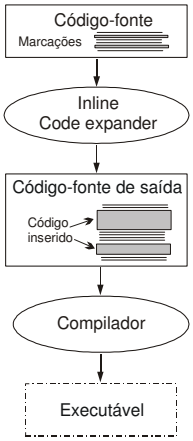

Figura 19 Processo de um gerador tipo "*inline code expander*"

3.2.5.3 "Mixed-code generation"

Um gerador de código misto ("*mixed-code generator*") lê um arquivo de código fonte de entrada, procurando por comentários específicos, preenchendo esses pontos com código, substituindo o arquivo original. A diferença do expansor de código em linha é que o gerador de código misto insere o código diretamente no código de entrada (Figura 20). Pode ser usado para implementar o mapeamento de controles da interface do usuário para variáveis de estrutura de dados, criar casos de teste a partir de dados de teste inseridos nos comentários e também para inserção de código de acesso a dados.

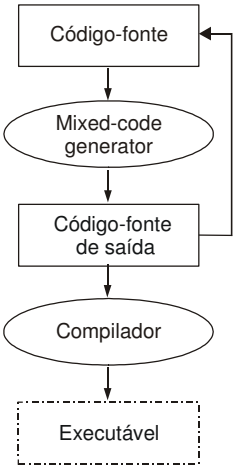

Figura 20 Processo de um gerador tipo "*mixed-code*"

3.2.5.4 "Partial-class generation"

Um gerador de classes parcial (Figura 21) lê um arquivo com uma definição abstrata e, baseado em *templates*, constrói um conjunto de bibliotecas de classes-base. Essas classes então são compiladas juntamente com as classes derivadas produzidas pelos engenheiros para completar o conjunto de produção de classes.

É um dos primeiros geradores a criar código a partir de uma especificação abstrata. Os modelos anteriores usavam código fonte em alguma linguagem (como C, Java, C++ ou SQL) como entrada. Em vez de inserir ou trocar fragmentos através da filtragem do código de entrada, este modelo toma uma descrição de código a ser criado e cria um conjunto completo de código implementado. Esse tipo de geração é um bom ponto de partida para construir um gerador que cria uma camada inteira de código.

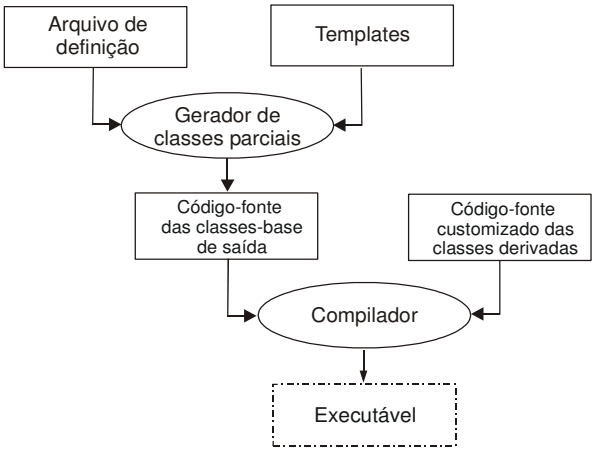

Figura 21 Processo de um gerador de classes parcial

3.2.5.5 Geração de camada

Na geração em camadas ("*tier generation*"), o gerador tem a função de gerar uma camada completa de uma aplicação multicamada. O processo desse tipo de geração é o mesmo existente na geração de classes parciais: o gerador lê um arquivo de especificação e usa templates para gerar classes de saída que implementam o que está definido no arquivo de entrada (Figura 22). A diferença é que neste caso o gerador constrói todo o código de uma camada, enquanto na geração parcial as classes-base precisam ser complementadas com classes derivadas para completar o código da camada.

Um exemplo desse tipo de geração é a geração dirigida a modelo (*model-driven generation*), na qual uma aplicação definida em UML e uma especificação de entrada em XML são usadas pelo gerador para construir uma ou mais camadas de um sistema. Outros exemplos de uso incluem a geração de camadas de *stored procedures* para acesso a banco de dados e a geração de camadas de importação, exportação ou conversão.

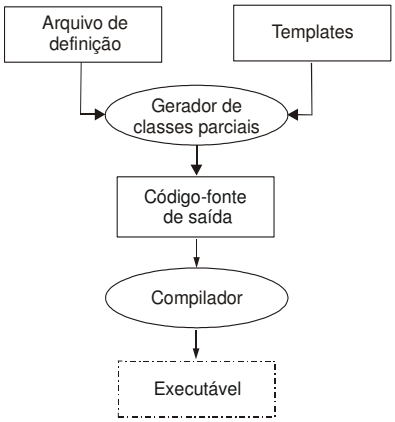

Figura 22 Processo de um gerador de camada

3.2.5.6 Linguagem de domínio completa

Uma linguagem de domínio completa é uma linguagem projetada para permitir que os engenheiros representem os conceitos do domínio de forma fácil. Portanto, este tipo de geração de código é a geração que usa uma Linguagem Específica de Domínio como entrada. As características, vantagens e processo de construção de LEDs foram discutidos no início deste capítulo. As idéias das formas de geração anteriores podem ser usadas de forma isolada ou conjunta na implementação de uma LED.

3.2.6 Abordagens gerativas

3.2.6.1 GenVoca

GenVoca [BATORY, O'MALLEY, 1992] é uma abordagem para a construção de geradores de software baseada na composição de camadas de abstração orientadas a objeto. Cada camada contém um número de classes e uma camada superior refina a camada inferior, adicionando novas classes ou métodos às classes existentes.

A abordagem é originária do trabalho de Don Batory com o Gênesis [BATORY et al., 1988], que é um gerador de sistemas gerenciadores de banco de dados, e do trabalho de O' Malley et al. no Avoca/x-kernel [HUTCHINSON et al., 1989], que é um gerador no domínio de protocolos de rede. A Tabela 12 relaciona alguns exemplos de sistemas e geradores baseados no GenVoca. O nome GenVoca originou-se da junção dos nomes Gênesis e Avoca.

Nome	**Descrição**
P1 e P2	Extensões da linguagem C para definição de geradores GenVoca
Predator	Gerador de contêineres de dados ("*data containers*")
P++	Extensões do C++ para definição de geradores GenVoca
Ficus	Sistema de arquivos distribuído
ADAGE	Geradores no domínio de sistemas de navegação aérea
Praise	Gerador de otimizadores de query
DiSTiL	Gerador de contêineres de dados no sistema IP ("*Intentional Programming*")

Tabela 12 Exemplos de sistemas baseados no GenVoca

No modelo GenVoca, cada tipo de dado abstrato ou característica é representada como uma camada separada e componentes concretos são definidos por expressões de tipo descrevendo composições de camadas. Um exemplo é a definição de um *bag* (coleção de objetos não ordenada que pode conter duplicidades) concorrente, redimensionável e gerenciado, que é definido pela expressão:

```
Bag[concurrent[size_of[unbounded[managed[heap]]]]].
```

A estrutura em camadas equivalente a essa expressão de tipo é mostrada na Figura 23, na qual cada camada (com exceção de heap) contribui com classes, atributos e métodos implementando as características correspondentes para a composição: *heap* implementa alocação de memória do total disponível, *managed* gerencia a memória alocada numa lista de disponibilidade, *unbounded* provê uma estrutura de dados redimensionável baseada em *managed*, *size_of* adiciona um atributo contador e método de leitura do tamanho, *concurrent* abriga os métodos de acesso num código de serialização semaforizado e *bag* implementa as operações de gerenciamento de elementos e *bag* necessárias.

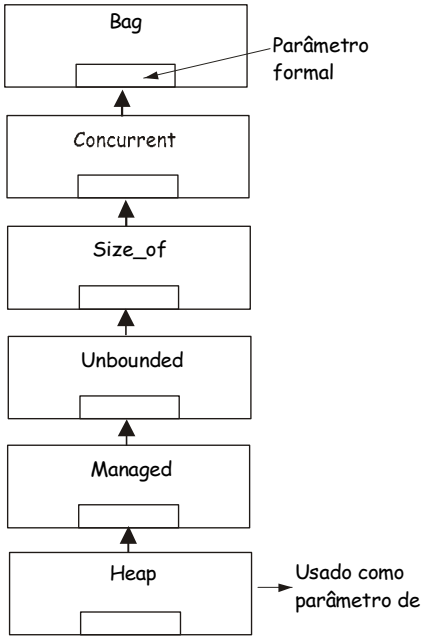

Figura 23 Exemplo de camadas GenVoca

Cada camada pode representar um componente parametrizado (exemplo: size_of recebe uma estrutura de dados como parâmetro) que pode ter mais de um parâmetro, fazendo com que a representação GenVoca tenha a forma de uma árvore. Portanto, cada camada exporta uma certa interface e espera seus parâmetros para exportar certas interfaces. Podemos ter camadas de interface definidas à parte, originando o conceito de *realm* (domínio), que é uma camada de interface padronizada, contendo coleção de classes e declarações.

3.3 Considerações finais

Este capítulo apresentou as principais soluções adotadas para promover o reuso gerativo na Engenharia de Domínio, dentre as quais destacamos as Linguagens Específicas de Domínio (LEDs)

As LEDs oferecem abstração de alto nível que facilita a criação da especificação das decisões de software necessárias para projetar um elemento da família de sistemas que serve como entrada para um gerador de aplicações ou outra ferramenta que promova seleção, adaptação e composição automática.

Foi feito também um estudo sobre ferramentas para o desenvolvimento de LEDs e geradores de aplicação, compilando informações que permitam um

comparativo com o trabalho que será proposto aqui e traçando diretrizes para a futura extensão do mesmo.

Este estudo serviu para termos um conhecimento das abordagens já existentes para o reuso gerativo na Engenharia de Domínio, um requisito para a propositura de uma abordagem similar para a Engenharia de Domínio Multiagente. A análise realizada também contribuiu para levantar parâmetros comparativos sobre as soluções que estão em uso, seus pontos fortes e deficiências.

4. UMA METODOLOGIA E UMA FERRAMENTA PARA O REUSO GERATIVO NA ENGENHARIA DE DOMÍNIO MULTIAGENTE

Os capítulos anteriores mostraram o estudo feito sobre os principais conceitos relacionados ao reuso gerativo de software. Esse estudo abordou a Engenharia de Domínio, que é o ponto de partida do reuso de software em famílias de sistemas; a modelagem de características como forma de abstração da variabilidade do domínio; a Engenharia de Domínio Multiagente e as iniciativas em técnicas e metodologias, incluindo as que serão utilizadas nesta pesquisa; as Linguagens Específicas de Domínio como uma forma de abstração do domínio na forma de uma especificação que será submetida a um gerador de aplicações; os geradores de aplicações e abordagens gerativas existentes.

Embasado nesse estudo, este capítulo apresenta as contribuições deste trabalho:

- A GENMADEM (*Generative Multi-Agent Domain Engineering Methodology)*, uma metodologia que integra e estende a metodologia MADEM [GIRARDI, LINDOSO, 2005] e a técnica TOD-LED [SERRA 2004], proporcionando uma abordagem gerativa e baseada em ontologias da Engenharia de Domínio Multiagente;
- A ONTOGENMADEM, uma ferramenta que suporta a aplicação da GENMADEM e que torna possível a especificação da LED e o projeto do gerador diretamente a partir do modelo de domínio criado nas fases de análise e projeto de domínio do processo da Engenharia de Domínio Multiagente. A ferramenta ONTOGENMADEM é composta por:
    - Uma ontologia que conceitualiza o conhecimento da GENMADEM e substitui as ontologias ONTOMADEM e ONTOLED, que dão suporte à MADEM e TOD-LED, respectivamente;
    - Um plugin para o editor de ontologias Protégé, que consolida o uso da ontologia como uma ferramenta para a análise, projeto e implementação de LEDs, complementando-a e automatizando algumas tarefas da GENMADEM.

4.1 GENMADEM: Uma metodologia para o reuso gerativo na Engenharia de Domínio Multiagente

O processo proposto pela GENMADEM estende o processo da Engenharia de Domínio Multiagente mostrado na Figura 11 (seção 2.6.1.2) para permitir a geração de famílias de sistemas multiagente. A Figura 24 mostra esse processo e como essas tarefas se integram com as atividades da Engenharia de Domínio Multiagente (análise, projeto e implementação do domínio), suas entradas / saídas e técnicas de suporte à execução das tarefas. O diagrama nessa figura é a representação do processo em forma de uma rede semântica, onde os conceitos entrada/saída (insumo ou produto do processo), fase e suporte (técnica que dá suporte a execução de uma tarefa) são representados pelas figuras de um círculo, quadrado arredondado e triângulo, respectivamente.

O processo consiste de seis fases: análise de domínio, projeto de domínio, implementação de domínio, extração e representação de padrões, especificação da LED e desenvolvimento do gerador.

A fase de *análise de domínio*, suportada pela técnica GRAMO e com base na análise de aplicações específicas existentes, no conhecimento do domínio e requisitos da família de sistemas, produz um modelo de domínio que representa os requisitos comuns e variáveis dos sistemas no domínio e as dependências entre eles. A fase de *projeto do domínio*, suportada pela técnica DDEMAS, produz um modelo do framework multiagente que é o modelo de solução reutilizável para os requisitos dos sistemas no domínio, usado na *implementação do domínio*, na qual os agentes de software que integram esse framework são implementados. A extração e representação de padrões, apoiada em diretrizes de extração e representação de padrões e com base nas experiências de desenvolvimento, produz os padrões de software e sistemas de padrões que podem ser reutilizados nas fases de análise, projeto e implementação de domínio [GIRARDI, LINDOSO, 2006] [GIRARDI, BALBY, OLIVEIRA, 2005]. Até este ponto, o reuso é composicional. As fases “especificação da LED” e “desenvolvimento do gerador” (em destaque na Figura 24) possibilitam o reuso gerativo. A tarefa especificação da LED, suportada por uma extensão da técnica TOD-LED, produz uma LED baseada num conjunto de modelos de domínio. A fase de desenvolvimento do gerador por fim produz um gerador de aplicações

para os sistemas no domínio, baseada na LED, no framework multiagente, e nos agentes de software.

Portanto, a GENMADEM tem como objetivo principal dar suporte à criação de LEDs no processo da Engenharia de Domínio Multiagente, mas não pretende oferecer suporte ao seu uso na Engenharia de Aplicações, não oferecendo ainda nenhuma infra-estrutura (na forma de interface para o usuário com gerador implementado ou similar) para que os artefatos criados pela mesma sejam utilizados.

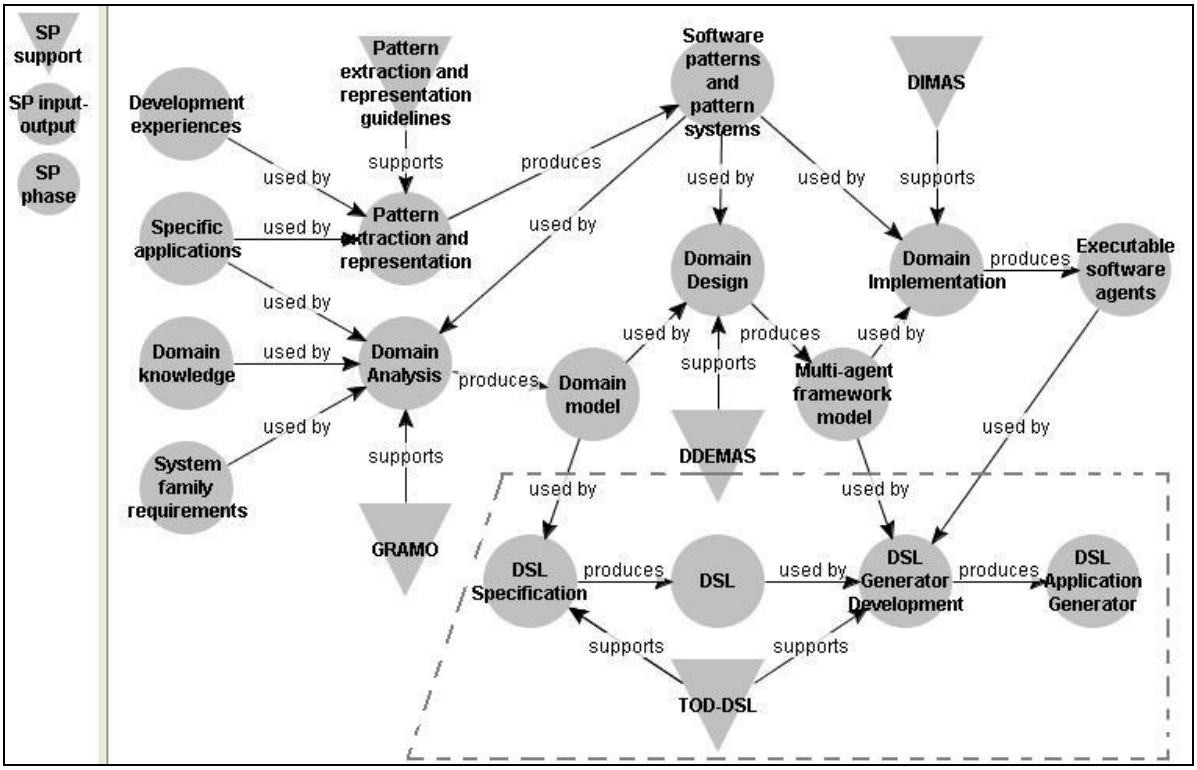

Figura 24 Processo gerativo da Engenharia de Domínio Multiagente da GENMADEM

A seção a seguir apresenta a extensão feita na técnica MADEM para aprimorar o modelo de domínio com maiores detalhes sobre a variabilidade e informações adicionais para o processo gerativo.

4.1.1 Extensão da metodologia MADEM para a abordagem gerativa

A metodologia MADEM, conforme apresentada na seção 2.6.1, já faz um tratamento da variabilidade do domínio classificando cada um dos seus conceitos como comuns (fixos) ou variáveis. Porém, esse tratamento não inclui informações adicionais sobre o domínio que são essenciais para a geração automática de

aplicações no domínio. A extensão proposta a esse tratamento da variabilidade permite a geração de famílias de sistemas, incluindo idéias da modelagem de características [CZARNECKI, 1998] para capturar a variabilidade com maiores detalhes e oferecer uma derivação automática do vocabulário da LED. Em nossa abordagem, as abstrações da modelagem de características foram mapeadas para as abstrações da MADEM da seguinte forma: os "conceitos" da modelagem de características equivalem aos objetivos e papéis existentes e as "características" dos conceitos equivalem às atividades e destrezas da MADEM, que em outras palavras são propriedades dos papéis (Figura 25).

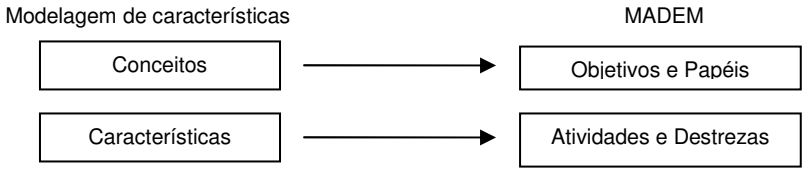

Figura 25 Mapeamento de abstrações da modelagem de características para a MADEM

Nesta nova concepção, que está resumida na Tabela 13, os conceitos no modelo de domínio podem ser *mandatórios* (estão presentes em todas as instâncias da família de sistemas), *opcionais* (integram um conjunto de conceitos onde um ou mais – mas nunca nenhum - podem estar presentes em uma instância da família de sistemas, como numa operação lógica "OR") e *alternativos* (integram um conjunto de conceitos onde somente um – mas nunca nenhum - pode estar presente numa instância da família, como numa operação lógica "XOR"). Para especificar conceitos alternativos ou opcionais o analista de domínio deve indicar na modelagem quais são os outros conceitos que integram o conjunto alternativo ou opcional. Os conceitos da modelagem onde essa variabilidade se aplica são objetivos específicos, papéis, atividades e destrezas, por serem os conceitos da metodologia que representam pontos de variação na modelagem. Portanto cada um deles pode ser mandatório, opcional ou alternativo (com outro conceito da mesma classe). Aos demais conceitos não é aplicada a noção de variabilidade.

Tipo de variabilidade	Exemplo
Mandatório	A responsabilidade "punctual information need specification" (Figura 26)
Opcional	Os objetivos relativos à recuperação e à filtragem (Figura 26)
Alternativo	Os objetivos relativos os três tipos de filtragem (Figura 26)

Tabela 13 Tipos de variabilidade considerados na GENMADEM

Podemos exemplificar essa variabilidade mostrando um produto do modelo de domínio ONTOINFO (vide APÊNDICE A), que é um modelo de domínio baseado em ontologias para a recuperação e filtragem de informação. A Figura 26 mostra um exemplo do modelo de objetivos da ONTOINFO [SERRA, GIRARDI, ALVES, 2004] estendida onde temos objetivos específicos opcionais e alternativos. Os objetivos “Satisfazer necessidades informação pontuais através de técnicas de recuperação” e “Satisfazer necessidades de informação de longo prazo através de técnicas de filtragem” compõem um grupo opcional: um dos dois ou ambos (mas não nenhum) podem estar presente em um sistema da família de sistemas. Os objetivos “Satisfazer necessidades de informação de longo prazo através de filtragem de conteúdo” e “Satisfazer necessidades de informação de longo prazo através de filtragem colaborativa” e “Satisfazer necessidades de informação de longo prazo através de filtragem híbrida” compõem um grupo alternativo: todos levam à realização do objetivo ao qual estão subordinados, mas somente um dos três deve estar presente em um sistema da família de sistemas.

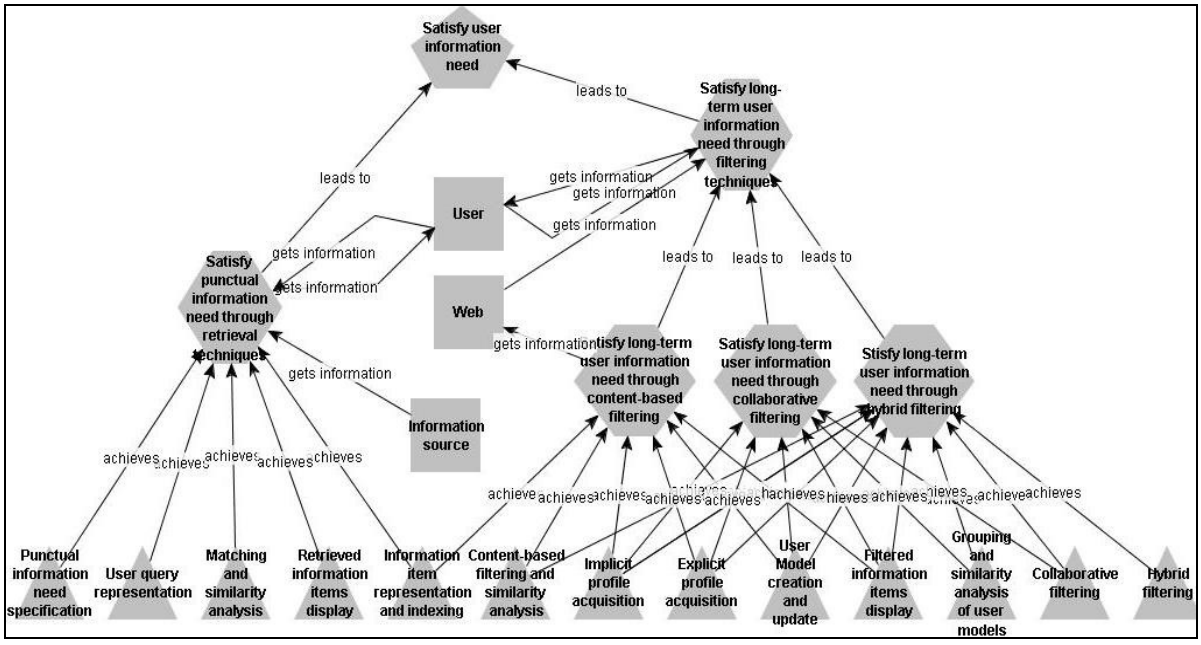

Figura 26 Modelo de objetivos da ONTOINFO

Em um comparativo com os diagramas de características podemos notar que a representação é similar e que a principal diferença é que o tipo de variabilidade (opcional ou alternativo) está definido na ontologia e não aparece graficamente. Podemos ver isso no diagrama de características equivalente ao

exemplo mostrado na Figura 26 (considerando somente o nível dos objetivos) que é apresentado na Figura 27.

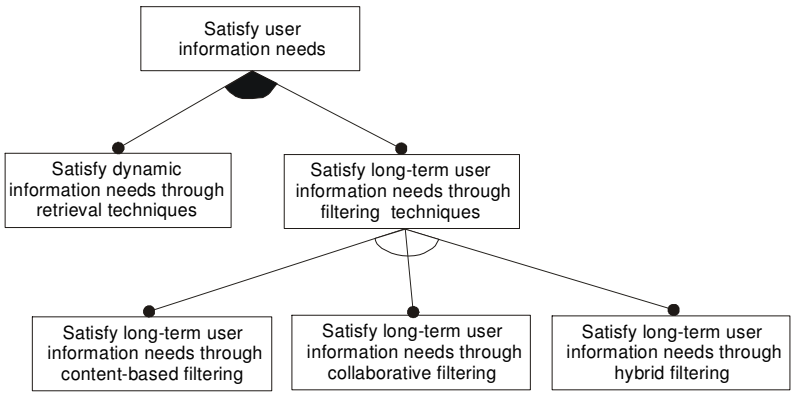

Figura 27 Diagrama de características do domínio de recuperação e filtragem de informação

4.1.2 Fases da GENMADEM

Esta seção apresenta as fases da metodologia e os produtos a serem obtidos com sua aplicação, que estão resumidos na Tabela 14. Essas fases assumem que o modelo de domínio (análise de domínio) e o modelo do framework multiagente (projeto de domínio) já tenham sido criados de acordo com o descrito na metodologia MADEM, com as técnicas GRAMO e DDEMAS respectivamente. Os principais produtos da GENMADEM são LEDs baseadas em ontologias e geradores de aplicações para as mesmas. Os conceitos referenciados nas fases e atividades da metodologia estão definidos na ontologia ONTOGENMADEM, que é descrita na seção 4.2.

Fases	Tarefas		Produtos
Especificação da LED	Especificação da Sintaxe	Definição do vocabulário	Vocabulário
		Definição da gramática	Gramática
	Especificação da semântica		Modelo de domínio criado pela GRAMO
	Especificação da pragmática		Documentação da LED, com áreas de aplicação, estudos de caso, etc.
Projeto do Gerador	Definição das regras de montagem		Regras de exigibilidade e exclusão mútua
	Definição dos mapeamentos		Mapeamentos de montagem

Tabela 14 Fases, tarefas e produtos da GENMADEM

4.1.2.1 Especificação da LED

Nesta fase define-se uma LED a ser usada na especificação de aplicações particulares da família de sistemas. Na GENMADEM, o produto principal desta fase é uma LED baseada em ontologias originada a partir do modelo de domínio baseado em ontologias criado na fase de análise do domínio. Essa LED é uma abstração dos conceitos variáveis de um modelo de domínio. Para entendermos melhor a concepção dessa LED a Tabela 15 faz um paralelo entre os conceitos de uma LED textual e os conceitos de uma LED baseada em ontologias (Tabela 21 e Figura 62 na seção 5.3.1), permitindo um melhor entendimento da abordagem proposta.

Conceitos	**LED Textual**	**LED baseada em ontologia**
Sintaxe	Palavras-chave derivadas dos conceitos-chave e características variáveis do domínio e gramática da LED, descritas em BNF ou outro formalismo	Instâncias de classes dos conceitos variáveis do modelo de domínio e definição das combinações válidas entre os conceitos que compõem o vocabulário, na forma de uma classe cujos *slots* referenciam instâncias de classes do vocabulário e definem a estrutura da linguagem.
Semântica	Atribuição de significados às sentenças, que serão ações do compilador ou gerador da LED que transformarão a especificação de entrada em uma outra forma de especificação ou programa executável	O domínio semântico é o próprio modelo de domínio criado, enquanto o mapeamento semântico dos conceitos selecionados na especificação para os agentes e software, está implícito no modelo do framework multiagente e nas classes e comportamentos concebidos na implementação desses agentes.
Pragmática	Instruções de uso da linguagem, na forma de manual que explica e exemplifica o uso do vocabulário e gramática	Instruções de uso da linguagem, na forma de manual que explica e exemplifica o uso do vocabulário e gramática

Tabela 15 Comparativo entre LED Textual e LED baseada em ontologia

Considerando esse comparativo, podemos ver que o vocabulário é representado numa classe que terá instâncias dos conceitos variáveis do domínio, conforme sua classificação na fase de análise de domínio. A gramática na ontologia será uma classe cujos slots referenciam e definem as combinações permissíveis entre os elementos do vocabulário em uma estrutura padrão. Essa abordagem permite especificar uma LED e escolher caminhos alternativos para a representação da mesma na fase de projeto e implementação (textual, visual, seletor baseado em características ou similar). Tendo o modelo de domínio e a definição da LED numa

mesma ontologia, temos a vantagem de poder referenciar diretamente os conceitos desse modelo de domínio na definição da LED. A semântica já está definida no próprio modelo de domínio e no modelo do framework multiagente. As atividades da fase de especificação da LED serão descritas a seguir.

4.1.2.1.1 Especificação da sintaxe

A especificação da sintaxe de uma linguagem envolve a definição do seu conjunto de termos válidos (vocabulário) e estrutura de escrita (gramática). A sintaxe deve contemplar, no mínimo, a parte variável do domínio, para permitir que abstrações de alto nível de elementos de uma família de aplicações de software sejam feitas na criação de aplicações específicas dessa família de sistemas na Engenharia de Aplicações.

- *Definição do vocabulário:*

Esta tarefa tem como insumo o modelo de domínio produzido na fase de análise de domínio e produz a lista de termos selecionados para o vocabulário da LED baseada nos conceitos que representam pontos de variação desse modelo de domínio. O vocabulário da LED é composto pelos conceitos variáveis do modelo de domínio e pelos conceitos comuns que tenham características variáveis (por exemplo, um papel mandatório que tenha destrezas alternativas). Na GENMADEM, a definição do vocabulário é realizada através da pesquisa dos conceitos que são variáveis no modelo de domínio, conceitos estes já especificados na fase de análise de domínio e que serão referenciados na definição do vocabulário da LED. Esses conceitos são: objetivos específicos, papéis, destrezas (*skills*) e atividades, opcionais ou alternativas, e também papéis mandatórios que tenham atividades ou destrezas variáveis. O objetivo específico compõe o vocabulário porque podemos ter um sistema na família em cuja implementação desejamos incluir apenas um (ou alguns) dos objetivos específicos.

O algoritmo da Figura 28 é usado pela GENMADEM para gerar o vocabulário a partir dos conceitos do modelo de domínio. Nesse algoritmo entende-se “variável” como qualquer conceito que seja alternativo ou opcional e assume-se que cada responsabilidade tem sempre um papel associado. As classes da ONTOGENMDEM referenciadas pelo algoritmo serão explicadas na seção 4.2.

Para exemplificar, vamos retornar ao exemplo citado para a variabilidade na Figura 26 e Figura 27. Podemos selecionar no mesmo dois objetivos opcionais (a recuperação e a filtragem de informação) e três alternativos (a filtragem baseada em conteúdo, colaborativa e híbrida) para o vocabulário, conforme podemos notar na Tabela 21 (seção 5.3.1).

- *Definição da gramática:*

A gramática de uma linguagem define suas regras de escrita. A técnica TOD-LED anteriormente definia a gramática na sua ontologia ONTOLED [SERRA, 2004] com uma classe gramática que tinha apenas um slot *nome* e um slot *especificação*, do tipo texto, no qual a gramática da LED era definida textualmente usando o formalismo BNF. A redefinição proposta pela GENMADEM permite a especificação da gramática na própria ontologia, sem a necessidade de especificação textual. A gramática da LED, assim como numa LED textual, será a definição das combinações válidas entre os elementos do vocabulário da LED.

```
Para cada instância de objetivo específico que faça parte do domínio
   Se variabilidade = opcional
      Instanciar a classe "Optional Goal"
   Senão Se variabilidade = alternativo
      Instanciar a classe "Alternative Goal"
   FimSe
FimPara

Para cada instância de responsabilidade que faça parte do domínio
   Identificar papel associado à responsabilidade
   Se variabilidade = opcional
      Instanciar a classe "Optional Role"
   Senão Se variabilidade = alternativo
      Instanciar a classe "Alternative Role"
   FimSe
FimPara

Para cada instância de atividade que faça parte do domínio
   Se variabilidade = opcional
      Instanciar a classe "Optional Activity"
   Senão Se variabilidade = alternativa
      Instanciar a classe "Alternative Activity"
   FimSe
FimPara

Para cada instância de destreza que faça parte do domínio
   Se variabilidade = opcional
      Instanciar a classe "Optional Skill"
   Senão Se variabilidade = alternativa
      Instanciar a classe "Alternative Skill"
   FimSe
FimPara

Para cada instância de responsabilidade que faça parte do domínio
   Se variabilidade = mandatória
      Identificar papel associado à responsabilidade
      Se papel tem destreza variável OU responsabilidade tem atividade variável
         Instanciar a classe "Mandatory Role with variable property"
   FimSe
FimPara
```

Figura 28 Algoritmo para obtenção do vocabulário a partir do modelo do domínio

Portanto, a definição da gramática na GENMADEM gera como produto uma sintaxe abstrata que representa os relacionamentos válidos entre os elementos do vocabulário, de acordo com sua hierarquia no modelo de domínio e com sua variabilidade. Essa sintaxe abstrata é estruturada como uma árvore onde os "nós" são os termos do vocabulário derivados de conceitos que têm sub-conceitos que parametrizam os mesmos, que são os objetivos e papéis; e as "folhas" são os sub-conceitos, que são as atividades e destrezas. O resultado final determina que conceito é parametrizado por quais outros conceitos. Um objetivo é parametrizado pelos objetivos ou papéis variáveis relacionados ao mesmo e um papel é parametrizado pelas destrezas ou atividades variáveis relacionadas ao mesmo. Estando definida a sintaxe dessa forma abstrata, fica em aberto a escolha de que representação usar na sintaxe concreta (se textual, gráfica ou interface de seleção, por exemplo).

Como exemplo, na Figura 26 e Figura 27, onde já selecionamos dois objetivos opcionais e três alternativos para o vocabulário, temos o objetivo "Filtragem" parametrizado pelos objetivos alternativos "Filtragem baseada em conteúdo", "Filtragem colaborativa" e "Filtragem híbrida", representando uma opção em aberto sobre qual será o tipo de filtragem do elemento da família de sistemas. A Figura 29 mostra esse fragmento da sintaxe como é representado na classe gramática da ontologia. Na Figura 62 do estudo de caso (seção 5.3.1), há uma representação completa de gramática.

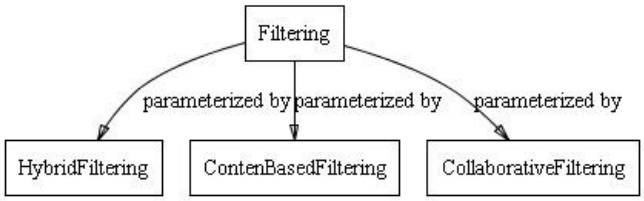

Figura 29 Exemplo de gramática obtida com a GENMADEM

4.1.2.1.2 Especificação da semântica

A semântica de uma linguagem corresponde ao significado que é dado aos termos que compõem seu vocabulário. Isso é definido através de um domínio semântico e de um mapeamento semântico [SCHMIDT, 1986]. Um domínio semântico é um conjunto de entidades do domínio e os relacionamentos entre eles.

Um mapeamento semântico associa os termos da linguagem às entidades do domínio semântico. Em LEDs definidas com a GENMADEM, o domínio semântico corresponde ao modelo de domínio obtido como produto na fase de análise de domínio. O mapeamento semântico é estabelecido pelos relacionamentos semânticos representados na ontologia entre os termos do vocabulário da LED e entidades do modelo de domínio.

Com isso, temos em uma só ontologia os conceitos que representam os produtos da análise e projeto de domínio e a própria definição da LED.

4.1.2.1.3 Especificação da pragmática

A pragmática de uma linguagem documenta todos os aspectos de uso da mesma. Nesta fase, uma descrição de como usar apropriadamente a LED desenvolvida é feita, tendo pelo menos as seguintes características:

- Descrição geral: descrição geral da LED e de suas aplicações;
- Descrição do domínio: descrição do domínio do problema para o qual a LED foi desenvolvida e descrição da cobertura de domínio do gerador de aplicações;
- Descrição do vocabulário: descrição do vocabulário da LED, com detalhes dos objetivos, papéis e atividades e destrezas;
- Descrição da sintaxe: a sintaxe da LED deve ser documentada, indicando como instanciar os conceitos na classe “DSL program” para iniciar e finalizar um programa, declarando os objetivos, papéis e propriedades de papéis;
- Gerador de aplicações: descrição de como usar o gerador de aplicações;
- Exemplos: uma lista de exemplos deve ser fornecida para dar aos usuários uma visão concreta do uso da LED e do gerador de aplicações;

Muitas das descrições acima podem ser obtidas a partir das descrições dos conceitos na ontologia. A pragmática pode incluir também um formulário para documentar novos requisitos ainda não cobertos pelo vocabulário da LED. Os

analistas do domínio podem usar esse formulário para uma futura revisão do modelo de domínio.

4.1.2.2 Projeto do Gerador

Conforme o que já foi descrito na seção 3.2.2, o gerador deve fazer o mapeamento automático de uma especificação de um programa na LED para a implementação de uma aplicação específica da família de sistemas, validando a sua entrada e gerando um código fonte na linguagem-alvo.

Na GENMADEM, os agentes de software são produtos da fase de projeto do domínio (técnica DDEMAS) e o gerador deverá então conhecer esses agentes e informações e sua implementação para fazer o reuso automático dos mesmos, apresentando a realização da especificação de alto nível da LED (Figura 30).

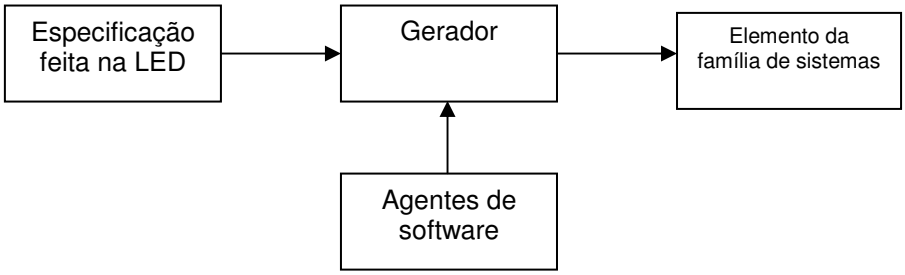

Figura 30 Papel do gerador na GENMADEM

Até a fase anterior da metodologia temos definida uma forma de abstração de alto nível para ser utilizada na especificação de elementos de uma família de sistemas multiagente. A fase de projeto do gerador visa definir o conhecimento necessário para que o gerador, recebendo como entrada uma especificação feita usando a LED, faça a validação e o mapeamento da mesma para os agentes de software implementados. Esse conhecimento é chamado de conhecimento de montagem e é definido com os produtos desta fase da metodologia.

Esta fase tem como entrada as definições da LED e o modelo do framework multiagente, e tem duas atividades: a definição das regras de montagem e a definição dos mapeamentos. Portanto, o pré-requisito para iniciar-se esta fase da metodologia é que as fases de análise, projeto de domínio e especificação da LED

estejam concluídas, e também que os agentes reutilizáveis, que são os principais produtos da engenharia de domínio, na sua fase de implementação do domínio, já estejam implementados. As atividades desta fase serão descritas nas seções que seguem.

4.1.2.2.1 Definição das regras de montagem

Fazendo um comparativo da abordagem gerativa multiagente com a linha de produção de uma fábrica, os agentes de software seriam os insumos pré-fabricados e o gerador seria a linha de montagem automatizada. Para que a montagem automática aconteça, os detalhes de configuração dos insumos devem ser conhecidos e bem detalhados. Portanto, este trabalho é pré-requisito para construção do gerador.

Nesta tarefa define-se esse conhecimento na forma de *regras de exigibilidade e exclusão mútua*. Essas regras irão permitir identificar combinações válidas e inválidas entre os conceitos selecionados num programa (especificação) criado com o uso as abstrações fornecidas pela LED. Os geradores em geral também incluem outras regras, chamadas regras de dependência padrão, que fornecem o conhecimento necessário para o gerador deduzir parâmetros não especificados pelo usuário durante o processo de configuração, a partir de outros existentes. A exigibilidade define quando a presença de um conceito exige que outro conceito também faça parte da montagem e a exclusão mútua define quando a presença de um conceito impede a inclusão de outro conceito na montagem. Essas regras serão usadas pelo gerador para a validação e montagem da aplicação. Para exemplificar, vamos retomar os exemplos dados para o vocabulário e gramática, onde identificamos três tipos de filtragem como conceitos alternativos (Figura 29). Para que ocorra a filtragem híbrida, precisamos dos papéis e destrezas dos outros dois tipos de filtragem, pois o papel da filtragem híbrida interage com os outros dois para atingir seu objetivo. Temos então duas regras de exigibilidade (Tabela 16).

Conceito existente	**Conceito exigível**
Filtragem Híbrida (papel alternativo)	Filtragem baseada em conteúdo (papel alternativo)
Filtragem Híbrida (papel alternativo)	Filtragem colaborativa (papel alternativo)

Tabela 16 Exemplo de regra de montagem

4.1.2.2.2 Definição dos mapeamentos

Os mapeamentos consolidam o conhecimento de como o gerador fará o *binding* (ligação) das abstrações oferecidas pela LED para as abstrações voltadas à implementação. Em outras palavras eles são os elementos essenciais para a implementação da LED (geração de código da aplicação a partir da especificação feita na LED) e devem contemplar a parte variável e a comum, pois o projeto do gerador deve incluir os mapeamentos de implementação tanto dos conceitos que estarão presentes em todos os elementos da família como dos conceitos que dependem de decisões de produto para fazer parte de um elemento da família. Na GENMADEM esses mapeamentos estão divididos em duas classes de mapeamentos: os *elementos JADE* e os *agentes de implementação*. JADE, ou "*Java Agent Development Framework*" [JADE, 2006] é a plataforma de implementação de sistemas multiagente escolhida para o projeto de domínio da Engenharia de Domínio Multiagente.

Esta fase usa o modelo de domínio e o modelo do framework como insumos na obtenção desses mapeamentos, cujos principais conceitos de modelagem são papéis (nos modelos de papéis), agentes (nos modelos da sociedade multiagente), responsabilidades e destrezas. Os papéis do modelo de domínio são substituídos pelos agentes no modelos do framework, sendo que um agente pode assumir mais de uma responsabilidade e o relacionamento entre os dois modelos é identificado pela responsabilidade. As classes de mapeamentos acrescentam detalhes de implementação que serão usados pelo gerador.

Os elementos JADE são mapeamentos dos conceitos do modelo do framework que são derivados de conceitos referenciados no vocabulário (agentes, responsabilidades e destrezas) para conceitos de implementação (agentes JADE e comportamentos JADE) (Figura 31). O mapeamento é realizado um para um de todos agentes do modelo do framework para agentes JADE, e também um para um de cada responsabilidade e destreza assumida por esses agentes para comportamentos JADE. Esses mapeamentos devem considerar todos os conceitos, tanto os comuns como variáveis. Exemplos de mapeamentos JADE podem ser encontrados na Tabela 22 e Tabela 23 da seção 5.4.2.

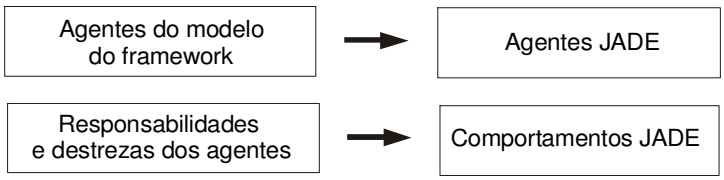

Figura 31 Concepção dos mapeamentos e elementos JADE

A outra classe de mapeamentos, a dos agentes de implementação, agrupa os agentes JADE obtidos no passo anterior de acordo com a variabilidade dos conceitos que definem seus comportamentos e associa os mesmos com seus respectivos comportamentos JADE. Os agentes são classificados, de acordo com a variabilidade, em três grupos: *agentes mandatórios com comportamento mandatório*, *agentes mandatórios com comportamento variável* e *agentes variáveis*.

Para obter esses mapeamentos, deve-se recorrer à variabilidade definida para as responsabilidades e destrezas no modelo de domínio. Os agentes que tiverem que tiverem pelo menos uma responsabilidade mandatória, e tiverem somente destrezas mandatórias, são incluídos no primeiro grupo (mandatórios com comportamento mandatório); os agentes tiverem pelo menos uma responsabilidade mandatórias mas tiverem pelo menos uma destreza variável são incluídos no segundo grupo (mandatórios com comportamento variáveis) e os agentes que tiverem somente responsabilidades variáveis são incluídos no terceiro grupo (agentes variáveis). A Tabela 17 resume essas regras para classificação dos agentes de implementação de acordo com a variabilidade dos seus comportamentos (responsabilidades e destrezas). Exemplos de agentes de implementação podem ser encontrados na Tabela 24, Tabela 25 e Tabela 26 da seção 5.4.2.

Todos esses mapeamentos são representados na ontologia, onde constam os relacionamentos necessários para a definição dos mesmos.

Responsabilidade do agente	**Destrezas do agente**	**Classificação do agente**
Pelo menos uma mandatória	Todas mandatórias	Mandatório com comportamento mandatório
Pelo menos uma mandatória	Pelo menos uma variável	Mandatório com comportamento variável
Todas variáveis	Mandatórias ou variáveis	variável

Tabela 17 Resumo das regras para classificação dos agentes de implementação

As atividades variáveis, que constituem uma divisão de uma responsabilidade, definindo caminhos alternativos ou opcionais para a realização da mesma, não foram abordadas na definição dos mapeamentos. Essa definição requer uma análise mais detalhada do impacto das atividades variáveis na implementação de aplicações da família de sistemas multiagente que o estudo de caso tratado neste trabalho não permitiu avaliar, constituindo, portanto, um item aberto para trabalhos futuros.

4.2 ONTOGENMADEM: uma ferramenta para o reuso gerativo na Engenharia de Domínio Multiagente

A ferramenta ONTOGENMADEM é composta por uma ontologia e um plugin construído para o editor de ontologias Protégé, cuja integração e relacionamento com as fases da metodologia serão descritos na seção a seguir.

4.2.1 Arquitetura da ferramenta ONTOGENMADEM

A ferramenta ONTOGENMADEM é composta por uma ontologia, que serve como ferramenta de análise e repositório de produtos, e por um plugin para o Protégé, que automatiza algumas tarefas da GENMADEM. A Figura 32 mostra esses elementos, como eles se integram e como os analistas e usuários interagem com a solução.

A ontologia (ONTOGENMADEM) e a ferramenta implementada como um plugin para o Protégé, integram o ambiente de desenvolvimento de sistemas baseado no conhecimento Protégé em funcionamento com o projeto ONTOGENMADEM aberto e o *plugin* habilitado. Os analistas / projetistas do domínio, através da interface padrão do Protégé, constroem um modelo de domínio instanciando as classes da ontologia. Após isso, o plugin pode ser usado para gerar o vocabulário e o gerador, como resumido a seguir: o módulo de geração da LED obtém os conceitos variáveis no modelo de domínio e instancia o vocabulário. O módulo de projeto do gerador identifica os mapeamentos. Os engenheiros de domínio complementam o vocabulário e a gramática conforme necessário, definem as regras de montagem e validam os produtos gerados. Na Engenharia de Aplicações, a LED criada guiará a especificação de programas e o projeto do

gerador guia a configuração automática a ser feita por um gerador de aplicações que gera uma aplicação específica da família de sistemas.

O plugin pode ser estendido para agregar uma interface para a Engenharia e Aplicações. A ontologia e o plugin serão descritos nas seções a seguir.

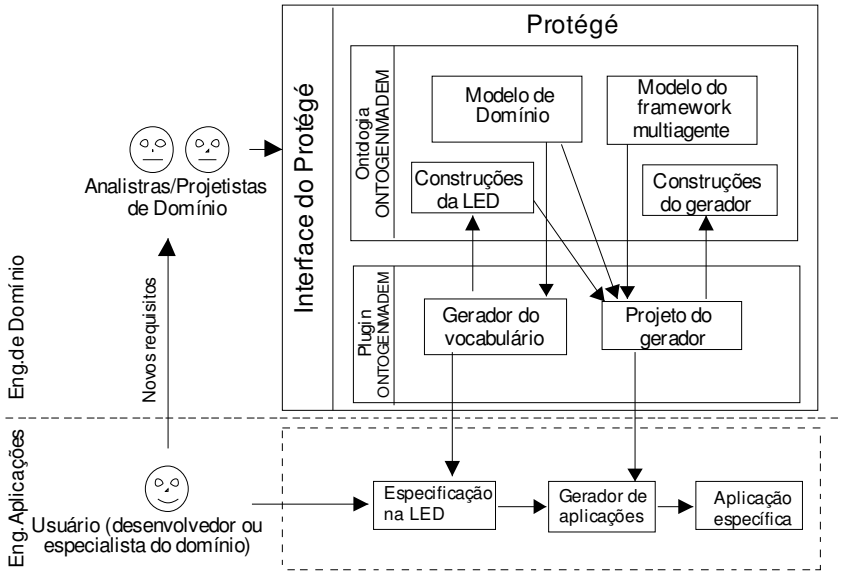

Figura 32 Arquitetura da ferramenta ONTOGENMADEM

4.2.2 A ontologia ONTOGENMADEM

Para dar suporte à aplicação da GENMADEM, foi criada a ONTOGENMADEM, que funciona como uma ferramenta para a modelagem e como repositório dos produtos criados pela mesma, representados na forma de conceitos semanticamente relacionados. Nessa ontologia, a aplicação da metodologia e obtenção dos produtos da mesma é feita através da instanciação das classes apropriadas da mesma.

A ONTOGENMADEM contém os conceitos e tarefas oriundas da MADEM juntamente com a conceitualização da metodologia GENMADEM (Figura 33), porém descreveremos somente esta última com maiores detalhes, fornecendo na seção a seguir apenas uma sucinta descrição dos conceitos oriundos das classes da MADEM.

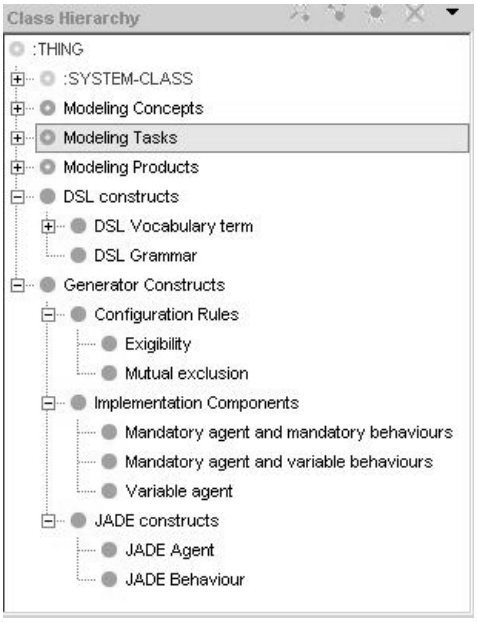

Figura 33 Hierarquia de classes da ONTOGENMADEM

4.2.2.1 Conceitos básicos

Os conceitos básicos referenciados na ontologia são os que estão em itálico junto às descrições a seguir. Os *conceitos* identificam as entidades do domínio e os relacionamentos existentes entre elas, refletindo o conhecimento dos especialistas e permitindo particularizar um problema para certa área. Os *objetivos* são o alvo que uma organização busca atingir, atribuindo *responsabilidades* aos indivíduos que a integram, os quais desempenham *papéis* podendo executar *atividades*. Para isso, usam e produzem *conhecimentos*, satisfazem *pré* e *pós-condições*e e exigem *destrezas*. Apesar de autônomos, os papéis, que são *entidades internas*, estabelecem interações entre si e também com *entidades externas* para executar atividades mais complexas, além de suas capacidades.

Um papel normalmente envolve uma responsabilidade, que pode ou não ser dividida em uma ou mais atividades. As destrezas são técnicas ou recursos necessários para a realização de uma responsabilidade. Uma entidade externa pode ser qualquer elemento externo ao sistema, como um cliente, usuário, outro sistema ou organização.

Por outro lado, os agentes são as entidades básicas do sistema, que assumem *responsabilidades*, executam *atividades*, usam e produzem *conhecimentos*, satisfazem *pré* e *pós-condições* e exigem *destrezas*. Eles se reúnem em *sociedades multiagente* e, apesar de serem autônomos, estabelecem *interações*

entre si para executar atividades mais complexas. Os *mecanismos de cooperação e coordenação* determinam a forma como os agentes se auxiliam e se organizam para atingir o fim da sociedade. O conhecimento da sociedade multiagente permite que os agentes se compreendam e se comuniquem. Tudo isso integra a *arquitetura geral do sistema*. Detalhando cada agente, há o conhecimento e as atividades, que informam o seu tipo – se reativo ou deliberativo – e seus aspectos estruturais, e os estados, que dizem o que ele faz, traduzindo aspectos comportamentais.

4.2.2.2 Tratamento da variabilidade

O uso de ontologias permite representar as mesmas variabilidades da modelagem de características, com a vantagem de termos a variabilidade armazenada, junto com outros detalhes do modelo de domínio, em uma base de conhecimento que permite que consultas e inferências sejam feitas. Esta seção mostra como a extensão da variabilidade explicada na seção 4.1.1 foi inserida na ontologia.

Os conceitos aos quais se aplica a variabilidade têm o slot "tipo de variabilidade", que pode assumir os valores possíveis para a variabilidade conforme descrito anteriormente, e o slot "Alternativo ou opcional" para especificar com qual(is) outro(s) objetivo(s) específico(s) este é alternativo ou opcional (conjunto de alternativos ou opcionais). A Figura 34 mostra o slot "tipo de variabilidade" na ontologia ONTOGENMADEM na classe "Objetivo Específico", onde podemos ver os valores que o slot "*variability type*", do tipo *symbol*, pode assumir.

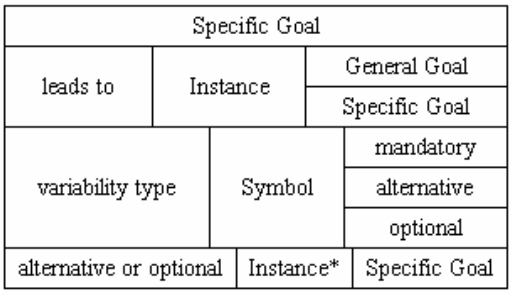

Figura 34 Concepção da variabilidade na ontologia ONTOGENMADEM

A Figura 35 detalha a variabilidade dos objetivos alternativos "Satisfazer necessidades de informação dinâmicas através de filtragem de conteúdo" e

“Satisfazer necessidades de informação dinâmicas através de filtragem colaborativa”, apresentados na seção 4.1.1, mostrando o conteúdo dos slots na ontologia.

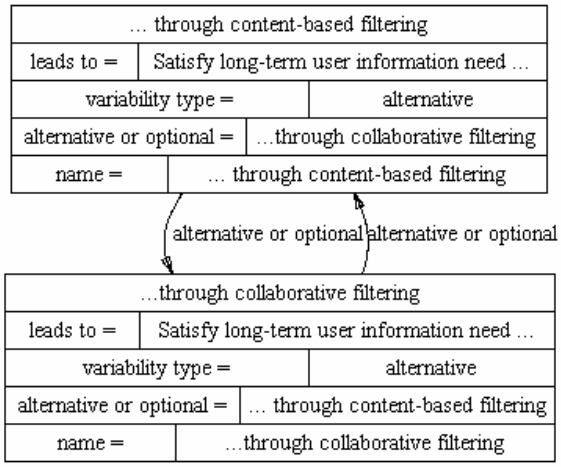

Figura 35 Detalhamento da variabilidade na ontologia

4.2.2.3 Classe “Construções da LED”

Esta classe representa os elementos que compõem a especificação da LED. Ela contém duas sub-classes: uma para representar o vocabulário e outra para representar a gramática.

4.2.2.3.1 Sub-Classe “termo do vocabulário da LED”

A Figura 36 mostra a hierarquia de classes criadas na ONTOGENMADEM para a definição do vocabulário da LED cuja superclasse é a “*DSL Vocabulary term*” e cada subclasse da mesma representa uma classe de conceitos variáveis permissíveis para a LED.

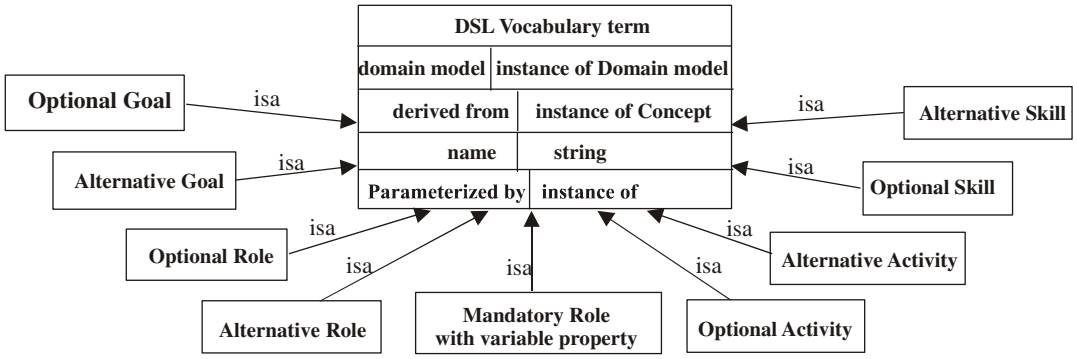

Figura 36 Hierarquia de classes da ONTOGENMADEM para definição do vocabulário da LED

A classe principal (*DSL Vocabulary*) contém os seguintes slots:

- *derived from* (derivado de): irá referenciar, nas subclasses, os conceitos do domínio que serão usados no vocabulário;
- *domain model* (modelo de domínio)*:* indica de qual modelo de domínio o conceito é oriundo, uma vez que podemos ter mais de um modelo de domínio na mesma ontologia;
- *name* (nome)*:* atribui um nome mais adequado ao termo do vocabulário, independente do nome do conceito.
- *parameterized by* (parametrizado por): indica quais são os conceitos que parametrizam o conceito atual (sub-conceitos), conforme explicado no item "definição da gramática", na seção 4.1.2.1.1. Permite instâncias de objetivos, papéis e destrezas, tanto alternativos como opcionais.

Esses slots são herdados pelas subclasses, que redefinirão o slot "*derived from*" para referenciar os conceitos apropriados e poderão acrescentar novos slots. Uma descrição das subclasses torna-se necessária para um melhor entendimento e uso da ontologia na especificação do vocabulário:

- *Alternative Goal* (objetivo alternativo): classe que conterá a especificação de quais objetivos alternativos farão parte do vocabulário. Seu slot "*derived from*" permite instâncias de "*specific goal*" (objetivos específicos), sendo que somente objetivos específicos alternativos devem ser inseridos.
- *Alternative Role* (papel alternativo): classe que define quais papeis alternativos farão parte do vocabulário. Aqui o slot "*derived from*" permite instâncias de "*role*" (papel), sendo que somente papéis alternativos devem ser inseridos.
- *Alternative Activity (*atividade alternativa): nesta classe teremos a definição de que atividades alternativas comporão o vocabulário. Aqui o slot "derived from" referencia instâncias da classe "acticity" (atividade), sendo que somente atividades alternativas devem ser inseridas.
- *Alternative Skill* (destreza alternativa): essa classe define quais destrezas alternativas farão parte do vocabulário. O slot "derived form"

permite instâncias da classe “skill” (destreza), sendo que somente destrezas cuja variabilidade seja alternativa devem ser incluídas.

- *Os slots Optional Goal* (objetivo opcional), *Optional Role* (papél opcional), *Optional Activity (*atividade opcional) e *Optional Skill* (destreza opcional) seguem a mesma descrição dos slots alternativos, sendo que no slots “*derived from*” dos mesmos só devem ser incluídas instâncias dos conceitos respectivos que tenham a variabilidade do tipo “opcional”;
- *Mandatory Role with variable skill* (papél mandatório com destreza variável): define que papéis mandatórios com destrezas variáveis farão parte do vocabulário. Como foi dito na seção 4.1.2.1.1, a definição do vocabulário da LED deve referenciar os conceitos variáveis do domínio. Porém podemos ter conceitos comuns que tenham características (sub-conceitos) variáveis, que é o caso dos papéis. Nesta classe o slot “derived from” permite instâncias da classe “*role*” (papel), porém somente instâncias papéis mandatórios que tenham propriedades (atividades ou destrezas) variáveis devem ser incluídas;

Todas as instâncias do vocabulário são derivadas de conceitos de um modelo de domínio em particular. É importante ressaltar que em todas as classes do vocabulário teremos apenas a adição de conceitos já existentes no modelo de domínio e nunca a criação de novos conceitos. Qualquer novo conceito deve ser incluído em uma revisão no próprio modelo de domínio (retornando à fase de análise) antes de ser adicionado ao vocabulário.

4.2.2.3.2 Sub-Classe “Gramática da LED”

Conforme descrito na seção 4.1.2.1, a gramática na ontologia será representada na forma de uma sintaxe abstrata, definindo as combinações permissíveis entre os elementos do vocabulário em uma estrutura de árvore. Na ONTOGENMADEM, isso será definido nos slots da classe “*DSL Grammar*” (Gramática da LED), que referenciam instâncias do vocabulário, expondo os relacionamentos criados através do slot “parameterized by” de cada elemento do vocabulário. Os slots da classe “*DSL grammar*” são:

- *name* (nome): atribui um nome à gramática
- *term* (termo): identifica um elemento do vocabulário como termo válido na sintaxe abstrata. Este slot referencia instâncias de todos os elementos do vocabulário;

A Figura 37 mostra o relacionamento exposto pelo slot da classe gramática da LED: o slot "term" referencia os elementos do vocabulário, que por sua vez vão tendo seus relacionamentos definidos no slot "parameterized by" de cada instância. Podemos ver que os objetivos são parametrizados por papéis (alternativos, opcionais e mandatório com destreza variável) e os papéis são parametrizados por atividades e destrezas (alternativas ou opcionais). Os relacionamentos abaixo dos objetivos e papéis opcionais foram omitidos nessa figura para maior clareza da mesma, mas são os mesmos relacionamentos que constam abaixo dos objetivos e papéis alternativos.

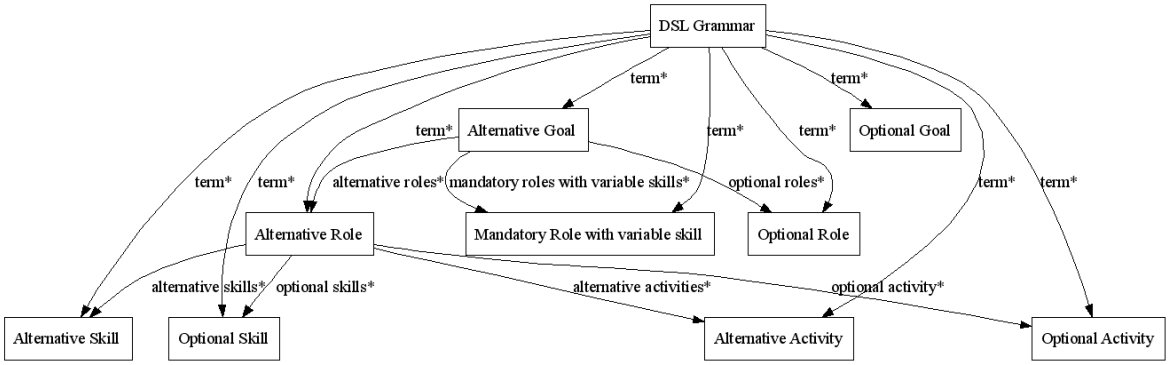

Figura 37 Relacionamentos definidos na classe "gramática da LED"

4.2.2.4 Classe "Construções do gerador"

As construções do gerador são aquelas definidas na seção 4.1.2.1.3 para se obter a concepção do gerador, que são as regras de montagem e os mapeamentos. Na ontologia, as regras de montagem estão definidas na classe "*Configuration Rules*" e os mapeamentos, nas classes "*Implementation Components*" e "*JADE Constructs*", sendo as mesmas subclasses da classe "*Generator Constructs*" (Figura 38).

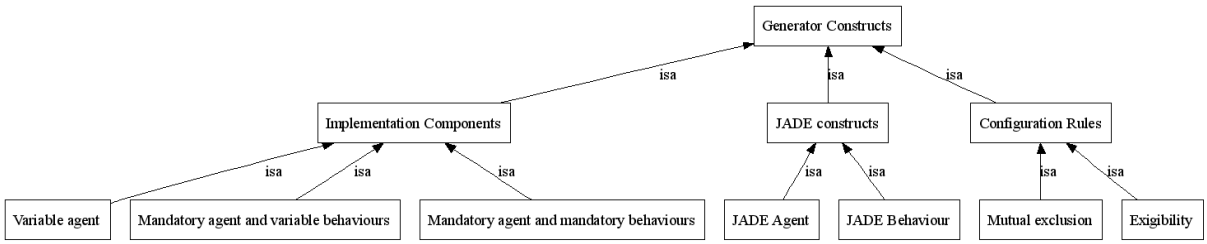

Figura 38 Hierarquia de classes do projeto do gerador na ONTOGENMADEM

4.2.2.4.1 Sub-Classe "Regras de montagem"

As regras de montagem foram definidas com a criação da classe "*Configuration Rules*". Existem duas sub-classes que definem as regras citadas no parágrafo anterior: a classe "*Exigibility"* (exigibilidade) e "*Mutual Exclusion"* (exclusão mútua). Essas subclasses têm dois slots que referenciam todos os coneitos do vocabulário: o slot "*existing concept*" (conceito existente) e o slot "*exigible concept*" (no caso da exigibilidade) ou "*excludable concept*" (no caso da exclusão mútua), sendo que na exigibilidade a presença do primeiro exige a presença do segundo e na exclusão mútua a presença do primeiro proíbe a existência do segundo.

4.2.2.4.2 Sub-Classe "Construções JADE"

Os mapeamentos estão definidos nas classes "*JADE Constructs*" e "*Implementation Agents*". A classe "*JADE Constructs*" (Figura 39) contém os mapeamentos dos conceitos da modelagem para os conceitos de implementação no ambiente JADE. Essa classe possui duas subclasses ("*JADE Agent*" e "*JADE Behaviour*") e tem o slot "*mapped from*" que é herdado pelas mesmas e define mapeamentos a partir de conceitos do modelo do framework. Essas subclasses são descritas a seguir:

- *Comportamento JADE ("JADE Behaviour"):* define os comportamentos dos agentes mapeados a partir das responsabilidades e destrezas que os agentes possuem no modelo do framework, com as quais devemos instanciar esta classe;
- *Agente JADE ("JADE Agent"):* Esta classe define os agentes JADE mapeados a partir dos agentes definidos no modelo do framework criado na fase de projeto de domínio da MADEM, e seus possíveis

comportamentos mapeados para instâncias da classe "*JADE Behaviour*". O slot "mapped from" permite instâncias da classe agente e o slot "*has behaviour*" aponta para os comportamentos do agente. Devemos instanciar esta classe com todos os agentes do modelo do framework e seus respectivos comportamentos JADE previamente instanciados.

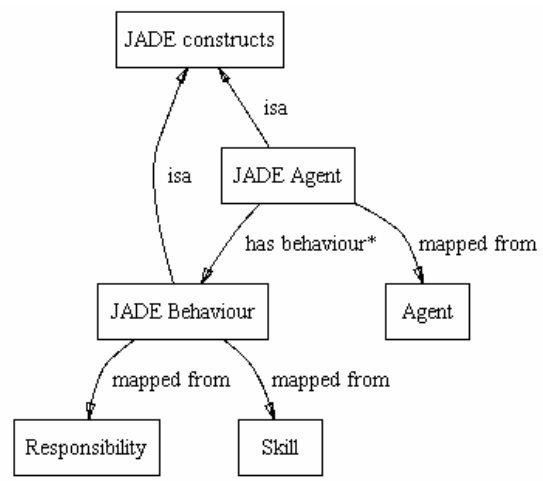

Figura 39 Sub-classe "Construções JADE"

4.2.2.4.3 Sub-Classe "Componentes de implementação"

A classe "*Implementation Components*" (Figura 40) define os componentes (agentes e seus respectivos comportamentos) que serão usados para construir qualquer elemento da família de sistemas. Em outras palavras são os componentes que o gerador utilizará na geração de um produto da família. Para isso, o gerador precisa conhecer não só os componentes variáveis, mas também a parte fixa que estará presente em qualquer elemento da família. A identificação desses agentes fixos e variáveis (e seus respectivos comportamentos) é possível através do que é definido nas seguintes subclasses da classe "*Implementation Components*":

- *Agentes mandatórios e comportamentos mandatórios ("Mandatory agent and mandatory behaviour"):* define o mapeamento de agentes mandatórios e comportamentos mandatórios para agentes JADE e comportamentos JADE respectivamente. Nesta classe estarão os agentes que compõem a parte fixa da família de

sistemas (agentes mandatórios que têm atividades e destrezas mandatórias);

- *Agentes mandatórios e comportamentos variáveis ("Mandatory agent and variable behaviour"):* define o mapeamento dos agentes mandatórios que têm comportamentos variáveis para agentes JADE e comportamentos JADE respectivamente, portanto são agentes que, por terem comportamento (responsabilidades e destrezas) variável, integram a parte variável da família de sistemas;
- *Agentes variáveis ("Variable agent"):* define o mapeamento dos agentes variáveis (agentes alternativos ou opcionais) e seus comportamentos (que podem ser fixos ou variáveis) para agentes JADE e comportamentos JADE.

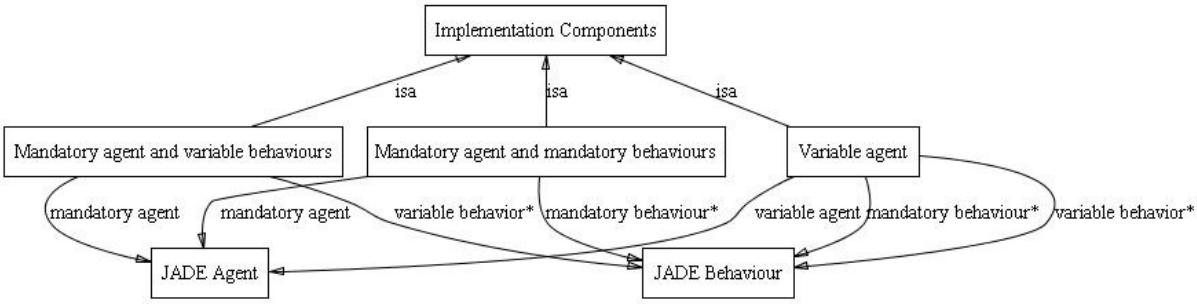

Figura 40 Classe "componentes de implementação"

4.2.3 O plugin ONTOGENMADEM

Para auxiliar a aplicação da metodologia GENMADEM, foi criada uma ferramenta cujo principal objetivo é automatizar as fases de projeto (sintaxe) e implementação (definição de mapeamentos de implementação) da LED, dando subsídios para a seleção, adaptação e composição automática a partir de uma especificação feita numa LED previamente definida com a GENMADEM.

Trata-se de um plugin do tipo "TabWidget" para o Protégé que, em conjunto com a ontologia ONTOGENMADEM, dá suporte à análise, projeto e implementação de LEDs. Trata-se de uma experiência pioneira, pois no estudo feito

nos capítulos 2 e 3 não foram encontradas ferramentas que abordam desde a fase de análise nem ferramentas voltadas para o paradigma multiagente.

4.2.3.1 Concepção

O primeiro passo na concepção da ferramenta foi a elaboração de scripts de consulta na linguagem baseada em regras Algernon [ALGERNON, 2005]. A criação desses scripts foi feita inicialmente visando dar suporte ao algoritmo de obtenção do vocabulário da LED. As experiências iniciais com os algoritmos no processo de copiar e colar os códigos na interface do Algernon (Algernon Tab) levou à idéia de criar uma interface que executasse esses scripts, fazendo também a criação do vocabulário e evitando o trabalho diretamente com os scripts.

Como as tarefas de modelagem do domínio serão executadas com o suporte do editor de ontologias Protégé, a concepção dessa interface para a GENMADEM foi na forma de um *plugin* do tipo "*TabWidget"* para o mesmo [PROTÉGÉ, 2005], resultando então num ambiente onde teremos a integração das atividades de modelagem da Engenharia de Domínio, e da ferramenta que dará suporte à abordagem gerativa num mesmo ambiente. O objetivo principal é prover uma interface que permita automatizar o processo de definição da LED, conforme já mencionado na seção 4.1.2.1.1, e apresentar a interface do gerador, de modo que essa interface tenha acesso aos produtos da modelagem de domínio, que serão usados pela mesma.

Os plugins são programas feitos na linguagem Java através da extensão de classes pré-definidas do próprio Protégé e podem ser usados para mudar ou estender o funcionamento do Protégé. O próprio Protégé é escrito como uma coleção de *plugins* e estes podem ser substituídos parcial ou integralmente para alterar a interface e comportamento do mesmo. A API (*Application Programming Interface*) do Protégé, que é um conjunto de classes Java, pode ser usada pelos plugins ou até por aplicações externas, que podem acessar as bases de conhecimento do Protégé e usar os formulários do mesmo sem executar a aplicação Protégé.

O Protégé tem alguns tipos básicos de plugins, dentre os quais o TabWidget, que foi escolhido para a ferramenta:

- **Tab Widget**: uma interface de usuário do tipo tab (ficha) que aparece na janela principal do Protégé, juntamente às tabs do sistema, como a tab de classes. Um exemplo de plugin do tipo Tab widget é a tab "Queries";
- **Slot Widget:** aparece um formulário e é usado para visualizar e receber um valor para um slot em uma instância. Um exemplo é o *TextFieldWidget*, que é usado para strings;
- **Back-end:** especifica o mecanismo que o Protégé usa para armazenamento (seja como texto ou banco de dados). Um exemplo é o *RDF back-end*.
- **CreateProject**: transforma um arquivo num formato produzido por outro programa e cria uma base de conhecimento Protégé mais similar possível;
- **Export**: provê um mecanismo extensível para exportar as bases de conhecimento do Protégé em vários outros formatos. Está relacionado com o plugin tipo back-end, mas é mais fácil de desenvolver e serve a outros propósitos;
- **Project:** permite a manipulação de um projeto Protégé

Nas seções a seguir teremos a descrição dos requisitos necessários ao uso da ferramenta, da implementação dos scripts Algernon e implementação do plugin, que descreve os passos comuns e específicos na criação de plugins.

4.2.3.2 Requisitos

Para o uso e aplicação da ferramenta GENMADEM, os seguintes requisitos tornam-se necessários:

- A ferramenta Protégé com o plugin Algernon instalado, que será o ambiente principal onde a ferramenta está inserida e onde todas as atividades do processo ocorrem;
- A ontologia ONTOGENMADEM, que consolida o conhecimento da abordagem e possibilita a execução das atividades de modelagem que ocorrem antes da utilização do plugin, além de armazenar todas os produtos tanto da modelagem como da GENMADEM;

- Arquivo “protege.lax” alterado, adicionando-se a a pasta onde fica o Algernon esteja na diretiva “lax.class.path”, para que o Protégé encontre o pacote de classes do Algernon.

4.2.3.3 Implementação do algoritmo de criação do vocabulário em Algernon

Algernon [ALGERNON, 2005] é um sistema de inferência baseado em regras implementado em Java e que usa a interface do Protégé através de um plugin do tipo TabWidget. Suas principais características são:

- Usa encadeamento de regras para trás e para frente para fazer inferência em dados armazenados em bases de conhecimento;

- Compatível com sistemas de gerenciamento de bases de conhecimento baseados em frames como Protégé;

- Linguagem nativa baseada em caminhos, mapeando diretamente para caminhos entre objetos relacionados numa base de conhecimento;

- Seus comandos podem ler e gravar valores em slots, criar novas classes, instâncias e slots, criar, executar e rastrear regras;

- É útil para sistemas que precisam processar informação armazenada em uma base de conhecimentos baseada em frames.

O Algernon foi escolhido para a implementação do algoritmo de derivação por permitir consultar e editar o projeto em uso no Protégé, possibilitando então a instanciação dos conceitos variáveis nas classes do vocabulário.

4.2.3.3.1 Conceitos

O Algernon, uma linguagem de programação em lógica, têm alguns conceitos que são peculiares a esse estilo de programação. Convém apresentarmos os principais conceitos para um melhor entendimento da implementação feita:

PATHS: Um “path” (caminho) numa base de conhecimento é uma sequência de frames que são conectadas por uma relação. Cada entrada para o Algernon deve ser um “path”. Por exemplo,

`A mãe B irmão C primo D esposa E irmã F` expressa o fato de que F é irmã da esposa do primo do irmão da mãe de F

CLÁUSULAS: especificam o relacionamento entre dois frames, na forma da tripla frame-slot-valor. A sintaxe no Algernon é (slot frame valor)

Exs: `(nome Pessoa Maria)`

`(cor Casa verde)`

`(nome Instance-0001 "Sodio")`

Um path é uma sequência de cláusulas:

`((sexo Pessoa "Masculino")(salario Pessoa 1000))`

RELAÇÃO: representa o slot de uma frame

VARIÁVEIS: em Algernon, variáveis começam com um sinal de interrogação. Ex.: ?name, ?x, ?tamanho. As variáveis em Algernon têm as seguintes características:

- Os nomes de variáveis devem ser significativos em relação ao que elas vão armazenar.
- Não precisam ser declaradas, elas assumem o tipo de acordo com o primeiro uso, mantendo esse tipo durante todo o path onde ela se encontra.
- Estão visíveis somente em um path. Não há variável global.
- Os tipos permitidos são: Boolean, List, Number, String, Symbol e qualquer classe da ontologia.

4.2.3.3.2 Comandos Algernon Utilizados

Na implementação do algoritmo de obtenção do vocabulário, os comandos do Algernon que foram usados são os que permitem a consulta, adição e exclusão de instâncias. Para entendimento dos algoritmos e das facilidades de consulta e manipulação da ontologia oferecidos pelo Algernon, esses comandos serão descritos a seguir com breves resumos da sua finalidade, sintaxe e exemplos. Nas descrições a seguir, as palavras entre os sinais “<” e “>” são os parâmetros esperados pelos comandos.

(:INSTANCE <classe> <instância>)

Finalidade: Verifica se um frame é uma instância direta ou indireta de outra, ou retorna todos as instâncias diretas ou indiretas de uma classe

Parâmetros:

- classe: frame da classe
- instância: instância para ser testada ou variável para ser ligada

Exemplos:

- `(:INSTANCE Pessoa Maria) ; --> verifica se Maria é uma instância de Pessoa`
- `(:INSTANCE Pessoa ?x) --> retorna todas as instâncias de Pessoa em ?x`

(:ANY <cláusula>)

Finalidade: Avalia a cláusula e retorna, aleatoriamente, somente um dos resultados produzidos pela mesma. Os outros resultados são descartados. Útil quando precisamos saber se pelo menos um resultado foi gerado pela cláusula que está sendo avaliada ou para restringir o resultado da mesma a apenas um elemento. Este comando será avaliado positivamente se a cláusula recebida retornar pelo menos um resultado ou negativamente se a cláusula não retornar resultados.

Parâmetros:

- cláusula: cláusula a ser avaliada

Exemplos:

- `(:ANY (:INSTANCE Pessoa ?x)) ; --> avalia a cláusula que obtém todas as instâncias de Pessoa em ?x e retorna somente um resultado da mesma`

(:OR <path 1> <path 2> ...)

Finalidade: testar se pelo menos um dos sub-paths passados como parâmetros obtém sucesso.

Parâmetros:

- path n: path do conjunto de paths a serem avaliados

Exemplos:

- `(:OR ((:INSTANCE Pessoa ?x)(nome ?x “Maria”)) ((:INSTANCE Pessoa ?y)(nome ?y “João”)) ) ; --> verifica se existem instâncias de Pessoa cujo nome seja “Maria” ou “João”`

(:ADD-INSTANCE (<variável> <classe>) <relações(slots)>...)

Finalidade: Cria uma instância de <classe> atribuindo aos slots (que são chamados de relações) os valores passados em <relações>.

Parâmetros:

- variável: variável que será ligada à nova instância que será criada
- classe: classe da nova instância
- relações: valores a armazenar nos slots, no formato (slot frame valor)

Exemplos:

- incluir uma instância de pessoa com nome="João", nascimento="01/01/1970" e apelido (que é um slot que aceita múltiplos valores) com os valores “Jota” e “Joãozinho”.

```
(:ADD-INSTANCE (?pess Pessoa)
  (nome ?pess João)
  (nascimento ?pess "01/01/1970")
  (apelido ?pess "Jota”)
  (apelido ?pess "Joãozinho")
)
```

- OBS: Para atualizar slots múltiplos no comando ADD-INSTANCE basta repetir a relação referente ao slot múltiplo com os valores a incluir, no mesmo comando.

(:DELETE-INSTANCE <instância>)

Finalidade: Exclui uma instância

Parâmetros:

- instância: instância a ser excluída

Exemplo:

- Excluir uma instância de pessoa cujo nome é João

```
(  (:INSTANCE Pessoa ?pess)
   (nome ?pess “João”)
   (:DELETE-INSTANCE ?pess)
)
```

(:CLEAR-RELATION <instância> <slot>)

Finalidade: Apaga o conteúdo de um slot de uma instância

Parâmetros:

- instância: nome da instância
- slot: nome do slot cujo conteúdo deve ser eliminado

Exemplo:

- Excluir uma instância de pessoa cujo nome é João

```
(   (:INSTANCE Pessoa ?pess)
    (nome ?pess "João")
    (:CLEAR-RELATION ?pess apelido)
)
```

- OBS: Pode ser executado com ask ou tell

Atualização de slot com "assert": ((<slot> <instância> <valor>))

Finalidade: atualizar o conteúdo de um slot de uma instância já existente com um novo valor. Para atualizar um slot, é feita uma afirmação com um valor explicitado.

Parâmetros:

- slot: slot a ser atualizado
- instância: instância cujo slot deve ser atualizado
- valor: valor a gravar no slot

Exemplo:

- Atualizar o slot "apelido" com "teste", na instância de pessoa cujo nome é "João"

```
(   (:INSTANCE Pessoa ?pess)
    (nome ?pess "João")
    (apelido ?pess "Teste") )
```

- Obs.1: deve ser executado com TELL.
- Obs.2: se o slot for múltiplo, os valores serão adicionados. Caso isso não seja desejado, deve-se usar o comando CLEAR-INSTANCE antes.

4.2.3.3.3 Implementação do algoritmo de obtenção do vocabulário em Algernon

A implementação do algoritmo de obtenção do vocabulário (Figura 28) foi feita aproveitando os recursos de consulta, inserção e alteração de instâncias da base de conhecimento oferecidas pela API do Algernon. Foi criado um *path* (conjunto de cláusulas) para cada classe de conceito a ser inserido na classe "DSL Vocabulary". Cada um desses paths consiste basicamente de comandos Algernon para obter a lista de instâncias de conceitos variáveis do domínio para os quais devem ser instanciadas as classes do vocabulário da LED (comandos :INSTANCE) seguida de comandos para adicionar a instância na devida classe a partir dos dados obtidos (comando :ADD-INSTANCE). Para algumas classes do vocabulário torna-se necessário utilizar comandos adicionais para atualizar os slots múltiplos (:CLEAR-INSTANCE e "asserts" a serem executados com TELL). Comandos de exclusão (:DELETE-INSTANCE) de instâncias também foram usados para eliminar todas as instâncias de cada classe do vocabulário do modelo de domínio em operação, para evitar duplicidades.

A seguir serão descritos os paths criados para instanciar cada classe de "DSL Vocabulary". Nos códigos apresentados nas figuras a seguir, o nome do modelo de domínio desejado referencia "ONTOINFO-DM: Domain Model of Information Retrieval and Filtering", mas essa constante é inserida no código Algernon pelo plugin, de acordo com o modelo de domínio selecionado na interface. Desse modo, esses scripts apresentados podem ser executados na interface do plugin Algernon no Protégé. É importante notar também que cada código tem duas partes: uma de consulta e outra de adição. A primeira parte (consulta) pode ser executada separada da segunda na interface do Algernon para vermos os resultados obtidos.

<u>Objetivos opcionais</u>: O *path* criado (Figura 41) obtém inicialmente todas as instâncias de objetivos específicos (linha 2) cuja variabilidade seja opcional (linha 3), guardando as instâncias na variável "?Specific_Goal" e os nomes das instâncias na variável "?name". Depois o conjunto de resultados obtido é restrito a somente as instâncias que pertencem ao modelo de domínio selecionado. Isso é feito obtendo-se a instância do modelo de domínio desejado (linhas 5 e 6), os modelos contidos nesse modelo de domínio (linha 8) que sejam instâncias da classe "Goal Model" (em

outras palavras, que sejam modelos de objetivos) (linha 9) e tenha como conceito o (s) objetivos específicos obtidos na consulta inicial. Algumas das variáveis usadas nessa parte inicial serão usadas na segunda parte, no comando ADD-INSTANCE, para atualizar os slots da classe que será instanciada. Nas linhas 11 a 16 temos o comando ADD-INSTANCE que irá instanciar a classe “Optional Goal” para cada resultado obtido na consulta anterior. Podemos ver que a consulta vai da linha 2 a 10 e a instanciação da classe “Specific Goal” vai da linha 11 a 16.

Objetivos Alternativos: o path criado aqui tem os mesmos comandos do anterior, com a diferença na linha, que procura por objetivos com variabilidade “alternative” em vez de “optional” e na linha 11, que instanciará a classe “Alternative Goal” em vez de “Optional Goal”.

```
1   (
2       (:instance "Specific Goal" ?Specific_Goal)
3       ("variability type" ?Specific_Goal "optional")
4       (name ?Specific_Goal ?name)
5       (:instance "Domain Model" ?Domain_Model)
6       (name ?Domain_Model "ONTOINFO-DM: Domain Model of Information Retrieval and
7   Filtering")
8       (contains ?Domain_Model ?Goal_Model)
9       (:instance "Goal Model" ?Goal_Model)
10      (concepts  ?Goal_Model ?Specific_Goal)
11      (:add-instance (?Optional_Goal "Optional Goal")
12              (name ?Optional_Goal ?name)
13              ("derived from" ?Optional_Goal ?Specific_Goal)
14              ("domain model" ?Optional_Goal ?Domain_Model)
15      )
16  )
```

Figura 41 Código Algernon para os objetivos opcionais

Papéis opcionais: Como os papéis estão ligados a responsabilidades e a variabilidade é um slot da responsabilidade, esse código (Figura 42) pesquisa instâncias de responsabilidades opcionais (linhas 2 e 3). Nas linhas 4 e 5 guardamos em uma variável nomes dos papéis associados a essas responsabilidades e nas linhas 6 a 11 restringimos os resultados obtidos a apenas os papéis que pertençam ao modelo de domínio desejado. Nas linhas 12 a 16 o comando ADD-INSTANCE é usado para instanciar a classe “Optional Role” com os papéis opcionais obtidos.

Papéis alternativos: o path para esta classe tem os mesmos comandos do anterior, com a diferença na linha, que procura por responsabilidades com variabilidade “alternative” em vez de “optional” e na linha 12, que instanciará a classe “Alternative Role” em vez de “Optional Role”.

```
1   ((
2   (:instance Responsibility ?Responsibility)
3   ("variability type" ?Responsibility "optional")
4   ("performed by" ?Responsibility ?Role)
5   (name ?Role ?name)
6   (:instance "Domain Model" ?Domain_Model)
7   (name ?Domain_Model "ONTOINFO-DM: Domain Model of Information Retrieval and
8   Filtering")
9   (contains ?Domain_Model ?Role_Model)
10  (:instance "Role Model" ?Role_Model)
11  (concepts  ?Role_Model ?Role)
12  (:add-instance          (?Optional_Role "Optional Role" )
13      (name ?Optional_Role ?name)
14      ("derived from" ?Optional_Role ?Role)
15      ("domain model" ?Optional_Role ?Domain_Model))
16  )
```

Figura 42 Código Algernon para os papéis opcionais

Atividades opcionais: um *path* similar ao usado para os objetivos foi criado para as atividades (Figura 43), fazendo a pesquisa das atividades opcionais nas linhas 2 a 5 e a confirmação se pertence ao modelo de domínio desejado nas linhas 6 a 11. Nas linhas 12 a 16 a instanciação da classe "Optional Activity" é feita.

Atividades alternativas: o *path* para esta classe tem os mesmos comandos do anterior, com a diferença na linha 3 que procura por atividades com variabilidade "*alternative*" em vez de "*optional*" e na linha 12, que instanciará a classe "*Alternative Activity*" em vez de "*Optional Activity*".

```
1   (
2   (:instance Activity ?Activity)
3   ("variability type" ?Activity "optional")
4   (exercises ?Activity ?Responsibility)
5   (name ?Activity ?name)
6   (:instance "Domain Model" ?Domain_Model)
7   (name ?Domain_Model "ONTOINFO-DM: Domain Model of Information Retrieval and
8   Filtering")
9   (contains ?Domain_Model ?Model)
10  (:instance "Role Model" ?Model)
11  (concepts ?Model ?Activity)
12  (:add-instance (?Optional_Activity "Optional Activity" )
13      (name ?Optional_Activity ?name)
14      ("derived from" ?Optional_Activity ?Activity)
15      ("domain model" ?Optional_Activity ?Domain_Model))
16  )
```

Figura 43 Código Algernon para as atividades opcionais

Destrezas opcionais: similar ao *path* dos objetivos, o *path* das destrezas opcionais (Figura 44), faz a pesquisa das destrezas opcionais nas linhas 2 a 4 e a confirmação se pertence ao modelo de domínio desejado nas linhas 5 a 10. Nas linhas 11 a 15 a instanciação da classe "*Optional Skill*" é feita.

Destrezas alternativas: o *path* para esta classe tem os mesmos comandos do anterior, com a diferença na linha 3 que procura por destrezas com variabilidade

"*alternative*" em vez de "*optional*" e na linha 12, que instanciará a classe "*Alternative Skill*" em vez de "*Optional Skill*".

```
1   (
2   (:instance Skill ?Skill)
3   ("variability type" ?Skill "optional")
4   (name ?Skill ?name)
5   (:instance "Domain Model" ?Domain_Model)
6   (name ?Domain_Model "ONTOINFO-DM: Domain Model of Information Retrieval and
7   Filtering")
8   (contains ?Domain_Model ?Model)
9   (:instance "Role Model" ?Model)
10  (concepts ?Model ?Skill)
11  (:add-instance (?Optional_Skill "_Skill")
12      (name ?_Skill ?name)
13      ("derived from" ?_Skill ?Skill)
14      ("domain model" ?_Skill ?Domain_Model))
15  )
```

Figura 44 Código Algernon para as destrezas opcionais

<u>Papéis mandatórios com propriedades variáveis</u>: Este path (Figura 45), que é um pouco mais complexo que os outros, tem como pré-requisito que as outras classes do vocabulário da LED já estejam instanciadas, pois o mesmo pesquisa essas instâncias para descobrir atividades e destrezas variáveis das responsabilidades mandatórias. O path pesquisa as responsabilidades mandatórias pertencentes ao domínio desejado (linhas 1 a 4), confirma o modelo de domínio (linhas 5-10), checa se a responsabilidade tem destrezas ou atividades variáveis (linhas 11 a 19), usando o comando :OR para suceder em qualquer caso e o :ANY para evitar a duplicidade de papéis (se houver mais de uma destreza ou atividade variável para a mesma responsabilidade, o que é bastante comum). As linhas 20 a 25 instanciam a classe "Mandatory Role with variable property".

Os scripts Algernon criados para a geração dos mapeamentos são similares aos apresentados para o vocabulário. A listagem completa do dos scripts consta no APÊNDICE B.

```
1  ((:instance Responsibility ?Responsibility)
2  ("variability type" ?Responsibility "mandatory")
3  ("performed by" ?Responsibility ?Role)
4  (name ?Role ?name)
5  (:instance "Domain Model" ?Domain_Model)
6  (name ?Domain_Model "ONTOINFO-DM: Domain Model of Information Retrieval and
7  Filtering")
8  (contains ?Domain_Model ?Model)
9  (:instance "Role Model" ?Model)
10 (concepts ?Model ?Responsibility)
11 (:OR ((:ANY (:instance "Alternative Skill" ?Alternative_Skill) ("derived from"
12 ?Alternative_Skill ?Skill) (requires ?Responsibility ?Skill)))
13      ((:ANY (:instance "Optional Skill" ?Optional_Skill) ("derived from"
14 ?Optional_Skill ?Skill) (requires  ?Responsibility ?Skill)))
15      ((:ANY (:instance "Optional Activity" ?Optional_Activity) ("derived from"
16 ?Optional_Activity ?Activity) ("exercised through" ?Responsibility ?Activity)))
17      ((:ANY (:instance "Alternative Activity" ?Alternative_Activity) ("derived from"
18 ?Alternative_Activity ?Activity) ("exercised through" ?Responsibility ?Activity)))
19 )
20 (:add-instance
21     (?Mandatory_Role_with_variable_property "Mandatory Role with variable property"
22 )
23     (name ?Mandatory_Role_with_variable_property ?name)
24     ("derived from" ?Mandatory_Role_with_variable_property ?Role)
25     ("domain model" ?Mandatory_Role_with_variable_property ?Domain_Model))
26 ))
```

Figura 45 Código Algernon para os papéis mandatórios com propriedades variáveis

4.2.3.4 Implementação do plugin

A implementação de um plugin Protégé segue alguns passos comuns para todos os tipos e outros passos específicos a cada tipo de plugin. A documentação do Protégé descreve o processo básico de criação de um plugin, porém em muitos casos a documentação das classes da API (em Javadoc) não contém detalhes que facilitariam bastante o desenvolvimento, levando o desenvolvedor a buscar soluções alternativas e demoradas para solucionar eventuais problemas, como ciclos de tentativa e erro, análise do código fonte de outros plugins, participação em fórums de mensagens relacionados ao tema. Os passos descritos como gerais são:

1- Desenvolver um “esqueleto” do plugin, que é um primeiro esboço com o nome desejado, e testar no Protégé:

1.1- Criar uma Subclasse da classe apropriada ao tipo de plugin desejado;

1.2- Implementar os métodos necessários para fazer algo simples;

1.3- Criar um arquivo manifesto *(manifest file*) com uma entrada apropriada para o plugin. Esse arquivo é necessário para que o Protégé “enxergue” e acione o plugin. A criação de um arquivo JAR é opcional e desejável apenas ao finalizar e distribuir o plugin. O Protégé pode carregar e executar plugins baseados

apenas em arquivos ".class" em um diretório. Importante: esse arquivo manifesto não deve ter nenhum espaço desnecessário após a assinatura da classe que contém o método initialize do plugin.

2- Incrementar o plugin, desenvolvendo o código que irá prover a funcionalidade específica do mesmo.

3- Empacotar e distribuir o plugin: concluído o desenvolvimento e teste do plugin este deve ser empacotado para que possa ser distribuído a outros usuários. A recomendação é que seja criado um arquivo JAR contendo a(s) classe(s) que integram o plugin e o arquivo manifesto.

A interface projetada para esse plugin é mostrada na Figura 46. Como podemos ter mais de um modelo de domínio no mesmo arquivo de projeto ONTOGENMADEM do Protégé, o primeiro campo da interface é um componente do tipo JcomboBox que permite selecionar para qual modelo de domínio deseja-se gerar o vocabulário da LED. As opções que aparecem nesse seletor são obtidas também através de consulta à ontologia com um script Algernon e são itens que teriam que ser editados manualmente caso os scripts fossem executados um a um na janela de interface do Algernon.

Logo em seguida temos um painel referente à geração da LED, com um campo de edição para dar um nome à LED que deseja-se gerar e um botão que inicia o processo de criação do vocabulário. Ao clicar-se nesse botão uma confirmação é solicitada, pois caso já exista um vocabulário para o modelo de domínio selecionado, este será excluído antes da geração (Figura 47).

O painel seguinte, que dá suporte à fase de projeto do gerador, contém um botão que gera os mapeamentos que compõem as construções do gerador. Esta fase deve ser executada após a geração da LED. Na parte inferior da interface ONTOGENMADEM temos uma caixa de texto do tipo JtextArea que mostra os resultados da geração do vocabulário.

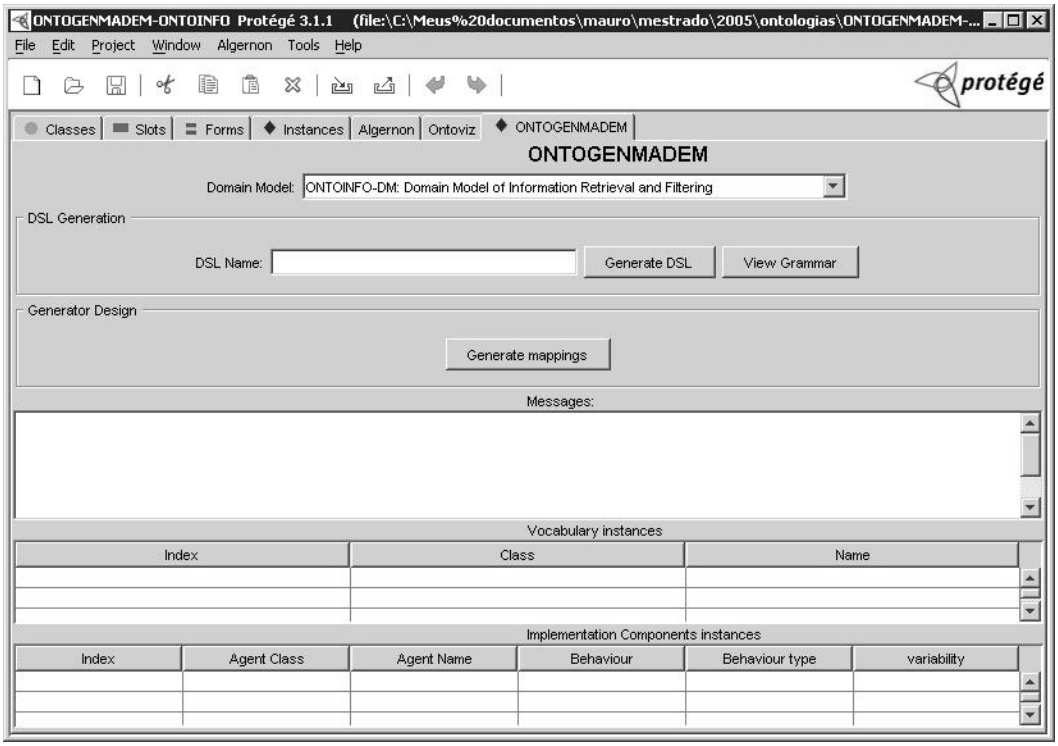

Figura 46 Interface do TabWidget ONTOGENMADEM no Protégé

Figura 47 Confirmação para geração do vocabulário

Para possibilitar o acesso à ontologia a partir do plugin, uma instância da classe “Algernon” e “AlgernonKB” são criadas na classe do plugin. Essa instância é criada referenciado-se o nome do projeto em uso no Protégé, que é obtido instanciando-se a classe Project com o construtor getProject() e usando-se o método getProjectFilePath() dessa instância. Esse método retorna o nome do arquivo Protégé em uso, com o caminho completo. O restante do plugin é composto de código Java que efetua chamada ao método “ask” da instância Algernon criada. Maiores detalhes da implementação podem ser observados no código completo do plugin está no APÊNDICE B e está bem comentado, para favorecer um bom entendimento e futura extensão.

O plugin foi inserido em um pacote cujo nome segue a regras de domínio invertido, tomando como base a instituição e laboratório onde este trabalho foi

desenvolvido (br.ufma.deinf.maae.ontogenmadem_plugin), para uma melhor identificação e evitar duplicidades. Ao final, foi criado um arquivo "ONTOGENMADEM.jar" contendo as classes do plugin. Esse arquivo ficará em uma sub-pasta da pasta plugins do Protégé, cujo nome será o nome do pacote, conforme será descrito na seção a seguir.

4.2.3.5 Instalação do plugin

Para instalar o plugin basta executar os seguintes passos:

- Criar a sub-pasta "br.ufma.deinf.maae.ontogenmadem_plugin" na pasta "plugins" do Protégé (\Arquivos de programas\Protege\plugins").
- Copiar o arquivo "ONTOGENMADEM.jar" (pacote que contém a classe principal do plugin) para essa pasta
- Iniciar o Protégé abrindo a ontologia ONTOGENMADEM e habilitar o plugin no menu "Project – Configure". O plugin só permite ser habilitado se o projeto aberto for a ontologia ONTOGENMADEM.

4.3 Considerações finais

Este capítulo apresentou a metodologia GENMADEM e a ontologia e ferramenta ONTOGENMADEM. A metodologia GENMADEM se propõe a ser uma solução viável para o desenvolvimento de LEDs a partir de um modelo de domínio e modelo de framework multiagente, onde o processo de desenvolvimento da LED referencia diretamente os conceitos do modelo de domínio criado. A ontologia ONTOGENMADEM suporta a aplicação da GENMADEM nas fases de análise e projeto do domínio e na especificação da LED. A ferramenta ONTOGENMADEM, concebida como um plugin para o editor de ontologias Protégé, automatiza a maior parte do processo de desenvolvimento da LED.

Para uma melhor classificação das mesmas, a Tabela 18 e Tabela 19 fazem um comparativo com as outras metodologias e ferramentas estudadas, respectivamente, permitindo uma análise comparativa.

A GENMADEM se diferencia das demais metodologias e técnicas para o desenvolvimento de LEDs por mostrar tarefas bem enquadradas nas fases clássicas de análise, projeto e implementação da Engenharia de Domínio; por ser baseada

em ontologias, que se mostram como uma boa solução para a abstração de software reutilizável; e por oferecer um processo que contempla desde a análise de domínio até a concepção da LED e projeto do gerador.

A ferramenta ONTOGENMADEM se diferencia das ferramentas analisadas por oferecer um suporte à análise, projeto e implementação (ainda que parcialmente) de LEDs voltadas para o paradigma multiagente, nada impedindo entretanto que esta seja aplicada em outro paradigma de desenvolvimento de software.

	Especifi-cação da sintaxe	**Especifi-cação da semântica**	**LED's desenvol-vidas**	**Fases abordadas**	**Técnica de Análise de Domínio**	**Implemen-tação**	**Documen-tação**
TOD-LED	BNF	Ontologias	LSRF	Especificação	GRAMO	-	Sim
Sprint	Gramática BNF	Semântica denota-cional	GAL, PLAN-P [THIBAULT, 1998]	Especificação Projeto e impleentação	Não especifica do	Interpretador e avaliação parcial	Não
DAS	Não especifica-do	Semântica denota-cional	MSL [WIDEN, Hook, 1998]	Projeto e implemen-tação	Técnica própria	Interpretador	Não
FAST	Gramática BNF	Não especificad o	Auditdraw [WEISS, 1995]	Especificação Projeto e implementa-ção	Não especifica do	Não especificado	não
DEMRAL	Árvore abstrata e BNF	Informal	GMCL [CZARNEC KI, 1998]	Análise, projeto e implementaçã o	Própria	Inserção (meta-programação com templates)	não
GENMADEM	Ontologias	Modelo de domínio baseado em ontologias	LSRF2	Especificação ,projeto e Implementa-Cão (parcial)	GRAMO	Gerador de aplicações	sim

Tabela 18 Comparativo da GENMADEM com outras técnicas estudades

Ferramenta	**Metodologia que suporta**	**Tecnologia base**	**Análise**	**Projeto**	**Implemen-tação**
JTS	-	Genvoca / pré-compilação	-	Sim	Sim
LaLa	-	Knowledge base/ seleção	-	Sim	Sim
Stratego	-	Regras de transformação	-	Parcial	Sim
ONTOGENMADEM	GENMADEM	Ontologias	Sim	Sim	parcial

Tabela 19 Comparativo da ONTOGENMADEM com outras ferramentas estudadas

5. ESTUDO DE CASO: DESENVOLVIMENTO DE UMA LED E PROJETO DE GERADOR NO DOMÍNIO DA RECUPERAÇÃO E FILTRAGEM DE INFORMAÇÃO

Para avaliar e demonstrar a aplicação da metodologia GENMADEM na Engenharia de Domínio Multiagente, este capítulo apresenta um estudo de caso envolvendo o domínio da recuperação e filtragem de informação. A escolha deste domínio deve-se ao fato de já existir um modelo de domínio inicialmente concebido sem a abordagem gerativa, e da variabilidade presente no mesmo, que se torna oportuna para demonstrar as extensões oferecidas pela GENMADEM para o tratamento da mesma. Dentre outros possíveis domínios de aplicação que podem ser explorados podemos citar o dos sistemas de tráfego, monitoramento de redes, inteligência de negócios e ensino interativo.

A seção 5.1 descreve o domínio do problema. A seção 5.2 apresenta a variabilidade do domínio da recuperação e filtragem e as seções seguintes apresentam a aplicação da GENMADEM no desenvolvimento da LED e projeto do gerador, com aplicação da ferramenta ONTOGENMADEM.

5.1 O Domínio da Recuperação e Filtragem de Informação

O acesso à informação [SALTON, McGILL, 1983] é uma área da Ciência da Computação que visa prover mecanismos para que seja possível localizar e recuperar informações relevantes aos usuários da forma mais eficiente possível. Os principais conceitos e processos envolvidos no acesso à informação [BAEZA-YATES, RIBEIRO-NETO, 1999] [SALTON, McGILL, 1983] [GIRARDI, 1998] são descritos a seguir.

A princípio, tem-se uma necessidade de informação que deve ser expressa em uma consulta ou perfil de usuário. Uma consulta caracteriza uma necessidade de informação pontual. Um perfil de usuário caracteriza uma necessidade de informação em longo prazo e esse perfil pode ser obtido de forma explícita, quando o usuário define explicitamente sua(s) área(s) de interesse ou implicitamente, através da mineração de uso, que consiste em monitorar as sessões

de navegação do usuário pelos tópicos integrantes da área de interesse e deduzir o seu grau de interesse por eles.

Se a necessidade de informação for pontual o sistema deve acessar uma base indexada e recuperar os elementos de informação relevantes ordenados segundo sua similaridade com a necessidade de informação.

Se a necessidade de informação for em longo prazo o sistema deve fazer a filtragem de informações (FI), que consiste em estar continuamente monitorando fontes de informação a procura de elementos de informação que possam ser relevantes a seus usuários representados através de seus perfis. Existem duas forma básicas de filtragem: a filtragem baseada no conteúdo (FBC) e a filtragem colaborativa (FC). Há também a filtragem pela união de características das duas anteriores: a filtragem híbrida (FH). A filtragem baseada em conteúdo consiste na análise da correlação entre o conteúdo dos itens de informação com o perfil do usuário para recomendar itens relevantes e descartar itens não pertinentes. A filtragem colaborativa baseia-se na correlação entre perfis, visando agrupar usuários com necessidades e interesses semelhantes, e na relevância que os usuários dão para os itens de informação através de avaliações.

Alguns conceitos se destacam no âmbito do acesso à informação, sendo os principais:

Necessidade de informação: é uma carência de informação por parte de um ou mais usuários. A necessidade de informação pode ser pontual, ou seja, uma necessidade a ser satisfeita imediatamente, ou em longo prazo, isto é, uma necessidade que se prolonga por um certo tempo;

Fonte de informação: é um repositório de elementos de informação. As fontes de informação podem ser estruturadas ou não estruturadas, podem ainda ser dinâmicas ou estáticas. Um exemplo é a Web;

Item de informação: é um item que contém informação. Os elementos de informação podem ser dos seguintes tipos: documentos textuais e hipermídia, vídeo, som, imagem;

Consulta: é uma representação da necessidade de informação pontual do usuário. Uma consulta é expressa em uma linguagem de especificação de consulta;

Perfil de usuário: é uma representação do usuário que identifica a sua necessidade de informação de longo prazo, a partir da qual o sistema filtra

elementos de informação que possam ser relevantes. O perfil do usuário pode ser capturado de forma explícita (ex: preenchimento de formulário) ou implícita (observação de links percorridos);

Representação: é uma representação interna de itens de informação, da consulta ou perfil do usuário;

Recuperação: é uma modalidade de acesso à informação utilizada no caso de uma necessidade de informação pontual.

Filtragem: é uma modalidade de acesso à informação utilizada no caso de uma necessidade de informação em longo prazo;

5.2 Variabilidade no domínio da recuperação e filtragem de informação

Como já foi mencionado na seção 4.1.2, para a aplicação da GENMADEM, precisamos ter um modelo de domínio previamente construído segundo a metodologia MADEM. Portanto, para a execução do estudo de caso apresentado neste capítulo, foi feita uma extensão da ONTOINFO [SERRA, GIRARDI, ALVES, 2004], que é um modelo de domínio baseado em ontologias para a recuperação e filtragem de informações. Como a extensão desse modelo de domínio não é o assunto central deste capítulo, apesar de ser umas das contribuições deste trabalho, os modelos estendidos e descrições constam no APÊNDICE A deste trabalho e esta seção então se restringirá a apresentar a variabilidade definida nesses modelos, que é o ponto de partida para as fases da GENMADEM. Caso seja desejado um entendimento maior acerca dos conceitos da modelagem cuja variabilidade está sendo descrita nesta seção, o modelo de domínio do APÊNDICE A deve ser consultado.

5.2.1 Variabilidade no modelo de objetivos

A variabilidade que estabelece a diferença entre os possíveis elementos da família de sistemas, começa a aparecer no modelo de objetivos a partir dos objetivos específicos, considerando-se que podemos ter elementos da família de sistemas que serão concebidos para atender apenas alguns dos objetivos específicos. Na Figura 48 e Figura 49 podemos ver os detalhes da variabilidade dos objetivos específicos presentes nesse modelo de objetivos.

Nesse modelo de objetivos os objetivos específicos “satisfazer necessidades de informação pontuais através de técnicas de recuperação” e “satisfazer necessidades de informação de longo prazo através de técnicas de filtragem” são opcionais entre si, e poderemos ter elementos da família de sistemas com os dois objetivos, com um dos dois, mas sempre um dos dois deverá estar presente.

Os objetivos específicos “satisfazer necessidades de informação de longo prazo através de filtragem de conteúdo”, “satisfazer necessidades de informação de logo prazo através de filtragem colaborativa” e “satisfazer necessidades de informação de longo prazo através de filtragem híbrida” são alternativos entre si. Portanto podemos ter aplicações da família de sistemas que terão um desses três objetivos específicos (mas nunca nenhum).

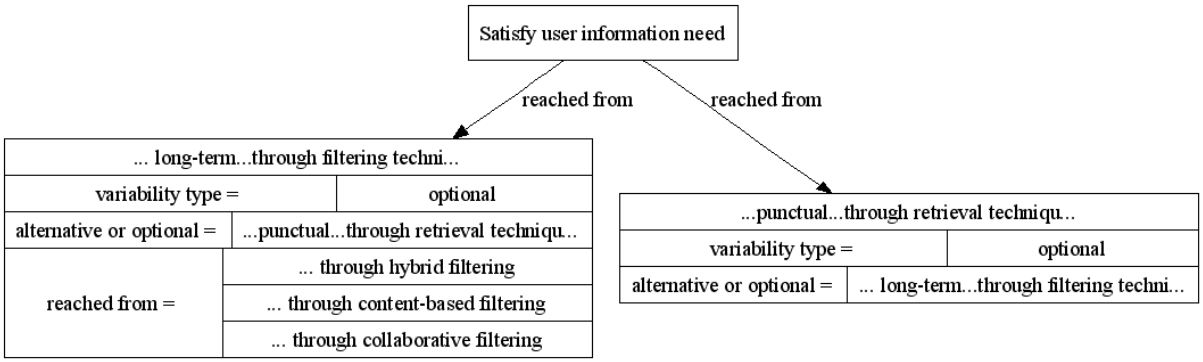

Figura 48 Objetivos opcionais na ONTOINFO

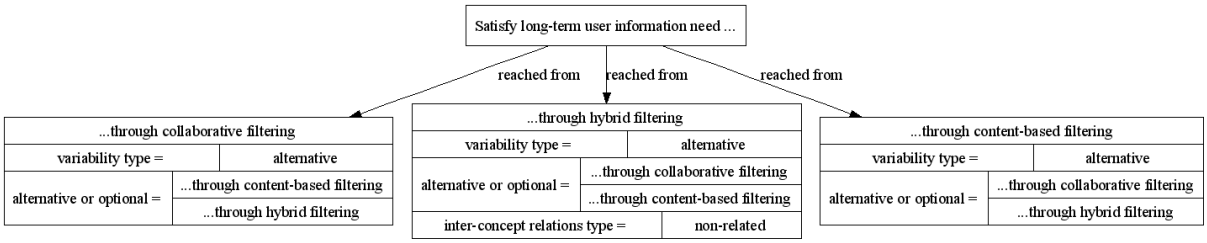

Figura 49 Objetivos alternativos na ONTOINFO

5.2.2 Variabilidade nos modelos de papéis

No modelo de papéis relativo à *recuperação da informação*, as responsabilidades são todas mandatórias e, conseqüentemente, os objetivos associados a elas. Isto significa que esses papéis estarão presentes numa instância

da família de sistemas se o objetivo “satisfazer necessidades de informação pontuais por técnicas de recuperação” estiver presente, visto que o mesmo é opcional. Porém temos destrezas alternativas. A representação da consulta e dos itens de informação pode ser feita através de três técnicas: espaço vetorial, linguagem natural e agrupamento (*clustering*) de termos ou documentos. Portanto, as responsabilidades “Representação e indexação dos itens de informação” e “Representação da consulta do usuário” têm três destrezas alternativas, que representam cada uma dessas formas de representação da consulta do usuário e dos itens de informação. Temos destrezas alternativas também para a responsabilidade “Comparação e análise de similaridade”, que são as técnicas do produto interno e co-seno, que podem ser usadas para a análise de similaridade entre a consulta do usuário e os itens de informação. Na Figura 50 podemos ver o detalhamento dessa variabilidade nos *slots* das instâncias, com o papel no primeiro nível, a responsabilidade no nível intermediário e as destrezas no terceiro nível.

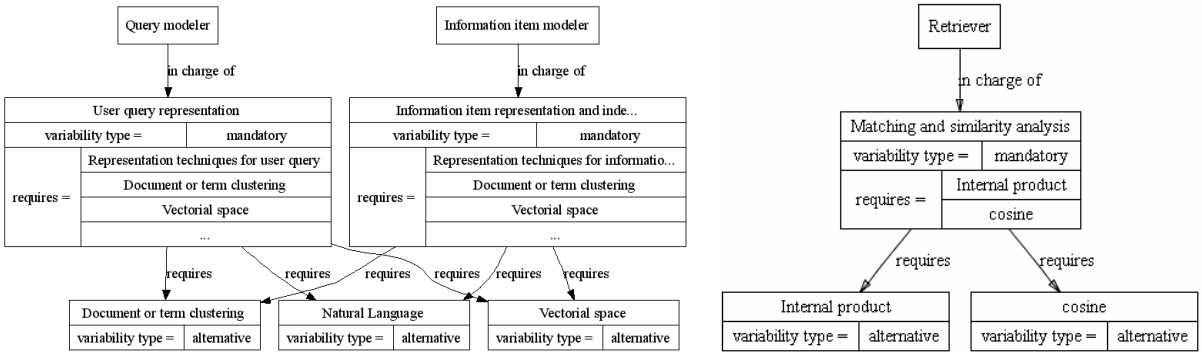

Figura 50 Destrezas alternativas no modelo de papéis relativo à recuperação

No modelo de papéis relativo à *modelagem do usuário*, as responsabilidades “Aquisição explícita do perfil” e “Aquisição implícita do perfil” são opcionais entre si, pois podemos ter sistemas com a aquisição implícita, aquisição explícita, ou ambas (Figura 51).

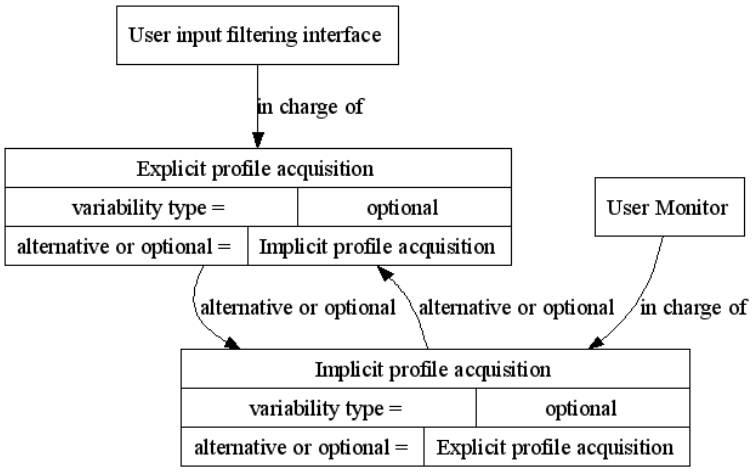

Figura 51 Papéis opcionais no modelo de papéis relativo à modelagem do usuário

Nesse mesmo modelo, considerou-se duas técnicas para se fazer a aquisição implícita do perfil do usuário, que são as duas destrezas alternativas da responsabilidade "Aquisição implícita do perfil": "Applet Java" e "Mosaic Browser" (Figura 52). De modo similar, existem técnicas conhecidas para realização da responsabilidade "Construção e atualização do modelo de usuário", que serão as destrezas alternativas "Kmeans clustering", "Regras de associação", "Descoberta de padrões seqüenciais" e "Classificação" (Figura 53).

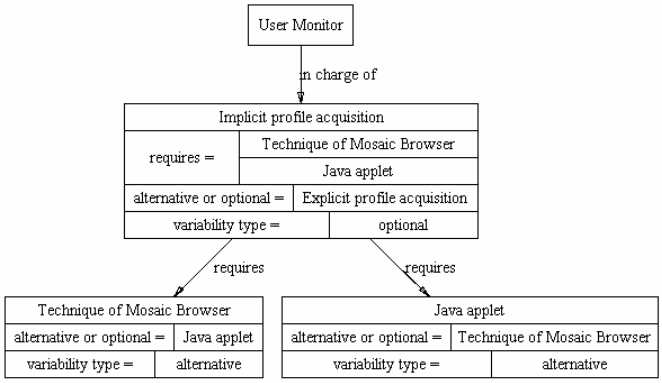

Figura 52 Destrezas alternativas da responsabilidade "Aquisição implícita do perfil"

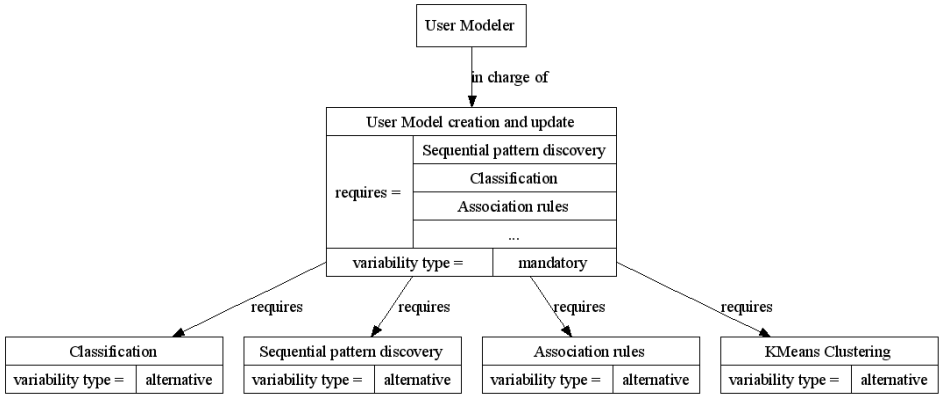

Figura 53 Destrezas alternativas da responsabilidade "Construção e atualização do modelo de usuário"

No modelo de papéis relativo à *filtragem de informação*, todas as responsabilidades são mandatórias, portanto estarão presentes numa instância da família de sistemas sempre que o objetivo a elas associado estiver presente.

Para a responsabilidade "Filtragem baseada em conteúdo" temos as destrezas alternativas que representam as técnicas consideradas para a filtragem baseada em conteúdo, que são "TF-IDF" ("*Total Frequency-Inverse Document Frequency*"), "Produto interno" e "co-seno" (Figura 54)

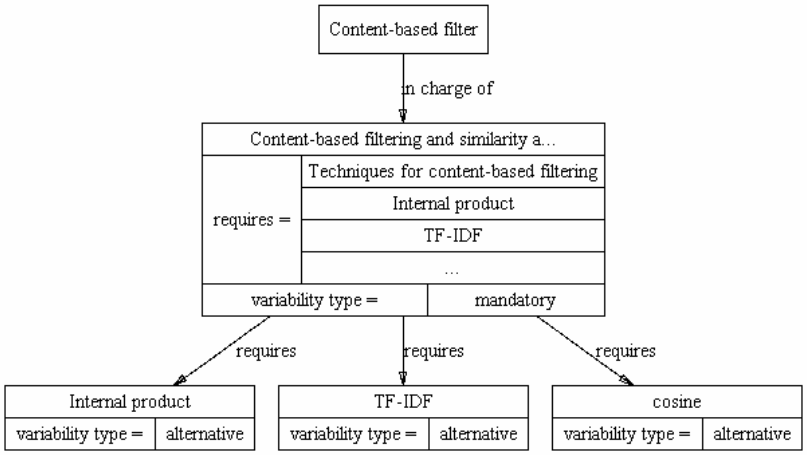

Figura 54 Destrezas alternativas da responsabilidade "Filtragem baseada em conteúdo"

Nas responsabilidades relativas à filtragem colaborativa temos as destrezas alternativas "Coeficiente de Pearson" e "co-seno" como técnicas para a classificação de vizinhanças necessárias à filtragem colaborativa, e as destrezas alternativas "Vizinhos próximos" e "*Clustering*" (agrupamento) como técnicas para a filtragem colaborativa propriamente dita. A Figura 55 apresenta esses conceitos variáveis do modelo.

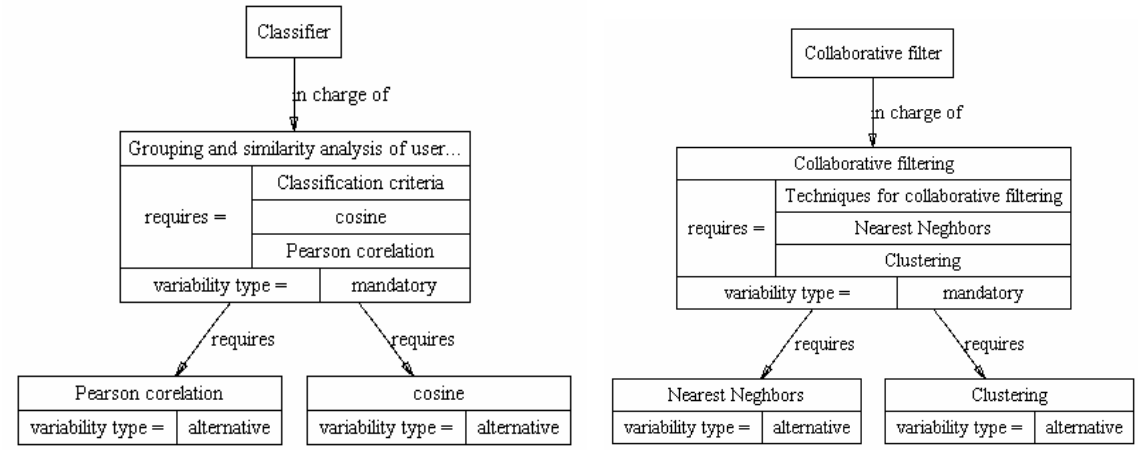

Figura 55 Destrezas alternativas nas responsabilidades relativas à filtragem colaborativa

Para a responsabilidade "filtragem híbrida" temos destrezas opcionais que são as técnicas que definem como os resultados da FBC e FC podem ser combinados para se obter a filtragem híbrida. Essas destrezas são opcionais porque podemos ter a combinação de mais de uma técnica na escolha dos resultados da filtragem híbrida (Figura 56).

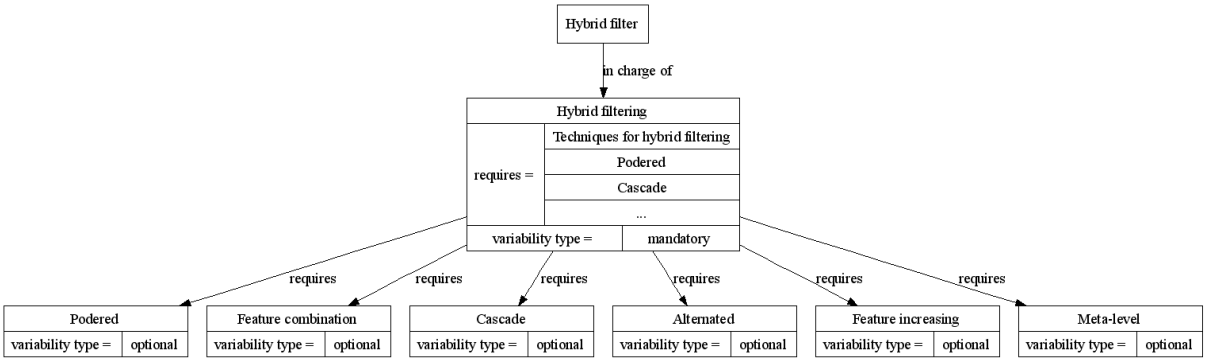

Figura 56 Destrezas opcionais da responsabilidade "Filtragem híbrida"

5.3 Especificação da LED

Uma vez que temos o modelo de domínio da recuperação e filtragem criado segundo a metodologia MADEM (APÊNDICE A), e a variabilidade dos conceitos do domínio descrita na seção anterior, podemos iniciar a especificação da LED segundo as fases da GENMADEM. Esta seção mostra como a metodologia e a ferramenta podem ser aplicadas para se obter a especificação da LED.

5.3.1 Especificação da Sintaxe

O primeiro passo da metodologia consiste na seleção dos termos variáveis do domínio que irão compor o vocabulário da LED. A seleção desses conceitos será descrita a seguir das duas formas possíveis: com a aplicação do algoritmo proposto (Figura 28) e logo em seguida com a utilização da ferramenta ONTOGENMADEM (plugin Protégé) para a obtenção automática do vocabulário. Segundo o algoritmo, procuramos os objetivos, papéis, atividades e destrezas opcionais ou alternativos e finalmente os papéis mandatórios com propriedades (destrezas) variáveis. As descrições a seguir apresentam a aplicação do algoritmo nessa mesma ordem, analisando o modelo de objetivos e cada um dos modelos de papéis criados.

Objetivos alternativos e opcionais: No modelo de objetivos (Figura 26) temos dois objetivos opcionais ("Satisfazer necessidades de informação pontuais através de técnicas de recuperação" e "Satisfazer necessidades de informação de longo prazo através de técnicas de filtragem") (Figura 48) para os quais devemos instanciar a classe "*Optional Goal*" e três objetivos alternativos ("Satisfazer necessidades de informação de longo prazo através de filtragem de conteúdo", "Satisfazer necessidades de informação de longo prazo através de filtragem colaborativa" e "Satisfazer necessidades de informação de longo prazo através de filtragem híbrida") (Figura 49) para os quais devemos instanciar a classe "*Alternative Goal*". A Figura 57 mostra o *instance browser* do Protégé após a instanciação dessas duas classes.

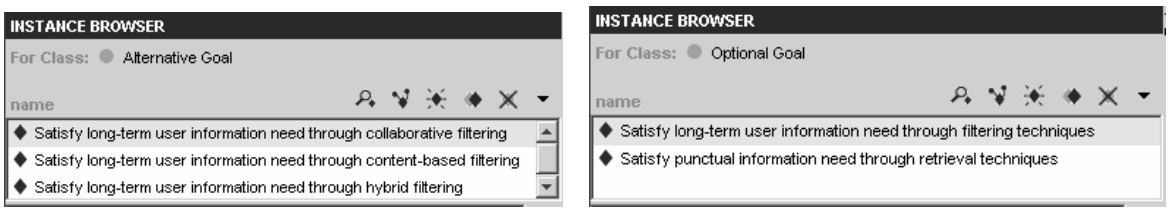

Figura 57 Instâncias de "Alternative Goal" e "Optional Goal"

Papéis alternativos e opcionais: No modelo de papéis relativo à recuperação da informação (APÊNDICE A, Figura 2A) não temos responsabilidades variáveis. No modelo de papéis relativo à modelagem do usuário (APÊNDICE A, Figura 3A) temos duas responsabilidades opcionais: "Aquisição explícita do perfil" e

“Aquisição implícita do perfil” (Figura 51). Devemos instanciar a classe “*Optional Role*” com os papéis associados a essas duas responsabilidades, que são “Interface de entrada do usuário da filtragem” e “Monitor do usuário”, respectivamente. A Figura 58 mostra essas instâncias criadas.

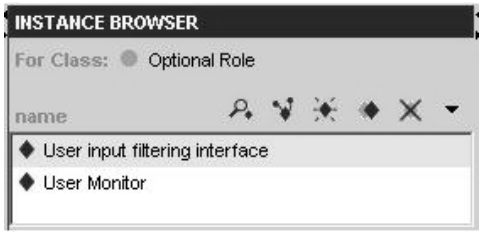

Figura 58 Instâncias de “Optional Role”

Atividades alternativas e opcionais: Não temos nenhuma atividade variável nos modelos de papéis criados.

Destrezas alternativas e opcionais: No modelo de papéis relativo à recuperação da informação (APÊNDICE A, Figura 2A) temos as destrezas alternativas das responsabilidades de representação da consulta e dos itens de informação, que são “Espaço vetorial”, “Linguagem natural” e “*document or term clustering*” (Figura 50). Devemos instanciar a classe “*Alternative skill*” para cada uma delas. No modelo de papéis relativo à modelagem do usuário (APÊNDICE A, Figura 3A) temos destrezas alternativas para a responsabilidade de aquisição implícita do perfil (“*Java Applet*” e “*Mosaic Browser*”) (Figura 52) e para a responsabilidade de construção e atualização do modelo do usuário (“*Kmeans clustering*”, “regras de associação”, “descoberta de padrões seqüenciais” e “classificação”) (Figura 53). Temos destrezas alternativas também para as responsabilidades da filtragem baseada em conteúdo e colaborativa (Figura 54 e Figura 55). Devemos instanciar a classe “*Alternative skill*” com essas destrezas. Finalmente, temos destrezas opcionais para a responsabilidade “Filtragem híbrida” (Figura 56), com as quais devemos instanciar a classe “*Optional Skill*”. A Figura 59 mostra essas instâncias criadas.

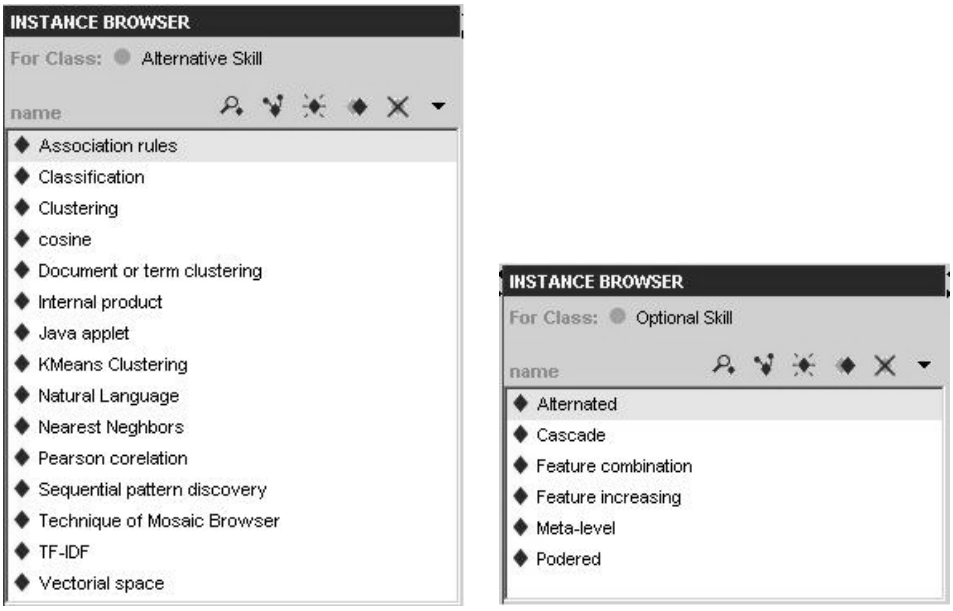

Figura 59 Instâncias de "Alternative Skill" e "Optional Skill"

Papéis mandatórios com propriedades variáveis: podemos encontrar responsabilidades mandatórias com propriedades variáveis (cujas destrezas encontramos no passo anterior) em todos os modelos de papéis. Os papéis associados a elas são: "Modelador de itens de informação", "Modelador da consulta" e "Recuperador" na Figura 50; "Modelador do usuário" na Figura 53 e os papéis "Filtrador baseado em conteúdo" (Figura 54), "Classificador" e "Filtrador colaborativo" (Figura 55), e "Filtrador híbrido" (Figura 56). A Figura 60 mostra essas instâncias criadas.

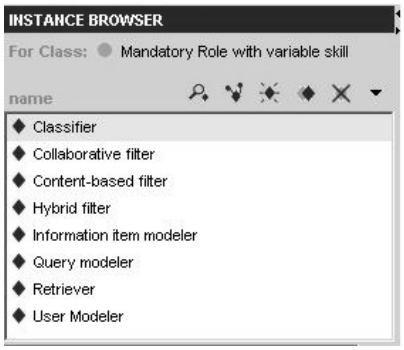

Figura 60 Instâncias de "Mandatory Role with variable skill"

Após a análise da variabilidade e definição do vocabulário podemos resumir os conceitos que não são variáveis e nem têm propriedades variáveis

(Tabela 20), e que portanto estarão presentes na implementação de qualquer elemento da família de sistemas; e os conceitos variáveis ou mandatórios com destrezas variáveis selecionados para compor o vocabulário da LED (Tabela 21), que estarão presentes ou não em um elemento da família de sistemas, dependendo da especificação feita com o uso da LED e das regras de configuração.

Tipo	**Conceito**	**Variabilidade**
Papel	Interface de saída do usuário da filtragem	Mandatório
Papel	Interface de entrada do usuário da recuperação	Mandatório
Papel	Interface de saída do usuário da recuperação	Mandatório
Destreza	Interface-suporte para a especificação da necessidade de informação pontual	Mandatória
Destreza	Interface-suporte para a especificação de necessidade de longo prazo	Mandatória
Destreza	Interface-suporte para a apresentação de itens de informação recuperados	Mandatória
Destreza	Interface-suporte para a apresentação de itens de informação filtrados	Mandatória

Tabela 20 Conceitos mandatórios do modelo de domínio da ONTOINFO

Os passos acima apesar de simples podem representar um grande trabalho, principalmente em um domínio com uma grande quantidade de conceitos. A ferramenta ONTOGENMADEM é de grande auxílio nesta fase, buscando os conceitos variáveis do modelo de domínio e instanciando automaticamente as classes do vocabulário, conforme apresentado a seguir.

A criação do vocabulário com a ferramenta ONTOGENMADEM é bem simples: com o projeto aberto e o *plugin* habilitado, pressiona-se o botão "*Generate DSL*". Uma confirmação é solicitada, pois a geração exclui as instâncias já existentes nas classes do vocabulário para evitar duplicidade. Uma vez confirmada, a geração ocorre de forma rápida e o *plugin* mostra os resultados numa área de texto destinada a mensagens e as instâncias criadas em duas tabelas, sendo uma para o vocabulário e outra para os componentes de implementação (Figura 61). Após a execução do plugin, podemos consultar as instâncias do vocabulário e veremos as mesmas instâncias criadas anteriormente (Figura 57 a Figura 60).

Tipo	Conceito	Variabilidade
Objetivo	Satisfazer necessidades de informação pontuais através de técnicas de recuperação	Opcional
Objetivo	Satisfazer necessidades de informação de longo prazo através de técnicas de filtragem	Opcional
Objetivo	Satisfazer necessidades de informação de longo prazo através de filtragem baseada em conteúdo	Alternativo
Objetivo	Satisfazer necessidades de informação de longo prazo através de filtragem colaborativa	Alternativo
Objetivo	Satisfazer necessidades de informação de longo prazo através de filtragem híbrida	Alternativo
Papel	Interface de entrada do usuário da filtragem	Opcional
Papel	Monitor do usuário	Opcional
Destreza	Espaço vetorial	Alternativa
Destreza	Linguagem natural	Alternativa
Destreza	*Document or term clustering*	Alternativa
Destreza	Java applet	Alternativa
Destreza	*Mosaic Browser*	Alternativa
Destreza	*Kmeans clustering*	Alternativa
Destreza	Regras de associação	Alternativa
Destreza	Descoberta de padrões seqüenciais	Alternativa
Destreza	Classificação	Alternativa
Destreza	TF-IDF	Alternativa
Destreza	Produto interno	Alternativa
Destreza	Co-seno	Alternativa
Destreza	Coeficiente de Pearson	Alternativa
Destreza	Vizinhos próximos	Alternativa
Destreza	Agrupamento	Alternativa
Destreza	Ponderada	Opcional
Destreza	Alternada	Opcional
Destreza	Cascata	Opcional
Destreza	Combinação de features	Opcional
Destreza	Aumento de features	Opcional
Destreza	Meta-level	Opcional
Papel	Modelador de itens de informação	Mandatório c/ destreza variável
Papel	Modelador da consulta	Mandatório c/ destreza variável
Papel	Recuperador	Mandatório c/ destreza variável
Papel	Modelador do usuário	Mandatório c/ destreza variável
Papel	Filtrador baseado em conteúdo	Mandatório c/ destreza variável
Papel	Classificador	Mandatório c/ destreza variável
Papel	Filtrador colaborativo	Mandatório c/ destreza variável
Papel	Filtrador híbrido	Mandatório c/ destreza variável

Tabela 21 Termos do vocabulário obtidos do modelo de domínio da ONTOINFO

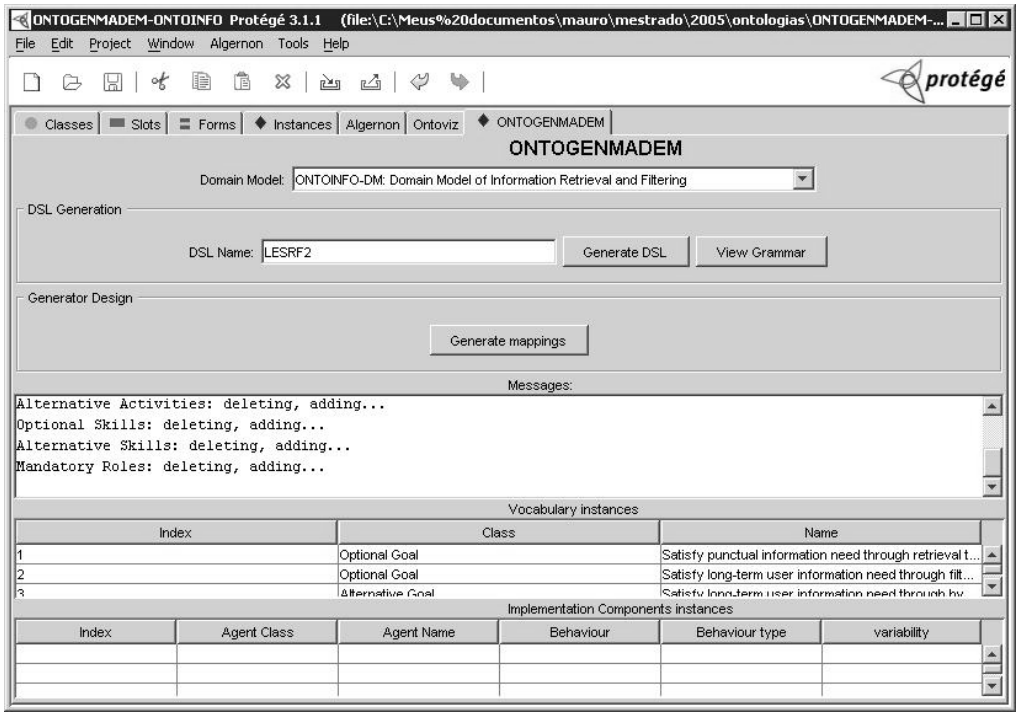

Figura 61 Geração do vocabulário com o *plugin* ONTOGENMADEM

A gramática é definida na forma de uma sintaxe abstrata definida pelos elementos que podem ocupar os slots da classe “DSL grammar”, conforme ilustrado na Figura 62. Essa sintaxe deixa claro que a LED será declarativa de modo a permitir que numa especificação sejam informados que caminhos da árvore de objetivos, papéis e atividades / destrezas foram escolhidos como decisões de projeto para gerar um elemento da família de sistemas. Essa sintaxe abstrata pode ser mapeada para representações diferentes na sintaxe concreta como a textual (Figura 63), visual, interface de seleção (Figura 64) ou outra representação. Podemos notar os objetivos no primeiro nível, os papéis no nível intermediário e as destrezas no nível inferior. As figuras que representam os conceitos na gramática são as mesmas usadas nos modelos do domínio, sendo que há tons de cinza diferentes para os mandatórios (cinza médio), alternativos (cinza escuro) e opcionais (cinza claro). Podemos notar também nessa sintaxe abstrata gerada que os papéis relativos à modelagem do usuário e obtenção do perfil de forma explícita foram colocados hierarquicamente como parâmetros diretos do objetivo principal da filtragem, já que os mesmos atendem a todos os tipos de filtragem.

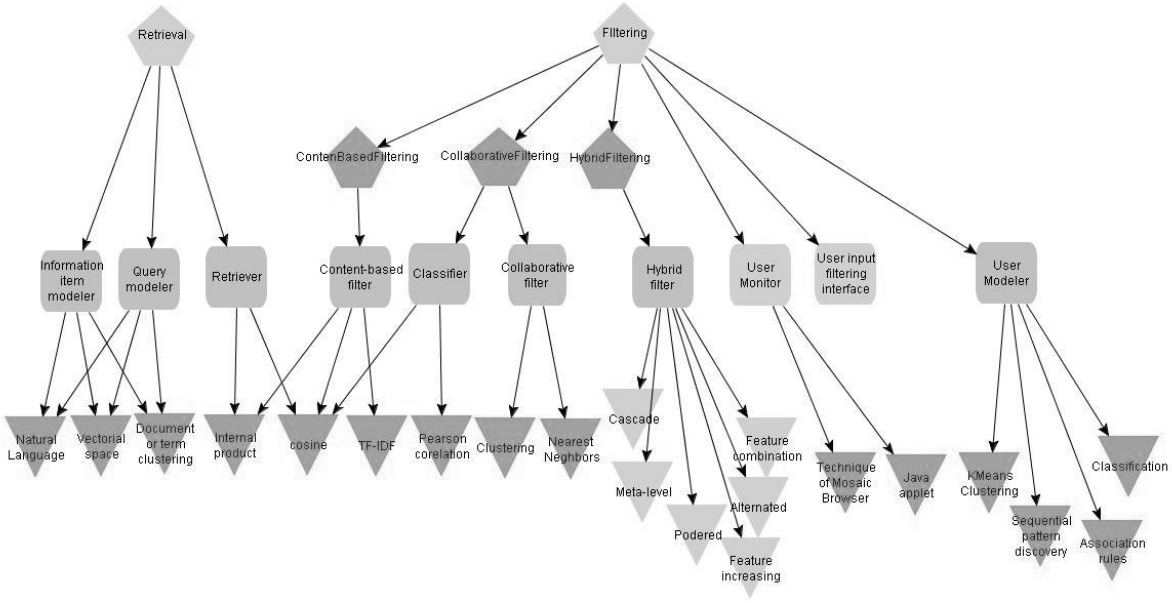

Figura 62 Instância da classe "DSL Grammar"

```
System::=Goal(GoalList)
GoalList::=Retrieval[RoleR],Filtering[SubF]
RoleR::=InformationModeler[modelingTech],QueryModeler[modelingTech],
  Retriever[retrieverTech]
SubF::=ContentBased[ContentFilter[contTech]] |
       CollaborativeFilter[colabTech] |
       HybridFilter[hybridTech]
modelingTech::=VectorialSpace | NaturalLanguage | DocumentTermClustering
retrieverTech::=InternalProduct|cosine
```

Figura 63 Fragmento de exemplo de representação textual para a sintaxe concreta

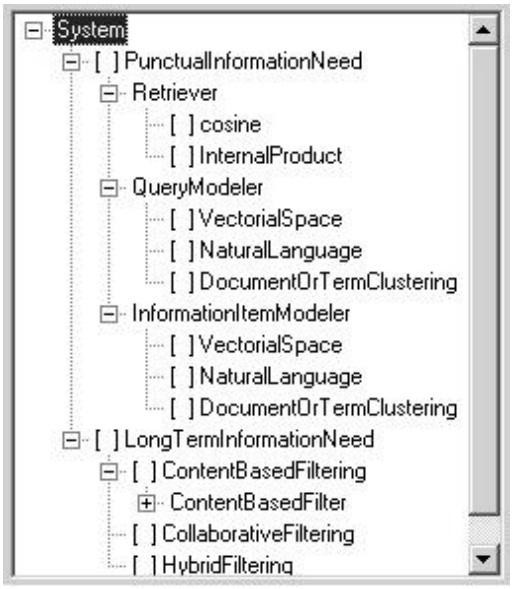

Figura 64 Exemplo de representação tipo interface de seleção para a sintaxe concreta

5.3.2 Especificação da pragmática

Na concepção atual da GENMADEM, as informações necessárias para especificar a pragmática da LED podem ser obtidas em parte no próprio modelo de domínio e em parte em documentações externas usadas na análise de domínio. A descrição do vocabulário, por exemplo, pode ser obtida nos slots “descrição” e “referências” que já existem para as classes dos conceitos dos quais derivam os termos do vocabulário (Figura 65). Um fragmento de uma possível documentação obtida neste estudo é mostrado na Figura 66.

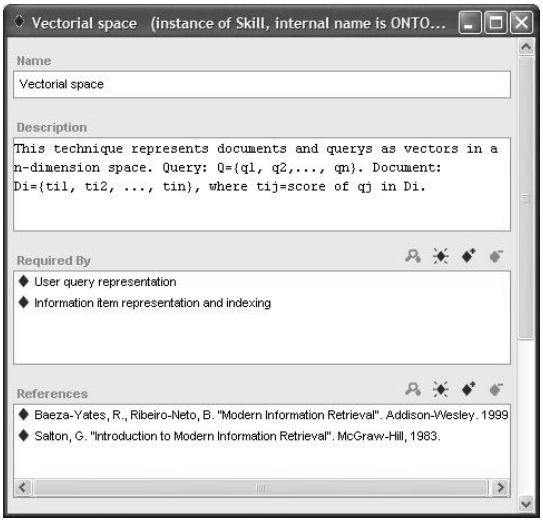

Figura 65 Exemplo de obtenção da descrição do vocabulário na ontologia

Descrição do domínio: O objetivo principal é prover mecanismos para que seja possível localizar e recuperar informações relevantes aos usuários da forma mais eficiente possível. A princípio, tem-se uma necessidade de informação que deve ser expressa em uma consulta ou perfil de usuário. Uma consulta caracteriza uma necessidade de informação pontual (...)

Descrição do vocabulário:
VetorialSpace (destreza alternativa): Esta técnica representa documentos e consultas como vetores num espaço n-dimensional. Consulta: Q={q1,q2,...,qn}. Documento: Di={ti1,ti2,...,tin}, onde tij=peso de qj em Di
(...)

Figura 66 Exemplo de documentação da LED

Nesta fase identificamos como ponto aberto para extensões futuras a recolocação dessa documentação externa em uma classe da ontologia, como era

definido na TOD-LED, de modo que a ferramenta ONTOGENMADEM possa gerar uma documentação (em HTML, por exemplo) a partir das descrições textuais contidas na própria ontologia.

5.4 Projeto do gerador

O projeto do gerador consiste em instanciar as classes "*Configuration Rules*", "*Implementation Components*" e "*JADE constructs*", subclasses de "*Generator Constructs*".

5.4.1 Regras de construção

Na Figura 49 temos três objetivos alternativos referentes aos três tipos de filtragem que os sistemas da família no domínio podem adotar. Porém, como a filtragem híbrida seleciona os resultados dos outros dois tipos de filtragem, sempre que a filtragem híbrida estiver presente na família de sistemas, a implementação dos outros dois tipos de filtragem também deverá ser incluída. Aqui temos duas regras de exigibilidade. Instanciamos então a classe "Exigibility" com essas regras.

5.4.2 Mapeamentos

Como foi explicado na seção 4.1.2.2.2, os mapeamentos referenciam tanto os conceitos variáveis como os comuns, visto que o gerador deverá conhecer a parte comum e variável para gerar os elementos da família de sistemas.

Para construir os mapeamentos, o primeiro passo é criar as instâncias das classes "JADE Behaviour" e "JADE Agent". A "JADE Behaviour" deve ser instanciada com todas as responsabilidades e destrezas relacionadas aos agentes, enquanto a "JADE Agent" deve ser instanciada com todos os agentes do modelo do framework multiagente, com seus respectivos comportamentos.

Para instanciar a classe "JADE Behaviour", os diagramas de papéis ou diagramas da sociedade multiagente (APÊNDICE A) devem ser consultados ou podemos usar os scripts Algernon para fazer a consulta ou usar o plugin ONTOGENMADEM para fazer toda a tarefa. A Tabela 22 relaciona as instâncias criadas para a classe "JADE Behaviour".

Agora que já temos a classe “JADE Behaviour” instanciada, instanciamos a classe “JADE Agents” com todos os agentes do modelo do framework multiagente, com seus respectivos comportamentos. O uso da ferramenta ONTOGENMAEM nesta tarefa é de grande utilidade. A mesma aplica um script Algernon similar ao da Figura 67 para obter todos os agentes com suas responsabilidades e destrezas e faz a devida instanciação da classe, ao pressionamento do botão “Generate mappings” na interface mostrada na Figura 61.

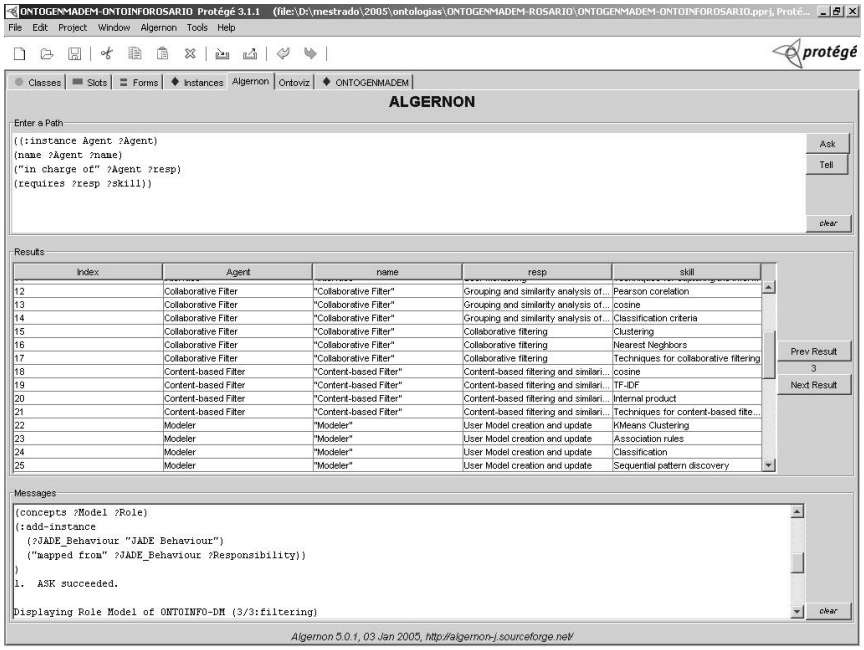

Figura 67 Consulta Algernon para instanciar a classe “JADE Agent”

A Tabela 23 relaciona os agentes instanciados na classe “JADE Agent”. Podemos notar claramente nesse mapeamento que alguns agentes assumem mais de uma responsabilidade com as respectivas destrezas associadas às mesmas, o que é normal acontecer na fase de projeto do domínio, durante a criação do modelo da sociedade multiagente (APÊNDICE A).

Instância	Tipo
Alterned	Destreza
Association Rules	Destreza
Cascade	Destreza
Classification	Destreza
Clustering	Destreza
Collaborative filtering	Responsabilidade
Content-based filtering and similarity analysis	Responsabilidade
Cosine	Destreza
Document or term clustering	Destreza
Explicit profile acquisition	Responsabilidade
Feature combination	Destreza
Feature increasing	Destreza
Filtered information item display	Responsabilidade
Grouping and similarity analysis of user models	Responsabilidade
Hybrid filtering	Responsabilidade
Implicit profile acquisition	Responsabilidade
Information item representing and indexing	Responsabilidade
Interface support for the display of filtered information items	Destreza
Interface support for the display of retrieved information items	Destreza
Interface support for the specification of long-term information need	Destreza
Interface support for the specification of punctual information need	Destreza
Internal product	Destreza
Java applet	Destreza
Kmeans clustering	Destreza
Matching and similarity analysis	Responsabilidade
Meta-level	Destreza
Natural language	Destreza
Nearest neighbors	Destreza
Pearson correlation	Destreza
Pondered	Destreza
Punctual information need specification	Responsabilidade
Retrieved information items display	Responsabilidade
Sequential pattern discovery	Destreza
Technique of mosaic browser	Destreza
TF-IDF	Destreza
User model creation and update	Responsabilidade
User monitoring	Responsabilidade
User query representation	Responsabilidade
Vectorial space	Destreza

Tabela 22 Instâncias da classe "JADE Behaviour"

Agente	Comportamento	Tipo
RetrievalInterface	Punctual information need specification	Responsabilidade
	Retrieved information items display	Responsabilidade
	Interface support for the display of retrieved information	Destreza
	Interface support for the specification of punctual information need	Destreza
Retriever	Matching and similarity analysis	Responsabilidade
	Cosine	Destreza
	Internal product	Destreza
IRConstructor	Information item representation and indexing	Responsabilidade
	User query representation	Responsabilidade
	Natural language	Destreza
	Vectorial space	Destreza
	Document or term clustering	Destreza
Monitor	Implicit profile acquisition	Responsabilidade
	Java applet	Destreza
	Technique of mosaic browser	Destreza
Modeler	User Model creation and update	Responsabilidade
	KMeans Clustering	Destreza
	Association rules	Destreza
	Classification	Destreza
	Sequential pattern discovery	Destreza
FilteringInterface	Explicit profile acquisition	Responsabilidade
	Filtered information items display	Responsabilidade
	Interface support for the display of filtered information items	Destreza
	Interface support for the specification of long-term information need	Destreza
ContentbasedFilter	Content-based filtering and similarity analysis	Responsabilidade
	Cosine	Destreza
	TF-IDF	Destreza
	Internal product	Destreza
CollaborativeFilter	Grouping and similarity analysis of user models	Responsabilidade
	Collaborative filtering	Responsabilidade
	Clustering	Destreza
	Nearest Neighbors	Destreza
	Pearson correlation	Destreza
	Cosine	Destreza
HybridFilter	Hibrid filtering	Responsabilidade
	Alterned	Destreza
	Meta-level	Destreza
	Feature Combination	Destreza
	Feature increasing	Destreza
	Cascade	Destreza
	Pondered	Destreza

Tabela 23 Instâncias da classe "JADE Agent"

O último passo da construção dos mapeamentos é a instanciação das classes “Mandatory agent and mandatory behaviour”, “Mandatory agent and variable behaviour” e “Variable agent”. Podemos recorrer à Tabela 23 e ao resultado da análise da variabilidade do modelo de domínio (Tabela 21) para instanciar essas classes ou usar a ferramenta ONTOGENMADEM que também automatiza essa tarefa. A TABELA resume os agentes obtidos para essas classes.

A Tabela 24 resume as instâncias criadas para a classe “Mandatory agent and madatory behaviour”, com seus respectivas comportamentos e variabilidade dos mesmos. O agente “Interface da filtragem” (“*FilteringInterface*”), por ter uma responsabilidade que é opcional e outra que é mandatória, deve ser mandatório.

Agente	**Comportamento**	**Tipo do comportamento**	**Variabilidade**
RetrievalInterface	Punctual information need specification	Responsabilidade	Mandatória
	Retrieved information items display	Responsabilidade	Mandatória
	Interface support for the specification of puntual information need	Destreza	Mandatória
	Interface support for the display of retrieved information	Destreza	Mandatória
FilteringInterface	Explicit profile acquisition	Responsabilidade	Opcional
	Filtered information items display	Responsabilidade	Mandatória
	Interface support for the display of filtered information items	Destreza	Mandatória
	Interface support for the specification of long-term information need	Destreza	mandatória

Tabela 24 Instâncias da classe “Mandatory agent and mandatory behaviour”

A Tabela 25 e a Tabela 26 resumem as instâncias criadas para as classes “Variable agent” e “Mandatory agent and variable behaviour”

Agente	**Comportamento**	**Tipo do comportamento**	**Variabilidade**
Monitor	Implicit profile acquisition	Responsabilidade	Opcional
	Java applet	Destreza	Alternativa
	Technique of mosaic browser	Destreza	Alternativa

Tabela 25 Instâncias da classe “Variable agent”

Agente	Comportamento	Tipo do comportamento	Variabilidade
Retriever	Matching and similarity analysis	Responsabilidade	Mandatória
	Cosine	Destreza	Alternativa
	Internal product	Destreza	Alternativa
IRConstructor	Information item representation and indexing	Responsabilidade	Mandatória
	User query representation	Responsabilidade	Mandatória
	Natural language	Destreza variável	Alternativa
	Vectorial space	Destreza variável	Alternativa
	Document or term clustering	Destreza variável	Alternativa
ContentbasedFilter	Content-based filtering and similarity analysis	Responsabilidade	Mandatória
	Cosine	Destreza	Alternativa
	TF-IDF	Destreza	Alternativa
	Internal product	Destreza	Alternativa
CollaborativeFilter	Grouping and similarity analysis of user models	Responsabilidade	Mandatória
	Collaborative filtering	Responsabilidade	Mandatória
	Clustering	Destreza	Alternativa
	Nearest Neghbors	Destreza	Alternativa
	Pearson corelation	Destreza	Alternativa
	Cosine	Destreza	Alternativa
HybridFilter	Hibrid filtering	Responsabilidade	Mandatória
	Alterned	Destreza	Opcional
	Meta-level	Destreza	Opcional
	Feature Combination	Destreza	Opcional
	Feature increasing	Destreza	Opcional
	Cascade	Destreza	Opcional
	Pondered	Destreza	Opcional
Modeler	User Model creation and update	Responsabilidade	Mandatória
	KMeans Clustering	Destreza	Alternativa
	Association rules	Destreza	Alternativa
	Classification	Destreza	Alternativa
	Sequential pattern discovery	Destreza	Alternativa

Tabela 26 Instâncias da classe "Mandatory agent and variable behaviour"

5.5 Considerações finais

Este capítulo apresentou um estudo de caso visando avaliar e demonstrar o uso da GENMADEM e da ferramenta ONTOGENMADEM na especificação de uma LED e projeto de um gerador para o domínio da recuperação e filtragem da informação. Esse estudo fez uso do modelo de domínio e de framework contidos na ONTOINFO estendida, que consta no APÊNDICE A e gerou como resultado a especificação da Linguagem para Especificação de Sistemas de Recuperação e Filtragem versão 2 (LESRF2).

Com essa avaliação notou-se que a GENMADEM facilitou bastante a especificação da LED e projeto do gerador, agilizando a obtenção dos conceitos variáveis do domínio e definição dos mapeamentos, principalmente com a aplicação da ferramenta ONTOGENMADEM.

6. CONCLUSÕES

Neste trabalho foi proposta a GENMADEM, uma metodologia para a Engenharia de Domínio Multiagente gerativa, que integra técnicas que abordam desde a modelagem do domínio até o desenvolvimento de LEDs e projeto de geradores a partir de modelos de domínio multiagente e modelos de framework multiagente baseados em ontologias.

Integra essa proposta a ferramenta ONTOGENMADEM, que é composta por uma ontologia que representa o conhecimento da metodologia e um plugin para o editor de ontologias Protégé que guia a aplicação da GENMADEM, tornando o conjunto uma ferramenta para a análise, projeto e implementação de LEDs.

A GENMADEM contribui com a extensão e integração de uma metodologia (MADEM) e uma técnica (TOD-LED) já existentes, e uma ferramenta que auxilia no desenvolvimento de LED, além de alguns esforços iniciais na direção da concepção de um gerador para o ambiente.

O uso de ontologias nesse processo demonstrou ser uma solução muito aceitável, visto que permite representar adequadamente os conceitos e produtos da Engenharia de Domínio além de permitir realizar consultas e inferências de forma rápida e simplificada, com o uso de ferramentas que estão disponíveis com código aberto e em evolução contínua.

6.1 Resultados e contribuições da pesquisa

As principais contribuições desta pesquisa foram:

- Análise de técnicas e ferramentas para construção de LEDs e geradores de aplicação (capítulo 3);
- Elaboração da metodologia GENMADEM (seção 4.1) para o reuso gerativo na Engenharia de Domínio Multiagente;
- Extensão da metodologia MADEM com melhorias no tratamento da variabilidade para uma melhor adequação à abordagem gerativa;
- Extensão da técnica TOD-LED para desenvolvimento de LEDs e integração da mesma à metodologia MADEM;

- Implementação de uma ferramenta (o *plugin* ONTOGENMADEM) que, aliado à ontologia ONTOGENMADEM, dá suporte à análise, projeto e implementação de LEDs (seção 4.2.3);
- Extensão do modelo de domínio para a recuperação e filtragem de informação ONTOINFO (APÊNDICE A);

6.2 Trabalhos futuros

A partir dos resultados obtidos neste trabalho podemos identificar algumas limitações e pontos ainda não contemplados no escopo deste trabalho. Não foi possível atingir o estágio de geração de código com a ferramenta proposta; os detalhes de implementação precisam ainda de uma maior validação e amadurecimento para se chegar a uma abordagem de geração de código; o conceito de *atividade* do modelo de domínio não foi contemplado nos mapeamentos, devido à ausência do mesmo no modelo de domínio criado. Esses e outros pontos podem ser superados com uma melhor validação das soluções encontradas numa revisão e aprofundamento do estudo de caso feito.

Com base nesses resultados e limitações, identificamos alguns itens que constituem itens abertos para trabalhos futuros:

- Avaliação da metodologia em outros domínios de aplicação;
- Inclusão das restrições e regras de dependência, que foram concebidas como regras de construção na proposta atual, como conceitos do modelo de domínio na fase de análise;
- Extensão da ferramenta ONTOGENMADEM para gerar a documentação da LED;
- Construção de um ambiente de uso dos produtos com a implementação de um gerador integrado à ferramenta ONTOGENMADEM;
- Inclusão na ONTOGENMADEM dos mapeamentos necessários à comunicação entre agentes;
- Integração dos plugins ONTOGENMADEM e ONTOCADE [ALVES, 2005] em um ambiente de auxílio às tarefas de análise, projeto e implementação do domínio.

REFERÊNCIAS

AHO, Alfred V., RAVI, S., ULLMAN, Jeffrey, D. **Compiladores: princípios, técnicas e ferramentas**. Rio de Janeiro: LTC, 1995.

ALGERNON. Rule-Based Programming. Disponível em: <http://algernon-j.sourceforge.net/docs/algernon-protege.html>. Acesso em: 07 nov. 2005.

ALVES, José Henrique. **ONTOCADE: Um Ambiente CASE baseado em Ontologias para Análise e Projeto na Engenharia de Domínio Multiagente**. Dissertação (Mestrado em Engenharia de Eletricidade), Universidade Federal do Maranhão, 2005.

BAEZA-YATES, R., RIBEIRO-NETO, B. **Modern Information Retrieval.** New York: ACM Press Series/Addison Wesley, 1999.

BATORY, D. et al. **GENESIS: An Extensible Database Management System**. In IEEE Transactions on Software Engineering, vol. 14, no. 11, 1711-1730, nov, 1988.

BATORY, D., LOFASO, B., SMARAGDAKIS, Y. **JTS: Tools for Implementing Domain-Specific Languages**. 5th International Conference on Software Reuse, Victoria, Canada, jun. 1998.

BATORY, D., O'MALLEY, S. **The Design and Implementation of Hierarchical Software Systems with Reusable Components**. In ACM Transactions on Software Engineering and Methodology, vol. 1, no. 4, pp. 355-398, 1992.

CLEAVELAND, J. C. **Building Application Generators**. IEEE Software 5, pp 25-33, jul. 1988.

COMPOSE PROJECT. Overview of the Compose Project: Domain-Specific Languages. Disponível em <http://compose.labri.fr/overview/overview.en.php3#dsl>. Acesso em: jun. 2004.

CONSEL, C., DANVY, O. **Tutorial Notes on Partial Evaluation**. In ACM Symposium on Principles of Programming Languages, pp 493-501, 1993.

CONSEL, C., MARLET, R. **Architecturing software using a methodology for language development.** In Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming, volume 1490 of Lecture Notes in Computer Science, Pisa, Italy, pp 170-194, sep. 1998.

CZARNECKI K., EISENECKER, U. **Components and Generative Programming**, in Proceedings of the Joint European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Springer-Verlag LNCS 1687, pp. 2-19, 1999.

CZARNECKI, K. **Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models**. Ph.D. thesis, Technische Universität Ilmenau, Germany, 1998.

CZARNECKI, K. **Overview of Generative Software Development**. In Unconventional Programming Paradigms (UPP), Mont Saint-Michel, France, LNCS 3566, pp. 313–328, 2004.

DELTA SOFTWARE. Generative Programming: From Theory to Practice. Disponível em: <http://www.d-s-t-g.com/neu/media/pdf/Facts_e/gp_hintergrnd.pdf>. Acesso em: mai. 2005.

DEURSEN, A., KLINT, P., VISSER, J. **Domain-specific languages: An annotated bibliografy**, ACM SIGPLAN Notices, 35(6): 26-36, jun. 2000.

FARIA, Carla Gomes de. **Uma Técnica para a Aquisição e Construção de Modelos de Domínio e Usuário Baseados em Ontologias para a Engenharia de Domínio Multiagente**, Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, 2004.

FERREIRA, Stefferson Lima Costa. **Uma Técnica e uma Ferramenta para o Projeto de Domínio Global e Detalhado de Sistemas Multiagente**. Dissertação do curso de Pós-Graduação em Engenharia de Eletricidade, na área de Ciência da Computação. Universidade Federal do Maranhão, 2004.

GESEC. Grupo de pesquisa em Engenharia de Software e Engenharia do Conhecimento. Disponível em: <http://maae.deinf.ufma.br/orientacoes/index.php>. Acesso em: 10 de mar. 2005.

GIRARDI, Rosario, BALBY, Leandro e OLIVEIRA, Ismênia. **A System of Agent-based Patterns for User Modeling based on Usage Mining.** Interacting with Computers, v. 17, n.5, Ed. Elsevier, pp. 567-591, sep. 2005.

GIRARDI, Rosario, LINDOSO , Alisson. **An Ontology-based Methodology for Multi-agent Domain Engineering.** In: 3rd Workshop on Multi-agent Systems: Theory and Applications (MASTA 2005) at 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Ed. IEEE. Covilhã, Portugal, dec. 2005.

GIRARDI, Rosario, LINDOSO, Alisson. **An Ontology-based Knowledge Base for the Representation and Reuse of Software Patterns.** ACM SIGSOFT Software Engineering Notes, Volume 31, Issue 1, Ed. ACM, pp. 1-6. New York. Janeiro, 2006.

GIRARDI, Rosario. **Agent-based Application Engineering.** In: 3rd Internacional Conference on Enterprise Information Systems (ICEIS 2001), Setúbal, Portugal, 2001.

GIRARDI, Rosario. **Engenharia de Software baseada em Agentes.** In: Anais do IV Congresso Brasileiro de Ciência da Computação, Itajaí, Santa Catarina, Brasil, 2004.

GIRARDI, Rosario. **Main Approaches to Software Classification and Retrieval**, En las actas del curso Ingeniería del Software y reutilización: Aspectos Dinámicos y Generación Automática. Editores J. L. Barros y A. Domínguez. (Universidad de Vigo), 1998.

GIRARDI, Rosario. **Software Architectures to Improve the Effectiveness of Reuse Techniques.** In: 8th Workshop on Software Reuse (WISR8), 1997.

GRUBER, T. R. **Towards Principles for the Design of Ontologies used for Knowledge Sharing**. International Journal Human-Computer Studies. Nº 43, pp. 907-928, 1995.

HERRINGTON, Jack. **Code Generation in Action.** Manning, Greenwich, CT, 2003.

HUTCHINSON, N. et al. **RPC in the x-Kernel: Evaluating New Design Technique.** In Proceedings of the Symposium on Operating System Principles, 91-101, dec. 1989.

JADE. Java Agent Development Framework. Disponível em: <http://jade.tilab.com/>. Acesso em: 03 fev. 2006.

KANG, K., COHEN, S., HESS, J., NOWAK, W. and PETERSON, S. **Feature-Oriented Domain Analysis (FODA) Feasibility Study.** Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

KRUEGER, C. W. **Software Reuse.** ACM Computing Surveys 24, 134-183. jun. 1992.

LEE, K., KANG, K., LEE, J. **Concepts and Guidelines of Feature Modeling for Product Line Software Engineering.** Proc. 7 Int. Conf. Software Reuse, pp. 62-77, 2002.

LIM, W. C. **Managing Software Reuse.** Prentice Hall PTR, 1998.

LUKER, P. A. **Program Generators and Generation Software.** The Computer Journal, No. 29 (4), pp. 315-321, 1986.

MERNIK, M., HEERING, J., SLOANE, A. **When and How to Develop Domain-Specific Languages.** Stichting Centrum voor Wiskunde en Informatica, 2003.

PARNAS, D. L. **On the design and development of program families.** IEEE Transactions on Software Engineering, SE-2(1):1 - 9, mar. 1976.

PFAHLER, P., KASTENS , U. **Language Design and Implementation by Selection**. Proc. 1st ACM-SIGPLAN Workshop on Domain-Specific-Languages, DSL '97, Paris, France, 1997.

PROTÉGÉ Project. Disponível em: http://protege.stanford.edu. Acesso em: 02 mai. 2005.

RUSSELL, Stuart, NORVIG, Peter. **Inteligência Artificial**. 2. ed. São Paulo: Campus, 2004.

SALTON, G. McGILL, M. **An Introduction to Modern Information Retrieval**, New York: McGraw-Hill, 1983.

SCHMIDT, D. A. **Denotational Semantics: A Methodology for Language Development,** Allyn and Bacon, Inc., Newton, MA, 1986.

SERRA, I., GIRARDI, R., ALVES, J. **Um Modelo de Domínio baseado em Ontologias para a Recuperação e Filtragem de Informação.** Anais do IV Congresso Brasileiro de Computação (CBCOMP), Ed. UNIVALI, pp. 124-129. Itajaí, Santa Catarina, Brasil, out. 2004.

SERRA, Ivo. **Uma Técnica para o Desenvolvimento de Linguagens Específicas de Domínio**. Dissertação do curso de Pós-Graduação em Engenharia de Eletricidade, na área de Ciência da Computação. Universidade Federal do Maranhão, 2004.

SIMOS, M., ANTHONY, J. **Weaving the model web: A multi-modeling approach to concepts and features in domain engineering.** In Proceedings of the Fifth International Conference on Software Reuse, pages 94–102. IEEE Computer Society, 1998.

SLONNEGER, K., KURTZ, B. **Formal Syntax and Semantics of Programming Languages**: A laboratory based approach. Addison-Wesley, 1995.

Smith, E., Medin, D. **Categories and concepts.** Harvard University Press, Cambridge, Massachusetts, 1981.

SOFTWARE PRODUCT LINES. Disponível em: <http://www.softwareproductlines.com. Acesso em: 29 mar. 2006.

SPINELLIS, D. **Notable design patterns for domain specific languages**. Journal of Systems and Software, 56(1):91–99, Feb. 2001.

STRATEGO. Program Transformation Language. Disponível em www.stratego-language.org. Acesso em: 16 abr. 2006.

TAYLOR, R., TRACZ, W., COGLIANESE, L.**Software development using domain-specific software architectures.** ACM SIGSOFT Software Engineering Notes, 20(5):27–37, 1995.

THIBAULT, S. **Domain-Specific Languages: Conception, Implementation and Application**, PhD thesis, IRISA/University of Rennes, 1998.

THIBAULT, S., MARLET, R., Consel, C. **A domain-specific language for video device drivers: from design to implementation**, In Conference on Domain Specific Languages, p. 11-26, Santa Barbara, Usenix, 1997.

WIDEN, T. **Formal Language Design in the Context of Domain Engineering.** Msc thesis, Oregon Graduate Institute of Science & Technology, 1995.

WIDEN, T., HOOK, J. **Software design automation: Language design in the context of domain engineering,** In the 10th International Conference on Software Engineering & Knowledge Engineering (SEKE'98), pages 308-317, San Francisco Bay, California, Jun. 1998.

WIESS, David. **Software Synthesis: The FAST process,** In: Proceedings of the Internacional Conference on Computing in High Energy Physics (CHEP), Sep. 1995.

APÊNDICE A – O MODELO DE DOMÍNIO E DE FRAMEWORK ONTOINFO ESTENDIDO

ONTOINFO é um modelo de domínio e de framework multiagente baseado em ontologias para a recuperação e filtragem de informação, construído segundo a metodologia MADEM. Esse modelo de domínio é composto pelos seguintes produtos, que serão apresentados nos próximos itens:

- Modelo de domínio
  - Modelo de objetivos
  - Modelo de Papéis
  - Modelo de interações entre papéis
- Modelo do framework multiagente
  - Modelo da sociedade multiagente
  - Diagramas de interações entre agentes

Modelo de objetivos

O modelo de objetivos (Figura 1A) visa apresentar o objetivo geral e os objetivos específicos da família de sistemas. No domínio da recuperação e filtragem, o objetivo geral é “satisfazer necessidades de informações e usuários”. Os objetivos específicos imediatos são “satisfazer necessidades pontuais por técnicas de recuperação” e “satisfazer necessidades de longo prazo por técnicas de filtragem”. Este último objetivo se subdivide em três sub-objetivos específicos referentes a cada uma das técnicas de filtragem “baseada no conteúdo”, “colaborativa” e “híbrida”.

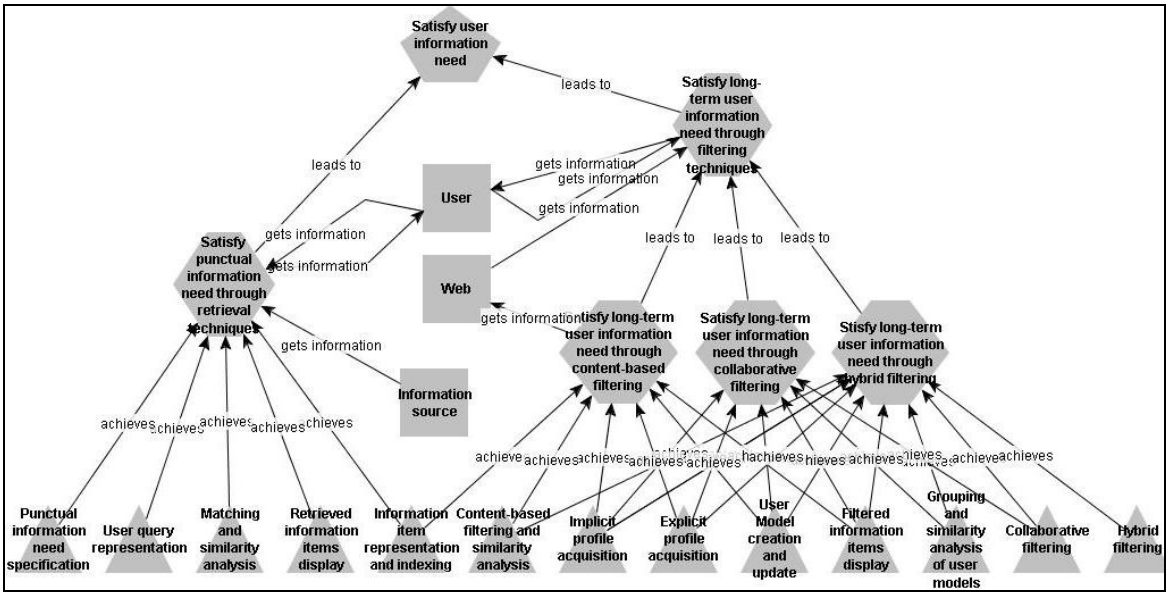

Figura 1A Modelo de objetivos da ONTOINFO

Modelo de papéis

O modelo de papéis é composto por um ou mais diagramas de papéis que mostram que condições, insumos (conhecimento), destrezas atividades são necessários para a realização de cada responsabilidade do diagrama de objetivos, bem como os produtos gerados e a interação de entidades externas com o processo. Inicialmente, cada papel assume uma responsabilidade, que serão apresentados a seguir.

O primeiro diagrama do modelo de papéis do modelo de domínio da ONTOINFO (Figura 2A) refere-se às responsabilidades associadas à recuperação de informação. Nesse modelo, temos cinco responsabilidades: “Especificação da necessidade pontual de informação” trata de receber a entrada da consulta pontual do usuário; “Representação da consulta do usuário” corresponde à criação de uma representação interna para a consulta do usuário; “Representação e indexação dos itens de informação” corresponde à criação da representação interna do e posterior indexação (na forma de índice invertido) dos itens de informação; “Comparação e análise de similaridade” trata da comparação da representação interna da consulta e itens de informação e “Apresentação dos itens de informação recuperados” trata da entrega dos resultados ao usuário.

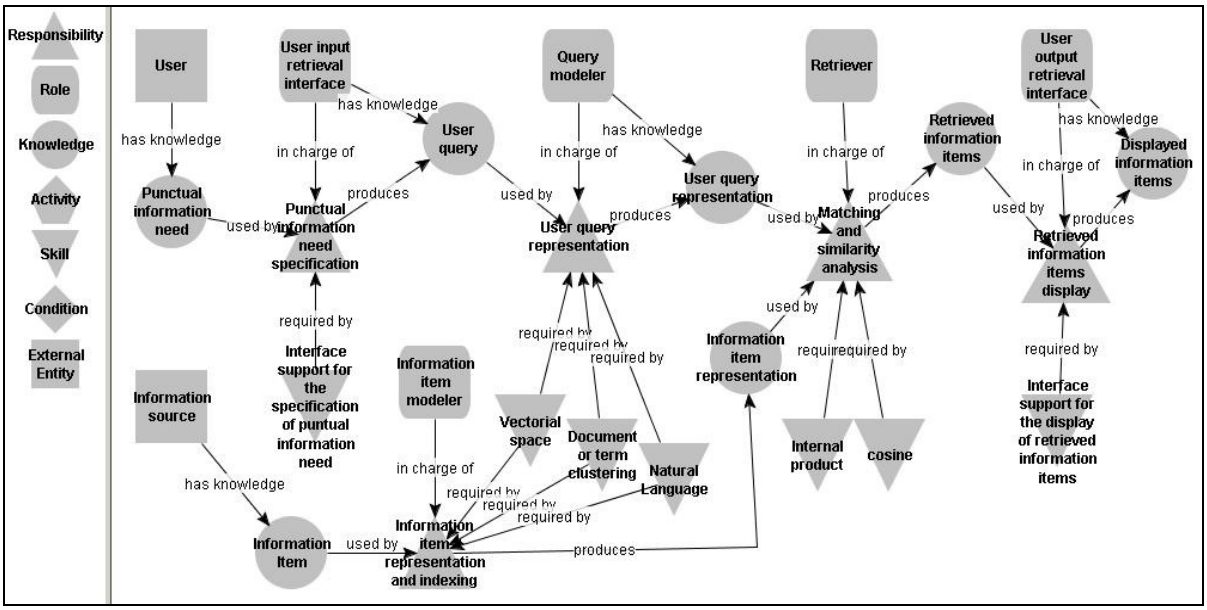

Figura 2A Modelo de papéis da ONTOINFO relativo à recuperação de informação

O segundo diagrama do modelo de papéis da ONTOINFO (Figura 3A) trata das responsabilidades associadas à modelagem do usuário, ou seja, à obtenção do perfil do usuário, de forma explícita ou implícita a partir da observação do comportamento do usuário durante o acesso à internet. Nesse modelo de papéis temos três responsabilidades: “Aquisição explícita do perfil”, “Aquisição implícita do perfil” e “criação e atualização do modelo do usuário”.

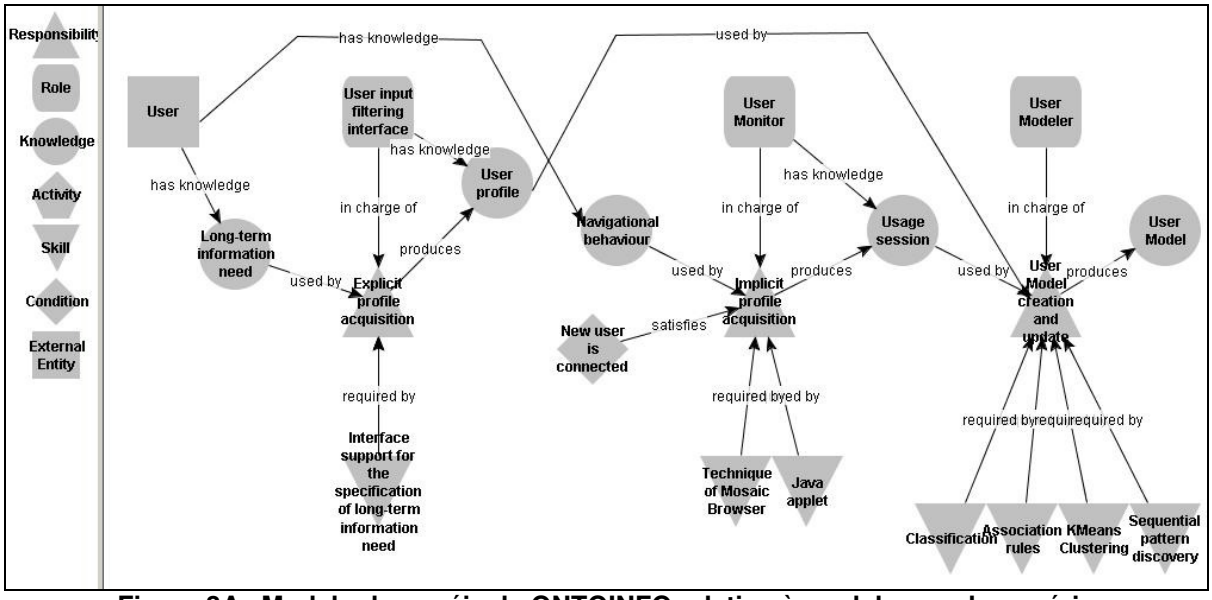

Figura 3A Modelo de papéis da ONTOINFO relativo à modelagem do usuário

O terceiro diagrama do modelo de papéis da ONTOINFO (Figura 4A) refere-se à filtragem de informação, atendendo aos objetivos específicos que tratam da filtragem baseada em conteúdo, filtragem colaborativa e filtragem híbrida. Nesse modelo de papéis temos cinco responsabilidades, onde quatro são relativas às modalidades de filtragem de informação existentes nos sistemas do domínio e outra que trata da entrega das informações filtradas. As responsabilidades que tratam da filtragem têm como entrada o modelo do usuário obtido na modelagem de usuário (conhecimento do papel “Modelador do usuário”).

A responsabilidade “Filtragem baseada em conteúdo e análise de similaridade” refere-se à filtragem baseada em conteúdo. As responsabilidades “Agrupamento e análise de similaridade de modelos de usuários” e “Filtragem colaborativa” referem-se à filtragem colaborativa. A primeira trata da análise de similaridade entre modelos de usuários, visando classificá-los em grupos de perfis

similares. Uma vez conhecido os grupos com perfis similares, a responsabilidade “Filtragem colaborativa” irá comparar os perfis de outros usuários do grupo com a representação dos itens de informação, visando sugerir novos itens ao usuário com base nessa similaridade entre perfis.

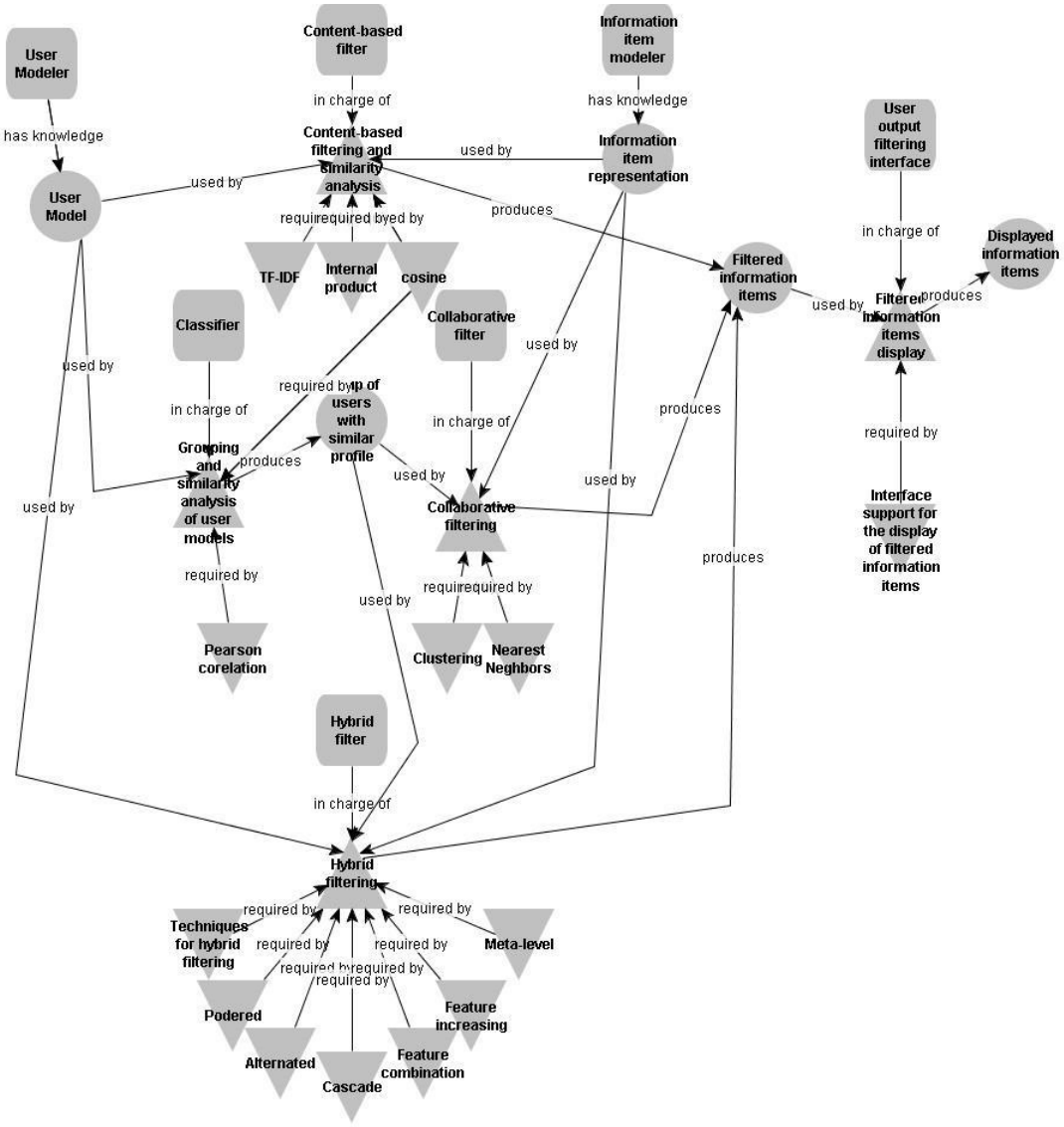

Figura 4A Modelo de papéis da ONTOINFO relativo à filtragem de informação

A responsabilidade “Filtragem híbrida” trata da filtragem híbrida, que é uma forma de filtragem que une características da filtragem baseada em conteúdo e da filtragem colaborativa. Existem modos distintos de se aproveitar os recursos de cada uma das filtragens na filtragem híbrida, e esses modos serão as destrezas da responsabilidade “Filtragem híbrida”, que serão opcionais entre si, permitindo então

a presença de mais de uma destreza (ou mais de um modo de filtragem híbrida) numa mesma instância da família de sistemas. Essas destrezas são “o modo ponderado”, “modo alternado”, “modo cascata”, “combinação de features”, “aumento de features” e “meta-level”. O modo como essa filtragem híbrida aproveitará os recursos de cada uma das filtragens (baseada em conteúdo e colaborativa) depende da técnica a ser usada.

A responsabilidade “Apresentação dos itens de informação filtrados” trata da entrega dos resultados (itens de informação filtrados) ao usuário.

Modelo de interação entre papéis

Este modelo mostra as interações que ocorrem entre papéis e entre papéis e entidades externas. Foi construído um modelo de interações para cada objetivo específico da ONTOINFO, conforme descrito na metodologia MADEM. Nos diagramas a seguir as entidades externas são representadas por quadrados e os papéis por quadrados com cantos arredondados e as interações são setas complementadas por um método (que é uma responsabilidade associada ao papel) e um parâmetro (que é o item recebido ou produzido pelo papel, e aparece entre parênteses).

O primeiro modelo de interação é o dos papéis relativos ao objetivo “Satisfazer necessidades pontuais de informação através de técnicas de recuperação” (Figura 5A). Inicia-se com a representação e indexação dos itens de informação pelo papel “Modelador de itens de informação”, que interage com a entidade externa “Fonte de informação”. Em seguida, a entidade externa “Usuário” expressa sua necessidade de informação pontual para o papel “Interface de entrada da recuperação”. Então o papel “Modelador da consulta” recebe a consulta do usuário para criar a representação interna da mesma e entregar ao papel “Recuperador”, que faz a comparação e análise de similaridade entre a representação da consulta e dos itens de informação criadas pelo papel “modelador de itens de informação”, produzindo então os itens de informação recuperados que serão entregues ao papel “Interface de saída da filtragem”, para serem apresentados ao usuário.

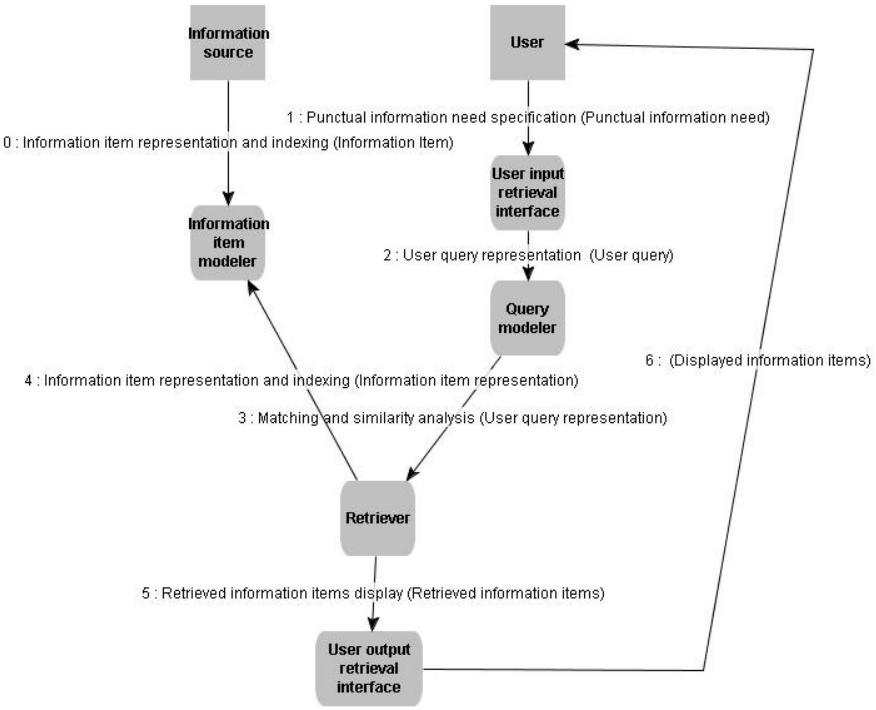

Figura 5A Modelo de interações de "Satisfazer necessidades pontuais através de técnicas de recuperação"

O modelo de interação entre os papéis relativos ao objetivo "Satisfazer necessidades de informação de longo prazo através de filtragem baseada em conteúdo" (Figura 6A) também inicia com o papel "Modelador de itens de informação" a fazer a representação e indexação dos itens de informação obtidos da entidade externa "Fonte de informação". Em seguida temos duas interações que podem ocorrer simultaneamente: a aquisição do perfil do usuário de forma implícita ou explícita, a partir da entidade externa "Usuário". Dessas interações participam os papéis "Monitor do usuário" e "Interface de entrada da filtragem", respectivamente. Desses dois papéis partem duas interações com o papel "Modelador do usuário", que também podem ocorrer independentemente de seqüência e tratam da criação do modelo do usuário a partir da sessão de uso do usuário (forma implícita) ou do perfil fornecido pelo usuário de forma explícita. Esse modelo do usuário é encaminhado ao papel "Filtrador baseado em conteúdo", que o compara com a representação dos itens de informação recebida do papel "Modelador de itens de informação" e então encaminha os itens de informação filtrados ao papel "Interface de saída da filtragem", que então os apresenta ao usuário.

O modelo de interações entre os papéis relativos ao objetivo "Satisfazer necessidades de informação de longo prazo através de filtragem colaborativa" é

apresentado na Figura 7A.

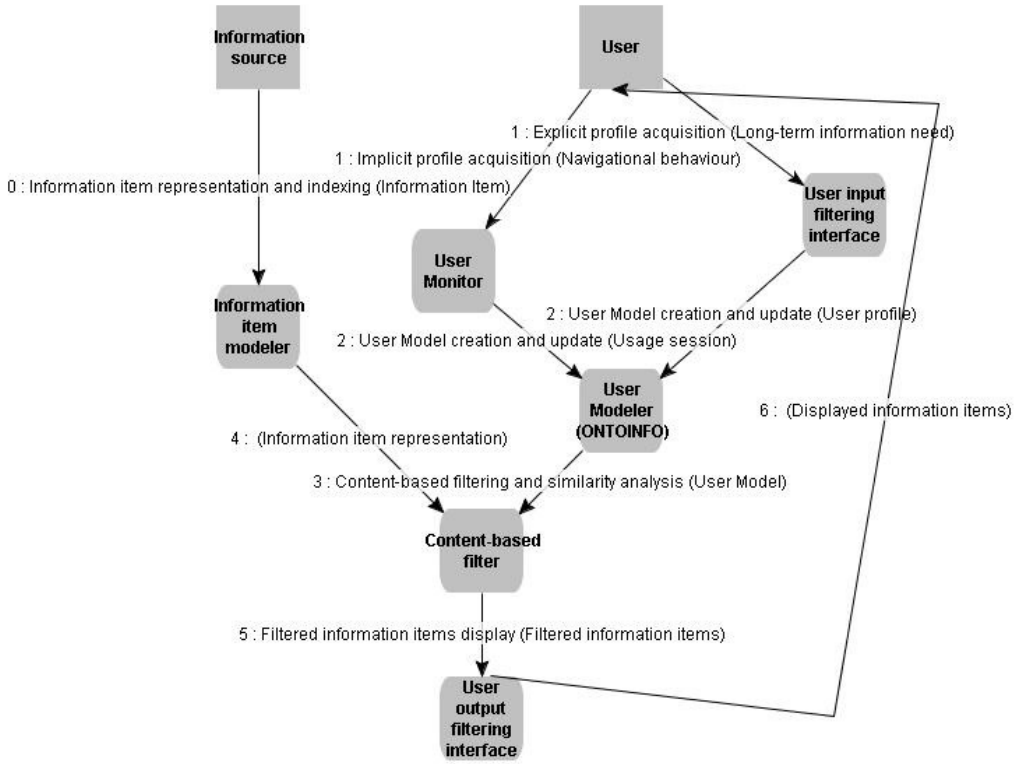

Figura 6A Modelo de interações de "Satisfazer necessidades de longo prazo através de filtragem baseada em conteúdo"

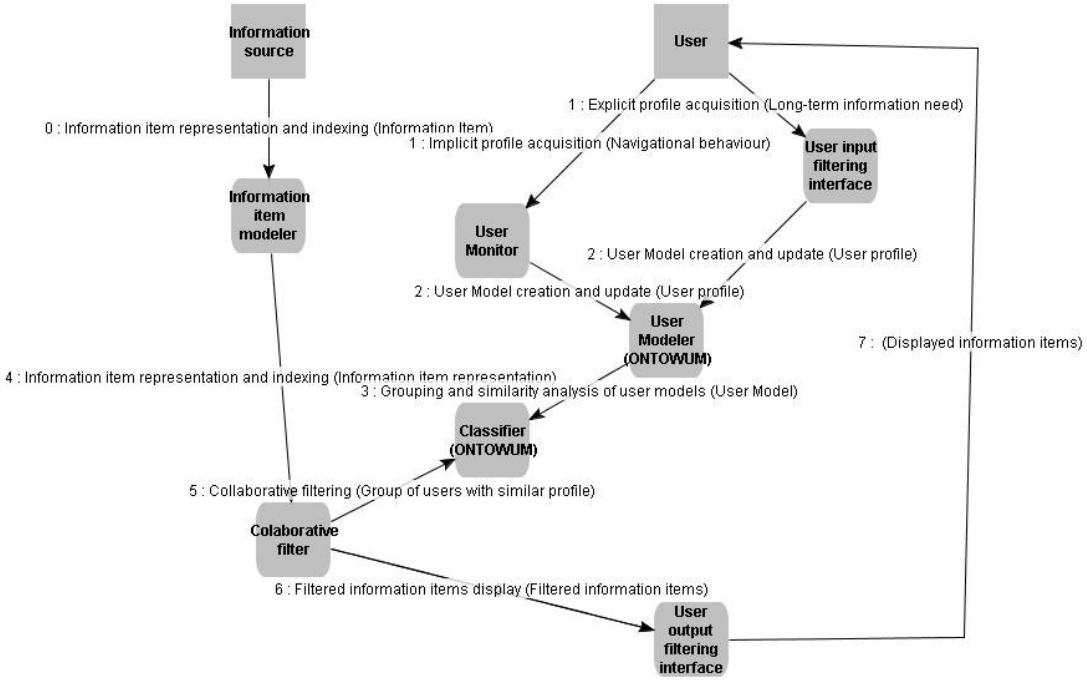

Figura 7A Modelo de interações de "Satisfazer necessidades de longo prazo através de filtragem colaborativa"

Modelo da sociedade multiagente

A seguir serão listados os diagramas do modelo da sociedade multiagente. Esse diagramas são similares aos diagramas de papéis, onde os papéis são substituídos por agentes e um agente pode assumir uma ou mais responsabilidade.

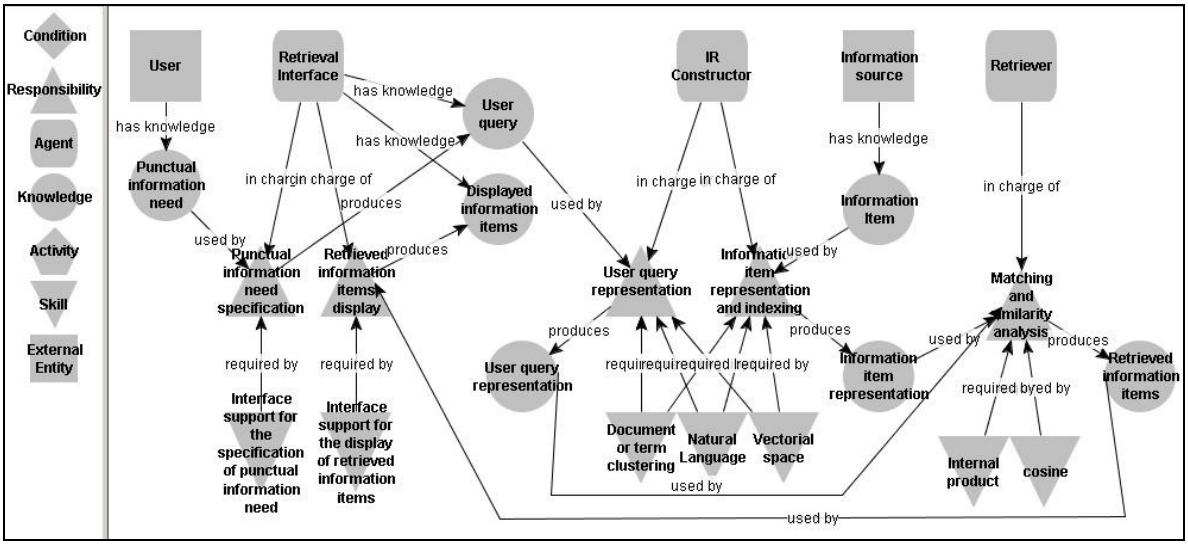

Figura 8A Modelo da sociedade multiagente relativo à recuperação da informação

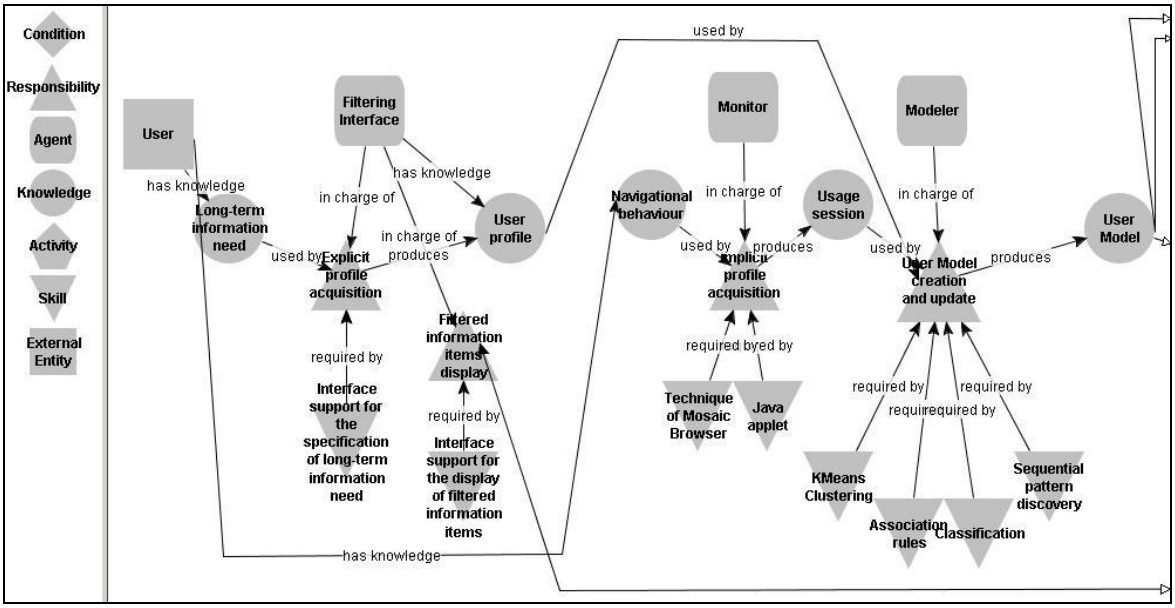

Figura 9A Modelo da sociedade multiagente relativo à modelagem do usuário e filtragem (parte 1)

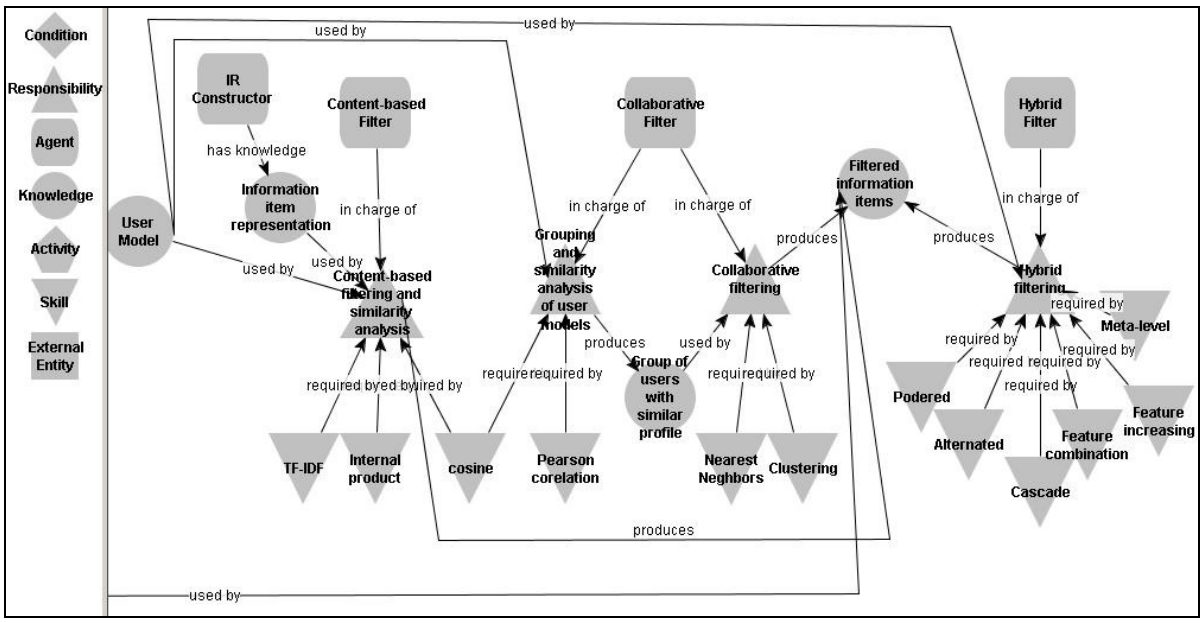

Figura 10A Modelo da sociedade multiagente relativo à modelagem do usuário e filtragem (parte 2)

Diagrama de interações entre agentes

A seguir serão listados os diagramas de interações entre agentes. Eles mostram as comunicações que existem entre os agentes para a realização de suas responsabilidades.

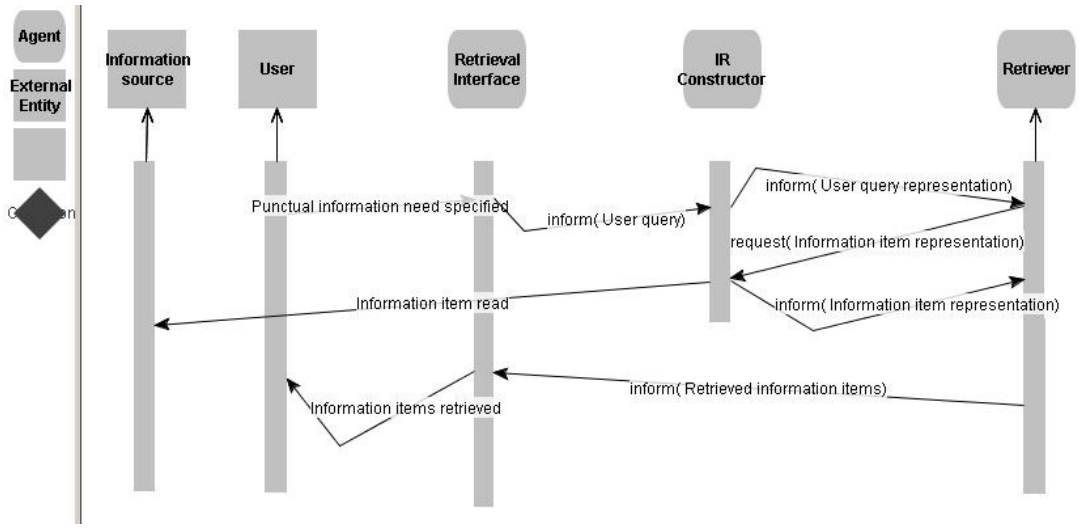

Figura 11A Diagrama de interações entre agentes relativo à recuperação de informação

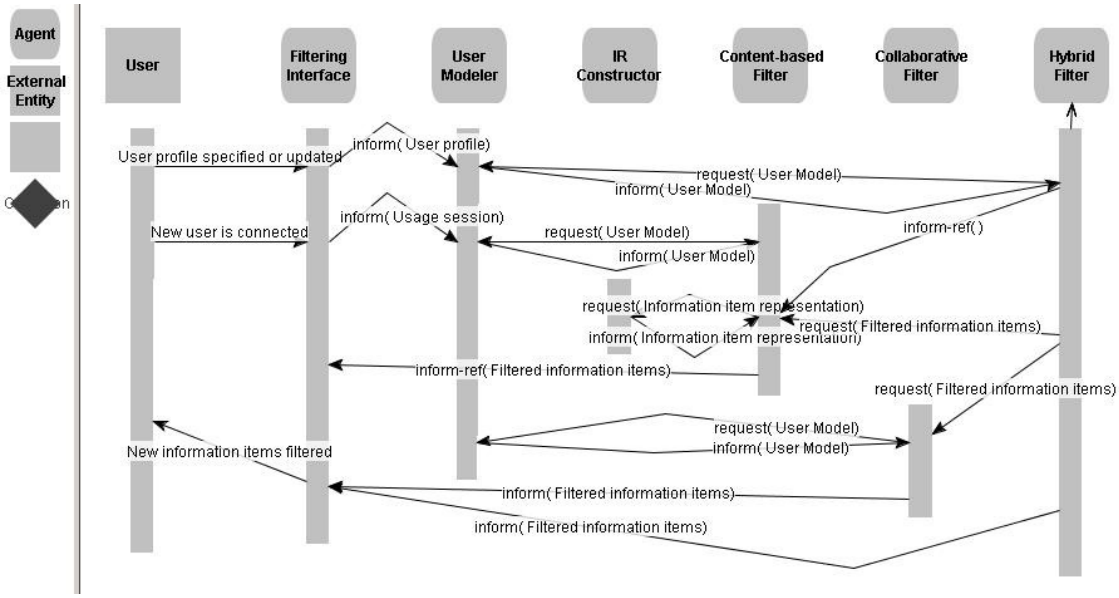

Figura 12A Diagrama de interações entre agentes relativo à filtragem

APÊNDICE B – CÓDIGO-FONTE DO PLUGIN ONTOGENMADEM

```
/*
 * ONTOGENMADEM - a tool for DSL development
 *
 */

package br.ufma.deinf.maae.ontogenmadem_plugin;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import edu.stanford.smi.protege.model.*;
import edu.stanford.smi.protege.widget.*;
import edu.stanford.smi.protege.resource.*;
import br.ufma.deinf.maae.ontogenmadem_plugin.*;

import org.algernon.Algernon;
import org.algernon.aam.Tracer;
import org.algernon.exception.AlgernonException;
import org.algernon.util.ErrorSet;
import org.algernon.util.gui.UneditableJTable;
import org.algernon.datatype.Result;
import org.algernon.datatype.BindingList;
import org.algernon.kb.okbc.protege.AlgernonProtegeKB;
import org.algernon.kb.okbc.protege.plugins.action.*;
import org.algernon.kb.okbc.protege.plugins.gui.LoadRulesDialog;
import org.algernon.kb.api.java.JavaAPIKB;
import org.algernon.kb.AlgernonKB;
import org.algernon.kb.*;
import org.jatha.dynatype.LispValue;
import org.jatha.read.LispParser;
import org.jatha.Jatha;

public class ONTOGENMADEM extends AbstractTabWidget {

   public static int ASK  = 1;
   public static int TELL = 2;

// public static AlgQuery alg;
   public String QueryText;
   public boolean eof;
   public static JTextArea taRes = new JTextArea(7,40);
   public JScrollPane scroll = null;

   Result result;          // variable to gets the result
   Iterator iterator;  // variable to bind the result
   BindingList bl;

   ErrorSet erros=null;
   protected Algernon  f_algy=null;
   protected Jatha     f_lisp = null;
   protected AlgernonKB f_kb=null;
   public int count_eof;   // control variable to make eof true only in the second next without hasNext
   private Project projeto = null;
```

```
   public String vtabvoctit[]={"Index","Class","Name"};            // conteudo da tabela de resultados do
vocabulario
   public String vtabvocdat[][]=new String[200][3];
   public      String      vtabagetit[]={"Index","Agent      Class","Agent      Name","Behaviour","Behaviour
type","variability"};   // conteudo da tabela de resultados do comp.impl.
   public String vtabagedat[][]=new String[200][6];

   /**
    * Metodo padrao de inicializacao do plugin
    */
   public void initialize() {

       // initialize the tab label
       setLabel("ONTOGENMADEM");
       setIcon(Icons.getInstanceIcon());

       setLayout(new BoxLayout(this,BoxLayout.PAGE_AXIS));

       // --- Gets path and nome of Protege project file in use (working)
       // getProjectFilePath() returns drive,path and filename of the project, with extension
       // getProjectName()     returns only filename of the project, without extension
       projeto = getProject();
       String varqprj = projeto.getProjectFilePath();

       // --- Initialize Algernon instances (init method from Algernon example)
       initAlg();

       JLabel titulo = new JLabel("ONTOGENMADEM",JLabel.CENTER);
       titulo.setFont(new Font( "Arial", Font.BOLD, 16));
       add(titulo);

       // --- Domain model selection combobox
       JPanel pnSel=new JPanel();
       JLabel lbModelo = new JLabel("Domain Model:");
       pnSel.add(lbModelo);
       final JComboBox cbModelo = new JComboBox();

       // --- Query the KB
       QueryText="((:instance \"Domain Model\" ?Domain_Model)(name ?Domain_Model ?name))";
       run(ASK);
       while (!eof) {
           cbModelo.addItem(BindByName("?name"));
           next();
       }

       pnSel.add(cbModelo);
       add(pnSel);

       // --- Desenv.da LED
       JPanel pnLED=new JPanel();
      pnLED.setBorder(BorderFactory.createCompoundBorder(
                       BorderFactory.createTitledBorder(" DSL Generation "),
                       BorderFactory.createEmptyBorder(5,5,5,5)));
       JLabel lbLED=new JLabel("DSL Name:");
       pnLED.add(lbLED);
       JTextField edNomeLED = new JTextField(30);
       edNomeLED.setHorizontalAlignment(SwingConstants.LEFT);
       pnLED.add(edNomeLED);
```

```
    JButton btGeraLED = new JButton("Generate DSL");
    btGeraLED.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            geraLED((String)cbModelo.getSelectedItem());
        }
    });
    pnLED.add(btGeraLED);
    JButton btViewGrammar = new JButton("View Grammar");
    btViewGrammar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            projeto.show("ONTOGENMADEM-ONTOINFOgrammar2_Instance_0");
        }
    });
    pnLED.add(btViewGrammar);
    add(pnLED);

    // --- Geracao dos mapeamentos
    JPanel pnMap=new JPanel();
   pnMap.setBorder(BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder(" Generator Design "),
                    BorderFactory.createEmptyBorder(5,5,5,5)));
    JButton btGeraMap = new JButton("Generate mappings");
    btGeraMap.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event) {
            geraMapeamentos((String)cbModelo.getSelectedItem());
        }
    });
    pnMap.add(btGeraMap);
    add(pnMap);

    // --- TextArea para mensagens
    JLabel lbMsg=new JLabel("Messages:");
    add(lbMsg);
    scroll = new JScrollPane(taRes,
       ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
       ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    add(scroll);

    // --- Tabela de instancias do vocabulario
    JLabel lbTabvoc = new JLabel("Vocabulary instances",JLabel.CENTER);
    JTable tbVocabulary = new JTable(vtabvocdat,vtabvoctit);
    JScrollPane scVocabulary = new JScrollPane(tbVocabulary,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    scVocabulary.setPreferredSize(new Dimension(700,90));
    add(lbTabvoc);
    add(scVocabulary,BorderLayout.CENTER);

    // --- Tabela de instancias dos agentes
    JLabel lbTabcomp = new JLabel("Implementation Components instances",JLabel.CENTER);
    JTable tbComponents = new JTable(vtabagedat,vtabagetit);
    JScrollPane scComponents = new JScrollPane(tbComponents,
        ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    scComponents.setPreferredSize(new Dimension(700,90));
    add(lbTabcomp);
    add(scComponents,BorderLayout.CENTER);
}
```

```
// --- allows the plugin work only with ONTOGENMADEM.pprj project file
public static boolean isSuitable(Project project, Collection errors) {
    boolean isSuitable;
    String filename;
    filename=project.getProjectName();
    filename=filename.substring(0,12);
    if (filename.equalsIgnoreCase("ONTOGENMADEM")) {
       isSuitable = true;
    } else {
       isSuitable = false;
       String text = "Works only with ONTOGENMADEM* projects";
       errors.add(text);
    }

    return isSuitable;
}

private void geraLED(String vmodel) {
    boolean verr=false;
    int vconf,vlin,vcol,vctab;
    String[ ] vaux1=new String[200];  // guarda resultados
    String vclasse,vvariab;           // classe e variabilidade a ser substituida nos scripts Algernon
    int vc,vqtres,vcclass;            // contadores e qt.de resultados
    vclasse="";

    vconf=JOptionPane.showConfirmDialog(null,"Any existing DSL will be deleted. Continue?",
       "DSL vocabulary generation",JOptionPane.YES_NO_OPTION);
    if (vconf!=JOptionPane.YES_OPTION)
       return;

    taRes.append("Processing...");

    // -- limpa resultados
    vctab=0;
    for(vlin=0;vlin<200;vlin++)
       for(vcol=0;vcol<3;vcol++)
          vtabvocdat[vlin][vcol]="";

    // === Goals ===
    // --- deletes existing optional goal vocabulary instances
    taRes.setText("");
    taRes.append("Optional Goals: deleting, ");
    verr=(DeleteInstances("Optional Goal",vmodel)==false||verr);
    // --- recria objetivos opcionais da DSL
    taRes.append("adding...\n");
    QueryText="";
    QueryText+="((:instance \"Specific Goal\" ?Specific_Goal)";
    QueryText+=" (\"variability type\" ?Specific_Goal \"optional\")";
    QueryText+=" (name ?Specific_Goal ?name)";
    QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
    QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
    QueryText+=" (contains ?Domain_Model ?Goal_Model)";
    QueryText+=" (:instance \"Goal Model\" ?Goal_Model)";
    QueryText+=" (concepts ?Goal_Model ?Specific_Goal)";
    QueryText+="   (:add-instance (?Optional_Goal \"Optional Goal\" )";
    QueryText+="      (name ?Optional_Goal ?name)";
    QueryText+="      (\"derived from\" ?Optional_Goal ?Specific_Goal)";
```

```
QueryText+="     (\"domain model\" ?Optional_Goal ?Domain_Model)";
QueryText+="))";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Optional Goal");                    // atualiza tabela
// --- deleta objetivos opcionais existentes no vocabulario
taRes.append("Alternative Goals: deleting, ");
verr=(DeleteInstances("Alternative Goal",vmodel)==false||verr);
// --- recria objetivos alternativos da DSL
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance \"Specific Goal\" ?Specific_Goal)";
QueryText+=" (\"variability type\" ?Specific_Goal \"alternative\")";
QueryText+=" (name ?Specific_Goal ?name)";
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Goal_Model)";
QueryText+=" (:instance \"Goal Model\" ?Goal_Model)";
QueryText+=" (concepts  ?Goal_Model ?Specific_Goal)";
QueryText+="   (:add-instance (?Alternative_Goal \"Alternative Goal\")";
QueryText+="      (name ?Alternative_Goal ?name)";
QueryText+="      (\"derived from\" ?Alternative_Goal ?Specific_Goal)";
QueryText+="      (\"domain model\" ?Alternative_Goal ?Domain_Model)";
QueryText+="))";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Alternative Goal");

// === ROLES ===
// --- exclui papeis opcionais do vocabulario
taRes.append("Optional Roles: deleting, ");
verr=(DeleteInstances("Optional Role",vmodel)==false||verr);
// --- adiciona papeis opcionais do vocabulario da DSL
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance Responsibility ?Responsibility)";
QueryText+=" (\"variability type\" ?Responsibility \"optional\")";
QueryText+=" (\"performed by\" ?Responsibility ?Role)";
QueryText+=" (name ?Role ?name)";
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Role_Model)";
QueryText+=" (:instance \"Role Model\" ?Role_Model)";
QueryText+=" (concepts  ?Role_Model ?Role)";
QueryText+=" (:add-instance (?Optional_Role \"Optional Role\")";
QueryText+=    (name ?Optional_Role ?name)";
QueryText+=    (\"derived from\" ?Optional_Role ?Role)";
QueryText+=    (\"domain model\" ?Optional_Role ?Domain_Model)) )";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Optional Role");                    // atualiza tabela
// --- exclui papeis alternativos do vocabulario
taRes.append("Alternative Roles: deleting, ");
verr=(DeleteInstances("Alternative Role",vmodel)==false||verr);
// --- adiciona papeis alternativos do vocabulario da DSL
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance Responsibility ?Responsibility)";
QueryText+=" (\"variability type\" ?Responsibility \"alternative\")";
QueryText+=" (\"performed by\" ?Responsibility ?Role)";
QueryText+=" (name ?Role ?name)";
```

```
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Role_Model)";
QueryText+=" (:instance \"Role Model\" ?Role_Model)";
QueryText+=" (concepts  ?Role_Model ?Role)";
QueryText+="   (:add-instance (?Alternative_Role \"Alternative Role\")";
QueryText+="     (name ?Alternative_Role ?name)";
QueryText+="     (\"derived from\" ?Alternative_Role ?Role)";
QueryText+="     (\"domain model\" ?Alternative_Role ?Domain_Model)))";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Alternative Role");                  // atualiza tabela

// === ACTIVITIES ===
// --- exclui atividades opcionais do vocabulario
taRes.append("Optional Activities: deleting, ");
verr=(DeleteInstances("Optional Activity",vmodel)==false||verr);
// --- adiciona atividades opcionais do vocabulario da DSL
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance Activity ?Activity)";
QueryText+=" (\"variability type\" ?Activity \"optional\")";
QueryText+=" (exercises ?Activity ?Responsibility)";
QueryText+=" (name ?Activity ?name)";
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Model)";
QueryText+=" (:instance \"Role Model\" ?Model)";
QueryText+=" (concepts ?Model ?Activity)";
QueryText+="   (:add-instance (?Optional_Activity \"Optional Activity\")";
QueryText+="     (name ?Optional_Activity ?name)";
QueryText+="     (\"derived from\" ?Optional_Activity ?Activity)";
QueryText+="     (\"domain model\" ?Optional_Activity ?Domain_Model)) )";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Optional Activity");                 // atualiza tabela
// --- exclui atividades alternativas do vocabulario
taRes.append("Alternative Activities: deleting, ");
verr=(DeleteInstances("Alternative Activity",vmodel)==false||verr);
// --- adiciona atividades alternativas do vocabulario da DSL
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance Activity ?Activity)";
QueryText+=" (\"variability type\" ?Activity \"alternative\")";
QueryText+=" (exercises ?Activity ?Responsibility)";
QueryText+=" (name ?Activity ?name)";
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Model)";
QueryText+=" (:instance \"Role Model\" ?Model)";
QueryText+=" (concepts ?Model ?Activity)";
QueryText+="   (:add-instance (?Alternative_Activity \"Alternative Activity\")";
QueryText+="      (name ?Alternative_Activity ?name)";
QueryText+="      (\"derived from\" ?Alternative_Activity ?Activity)";
QueryText+="      (\"domain model\" ?Alternative_Activity ?Domain_Model)) )";
verr=(run(ASK)==false||verr);
vctab=atuTabVoc(vctab,"Alternative Activity");              // atualiza tabela

// === SKILLS ===
for (vcclass=1;vcclass<=2;vcclass++) {
```

```
    vvariab=(vcclass==1 ? "optional" : "alternative");
    vclasse=(vcclass==1 ? "Optional" : "Alternative")+" Skill";
    // --- exclui skills ja existentes
    taRes.append(vclasse+"s"+": deleting, ");
    verr=(DeleteInstances(vclasse,vmodel)==false||verr);
    // --- adiciona skills ao vocabulario da DSL
    taRes.append("adding...\n");
    QueryText="";
    QueryText+="((:instance Skill ?Skill)";
    QueryText+=" (\"variability type\" ?Skill \""+vvariab+"\")";
    QueryText+=" (name ?Skill ?name)";
    QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
    QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
    QueryText+=" (contains ?Domain_Model ?Model)";
    QueryText+=" (:instance \"Role Model\" ?Model)";
    QueryText+=" (concepts ?Model ?Skill) )";
    verr=(run(ASK)==false||verr);
    // --- guarda skills sem repeticao
    for (vc=0;vc<vaux1.length;vc++) vaux1[vc]="";
    vqtres=0;
    while (!eof) {
       if (!afind(vaux1,BindByName("?name")))   // se ainda nao existe, guarda
          vaux1[vqtres++]=BindByName("?name");
       next();
    }
    // --- inclui skills sem repeticao
    for (vc=0;vc<vqtres;vc++) {
       QueryText="";
       QueryText+="((:instance Skill ?Skill)";
       QueryText+=" (name ?Skill \""+vaux1[vc]+"\")";
       QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
       QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
       QueryText+="   (:add-instance (?_Skill \""+vclasse+"\")";
       QueryText+="      (name ?_Skill \""+vaux1[vc]+"\")";
       QueryText+="      (\"derived from\" ?_Skill ?Skill)";
       QueryText+="      (\"domain model\" ?_Skill ?Domain_Model)) )";
       verr=(run(ASK)==false||verr);
       vtabvocdat[vctab][0]=Integer.toString(vctab+1);
       vtabvocdat[vctab][1]=vclasse;
       vtabvocdat[vctab++][2]=vaux1[vc];
    }
}

// === PAPEIS MANDATORIOS COM PROPR.VARIAVEIS ===
// --- exclui papeis mandatorios com destrezas.variaveis
taRes.append("Mandatory Roles: deleting, ");
verr=(DeleteInstances("Mandatory Role with variable skill",vmodel)==false||verr);
// --- adiciona papeis mandatorios com destrezas.variaveis
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance Responsibility ?Responsibility)";
QueryText+=" (\"variability type\" ?Responsibility \"mandatory\")";
QueryText+=" (\"performed by\" ?Responsibility ?Role)";
QueryText+=" (:instance Role ?Role)";
QueryText+=" (name ?Role ?name)";
QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
QueryText+=" (contains ?Domain_Model ?Model)";
```

```
        QueryText+=" (:instance \"Role Model\" ?Model)";
        QueryText+=" (concepts ?Model ?Responsibility)";
        QueryText+=" (:OR ((:ANY (:instance \"Alternative Skill\" ?Alternative_Skill) (\"derived from\"
?Alternative_Skill ?Skill) (requires ?Responsibility ?Skill)))";
        QueryText+="            ((:ANY (:instance \"Optional Skill\" ?Optional_Skill) (\"derived from\"
?Optional_Skill ?Skill) (requires ?Responsibility ?Skill)))";
        QueryText+="         ((:ANY (:instance \"Optional Activity\" ?Optional_Activity) (\"derived from\"
?Optional_Activity ?Activity) (\"exercised through\" ?Responsibility ?Activity)))";
        QueryText+="          ((:ANY (:instance \"Alternative Activity\" ?Alternative_Activity) (\"derived
from\" ?Alternative_Activity ?Activity) (\"exercised through\" ?Responsibility ?Activity)))";
        QueryText+=" )";
        QueryText+=" (:add-instance (?Mandatory_Role_with_variable_skill \"Mandatory Role with variable
skill\")";
        QueryText+="   (name ?Mandatory_Role_with_variable_skill ?name)";
        QueryText+="     (\"derived from\" ?Mandatory_Role_with_variable_skill ?Role)";
        QueryText+=" (\"domain model\" ?Mandatory_Role_with_variable_skill ?Domain_Model))";
        QueryText+=")";
        verr=(run(ASK)==false||verr);
        vctab=atuTabVoc(vctab,"Mandatory Role with variable skill");              // atualiza tabela

        // ==== GRAMATICA
/***       // --- parametros de objetivos
        for (vcclass=1;vcclass<=2;vcclass++) {
            vvariab=(vcclass==1 ? "optional" : "alternative");
            vclasse=(vcclass==1 ? "Optional" : "Alternative")+" Goal";
            // --- atualiza slot "parameterized by" (objetivos como parametros)
            taRes.append("Grammar: "+vclasse+"...\n");
            QueryText="";
            QueryText+="((:instance \""+vclasse+"\" ?dsl_goal)";
            QueryText+=" (\"derived from\" ?dsl_goal ?derived)";
            QueryText+=" (\"reached from\" ?derived ?maps_to)";
            QueryText+=" (:instance \"DSL Vocabulary term\" ?par)";
            QueryText+=" (\"derived from\" ?par ?maps_to)";
            QueryText+=" (\"parameterized by\" ?dsl_goal ?par) )";
            verr=(run(TELL)==false||verr);
            // --- atualiza slot "parameterized by" (papeis como parametros)
            QueryText="";
            QueryText+="((:instance \""+vclasse+"\" ?dsl_goal)";
            QueryText+=" (\"derived from\" ?dsl_goal ?derived)";
            QueryText+=" (:instance \"Responsibility\" ?resp)";
            QueryText+=" (achieves ?resp ?derived)";
            QueryText+=" (:instance Role ?role)";
            QueryText+=" (\"in charge of\" ?role ?resp)";
            QueryText+=" (:instance \"DSL Vocabulary term\" ?par)";
            QueryText+=" (\"derived from\" ?par ?role)";
            QueryText+=" (\"parameterized by\" ?dsl_goal ?par) )";
            verr=(run(TELL)==false||verr);
        }
        // --- parametros de papeis
        for (vcclass=1;vcclass<=3;vcclass++) {
            vvariab=(vcclass==1 ? "optional" : "alternative");
            if (vcclass==1)               vclasse="Optional Role";
            else if (vcclass==2) vclasse="Alternative Role";
            else if (vcclass==3) vclasse="Mandatory Role with variable skill";
            // --- atualiza slot "parameterized by" (objetivos como parametros)
            taRes.append("Grammar: "+vclasse+"...\n");
            QueryText="";
            QueryText+="((:instance \""+vclasse+"\" ?dsl_role)";
```

```
            QueryText+=" (\"derived from\" ?dsl_role ?derived)";
            QueryText+=" (\"in charge of\" ?derived ?resp)";
            QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
            QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
            QueryText+=" (contains ?Domain_Model ?Model)";
            QueryText+=" (:instance \"Role Model\" ?Model)";
            QueryText+=" (concepts ?Model ?resp)";
            QueryText+=" (requires ?resp ?skill)";
            QueryText+=" (:instance \"DSL Vocabulary term\" ?par)";
            QueryText+=" (\"derived from\" ?par ?skill)  )";
            QueryText+=" (\"parameterized by\" ?dsl_role ?par) )";
            verr=(run(TELL)==false||verr);
        }

        // --- inclui termos na gramatica
***/

        if (!verr)
            JOptionPane.showMessageDialog(null,"DSL syntax generated!","Done",
                JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null,"DSL syntax generated!!","Done",
                JOptionPane.INFORMATION_MESSAGE);

    }

    private void geraMapeamentos(String vmodel) {
        boolean verr=false;
        int vconf,vlin,vcol,vctab;
        String[ ] vaux1=new String[100];               // guarda resultados
        String vauxres,vclasse,vrm,vsv;
        int vqtres,vc;

        vconf=JOptionPane.showConfirmDialog(null,"Existing mappings will be deleted. Continue?",
            "Mappings generation",JOptionPane.YES_NO_OPTION);
        if (vconf!=JOptionPane.YES_OPTION)
            return;

        taRes.append("Processing...");

        // -- limpa resultados
        vctab=0;
        for(vlin=0;vlin<200;vlin++)
            for(vcol=0;vcol<6;vcol++)
                vtabagedat[vlin][vcol]="";

        // === classe "JADE Behaviour" ===
        // --- exclui instancias ja existentes em JADE Behaviour
        taRes.setText("");
        repaint();
        taRes.append("JADE Behaviours: deleting, ");
        repaint();
        verr=(DeleteInstances("JADE Behaviour",null)==false||verr);
        // --- inclui comportamentos JADE
        taRes.append("adding...\n");
        QueryText="";
        // --- RESPONSABILIDADES como JADE Behaviour
        QueryText="";
```

```
        QueryText+="((:instance Responsibility ?Responsibility)";
        QueryText+=" (\"performed by\" ?Responsibility ?Role)";
        QueryText+=" (name ?Responsibility ?name)";
        QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
        QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
        QueryText+=" (contains ?Domain_Model ?Model)";
        QueryText+=" (:instance \"Role Model\" ?Model)";
        QueryText+=" (concepts ?Model ?Role) )";
//      QueryText+=" (:add-instance (?JADE_Behaviour \"JADE Behaviour\")";
//      QueryText+="   (\"mapped from\" ?JADE_Behaviour ?Responsibility)) )";
        verr=(run(ASK)==false||verr);
        // --- guarda responsabilidades sem repeticao
        vqtres=0;
        while (!eof) {
           if (!afind(vaux1,BindByName("?name")))      // se ainda nao existe, guarda
              vaux1[vqtres++]=BindByName("?name");
           next();
        }
        // --- inclui responsabilidades sem repeticao
        for (vc=0;vc<vqtres;vc++) {
           QueryText="";
           QueryText+="((:instance Responsibility ?Responsibility)";
           QueryText+=" (name ?Responsibility \""+vaux1[vc]+"\")";
           QueryText+=" (:add-instance (?JADE_Behaviour \"JADE Behaviour\")";
           QueryText+="   (\"mapped from\" ?JADE_Behaviour ?Responsibility)) )";
           verr=(run(ASK)==false||verr);
        }
        // --- SKILLS como JADE Behaviour
        QueryText="";
        QueryText+="((:instance Skill ?Skill)";
        QueryText+=" (name ?Skill ?name)";
        QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
        QueryText+=" (name ?Domain_Model \""+vmodel+"\")";
        QueryText+=" (contains ?Domain_Model ?Model)";
        QueryText+=" (:instance \"Role Model\" ?Model)";
        QueryText+=" (concepts ?Model ?Skill) )";
        verr=(run(ASK)==false||verr);
        // --- guarda skills sem repeticao
        for (vc=0;vc<vaux1.length;vc++) vaux1[vc]="";
        vqtres=0;
        while (!eof) {
           if (!afind(vaux1,BindByName("?name")))      // se ainda nao existe, guarda
              vaux1[vqtres++]=BindByName("?name");
           next();
        }
        // --- inclui skill sem repeticao
        for (vc=0;vc<vqtres;vc++) {
           QueryText="";
           QueryText+="((:instance Skill ?Skill)";
           QueryText+=" (name ?Skill \""+vaux1[vc]+"\")";
           QueryText+=" (:add-instance (?JADE_Behaviour \"JADE Behaviour\")";
           QueryText+="   (\"mapped from\" ?JADE_Behaviour ?Skill)) )";
           verr=(run(ASK)==false||verr);
        }

        // === classe "JADE Agent" ===
        // --- exclui instancias ja existentes em JADE Agent
        taRes.append("JADE Agents: deleting, ");
```

```
verr=(DeleteInstances("JADE Agent",null)==false||verr);
// --- inclui agentes JADE
taRes.append("adding...\n");
scroll.repaint();
// --- inclui primeiro os agentes sem os comportamentos
QueryText="";
QueryText+="((:instance Agent ?Agent)";
QueryText+="(name ?Agent ?name)";
QueryText+="(:ANY (:instance \"Domain Model\" ?Domain_Model)";
QueryText+="   (name ?Domain_Model \""+vmodel+"\")";
QueryText+="   (contains ?Domain_Model ?Model)";
QueryText+="   (:instance \"Role Model\" ?Model)";
QueryText+="   (concepts ?Model ?resp)";
QueryText+="   (\"in charge of\" ?Agent ?resp) )";
QueryText+=" (:add-instance (?JADE_Agent \"JADE Agent\")";
QueryText+="   (\"mapped from\" ?JADE_Agent ?Agent) ) )";
verr=(run(ASK)==false||verr);
// --- obtem nomes dos agentes e suas responsabilidades
// --- cujas responsabilidades pertecem ao modelo de dominio especificado
QueryText="";
QueryText+="((:instance Agent ?Agent)";
QueryText+="(name ?Agent ?name)";
QueryText+="(\"in charge of\" ?Agent ?resp)";
QueryText+="(name ?resp ?respname)";
QueryText+="(:instance \"Domain Model\" ?Domain_Model)";
QueryText+="(name ?Domain_Model \""+vmodel+"\")";
QueryText+="(contains ?Domain_Model ?Model)";
QueryText+="(:instance \"Role Model\" ?Model)";
QueryText+="(concepts ?Model ?resp) )";
verr=(run(ASK)==false||verr);
// --- guarda nomes dos agentes e suas responsabilidades
for (vc=0;vc<vaux1.length;vc++) vaux1[vc]="";
vqtres=0;
while (!eof) {
   vauxres=BindByName("?name")+";"+BindByName("?respname");
   if (!afind(vaux1,vauxres))    // se ainda nao existe, guarda
      vaux1[vqtres++]=vauxres;
   next();
}
// --- atualiza slot "has behaviour" de cada agente com as responsabilidades
for (vc=0;vc<vqtres;vc++) {
   QueryText="";
   QueryText+="((:instance Agent ?Agent)";
   QueryText+=" (name ?Agent \""+vaux1[vc].substring(0,vaux1[vc].indexOf(";"))+"\")";
   QueryText+=" (:instance \"JADE Agent\" ?JADE_Agent)";
   QueryText+=" (\"mapped from\" ?JADE_Agent ?Agent)";
   QueryText+=" (:instance Responsibility ?resp)";
   QueryText+=" (name ?resp \""+vaux1[vc].substring(vaux1[vc].indexOf(";")+1)+"\")";
   QueryText+=" (:instance \"JADE Behaviour\" ?behaviour)";
   QueryText+=" (\"mapped from\" ?behaviour ?resp)";
   QueryText+=" (\"has behaviour\" ?JADE_Agent ?behaviour) )";
   verr=(run(TELL)==false||verr);
}
// --- obtem todos os agentes com suas responsabilidades e instancia JADE Behaviour correspondente
// --- cujas responsabilidades pertecem ao modelo de dominio especificado
QueryText="";
QueryText+="((:instance Agent ?Agent)";
QueryText+="(name ?Agent ?name)";
```

```
QueryText+="(\"in charge of\" ?Agent ?resp)";
QueryText+="(requires ?resp ?skill)";
QueryText+="(name ?skill ?skillname)";
QueryText+="(:instance \"Domain Model\" ?Domain_Model)";
QueryText+="(name ?Domain_Model \""+vmodel+"\")";
QueryText+="(contains ?Domain_Model ?Model)";
QueryText+="(:instance \"Role Model\" ?Model)";
QueryText+="(concepts ?Model ?resp)";
QueryText+="(:instance \"JADE Behaviour\" ?behaviour_resp)";
QueryText+="(\"mapped from\" ?behaviour_resp ?resp)";
QueryText+="(:instance \"JADE Behaviour\" ?behaviour_skill)";
QueryText+="(\"mapped from\" ?behaviour_skill ?skill) )";
verr=(run(ASK)==false||verr);
// --- guarda nomes dos agentes e suas destrezas
for (vc=0;vc<vaux1.length;vc++) vaux1[vc]="";
vqtres=0;
while (!eof) {
   vauxres=BindByName("?name")+";"+BindByName("?skillname");
   if (!afind(vaux1,vauxres))    // se ainda nao existe, guarda
      vaux1[vqtres++]=vauxres;
   next();
}
// --- atualiza slot "has behaviour" de cada agente com as destrezas
for (vc=0;vc<vqtres;vc++) {
   QueryText="";
   QueryText+="((:instance Agent ?Agent)";
   QueryText+=" (name ?Agent \""+vaux1[vc].substring(0,vaux1[vc].indexOf(";"))+"\")";
   QueryText+=" (:instance \"JADE Agent\" ?JADE_Agent)";
   QueryText+=" (\"mapped from\" ?JADE_Agent ?Agent)";
   QueryText+=" (:instance Skill ?skill)";
   QueryText+=" (name ?skill \""+vaux1[vc].substring(vaux1[vc].indexOf(";")+1)+"\")";
   QueryText+=" (:instance \"JADE Behaviour\" ?behaviour)";
   QueryText+=" (\"mapped from\" ?behaviour ?skill)";
   QueryText+=" (\"has behaviour\" ?JADE_Agent ?behaviour) )";
   verr=(run(TELL)==false||verr);
}

// --- Classe "Implementation Components"
// --- obtem lista de agentes
taRes.append("Implementation Components: deleting, ");
verr=(DeleteInstances("Mandatory agent and mandatory behaviours",null)==false||verr);
verr=(DeleteInstances("Mandatory agent and variable behaviours",null)==false||verr);
verr=(DeleteInstances("Variable agent",null)==false||verr);
// --- inclui agentes JADE
taRes.append("adding...\n");
QueryText="";
QueryText+="((:instance \"JADE Agent\" ?JAgent)";
QueryText+=" (\"mapped from\" ?JAgent ?agent)";
QueryText+=" (name ?agent ?name))";
verr=(run(ASK)==false||verr);
// --- guarda nome dos agentes
for (vc=0;vc<vaux1.length;vc++) vaux1[vc]="";
vqtres=0;
while (!eof) {
   if (!afind(vaux1,BindByName("?name")))      // se ainda nao existe, guarda
      vaux1[vqtres++]=BindByName("?name");
   next();
}
```

```
        // --- checa regras para cada agente
        for (vc=0;vc<vqtres;vc++) {
            // --- checa se tem alguma responsabilidade mandatoria
            QueryText="";
            QueryText+="((:instance \"JADE Agent\" ?JAgent)";
            QueryText+=" (\"mapped from\" ?JAgent ?agent)";
            QueryText+=" (name ?agent \""+vaux1[vc]+"\")";
            QueryText+=" (\"has behaviour\" ?JAgent ?behav)";
            QueryText+=" (\"mapped from\" ?behav ?mapped_from)";
            QueryText+=" (:instance Responsibility ?mapped_from)";
            QueryText+=" (\"variability type\" ?mapped_from \"mandatory\") )";
            verr=(run(ASK)==false||verr);
            vrm=(!eof ? "S" : "N");                    // se not eof, tem resp.mand. (S), senao (N)
            // --- checa se tem alguma destreza variavel
            QueryText="";
            QueryText+="((:instance \"JADE Agent\" ?JAgent)";
            QueryText+=" (\"mapped from\" ?JAgent ?agent)";
            QueryText+=" (name ?agent \""+vaux1[vc]+"\")";
            QueryText+=" (\"has behaviour\" ?JAgent ?behav)";
            QueryText+=" (\"mapped from\" ?behav ?mapped_from)";
            QueryText+=" (:instance Skill ?mapped_from)";
            QueryText+="   (:OR   ((:instance   Skill   ?mapped_from)(\"variability   type\"   ?mapped_from
\"alternative\"))";
            QueryText+="              ((:instance  Skill  ?mapped_from)(\"variability  type\"  ?mapped_from
\"optional\")) ) )";
            verr=(run(ASK)==false||verr);
            vsv=(!eof ? "S" : "N");                    // se not eof, tem skill variavel (S), senao (N)
            // --- define a classe do agente
            // -- resp.mand=S e destr.var=N
            vclasse="";
            if ( vrm=="S" && vsv=="N")
               vclasse="Mandatory agent and mandatory behaviours";
            // -- resp.mand=S e destr.var=S
            else if ( vrm=="S" && vsv=="S")
               vclasse="Mandatory agent and variable behaviours";
            // -- resp.mand=N
            else if ( vrm=="N" )
               vclasse="Variable agent";
            vauxres=vclasse.split(" ")[0];
            vauxres=vauxres.toLowerCase();

            // --- insere o agente na classe devida
            QueryText="";
            QueryText+="((:instance \"JADE Agent\" ?JAgent)";
            QueryText+=" (\"mapped from\" ?JAgent ?agent)";
            QueryText+=" (name ?agent \""+vaux1[vc]+"\")";
            QueryText+=" (:add-instance (?ImplComp \""+vclasse+"\")";
            QueryText+="   (\""+vauxres+" agent\" ?ImplComp ?JAgent)) )";
            verr=(run(ASK)==false||verr);
        }

        // --- mostra resultados - agentes variaveis
        vclasse="";
        for(vc=1;vc<=3;vc++) {
            if (vc==1) vclasse="Mandatory agent and mandatory behaviours";
            else if (vc==2) vclasse="Mandatory agent and variable behaviours";
            else if (vc==3) vclasse="Variable agent";
            QueryText="";
```

```
        QueryText+="((:instance \""+vclasse+"\" ?a)";
        if (vc<3) QueryText+=" (\"mandatory agent\" ?a ?Jagent)";
        else      QueryText+=" (\"variable agent\" ?a ?Jagent)";
        QueryText+="(\"mapped from\" ?Jagent ?agent)";
        QueryText+="(name ?agent ?nm_agent)";
        QueryText+="(\"has behaviour\" ?Jagent ?beh)";
        QueryText+="(\"mapped from\" ?beh ?map)";
        QueryText+="(name ?map ?nm_behaviour)";
        QueryText+="(:OR ((:instance \"Responsibility\" ?map) (:bind ?tipo \"Responsibility\"))";
        QueryText+="     ((:instance \"Skill\" ?map) (:bind ?tipo \"Skill\"))  )";
        QueryText+="(\"variability type\" ?map ?variabilidade)  )";
        verr=(run(ASK)==false||verr);
        vctab=atuTabComp(vctab,vclasse);
    }

    if (!verr)
        JOptionPane.showMessageDialog(null,"Mappings generated!","Done",
           JOptionPane.INFORMATION_MESSAGE);
    else
        JOptionPane.showMessageDialog(null,"Mappings generated!!","Done",
           JOptionPane.INFORMATION_MESSAGE);
}

/**
 * Exclui todas instancias de uma classe da ontologia que pertencam a um determinado modelo de dominio
 *
 * @param  vclass       nome da classe da ontologia
 * @param  vDomainModel name the location of the image, relative to the url argument
 * @return              logico indicando se exclusao foi bem-sucedida
 */
boolean DeleteInstances(String vclass, String vDomainModel) {
    QueryText="";
    QueryText+="((:instance \""+vclass+"\" ?Instance)";
    if (vDomainModel!=null) {
        QueryText+=" (:instance \"Domain Model\" ?Domain_Model)";
        QueryText+=" (name ?Domain_Model \""+vDomainModel+"\")";
        QueryText+=" (\"domain model\" ?Instance ?Domain_Model)";
    }
    QueryText+=" (:DELETE-INSTANCE ?Instance))";
    return run(ASK);
}

/**
 * Atualiza tabela de resultados do vocabulario
 *
 * @param  vlinatu numero da linha atual na tabela
 * @param  vclasse nome da classe do vocabulario em questao
 * @return         num da linha atual apos a atualizacao
 */
int atuTabVoc(int vlinatu,String vclasse) {
    while (!eof) {
        vtabvocdat[vlinatu][0]=Integer.toString(vlinatu+1);
        vtabvocdat[vlinatu][1]=vclasse;
        vtabvocdat[vlinatu][2]=BindByName("?name");
        next();
        vlinatu++;
    }
    return(vlinatu);
```

```
    }

    /**
     * Atualiza tabela de resultados dos componentes de implementacao
     *
     * @param  vlinatu numero da linha atual na tabela
     * @param  vclasse nome da classe do vocabulario em questao
     * @return         num da linha atual apos a atualizacao
     */
    int atuTabComp(int vlinatu,String vclasse) {
        while (!eof) {
            vtabagedat[vlinatu][0]=Integer.toString(vlinatu+1);
            vtabagedat[vlinatu][1]=vclasse;
            vtabagedat[vlinatu][2]=BindByName("?nm_agent");
            vtabagedat[vlinatu][3]=BindByName("?nm_behaviour");
            vtabagedat[vlinatu][4]=BindByName("?tipo");
            vtabagedat[vlinatu][5]=BindByName("?variabilidade");
            next();
            vlinatu++;
        }
        return(vlinatu);
    }

    /**
     * Verifica se uma string consta num array de string
     *
     * @param  varray array de strings
     * @param  vstr   string a ser procurado
     * @return        logico indicando se string foi encontrado
     */
    boolean afind(String[] varray, String vstr) {
        int vc;
        boolean vfound=false;
        for (vc=0;vc<varray.length;vc++) {
            if (varray[vc]==vstr) {
               vfound=true;
               break;
            }
        }
        return(vfound);
    }

    // this method is useful for debugging
    public static void main(String[] args) {
        edu.stanford.smi.protege.Application.main(args);
    }

    // --- constructor method: initialize Algernon instance
    public void initAlg() {
        try {
            f_algy = new Algernon();
            if (f_algy !=null) {
               f_lisp    = f_algy.getLisp();
//             f_kb = new AlgernonProtegeKB(f_algy,vprj);
               f_kb = new AlgernonProtegeKB(f_algy,getKnowledgeBase());
               if (f_kb!=null) {
                  f_algy.addKB(f_kb);
                  erros=new ErrorSet();
```

```
            } else
                JOptionPane.showMessageDialog(null,"Nao instanciou AlgernonProtegeKB.","AlgQuery()",
                    JOptionPane.ERROR_MESSAGE);
        }
    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(null,"Erro ao acessar a KB.","AlgQuery()",
            JOptionPane.ERROR_MESSAGE);
        if (f_kb != null)
            f_kb.close();
    }
}

// --- Executes a query on the KB
public boolean run(int action) {
    boolean vres=false;
    LispValue vrq = f_lisp.NIL;
    try {
        erros.clear();
        if (action==ASK)
            vrq=f_algy.ask(QueryText,erros);
        else
            vrq=f_algy.tell(QueryText,erros);
        if (vrq!=null && vrq!=f_lisp.NIL)
            result=(Result)vrq;
        else
            result=null;
        vres=(result!=null);
        if (result==null) {
            System.err.println("Erro durante ask de '"+QueryText+"'.");
            System.err.println(erros.toString());
            eof=true;
        }
        else {
            iterator=result.iterator();
            count_eof=0;                // set eof count=0 (eof is set when overstep the last item)
            next();       // point to the first result item
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,"Erro rodando Algernon: "+e,"AlgQuery()",
            JOptionPane.ERROR_MESSAGE);
        System.err.println("Erro rodando o Algernon: "+e);
        e.printStackTrace();
        return vres;
    }
    return vres;
}

public String BindByName(String BindName) {
    return (String) f_algy.getBinding(BindName,bl);
}

public void next() {
    if (iterator.hasNext())
        bl=(BindingList) iterator.next();
    else
        count_eof++;
    eof=(!iterator.hasNext()) && (count_eof>0);
```

```
    }

    public void close() {
        f_kb.close();
    }

}
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas