
**UNIVERSIDADE FEDERAL DO MARANHÃO**
**CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE**
**ÁREA: CIÊNCIA DA COMPUTAÇÃO**

# UMA TÉCNICA PARA O DESENVOLVIMENTO DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO

*Ivo José da Cunha Serra*

**São Luís, MA**
**2004**

# UMA TÉCNICA PARA O DESENVOLVIMENTO DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO

**Ivo José da Cunha Serra**

Bacharel em Ciência da Computação
Universidade Federal do Maranhão, 2000

**Dissertação 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 Ciência da Computação**.

Orientadora: Profª. Drª. Rosario Girardi

**São Luís, MA**
**2004**

# UMA TÉCNICA PARA O DESENVOLVIMENTO DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO

*Ivo José da Cunha Serra*

Dissertação aprovada em / /

_____

Profª. Drª. Maria del Rosario Girardi
Universidade Federal do Maranhão
(Orientadora)

_____

Prof. Dr. Zair Abdelouahab
Universidade Federal do Maranhão

_____

Prof. Dr. Evandro de Barros Costa
Universidade Federal de Alagoas

*Aos meus pais.*

# AGRADECIMENTOS

Agradeço a todos que contribuíram direta ou indiretamente para a elaboração desta dissertação, em especial:

A Deus, o criador de todas as coisas.

Ao meu pai Tadeu, à minha mãe Vani, ao meu irmão Gustavo e à minha irmã Louise, pelo amor, dedicação e incentivo.

À Professora Rosario, pela orientação segura e presente e pelos ensinamentos, imprescindíveis para a realização deste trabalho.

Aos meus companheiros de trabalho, em especial, Alisson, Carla e Ismênia pelas suas inestimáveis contribuições.

À toda Coordenação do Mestrado, coordenadora, funcionários e professores, pelos bons serviços oferecidos, que foram fundamentais para a conclusão deste curso.

# RESUMO

Por motivos de qualidade e produtividade, o reuso de software é uma prática necessária no desenvolvimento dos sistemas atuais. Uma das formas do reuso de software é a reutilização gerativa, que consiste em selecionar e agrupar componentes de software de forma automática. A reutilização gerativa pode ser feita com o uso de linguagens específicas de domínio (LED's), que especificam sistemas em alto nível de abstração.

Neste trabalho é proposta a TOD-LED, uma técnica baseada em ontologias para o desenvolvimento de LED's na Engenharia de Domínio Multiagente. Esta técnica guia a especificação de LED's a partir de modelos de domínio desenvolvidos com a GRAMO, uma técnica para a análise de domínio na Engenharia de Domínio Multiagente.

A TOD-LED utiliza a ONTOLED, uma ontologia que representa o conhecimento acerca do desenvolvimento de LED's. A especificação de uma LED é representada por uma instância da ONTOLED.

É também apresentado um estudo de caso para avaliar a técnica proposta. O estudo de caso consiste na especificação da LESRF (Linguagem de Especificação de Sistemas para a Recuperação e Filtragem de informação), uma LED para o desenvolvimento de aplicações para o acesso à informação dinâmica e não estruturada.



Palavras - chave: Linguagens Específicas de Domínio, Engenharia de Domínio, Ontologias, Sistemas Multiagente.

**Fonte:** Serra, Ivo. **UMA TÉCNICA PARA O DESENVOLVIMENTO DE LINGUAGENS ESPECÍFICAS DE DOMÍNIO.** 2004. 130 p. Dissertação (Mestrado em Engenharia de Eletricidade), Universidade Federal do Maranhão, São Luís.

# ABSTRACT

To achieve quality and productivity in software development, software reuse is necessary nowadays. One way of doing so is generative reuse, which consists of automatic selecting and grouping software components. Generative software reuse can be done by employing Domain Specific Languages (DSL's). These languages specify a system in a high level of abstraction.

This work proposes TOD-LED, a technique based on ontologies for the development of DSL's on Multi-Agent Domain Engineering. This technique guides the specification of DSL's using domain models developed with GRAMO, a technique for Domain Analysis in Multi-Agent Domain Engineering.

TOD-LED uses ONTOLED, an ontology that represents the knowledge about the development of DSL's. The specification of a DSL is represented as an instance of ONTOLED.

A case study was developed to evaluate TOD-LED. The case study consists of the specification of LESRF, a DSL for the development of systems for dynamic and non structured information access.



Keywords: Domain Specific Languages, Domain Engineering, Ontologies, Multi-Agent Systems.

**Source:** Serra, Ivo. **A TECNIQUE FOR DOMAIN SPECIFIC LANGUAGE DEVELOPMENT.** 2004. 130 p. Thesis (Graduation in Eletrical Engineering), Universidade Federal do Maranhão, São Luís.

SUMÁRIO

LISTA DE FIGURAS

# LISTA DE TABELAS

# ABREVIATURAS E SÍMBOLOS

| | |
|---|---|
| BNF | Backus-Naur Form |
| GAL | Graphics Adaptor Language |
| GRAMO | Generic Requirement Analysis Method based on Ontologies |
| HTML | Hypertext Marckup Language |
| LED | Linguagem Específica de Domínio |
| LESRF | Linguagem de Especificação de Sistemas para Recuperação e Filtragem de informação |
| LMPL | Linguagem de Marcação da Plataforma Lattes |
| LPG | Linguagem de Propósito Geral |
| ONTODD-INFO | Framework multiagente baseado em ontologias para o acesso à informação |
| ONTODUM | Ontologia para a representação de modelos de domínio e usuários |
| ONTOINFO | Ontologia para a área do acesso à informação |
| ONTOLED | Ontologia que representa o conhecimento acerca do desenvolvimento de LED's segundo a TOD-LED |
| SDA | Software Design Automation |
| SQL | Structured Query Language |
| TOD-LED | Técnica baseada em Ontologias para o Desenvolvimento de LED's |
| XML | Extensible Marckup Language |

# 1 INTRODUÇÃO

## 1.1 Motivação

A Engenharia de Domínio [Werner, Braga, 1998] é uma atividade que visa identificar e delimitar domínios de problema e também produzir artefatos de software que possam ser reutilizados no desenvolvimento de sistemas pertencentes a um domínio específico.

Entre os artefatos produzidos pela Engenharia de Domínio está o Modelo de domínio que é uma representação conceitual das entidades do domínio e seus relacionamentos. Outros artefatos são componentes de software, que podem ser agrupados em bibliotecas especificas de domínio, úteis na fase de implementação de sistemas. Um terceiro produto são as Linguagens Específicas de Domínio (LED's).

As LED's [Deursen, Klint, Visser, 2000] [Thibault, 1998] [Thibault, Marlet, Consel, 1997] são linguagens de programação que possuem um alto nível de abstração e um vocabulário próximo ao utilizado pelos especialistas de domínio. É importante ressaltar que as LED's representam um passo a frente das bibliotecas de domínio.

Quando os desenvolvedores fazem uso de uma biblioteca de domínio, estão fazendo reutilização composicional, ou seja, eles próprios selecionam e integram um componente ao sistema que está sendo desenvolvido. Diferentemente disso, as LED's possibilitam a reutilização gerativa, ou seja, os desenvolvedores especificam uma aplicação em um alto nível de abstração que será interpretada ou compilada para construir o sistema através da reutilização automática de artefatos de software, tais como, frameworks, padrões de projeto e padrões de implementação.

## 1.2 Contexto de Pesquisa

MaAE (Multi-agent Application Engineering) é um projeto desenvolvido no contexto do grupo de pesquisa GESEC (Grupo de pesquisa em Engenharia de Software e Engenharia de Conhecimento) [Girardi, 2003], que busca a sistematização de metodologias de desenvolvimento para a Engenharia de Domínio Multiagente e para a reutilização de seus produtos no desenvolvimento de aplicações específicas.

O projeto trata-se de um empreendimento realizado em parceria entre professores e pesquisadores das Universidades Federal de Alagoas (UFAL), do Maranhão (UFMA) e de Santa Catarina (UFSC) que integra experiências interdisciplinares vindas das áreas da Engenharia de Software e da Inteligência Artificial e propõe aplicações concretas de resultados já obtidos de projetos de pesquisa em andamento. A proposta está sendo avaliada através do desenvolvimento de vários estudos de caso envolvendo a tecnologia multiagente, como o Aceso à Informação [Girardi, 2003].

## 1.3 Objetivo do Trabalho

Este trabalho tem como objetivo principal a definição de um roteiro para a especificação de linguagens específicas de domínio na Engenharia de Domínio Multiagente.

## 1.4 Estrutura da Dissertação

Esta dissertação esta estruturada em sete capítulos.

O segundo capítulo faz uma discussão sobre linhas de produção de software, Engenharia de Domínio e como as LED’s se constituem em um meio de promover a reutilização gerativa dos artefatos produzidos nesta última.

O terceiro capítulo apresenta uma discussão a respeito de LED's, conceitos, características, LED's existentes e metodologias para seu desenvolvimento.

No quarto capítulo é apresentada a técnica GRAMO [Faria, 2004] para a construção de modelos de domínio na Engenharia de Domínio Multiagente. Estes modelos constituem os insumos da técnica proposta.

O quinto capítulo apresenta a ONTOLED, uma ontologia que representa o conhecimento acerca do desenvolvimento de LED's e a TOD-LED, uma técnica para o desenvolvimento de LED's na Engenharia de Domínio Multiagente.

O sexto capítulo apresenta um estudo de caso para avaliar a TOD-LED. O estudo de caso consiste no desenvolvimento da LESRF, uma LED para a especificação de sistemas na área do acesso à informação dinâmica e não estruturada.

No sétimo capítulo são apresentadas as considerações finais do trabalho, destacando os resultados obtidos e os trabalhos futuros que poderão ser desenvolvidos a partir desta pesquisa.

# 2 ENGENHARIA DE DOMÍNIO

A Reutilização [Kruegar, 1992] é uma abordagem dentro da Engenharia de Software que tem se mostrado de grande utilidade ou quase imprescindível no desenvolvimento dos sistemas atuais, onde a complexidade é uma característica marcante e o tempo de desenvolvimento deve ser cada vez menor.

Um dos maiores problemas na reutilização de software é o de projetar e implementar componentes de tal forma que possam ser usados em situações para as quais não foram inicialmente concebidos.

Para amenizar este problema, é proposta a idéia de que seja feita a reutilização em domínios específicos. Isso significa desenvolver componentes que possam ser reutilizados entre sistemas pertencentes a um mesmo domínio. Um domínio pode ser definido como um conjunto de sistemas que possuem características em comum de tal forma que é conveniente estuda-los juntos [Arango, 1988].

Nesse sentido é proposta a Engenharia de Domínio [Arango, 1988] [Harsu, 2002] [Magnan, et. al, 2001], que é uma atividade que precede a fase de análise no desenvolvimento de sistemas. A Engenharia de Domínio é uma atividade que tem como finalidade criar uma série de artefatos que permitam a reutilização do conhecimento do domínio, na forma de seus vários produtos como modelos de domínio, frameworks, componentes e linguagens específicas de domínio.

## 2.1 Processo da Engenharia de Domínio

A Engenharia de Domínio representa um enfoque sistematizado da análise, projeto e implementação de domínio, sob uma perspectiva voltada para a construção de componentes, criando-se um processo completo para a especificação

de componentes reutilizáveis (análise, projeto e implementação do domínio) [Werner, Braga, 2000].

O conhecimento de domínio é um fator importante a ser levado em consideração no desenvolvimento de software, pois se o desenvolvedor conhece a área de aplicação na qual está desenvolvendo irá fazer uma boa especificação de requisitos e em conseqüência gerar um software de boa qualidade.

A Engenharia de Domínio tem como principal objetivo disponibilizar artefatos reutilizáveis, que possam ser utilizados no desenvolvimento de novas aplicações. A Engenharia de Domínio possui três etapas: Análise de domínio, Projeto de domínio e Implementação do Domínio [Harsu, 2002].

- *Análise do Domínio*: Esta fase tem como objetivo identificar as oportunidades para a reutilização e determinar os requisitos comuns de uma família de sistemas. O domínio do problema representa um conjunto de elementos de informação presente em um certo contexto do mundo real, inter-relacionado de forma bastante coesa, e que desperta o interesse de uma certa comunidade. O produto desta fase é um modelo de domínio.
- *Projeto do Domínio*: Esta fase tem como objetivo o refinamento das oportunidades de reutilização identificadas na fase de Análise do Domínio e identificar e generalizar soluções para os requisitos comuns, através da especificação de um framework (uma arquitetura de software reutilizável) e da especificação de padrões de projeto para o domínio. Os produtos desta fase são frameworks e padrões de projeto.

- *Implementação do Domínio*: Esta fase tem como objetivo implementar componentes, geradores de aplicação e linguagens específicas de domínio.

## 2.2 Linhas de produção de software

Os engenheiros de software da atualidade são pressionados pelo mercado a satisfazer dois objetivos que são conflitantes: precisam fazer desenvolvimento de software rápido e produzir software de qualidade. Este problema não é exclusividade da Engenharia de Software. Outras áreas como a produção de automóveis, Engenharia Mecânica, Engenharia Aeronáutica, se deparam com a mesma situação.

Para tentar conciliar estes objetivos, são propostas as linhas de produção. A idéia de linha de produção não é nova. Na verdade ela nasceu em 1913 com Henry Ford [Wikipedia, 2004]. Ela consiste em criar uma infraestrutura que permita a produção de algum bem de forma rápida, em larga escala e com menores custos.

Para trabalhar em linhas de produção de software é necessário identificar famílias de sistemas e criar um processo para o reuso sistemático dos artefatos específicos do domínio identificados na Engenharia de Domínio.

Segundo Parnas [Parnas, 1976]:

“Entende-se por família de produtos de software um conjunto de produtos de software com características suficientemente similares para permitir a definição de uma infraestrutura comum de estruturação dos itens que compõem os produtos e a parametrização das diferenças entre os produtos”.

Primeiramente é feita a identificação de famílias de sistemas e a construção dos artefatos do domínio através da Engenharia de Domínio.

Na segunda etapa, também conhecida como Engenharia de Aplicação os artefatos produzidos na Engenharia de Domínio são reutilizados em cada uma das atividades (análise, projeto e implementação) do desenvolvimento de sistemas pertencentes a uma família de sistemas [Girardi, Faria, 2003]. São também identificados os requisitos específicos de cada sistema. A **Figura 1** mostra o processo de uma linha de produção de software.

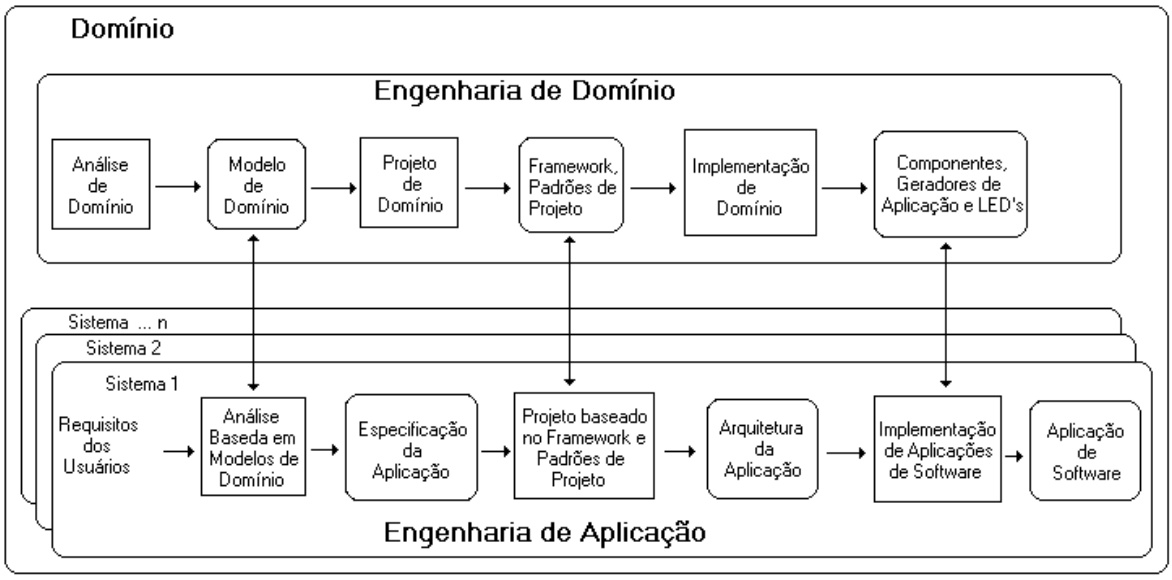


**Figura 1:** Processo de uma linha de produção de software [Foreman, 1996]

A reutilização, no desenvolvimento de sistemas em uma linha de produção de software pode ser feita de duas formas. Na primeira, os desenvolvedores fazem a busca, seleção e adaptação dos componentes a ser reutilizados quando do desenvolvimento de um novo sistema. Essa abordagem é chamada de composicional.

Uma abordagem menos explorada é a reutilização gerativa, na qual a reutilização é feita de forma automática, ou seja, de forma transparente aos desenvolvedores. A reutilização gerativa pode ser feita utilizando-se uma linguagem específica de domínio. Essa linguagem é um dos possíveis produtos resultantes da

Engenharia de Domínio e é utilizada para especificar todas as aplicações desenvolvidas em um domínio [Arango, 1988].

As LED's são a entrada para os geradores de aplicação [Cleaveland, 1988], programas que geram uma aplicação pertencente a uma família de sistemas a partir de uma especificação em alto nível de abstração. Os geradores de aplicação têm embutidos os artefatos do domínio que são reutilizados na produção de cada membro da família de sistemas. Uma LED serve como parâmetro para o gerador de aplicação, especificando as características particulares de cada sistema.

As LED's servem para especificar a parte que varia entre sistemas pertencentes a uma família de sistemas. Portanto, a identificação das partes comuns e variáveis em uma família de sistemas é fundamental para o desenvolvimento de tais linguagens. Este aspecto é abordado na próxima seção.

**2.3 Identificação dos conceitos fixos e variáveis**

Conceitos fixos são conceitos que estão presentes em todos os sistemas de uma família de sistemas. Eles representam as características comuns entre todos os sistemas. Conceitos variáveis são conceitos que podem estar presentes ou não em sistemas de uma mesma família. Eles representam as características que variam de um sistema para outro em uma família de sistemas [Mauw, Wiersma, Willemse, 2002].

A identificação dos conceitos fixos e variáveis é de fundamental importância por alguns motivos:

- *Projeto da LED*: Os conceitos fixos de uma família de sistemas não devem constar do vocabulário de uma LED, uma vez que são aspectos comuns e devem ser reutilizados para todos os sistemas da família de sistemas. Já os conceitos variáveis representam a parte que difere um

sistema do outro em uma família de sistemas e portanto devem constar do vocabulário da LED.

- *Treinamento para os usuários da LED*: A especificação dos conceitos fixos e variáveis é uma das principais fontes de informação que um novo membro de uma equipe de desenvolvimento precisa conhecer para entender uma LED e a família de sistemas a qual ela é associada.
- *Referência histórica*: A documentação permite aos arquitetos, desenvolvedores e demais envolvidos, manter e desenvolver uma família de sistemas a entender porque a família é estruturada e implementada de determinada forma.

Algumas das principais metodologias para o desenvolvimento de LED's, como Sprint [Consel, Marlet, 1998] e SDA [Widen, Hook, 1998] sugerem a utilização da técnica de análise dos aspectos comuns e variáveis proposto na abordagem FAST [Widen, 1995], (para o desenvolvimento de famílias de sistemas e também desenvolvimento de linguagens de domínio), chamada *Análise de semelhanças*, para a classificação dos conceitos do domínio. Isso se deve ao fato de essa técnica ser uma das mais bem documentadas, e também já testada em diferentes domínios. A *Análise de semelhanças* consiste de uma série de reuniões entre especialistas do domínio comandadas por um individuo conhecedor do processo FAST [Widen, 1995] e tem como produto um documento que especifica, entre outras coisas, os conceitos fixos e variáveis de uma família de sistemas.

As reuniões são geralmente feitas em intervalos regulares, mas sua duração e frequência podem variar bastante dependendo de cada projeto. Podem ainda existir questões que não representam um consenso entre os desenvolvedores

com relação à classificação de certos conceitos em fixos ou variáveis. Tais aspectos são incluídos em uma seção do documento chamada, *questões*.

Por esses motivos a *Análise de semelhanças* pode consumir tempo considerável e está sujeita a discordâncias entre os desenvolvedores com relação a classificação de certos conceitos do domínio. A **Tabela 1** mostra a estrutura do documento produzido na *Análise de semelhanças*.

| Seção | Propósito |
|---|---|
| Introdução | Descreve o propósito da análise e seu uso esperado. Tipicamente, o propósito é o de analisar ou definir os requisitos para uma família de sistemas. |
| Visão geral | Consiste de uma descrição rápida do domínio e seu relacionamento com outros domínios. |
| Dicionário de termos | Dá o significado de termos técnicos usados na descrição do domínio. |
| Aspectos comuns | Provê uma lista estruturada de considerações que são verdadeiras para todos os membros de uma família de sistemas. |
| Aspectos variáveis | Provê uma lista estruturada de considerações sobre como membros de uma família de sistemas podem variar. |
| Parâmetros de variação | Quantifica os aspectos variáveis, especificando a faixa de valores que esses podem assumir. |
| Questões | Provê um registro das alternativas existentes quando da análise da família de sistemas. Isso inclui conceitos que apresentaram maior grau de dificuldade para serem classificados em fixos ou variáveis, e por que foram classificados de uma forma ou de outra. |
| Apêndice | Inclui qualquer informação que possa ser útil para os revisores, projetistas da linguagem, projetistas de ferramentas automatizadas e quaisquer outros potenciais usuários da análise. |

**Tabela 1:** Organização do documento de *Análise de semelhanças* [Widen, 1995]

A **Figura 2**, mostra a título ilustrativo um trecho do documento *Análise de semelhanças* produzido para o domínio de sistemas de bóias meteorológicas.

**Dicionário de termos**

| Termo | Significado |
|---|---|
| Período do sensor | O tempo entre cada medição, em segundos |
| Período de transmissão | O tempo entre cada transmissão de mensagens, em segundos |
| Temperatura da água | A temperatura da água em graus Celsius |

**Aspectos comuns**

C1. Em intervalos fixos, a bóia monitora a velocidade do vento, temperatura da água e temperatura do ar na sua posição.

C2. A bóia é equipada com um ou mais sensores que monitoram a velocidade do vento.

C3. A bóia é equipada com um ou mais sensores que monitoram a temperatura do ar.

C4. A bóia é equipada com um ou mais sensores que monitoram a temperatura da água.

C5. A bóia é equipada com um rádio transmissor que permite a ela enviar mensagens aos satélites.

S6. Em intervalos fixos, a bóia transmite mensagens contendo uma aproximação dos valores referentes a velocidade do vento e temperatura. Os valores enviados são a média de todas as medições.

**Aspectos variáveis**

V1. O número de cada tipo de sensor em uma bóia pode variar.

V2. O período de transmissão de mensagens da bóia pode variar.

V3. O dispositivo de hardware para cada tipo de sensor pode variar.

V4. O dispositivo de hardware para transmissão de rádio pode variar.

**Parâmetros de variação**

| Parâmetro | Significado | Domínio |
|---|---|---|
| Período do sensor | Período do sensor | [1..600] segundos |
| Período de transmissão | Período de transmissão | [1..600] segundos |

**Figura 2:** Trecho do documento Análise de semelhanças do domínio das bóias meteorológicas [Widen, 1995]

Os sistemas das bóias meteorológicas são uma família de sistemas que provêem dados climáticos aos tráfegos marítimo e aéreo sobre o mar. As bóias coletam dados como, temperatura da água e velocidade do vento através de vários sensores. Cada bóia possui um transmissor de rádio para enviar os dados

meteorológicos a satélites que os repassam ao trafego marítimo e aéreo. Para cada bóia é desenvolvido um sistema que controla sua operação. Este sistema deve coordenar a operação dos sensores e o envio de relatórios aos satélites. As bóias irão conter um ou mais computadores e sensores para fazer medições como, a velocidade do vento e a temperatura da água.

## 2.4 Considerações finais

Neste capítulo foi feita uma discussão sobre a Engenharia de Domínio, suas fases e produtos. Os artefatos de software produzidos na Engenharia de Domínio podem ser reutilizados de duas formas, composicional ou gerativa. Na reutilização composicional os desenvolvedores fazem a busca e seleção manual dos artefatos a serem reutilizados. Na abordagem gerativa a reutilização é feita de forma automática. A reutilização gerativa pode ser feita com o uso de LED's, linguagens de alto nível de abstração utilizadas para especificar sistemas pertencentes a uma mesma família.

# 3 LINGUAGENS ESPECÍFICAS DE DOMÍNIO

As Linguagens Específicas de Domínio (LED's) não se constituem em um assunto novo, linguagens como Fortran, para o domínio de cálculos matemáticos e Cobol para o domínio das finanças foram concebidas há décadas. Entretanto, recentemente elas têm sido alvo de maior atenção tanto da comunidade científica quanto industrial. Um dos motivos para a atenção dedicada as LED's é que elas são vistas como uma boa opção para a especificação de sistemas em linhas de produção de software. Existem numerosos exemplos de domínios nos quais as LED's têm sido utilizadas tais como, gráficos [Elliott,1997] [Kamin, Hyatt, 1997], produtos financeiros [Arnol, Deursen, 1995], protocolos de rede [Thibault,1998] e drivers de dispositivos [Thibault, Marlet, Consel, 1997]. Essa profusão, assim como o número de eventos e projetos relacionados as LED's, demonstra a atenção que elas têm recebido das comunidades científica e industrial.

Estas linguagens são também um dos produtos da Engenharia de Domínio quando se possui um conhecimento estável e maduro com relação a uma área de aplicação. Neighbors inclusive defende que o principal produto da Engenharia de Domínio é a definição de uma linguagem específica de domínio [Werner, Braga, 1998].

Um outro motivo que fomenta o uso das LED's é a idéia de programação pelo usuário final, que consiste em permitir que os usuários de um sistema executem tarefas de programação simples. Algumas linguagens específicas de domínio bastante conhecidas e largamente utilizadas atualmente são SQL, linguagem de consulta a bases de dados e HTML, linguagem de marcação para páginas Web.

Este capítulo está organizado em três seções. Na seção 3.1 são apresentados conceitos e aspectos relacionados ao uso de LED's. Na seção 3.2 são

apresentadas metodologias para o seu desenvolvimento. Na seção 3.3 é apresentada uma análise comparativa das metodologias estudadas.

### 3.1 Conceitos básicos

Uma LED pode ser definida como uma linguagem de programação dedicada a um domínio particular. Ela provê abstrações e notações apropriadas para o domínio para o qual foi desenvolvida, é geralmente mais declarativa que imperativa e mais expressiva que uma Linguagem de Propósito Geral (LPG) [Compose project, 2003]

Uma vez que as LED's podem ser altamente declarativas e escondem muito dos detalhes de implementação, algumas delas podem ser consideradas mais como linguagens de especificação que linguagens de programação. Entretanto, essas especificações são geralmente executáveis.

A **Tabela 2** resume algumas LED's, classificando-as com relação a três critérios: se são imperativas ou declarativas, o domínio para os quais foram desenvolvidas e a notação que utilizam (ex.: textual, gráfica).

| LED's | Classificação | Domínio de aplicação | Notação | Referências |
|---|---|---|---|---|
| **HTML** | Declarativa | Exibição de páginas Web | Textual | [W3 Schools, 2003] |
| **XML** | Declarativa | Estruturação e intercâmbio de informação | Textual | [W3 Schools, 2003] |
| **GAL** | Declarativa | *Drivers* de dispositivos de vídeo | Textual | [Thibault, 1998] |
| **LMPL** | Declarativa | Formatação de informação sobre ciência e tecnologia | Textual | [Plataforma Lattes, 2003 ] |
| **SQL** | Imperativa | Consulta a base de dados | Textual | [W3 Schools, 2003] |

**Tabela 2:** Exemplos de LED's

### 3.1.1 Vantagens e desvantagens do uso de LED's

O uso das LED's traz vantagens em relação às linguagens de propósito geral e também alguns riscos. A seguir são discutidos alguns destes aspectos.

- *Programação facilitada*: Devido a abstrações e notações apropriadas e a formulação declarativa, um programa escrito em uma LED é mais conciso e de mais fácil leitura que um similar escrito em uma LPG. Por isso, o tempo de desenvolvimento é diminuído e a manutenção é melhorada. Uma vez que a programação é focada em o que deve ser feito e não em como, o usuário não precisa ser um programador experiente.
- *Reuso sistemático*: A maioria dos ambientes de programação de LPG permitem agrupar operações comumente usadas em bibliotecas. Nesse caso, o reuso fica a cargo do programador. Diferentemente, uma LED força a reutilização gerativa. As LED's capturam experiência do domínio de forma implícita, ocultando padrões de programação em sua implementação, ou explícita, expondo parametrizações específicas para o programador. Portanto, o programador necessariamente reusa componentes e conhecimento do domínio.
- *Alto nível de abstração:* As construções de alto nível das LED's servem para evitar possíveis erros de programação e escondem detalhes de implementação. Essas construções também permitem especificações mais concisas e portanto encurtam o tempo de desenvolvimento de novos produtos bem como sua manutenção.

- *O custo de projetar, implementar e manter uma LED:* O custo de desenvolver uma LED tem que ser compensado pelo número de sistemas que serão desenvolvidos. Por isso estas linguagens devem ser desenvolvidas para especificar sistemas pertencentes a uma família de sistemas.
- *A potencial perda de eficiência quando comparado com códigos escritos em LPG's:* O código escrito em LPG's é feito sob medida para cada situação. Isso não ocorre no caso das LED's, onde a aplicação é gerada automaticamente, o que pode ocasionar uma perda de eficiência.

### 3.2 Metodologias para o desenvolvimento de LED's

Apesar de existirem muitas LED's desenvolvidas e documentadas em várias áreas de aplicação, como protocolos de rede e consulta a bases de dados, ainda existe pouca informação sistematizada a respeito de metodologias para seu desenvolvimento. O desenvolvimento de LED's ainda é, na maioria das vezes, um processo artesanal ao invés de um processo claramente definido.

Existem algumas iniciativas em direção a metodologias para o desenvolvimento de LED's que diferem, por exemplo, com relação aos formalismos que utilizam e aos artefatos que produzem. Para a realização deste trabalho foram estudadas quatro metodologias para o desenvolvimento de LED's: Metodologia de Guizzardi [Guizzardi, Pires, Sinderen, 2002], *Sprint* [Consel, Marlet, 1998], *SDA* [Widen, Hook, 1998] e a metodologia de Mauw, Wiersma e Willemse [Mauw, Wiersma, Willemse, 2002] que são apresentadas nas próximas seções.

A metodologia de Guizzardi [Guizzardi, Pires, Sinderen, 2002], apesar de ainda não estar totalmente definida, foi de particular interesse, uma vez que sugere a utilização de modelos de domínio baseados em ontologias para a concepção de LED's, o que coincide com a proposta desta dissertação.

### 3.2.1 Sprint

Sprint é uma metodologia para o projeto e implementação de LED's baseada no *framework* proposto por Consel e Marlet [Consel, Marlet, 1998] e colocada em prática no desenvolvimento de algumas LED's como GAL [Thibault, 1998], para o domínio de drivers de dispositivos de vídeo e PLAN-P [Thibault, 1998], para o domínio de protocolos de aplicação em redes ativas (redes com roteadores programáveis).

Para efeito de simplificação, as fases da metodologia (análise da linguagem, definição da interface, semântica particionada, definição formal, máquina abstrata, implementação, avaliação parcial) são apresentadas seqüencialmente. Na verdade, todo o processo é interativo.

Para ilustrar a aplicação da metodologia são usados exemplos referentes ao desenvolvimento de uma LED denominada MAILSH [Consel, Marlet, 1998]. A MAILSH é uma linguagem simples que se propõe a especificar como devem ser tratados automaticamente e-mails recebidos. Este exemplo é inspirado em um programa UNIX chamado Slocal que oferece aos usuários uma forma de processar mensagens recebidas. No Slocal, os tratamentos definidos pelos usuários são especificados na forma de regras. Cada regra consiste de uma string a ser pesquisada em um campo da mensagem (ex.: assunto, de) e uma ação a ser executada de acordo com o valor da string.

3.2.1.1 Análise do domínio da linguagem

Na primeira fase da metodologia é analisada uma família de problemas. Durante essa análise as características comuns a todos os sistemas pertencentes a uma família de sistemas e as características variáveis que são as que diferem entre sistemas pertencentes a uma mesma família, são identificados.

A análise leva em consideração conhecimento do domínio tais como literatura técnica, programas existentes e requisitos atuais e futuros. As atividades e os principais produtos dessa análise são: a especificação dos requisitos da linguagem, uma descrição dos objetos e operações comuns e a especificação dos elementos de projeto da LED.

- *Análise dos requisitos da linguagem*

Esta atividade é semelhante a análise do problema que ocorre no início do desenvolvimento de qualquer software. A diferença é que os requisitos são expressos em termos de aspectos da linguagem ao invés de características gerais de uma aplicação.

Por exemplo, a linguagem MAILSH [Consel, Marlet, 1998] deve permitir os seguintes comandos: *copiar*, *mover*, *deletar*, *encaminhar* e *responder* a uma mensagem. Estas ações devem ser disparadas de acordo com condições dependentes das mensagens recebidas. Para tanto, é feita uma comparação entre valores previstos e os valores assumidos pelos campos das mensagens.

- *Descrição de objetos e operações*

Esta atividade corresponde essencialmente a definição dos artefatos de software necessários para expressar soluções para uma família de problema.

Por exemplo, na modelagem da MAILSH [Consel, Marlet, 1998], a análise de quatro famílias de aplicações de e-mail resultou nos seguintes objetos e operações fundamentais:

- Mensagens: Uma mensagem eletrônica consiste de campos de cabeçalho e um corpo. São necessárias operações para manipular esses campos e criar novas mensagens.
- Pastas: Pastas contém um conjunto de mensagens. Levando em consideração que só será possível despachar mensagens, a única operação necessária é adicionar uma mensagem a uma pasta.
- Hierarquia de pastas: Um usuário geralmente tem várias pastas para as quais direciona as mensagens. Dessa forma os sistemas de e-mail tem que oferecer a possibilidade de criação de hierarquia de pastas.
- Arquivos de pastas: Operações para ler e escrever uma pasta de um sistema de arquivos precisam ser criadas.
- Filas: Mensagens precisam ser enviadas e recebidas. Para modelar esse fato é necessário criar filas de mensagens recebidas e enviadas, assim como uma fila de comandos.
- Miscelânea: Há ainda objetos e operações de importância secundária. Eles incluem, por exemplo, a habilidade de saber o nome do usuário (para enviar mensagens) e a data do envio.

- *Elementos de projeto*

Nesta atividade são determinados os elementos do projeto da LED. Estes elementos incluem o paradigma da linguagem (ex: declarativo, imperativo), a

terminologia e a notação. A notação deve corresponder à forma como os especialistas do domínio expressam suas soluções.

3.2.1.2 Definição da interface da LED

A interface da linguagem são os elementos da linguagem, com os quais os desenvolvedores irão lidar. Dada a informação coletada anteriormente é possível desenvolver uma especificação preliminar da LED. Essa especificação preliminar consiste em definir interfaces: um esboço da álgebra semântica e a sintaxe da LED. A semântica é definida informalmente, ela será totalmente definida em uma fase posterior.

- *Definição da álgebra semântica*

Os objetos e operações comuns identificados na fase anterior são agora agrupados com relação aos objetos e operações que manipulam para produzir tipos abstratos de dados. Os tipos abstratos de dados podem ser formalizados em álgebra semântica. Uma álgebra semântica define formalmente um domínio e um conjunto de operações naquele domínio.

Por exemplo, a **Figura 3** mostra um trecho da álgebra semântica da linguagem MAILSH [Thibault, Marlet, Consel, 1997]. Nela são definidos sete domínios semânticos: *Message*, *Folders*, *FolderHierarchy*, *FolderFiles*, *InStream*, *OutStream*, *CmdStream*. São listadas também as operações válidas em cada domínio. Por exemplo, no domínio *Message* a operação *new-msg* cria uma nova mensagem, no domínio *Folder*, a operação *add-msg* adiciona uma mensagem a uma pasta.

**Messages**
Domain: *Message*
Operations:
*new-msg* : *Message*
*get-field* : *FieldName* → *Message* → *String*
*set-field* : *FieldName* → *String* → *Message* → *Message*
*get-body : Message* → *String*
*set-body : String* → *Message* → *Message*
*msg-to-string : Message* → *String*

**Folders**
Domain: *Folder*
Operations:
*add-msg* : *Message* → *Folder* → *Folder*

**Hierarchy of Folders**
Domain: *FolderHierarchy*
Operations:
*get-filename* : *FolderPath* → *FolderHierarch* → *FileName*

**Files of Folders**
Domain: *FolderFiles*
Operations:
*read-folder* : *FileName* → *FolderFiles* → *Folder*
*Write-folder : FileName* → *Folder* → *FolderFiles* → *FolderFiles*

**Streams**
Domain: *InStream*, *OutStream*, *CmdStream*
Operations:
*next-msg* : *InStream* →(*Message* x *InStream*)
*send-msg* : *Message* → *OutStream* → *OutStream*
*pipe-msg* : *Message* → *CmdString* → *CmdStream* → *CmdStream*

**Figura 3**: Trecho da álgebra semântica da MAILSH [Consel, Marlet, 1998]

```
If match "Subject" "DSL" then
  forward "jake";
    copy Research.Lang.DSL; delete
else if match "From" "hotmail.com" then
  reply "Leave me alone!" delete
else if match "Subject" "seminar" then
  pipe "agenda - stdin"; delete
else
  skip
```

**Figura 4:** Demonstração das regras sintáticas da MAILSH [Consel, Marlet, 1998]

- *Definição da sintaxe da LED*

A sintaxe da LED é definida baseada nos requisitos da linguagem (funcionalidades e restrições) e nos elementos de projeto (paradigma da linguagem, terminologia e notação), informações essas que são coletadas anteriormente.

Por exemplo, a **Figura 4** demonstra as regras sintáticas da MAILSH [Consel, Marlet, 1998].

3.2.1.3 Definição da semântica particionada

A semântica de linguagens de propósito geral é tipicamente dividida em ações de tempo de compilação e tempo de execução. Essas duas partes são também reconhecidas como semântica estática e dinâmica da linguagem. Sprint propõe realizar a mesma separação na semântica das LED's.

Do ponto de vista da arquitetura de software, a semântica estática corresponde à computação que determina o membro de uma família de sistemas. A semântica dinâmica corresponde à computação que produz respostas ao problema correspondente ou a execução do programa.

Do ponto de vista da implementação, processar a semântica estática de uma aplicação corresponde a configurar componentes de software genéricos com relação a um dado contexto. Mais precisamente corresponde a selecionar componentes apropriados e combina-los para produzir um software customizado. Já a semântica dinâmica corresponde a executar o software customizado.

3.2.1.4 Definição formal

Uma vez que os componentes estáticos e dinâmicos da linguagem tenham sido determinados, a LED é formalmente definida. Funções de avaliação definem a semântica das construções sintáticas. Elas especificam como as operações da álgebra semântica são combinadas.

Por exemplo, a **Figura 5** mostra as funções de avaliação definidas para a MAILSH. É usada a seguinte notação: a projeção da tupla $\sigma$ no domínio X (ex: *Message*) é denotado por $\sigma_x$ (ex: $\sigma_{message}$). E a atualização do elemento x da tupla $\sigma$ com um valor *y* é expresso por $[x \rightarrow y]\ \sigma$.

```
C : Command → StaticState → DynamicState → DynamicState where
StaticState = FoldersHierarchy x UserName
DynamicState = FolderFiles x OutStream x CmdStream x Date x Message

C [[C1;C2]] ρ = (C[[C2]] ρ) ∘ (C[[C1]] ρ)
C [[if B then C1 else C2]] ρ = cond (B[[B]]) (C[[C1]] ρ) (C[[C2]] ρ)
C [[skip]] ρ σ - σ
C [[copy F]] ρ σ =
      let σ = get-filename (F[[F]]) ρ folder-hierarchy
          φ = add-msg (set-field "Delivery-Date" σ date σ message)
                      (read-folder v σ folder-files)
in [folder-files → write-folder v φ σ folder-files] σ

C [[forward S]] ρ σ =
     [out-stream → send-msg
       (set-field "Resent-by" (concat ρ user-name (get-field "Resent-by" σ message))
         (set-field "Subject" (concat "Fwd: " (get-field "subject" σ message))
           (set-body (msg-to-string σ message)
             (set-field "From" ρ username
               (set-field "To" (S[[S]])
                 (set-field "Date" σ date (new-msg))))))) σ out-stream] σ

C [[reply S1]] ρ σ =
     [out-stream - send-msg
       (set-field "Subject" (concat "Re: " (get-field "Subject" σ message))
         (set-body (S[[S1]])
           (set-field "From" ρ user-name
             (set-field "To" (get-field "From" σ message)
               (set-field "Date" σ date (new-msg)))))) σ out-stream] σ

B : BoolExpr - DynamicState - DynamicState
B : [[match S1 S2]] σ = match (get-field (S[[S1]]) σ message) (S[[S2]])
```

**Figura 5:** Funções de avaliação definidas para a MAILSH [Consel, Marlet, 1998]

### 3.2.1.5 Máquina abstrata

Nesta fase é definida uma máquina abstrata. Uma maquina abstrata é um modelo computacional que suporta a implementação de todos os sistemas de uma família de sistemas. Em particular, uma vez que uma máquina abstrata pode expressar um grande número de aplicações em um domínio e uma LED expressa apenas um subconjunto delas, mais de uma LED's pode compartilhar a mesma máquina abstrata. Por exemplo, pode ser útil ter diferentes LED's para cada tipo de usuário; uma LED poderia ser usada para gerenciar todo um sistema de Banco de

dados, enquanto um subconjunto dessa LED poderia ser definida apenas para expressar consultas.

Por exemplo, a **Figura 6** mostra a definição de uma máquina abstrata para a MAILSH [Consel, Marlet, 1998].

C : *Command* → *StaticState* → *AbsMachState* → *AbsMachState* where
*StaticState* = *FoldersHierarchy* x *UserName*
*AbsMachState* = *FolderFiles* x *OutStream* x *CmdStream* x *Date*
x *Message* x *Message*

C [[$C_1$;$C_2$]] ρ = (C[[$C_2$]] ρ) o (C[[$C_1$]] ρ)
C [[if B then $C_1$ else $C_2$]] ρ = cond (B[[B]]) (C[[$C_1$]] ρ) (C[[$C_2$]] ρ)
C [[skip]] ρ = no-op
C [[copy F]] $\rho\ \sigma$ =
  let *v* = get-filename (*F*[[*F*]]) $\rho$ folder-hierarchy
  in ((write-folder *v*) o
    (add-msg) o
    (set-field, "Delivery-Date" $\sigma_{date}$) o
    (read-folder *v*)) $\sigma$

C [[forward S]] $\rho\ \sigma$ =
  ((send-msg) o
  (set-field$_c$ "Resent-by" (concat $\rho$ user-name (*get-field$_i$* "Resent-by" σ)) o
  (set-field$_c$ "Subject" (concat "Fwd: " (*get-field$_i$* "subject" σ)) o
  (set-body$_c$ (*msg-to-string$_i$* σ)) o
  (set-field$_c$ "From" $\rho$ username) o
  (set-field$_c$ "To" (S[[S]]) o
  (set-field$_c$ "Date" $\sigma$ date ) o
(new-msg$_c$)) $\sigma$

C [[reply $S_1$]] $\rho\ \sigma$ =
  ((send-msg) o
  (set-field$_c$ "Subject" (concat "Re: " (get-field$_i$ "Subject" $\sigma$))) o
  (set-body$_c$ (S[[$S_1$]]) o
  (set-field$_c$ "From" $\rho$ user-name) o
  (set-field$_c$ "To" (get-field$_i$ "From" $\sigma$)) o
  (set-field$_c$ "Date" $\sigma$ date) o
(new-msg$_c$)) $\sigma$

C[[pipe S]] $\rho$ = pipe-msg (S[[S]])

B : *BoolExpr - AbsMachState - AbsMachState*
B : [[match $S_1$ $S_2$]] $\sigma$ = match (*get-field$_i$* (S[[$S_1$]]) $\sigma$) (S[[$S_2$]])

**Figura 6:** Máquina abstrata definida para a MAILSH [Consel, Marlet, 1998]

### 3.2.1.6 Implementação

A implementação de uma LED pode ser derivada a partir da implementação de suas funções de avaliação e da implementação da correspondente maquina abstrata.

Assim como nas linguagens de programação de propósito geral, as LED's podem ser implementadas por um interpretador ou compilador. O interpretador é geralmente a mais fácil implementação, uma vez que processa um programa na presença dos dados e portanto produz diretamente uma resposta. Em contraste, um compilador produz um programa que só quando executado produz um resultado.

```
cond (match (get-field_i "Subject") "DSL") (
    new-msg;
    set-field_c "Date" (date);
    set-field_c "To" "jake";
    set-field_c "From" "bob";
    set-body_c (msg-to-string_i);
    set-field_c "Subject" (concat "Fwd: " (get-field_i "Subject"));
    set-field_c "Resent-by" (concat "bob" (get-field_i "Resent-by"));
    send-msg;
    read-folder "/home/bob/Mail/Research/Lang/DSL";
    set-field_c "Delivery-Date" (date);
    write-folder "/home/bob/Mail/Research/Lang/DSL"
) (
cond (match (get-field_i "From") "hotmail.com") (
   new-msg;
   set-field_c "Date" (date);
   set-field_c "To" (get-field, "From");
   set-field_c "From" "bob";
   set-body_c "Leave me alone!";
   set-field_c "Subject" (concat "Re: " (get-field_i "Subject"));
   send-msg;
) (
   cond (match (get-field_i "Subject") "seminar") (
      pipe-msg "agenda --stdin"
   ) (
     no-op)))
```

**Figura 7:** Exemplo de programa escrito com a MAILSH [Consel, Marlet, 1998]

### 3.2.1.7 Avaliação parcial

Enquanto a interpretação é mais flexível, a compilação é mais eficiente. Para obter o melhor das duas abordagens é usada uma técnica de transformação de programa chamada avaliação parcial [Consel, Danvy, 1993] [Jones, Gomard, Sestoft, 1993] para transformar automaticamente um programa escrito em uma LED em um programa compilado, usando um interpretador. A **Figura 7**, mostra um programa escrito na linguagem MAILSH.

### 3.2.2 Metodologia de Mauw, Wiersma e Willemse

A metodologia proposta por Mauw, Wiersma e Willemse [Mauw, Wiersma, Willemse, 2002] para o desenvolvimento de LED's baseia-se no conhecimento e técnicas que são necessárias para a condução da Análise de Domínio como ODM [Simos, 1996], FODA [Cohen, Kang, Hess, Peterson, 1990] [Krut, 1993] e DRACO [Neighbors, 1984]. O desenvolvimento da LED é descrita como uma coleção de artefatos. Esses artefatos incluem a definição da sintaxe, semântica e pragmática.

A metodologia de Mauw, Wiersma e Willemse compreende três fases: identificação do domínio do problema, identificação do espaço do problema e formulação da definição da linguagem. A **Figura 8** dá uma visão geral da metodologia, mostrando os artefatos produzidos (sintaxe, semântica e pragmática) e as diferentes opções para a especificação destes três.

#### 3.2.2.1 Identificação do domínio do problema

Nesta fase, o foco é em uma classe de problemas ao invés de um problema específico. Uma análise de domínio é necessária para possibilitar uma identificação completa e precisa de todos os conceitos essenciais no domínio do problema. Para tanto, podem ser usadas diferentes técnicas de Análise de Domínio existentes como ODM [Simos, 1996], FODA [Cohen,Kang, Hess, Peterson, 1990] [Krut, 1993] ou DRACO [Neighbors, 1984].

O domínio do problema geralmente é bem mais amplo tanto em generalidade dos conceitos quanto no número de conceitos do que é realmente necessário para resolver o problema específico.

#### 3.2.2.2 Identificação do Espaço do Problema

Ao contrário da fase anterior que consiste em uma coleta exaustiva de todos os conceitos de uma área de aplicação, nesta fase, uma restrição do domínio

do problema é necessária. Decisões de projeto têm que ser tomadas e tais decisões geralmente levam a conceitos não identificados no domínio do problema. Todos os conceitos identificados são então classificados em três tipos:

- *Conceitos irrelevantes:* Um conceito é irrelevante se ele não tem participação na solução de um problema específico.
- *Conceitos Variáveis:* São conceitos que variam entre instâncias de problema ou dentro de uma mesma instância.
- *Conceitos Fixos:* São conceitos que são idênticos para todos os problemas pertencentes a uma classe de problemas.

O conjunto de todos os conceitos fixos e variáveis é chamado espaço do problema (**Figura 8**).

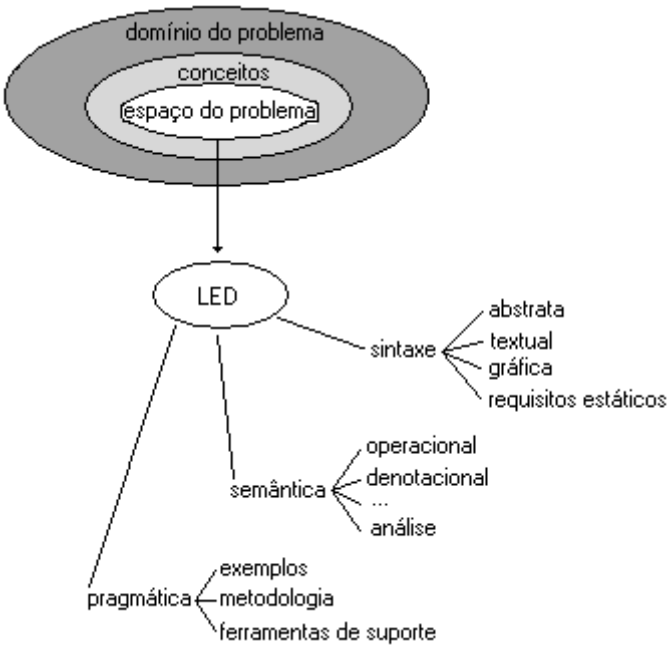


**Figura 8:** Metodologia de Mauw, Wiersma e Willemse [Mauw, Wiersma, Willemse, 2002]

3.2.2.3 Formulação da Definição da Linguagem

Esta fase é dividida em três subfases nas quais são definidas a semântica, a sintaxe e a pragmática da LED:

*Definição da Sintaxe*: A interface da linguagem com seus usuários é definida pela sua sintaxe. A sintaxe da LED consiste de expressões formadas a partir dos conceitos variáveis que foram identificados no espaço do problema.

Uma linguagem pode ter uma ou mais descrições sintáticas. Essas descrições dependem do uso que será feito da linguagem. Os formatos mais populares são: sintaxe textual ou linear e sintaxe gráfica.

*Definição da Semântica*: Assim como para as LPG's, existem várias abordagens para definir a semântica de uma linguagem. A escolha da abordagem semântica mais adequada depende das características da linguagem, por exemplo, se é uma linguagem textual ou gráfica. A maioria das abordagens semânticas leva em consideração a existência de um domínio semântico. Esse domínio geralmente consiste de um conjunto de entidades e seus relacionamentos. Expressões da LED se relacionam às entidades do domínio e obtêm seus significados através das propriedades dessas entidades. Três das principais abordagens para definição da semântica são semântica operacional, semântica denotacional e semântica axiomática.

*Definição da Pragmática*: A pragmática de uma linguagem se relaciona a todos os aspectos de seu uso. Ela pode ser expressa através de documentação e de um conjunto de exemplos que ilustrem sua aplicação.

### 3.2.3 Metodologias baseadas em ontologias

Uma ontologia é uma especificação explícita de uma conceitualização, ou seja, uma especificação explícita dos dos objetos, conceitos e outras entidades que assumimos que existem em uma área de interesse, além das relações entre esses conceitos e restrições expressadas através de axiomas [Gruber, 1995]. Quando são definidas ontologias, trabalha-se diretamente com o domínio em si,

independentemente dos módulos e sistemas que serão construídos para serem reutilizados [Guarino, 1998]. As ontologias permitem um nível mais alto de abstração, o reuso do conhecimento. Uma ontologia pode ser descrita como uma hierarquia de conceitos relacionados e em casos mais sofisticados, axiomas podem ser adicionados para expressar outros relacionamentos entre conceitos e para restringir sua interpretação pretendida.

3.2.3.1 Metodologia de Guizzardi

Esta é uma metodologia para o desenvolvimento de LED's proposta por Guizzardi, Ferreira Pires e Sinderen [Guizzardi, Pires, Sinderen, 2002]. A metodologia não é definida em detalhes, precisando ser ainda mais elaborada. Entretanto, como já foi mencionado, essa proposta tem um aspecto interessante e que foi de particular interesse para o presente trabalho. É que a metodologia faz uso de ontologias para o desenvolvimento de LED's. A metodologia é constituída de quatro fases: selecionar ou desenvolver uma ontologia de domínio, extrair da ontologia de domínio o meta-modelo inicial da linguagem, refinar o meta-modelo da linguagem e desenvolver a sintaxe concreta da linguagem.

- *Selecionar ou desenvolver uma ontologia de domínio*

Se uma ontologia que descreva o domínio de interesse já existir, faz-se sua reutilização. Caso contrário, é necessário desenvolver uma ontologia. Há bastante conhecimento a respeito de como modelar, formalizar, reutilizar e integrar ontologias. Os autores se referem a [Fernández, Gomez-Perez, Pazos, 1999] para uma metodologia de engenharia ontológica.

- *Extrair da ontologia de domínio o meta-modelo inicial da linguagem*

Nesta fase é necessário definir um procedimento para selecionar os conceitos e correspondentes axiomas da ontologia de domínio que são relevantes para a linguagem.

- *Refinar o meta-modelo da linguagem*

Nesta fase são acrescentados conceitos e axiomas suficientes para a definição da semântica da linguagem. Se o meta-modelo produzido na fase anterior não possuir axiomas suficientes para definir a semântica da linguagem, os desenvolvedores devem definir a semântica em termos dos axiomas da ontologia original.

- *Desenvolver a sintaxe concreta da linguagem*

Nesta fase, o meta-modelo é usado no desenvolvimento de um sistema de representação eficiente para ser usado como a sintaxe concreta da LED. A eficiência pragmática é aumentada à medida que uma representação exibe as mesmas propriedades da abstração que representa. Por essa, razão o meta-modelo deve representar os axiomas que restringem os relacionamentos entre as entidades do domínio (ex: transitividade, irreflexibilidade).

### 3.2.4 SDA

SDA [Widen, Hook, 1998] é um método para o projeto e implementação de LED's que deve ser usado para estender métodos existentes da Engenharia de Domínio. O método SDA é uma abordagem matemática para o projeto e implementação de LED's. Os conceitos fundamentais utilizados pelo SDA são projeto de linguagem baseado em semântica e modelos matemáticos.

Um benefício do uso de modelos matemáticos é tornar possível expressar os requisitos do domínio do problema formalmente. Os modelos matemáticos

permitem que as abstrações que eles representam sejam conhecidas e bem entendidas. Uma vez que os modelos usam um formalismo matemático eles apresentam o benefício adicional de capturar propriedades do domínio que podem ser analisadas formalmente.

O projeto de linguagem baseado em semântica é outro conceito fundamental em SDA. A metodologia usa semântica denotacional para capturar as especificações das LED's. A semântica denotacional abstrai detalhes computacionais que podem estar presentes em outras abordagens como semântica operacional. De fato, a semântica denotacional pode também ser usada na especificação de linguagens não computacionais.

SDA é constituído de três fases: Analise do domínio, definição da linguagem e implementação do gerador. Para ilustrar a aplicação da metodologia são usados exemplos referentes ao desenvolvimento de uma LED para a especificação de sistemas de controle de bóias meteorológicas (Estações Meteorológicas Flutuantes - EMF). Os sistemas EMF constituem uma família de sistemas que provêem dados sobre as condições climáticas para orientar o tráfego marítimo e aéreo (**Figura 9**). 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.

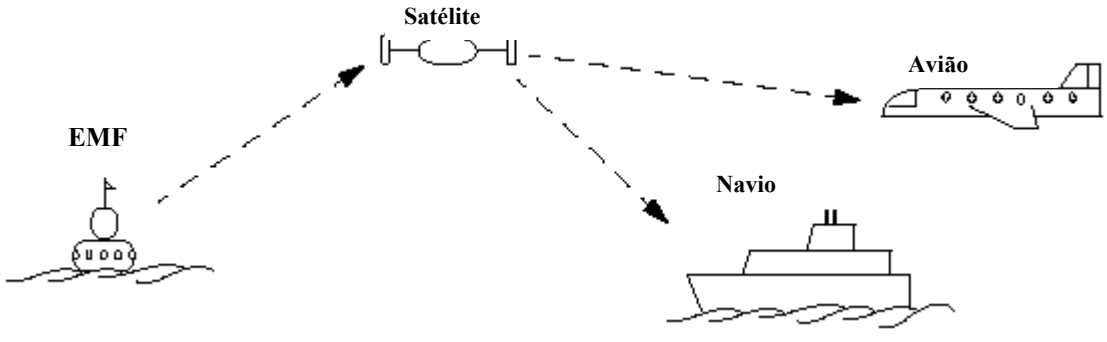


**Figura 9**: Contexto das bóias meteorológicas [Widen, 1995]

3.2.4.1 Analise do Domínio

Esta fase é a interface entre as atividades da Engenharia de Domínio e as atividades do projeto da LED. SDA faz uso dos conhecimentos fornecidos pelos especialistas do domínio para prover um documento de análise de domínio que apresente a maioria da informação necessária para o projeto da LED e do gerador de aplicação. Esta fase é dividida em quatro etapas: capturar uma definição escrita do domínio, formulação do modelo de domínio, definir o modelo de domínio e capturar a interface do sistema legado.

- *Capturar uma definição escrita do domínio*

O propósito deste passo é capturar informalmente informações relevantes sobre o domínio. Este passo provê aos projetistas da LED um entendimento inicial do domínio. Durante esse processo a informação é sintetizada entre os vários desenvolvedores e validada pelos especialistas do domínio para garantir sua veracidade.

Por exemplo, A **Figura 10** mostra uma descrição simplificada do domínio das bóias meteorológicas.

*O domínio que está sendo estudado é o de bóias meteorológicas. Os sistemas das bóias meteorológicas são uma família de sistemas que provêem dados climáticos aos tráfegos marítimo e aéreo sobre o mar. As bóias coletam dados como, temperatura da água e velocidade do vento, através de vários sensores. Cada bóia possui um transmissor de rádio para enviar os dados meteorológicos a satélites que os repassam ao trafego marítimo e aéreo. Para cada bóia é desenvolvido um sistema que controla sua operação. Esse sistema deve coordenar a operação dos sensores e o envio de relatórios aos satélites. As bóias irão conter um ou mais computadores, sensores para medir, por exemplo, a velocidade do vento e a temperatura da água. Eventualmente outros sensores podem ser acrescentados.*

**Figura 10**: Definição do domínio das bóias meteorológicas [Widen, 1995]

Outro artefato produzido neste passo é a arquitetura da família de sistemas. O propósito da arquitetura é ajudar a definir o escopo do domínio.

Por exemplo, a **Figura 11** mostra a arquitetura para o domínio das bóias meteorológicas.

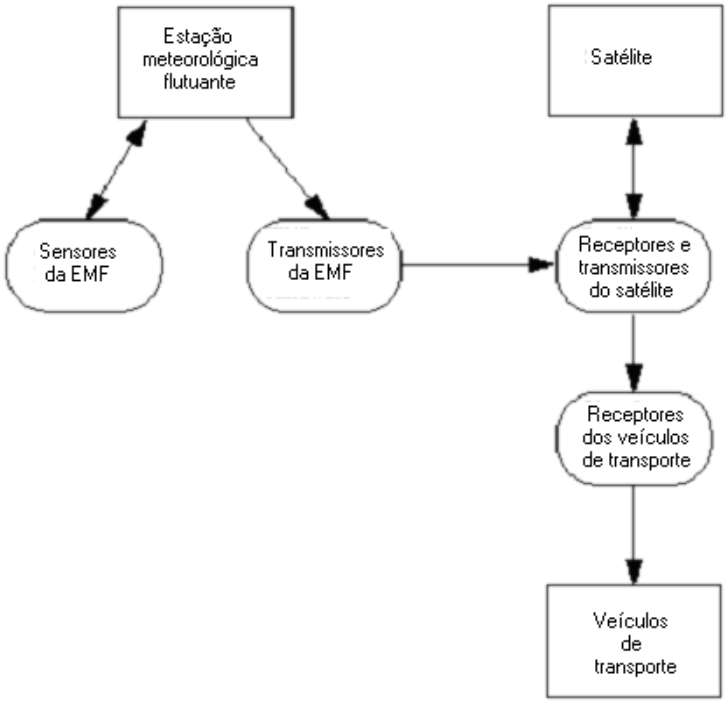


**Figura 11**: Arquitetura para o domínio das bóias meteorológicas [Widen, 1995]

Por último, é capturado um conjunto inicial de requisitos do domínio. Os requisitos definem os comportamentos e restrições que devem estar presentes em todas as instâncias de um domínio. Por exemplo, A **Figura 12** mostra alguns dos requisitos para o domínio das bóias meteorológicas.

*As medições feitas pelos sensores são coletadas regularmente. Periodicamente a média da temperatura e a média da velocidade do vento são enviados aos satélites em um relatório. O sistema deve também enviar um relatório de histórico das últimas dez medições.*

- *As medições tem que ser feitas pelos sensores em intervalos específicos.*
- *As médias das medições tem que ser informadas em intervalos específicos.*
- *O sistema das bóias deve poder armazenar mais de 10 medições feitas por cada sensor.*

**Figura 12**: Alguns requisitos para o domínio das bóias meteorológicas [Widen, 1995]

- *Formulação do Modelo de Domínio*

O objetivo desta atividade é formular um modelo formal do domínio. Um modelo de domínio é uma abstração das entidades do domínio. Ele captura informações de alto nível a respeito do domínio sem levar em consideração detalhes de implementação. Para especificar uma nova instância do domínio apenas suas características particulares (aquelas que diferem um sistema de outro em uma família de sistemas) precisam ser observadas. A **Figura 13** mostra o modelo formal para o domínio das bóias meteorológicas.

```
data FWS =
     FWS of (sensor list) * sensor-period * transmitter-type * transmitter-period *
     msg-format;
data msg-format = short-msg | long-msg;
data transmitter-period =
     transmitter-period of int;
data transmitter-type = am | fm;
data sensor-period=
     sensor-period of int;
data sensor =
     sensor of sensor-type * sensor-address;
data sensor-type = AirTemp | WindTemp | WindSpeed;
data sensor-address =
     sensor-address of int;
```

**Figura 13:** Modelo formal do domínio das bóias meteorológicas [Widen, 1995]

*Definir o Modelo de Solução*

O foco desta atividade é analisar, capturar e formatar o modelo de solução do domínio. O modelo de solução é uma abstração do domínio assim como o modelo de domínio. Entretanto ele está no nível da solução ao invés do nível do problema.

Por exemplo, a **Figura 14** mostra um modelo de solução parcial para o domínio das bóias meteorológicas.

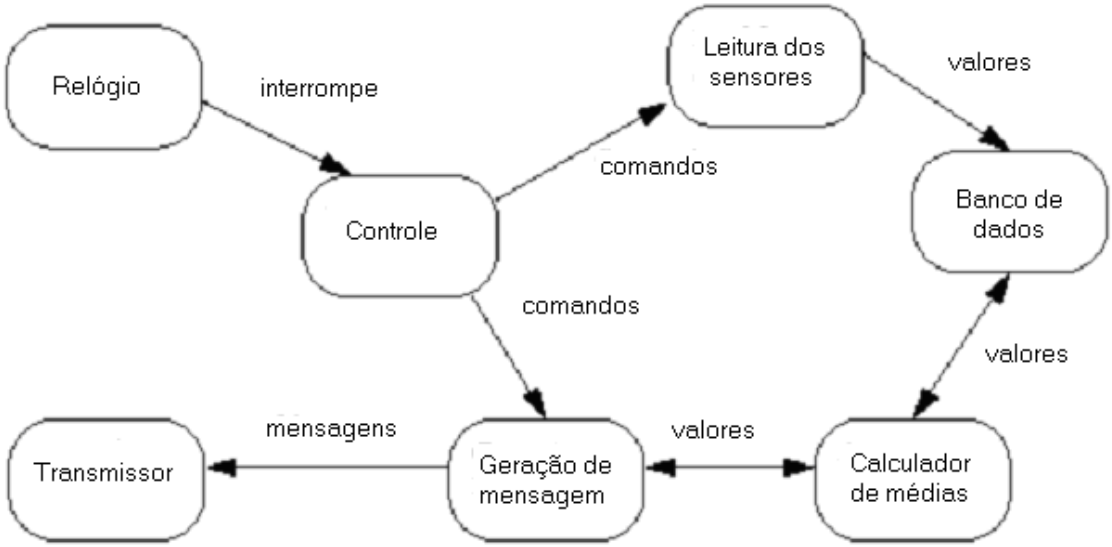


**Figura 14**: Modelo de solução parcial para as bóias meteorológicas [Widen, 1995]

- *Capturar a Interface do Sistema Legado*

O ambiente de execução ou sistema legado, se existir, deve ser analisado para garantir que os sistemas desenvolvidos com a LED irão se integrar corretamente. As tarefas realizadas nesta atividade capturam as interfaces e restrições, com o objetivo de garantir uma boa integração entre os sistemas.

- *Validar os modelos*

Neste ponto é possível fazer uma validação preliminar dos modelos, mostrando que uma solução pode ser derivada a partir de uma instância do problema descrito no modelo.

3.2.4.2 Definição da Linguagem

Esta fase tem dois objetivos. O primeiro é preparar uma definição inicial da linguagem. O segundo objetivo é capturar a semântica da linguagem na forma de um interpretador.

- *Definição Inicial da Linguagem*

Esta atividade consiste inicialmente em definir um vocabulário e notação para a LED. A LED preferencialmente deve usar a mesma notação e conceitos que

os engenheiros de aplicação usam para expressar suas soluções, isso torna o seu uso mais intuitivo.

É também definida a sintaxe da LED. A sintaxe é capturada como uma gramática que especifica as combinações válidas do vocabulário da LED.

- *Formalizar a semântica*

O propósito deste atividade é produzir um interpretador que captura a semântica da linguagem formalmente de acordo com a abordagem de semântica denotacional.

3.2.4.3 Implementação do gerador

Esta é a última fase do método SDA. Para cada interação do método SDA em um domínio, a maioria do trabalho já foi feito antes desta fase ser alcançada. Os modelos estão definidos, um interpretador já existe para a sintaxe abstrata da LED e a LED também já está definida.

Entretanto ainda há trabalho a ser feito. O objetivo desta fase é projetar, implementar e integrar todas as partes de um gerador e também produzir um ambiente de usuário.

## 3.3 Análise comparativa das metodologias estudadas

A **Tabela 3** apresenta uma comparação entre as metodologias estudadas segundo os seguintes critérios:

1. A abordagem utilizada para a especificação da semântica das LED's.
2. A abordagem utilizada para a especificação da sintaxe das LED's.
3. LED's desenvolvidas com a aplicação das metodologias.
4. Se a metodologia cobre apenas o projeto ou o projeto e implementação de LED's.

5. Se as metodologias especificam uma técnica de análise de domínio a ser utilizada.
6. Como as LED's são implementadas.
7. Se incluem um roteiro para a documentação da LED.
8. Se fazem uma análise dos aspectos fixos e variáveis dos sistemas pertencentes a um domínio.

| | **Especificação da semântica** | **Especificação da sintaxe** | **LED's desenvolvidas** | **Especificação/ Projeto/ Implementação** | **Técnica de Análise de Domínio** | **Implementação** | **Documentação** | **Analise de conceitos fixos e variáveis** |
|---|---|---|---|---|---|---|---|---|
| **Sprint** | Semântica denotacional | Gramática BNF | GAL, PLAN-P [Thibault, 1998] | Especificação, Projeto e implementação | Não especificado | Interpretador e avaliação parcial | Não | Não |
| **Metodologia de Mauw, Wiersma e Willemse** | Não especificado | BNF,gramaticas gráficas, gramática de atributos, predicados lógicos | Regulagem de sinais de transito [Mauw, Wiersma, Willemse, 2002] | Especificação e Projeto | ODM, FODA, DRACO | - | Sim | Não |
| **Metodologia de Guizzardi** | Ontologia | Não especificado | - | Especificação e Projeto | Não específicado | - | Não | Não |
| **SDA** | Semântica denotacional | Não especificado | MSL [Widen, Hook, 1998] | Projeto e implementação | Técnica própria | Interpretador | Não | Não |

**Tabela 3:** Análise comparativa das metodologias para o desenvolvimento de LED's

Nenhuma das metodologias estudadas possui um roteiro próprio para a classificação dos conceitos em fixos e variáveis, para tanto devem ser utilizadas técnicas como a análise de semelhanças do método FAST [Widen, 1995]. Uma deficiência encontrada em todas as metodologias estudadas é que elas ignoram a documentação, exceção feita a metodologia de Mauw, Wiersma e Willemse [Mauw, Wiersma, Willemse, 2002], entretanto ela não dá especificações mais precisas nesse sentido. Mesmo nas metodologias que não especificam como deve ser descrita a sintaxe, em seus estudos de caso são usadas gramáticas BNF. Das metodologias estudadas, *Sprint* e a *Metodologia de Guizzardi* não especificam a técnica de análise de domínio a ser utilizada. A metodologia de Mauw prevê a utilização de ODM

[Simos, 1996], FODA [Cohen, Kang, Hess, Peterson, 1990] [Krut, 1993] ou DRACO [Neighbors, 1984] como técnicas de análise de domínio. Já SDA define um processo próprio.

### 3.4 Considerações finais

Neste capítulo foi feita uma discussão a respeito das LED's, incluindo conceitos, exemplos, aspectos favoráveis e desfavoráveis do uso dessa abordagem.

Foram também apresentadas algumas das mais bem documentadas metodologias para o desenvolvimento de LED's, que serviram de subsídio para o desenvolvimento da TOD-LED, a técnica para o desenvolvimento de LED's proposta neste trabalho. Por fim, foi feita uma análise comparativa das metodologias estudadas.

No próximo capítulo é apresentada a GRAMO, uma técnica para a fase de análise de domínio da Engenharia de Domínio Multiagente. A GRAMO tem como produto modelos de domínio segundo o paradigma computacional de agentes, que são o insumo para a TOD-LED.

# 4 MODELOS DE DOMÍNIO BASEADOS EM ONTOLOGIAS

O principal produto de qualquer técnica para a análise de domínio é um modelo de domínio. Um modelo de domínio é uma abstração de alto nível dependente de um domínio de aplicação particular, que representa a formulação de um problema, conhecimento ou atividades do mundo real.

Neste capítulo é apresentada a técnica GRAMO [Faria, 2004] para a captura de requisitos genéricos de sistemas na Engenharia de Domínio Multiagente. A GRAMO [Faria, 2004] tem como produto um modelo de domínio baseado em ontologias. O modelo de domínio obtido como produto da GRAMO é resultante da instanciação da ontologia genérica ONTODUM [Faria, 2004].

A ONTODUM [Faria, 2004] é uma ontologia genérica que representa o conhecimento de metodologias da análise de requisitos e de métodos de análise de domínio, visando a captura e especificação de requisitos genéricos de sistemas, segundo o paradigma computacional de agentes. Na seção 4.1 é apresentada a ONTODUM e na seção 4.2 é apresentada a técnica GRAMO.

## 4.1 ONTODUM: Uma Ontologia genérica para a Análise de Domínio e Usuários na Engenharia de Domínio Multiagente

A ONTODUM [Faria, 2004] é uma ontologia genérica que representa o conhecimento de técnicas da análise de requisitos, da análise de domínio e da modelagem de usuários, para a captura e especificação dos requisitos genéricos de uma família de aplicações, segundo o paradigma computacional de agentes. Ela guia a captura e especificação do conhecimento dos conceitos do domínio, das tarefas a serem realizadas no domínio e dos perfis dos usuários que atuam no domínio.

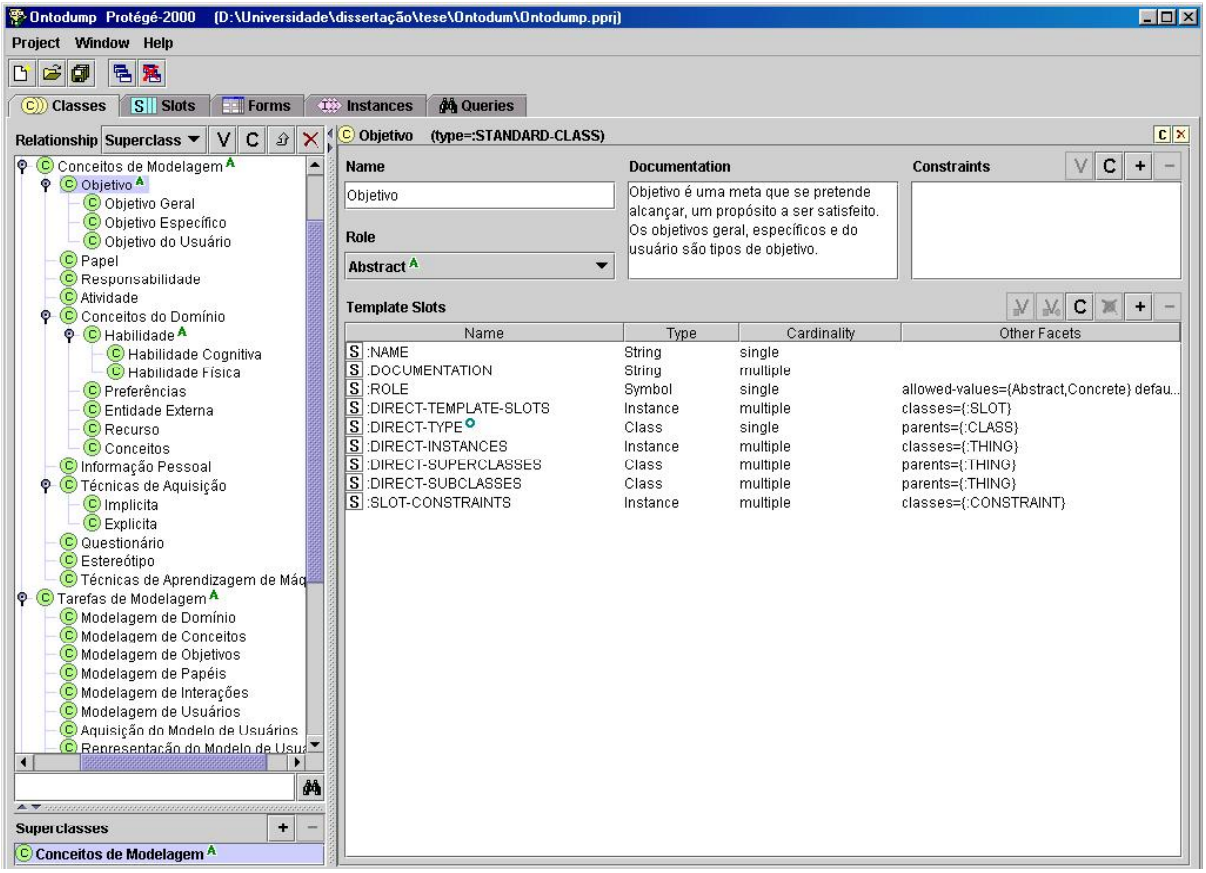


**Figura 15:** Hierarquia de meta-classes da ONTODUM e a meta-classe *Objetivo* [Faria, 2004]

A solução por ela proposta é baseada no levantamento de conceitos relevantes do domínio e na identificação de um objetivo geral a ser alcançado. Do objetivo geral derivam objetivos específicos que são atingidos através do cumprimento de responsabilidades. As responsabilidades são associadas a papéis e são exercidas através da execução de atividades, para tanto os papéis interagem entre si e fazem uso de recursos, no intuito de realizar suas atribuições. A ONTODUM consiste de uma hierarquia de meta-classes que reflete como os conceitos discutidos anteriormente estão estruturados e se relacionam, como objetivo, papel, responsabilidade, atividade. As figuras seguintes dão uma visão geral da hierarquia de classes da ONTODUM no editor de ontologias Protègè [Protègè Project, 2002]. Na **Figura 15** é mostrada a meta-classe *Objetivo* da ONTODUM, que possui as subclasses *Objetivo Geral*, *Objetivo Específico* e *Objetivo*

*do Usuário*. Na **Figura 16** é mostrada a meta-classe *Papel* da ONTODUM, que possui os slots *tem responsabilidade* e *usa recurso* do tipo instância da meta-classe *Responsabilidade* e *Recurso*, respectivamente. Na **Figura 17** é mostrada a meta-classe *Informação Pessoal* da ONTODUM, que tem os slots *idade*, *endereço*, *formação*, *língua materna*, *ocupação* e *sexo*.

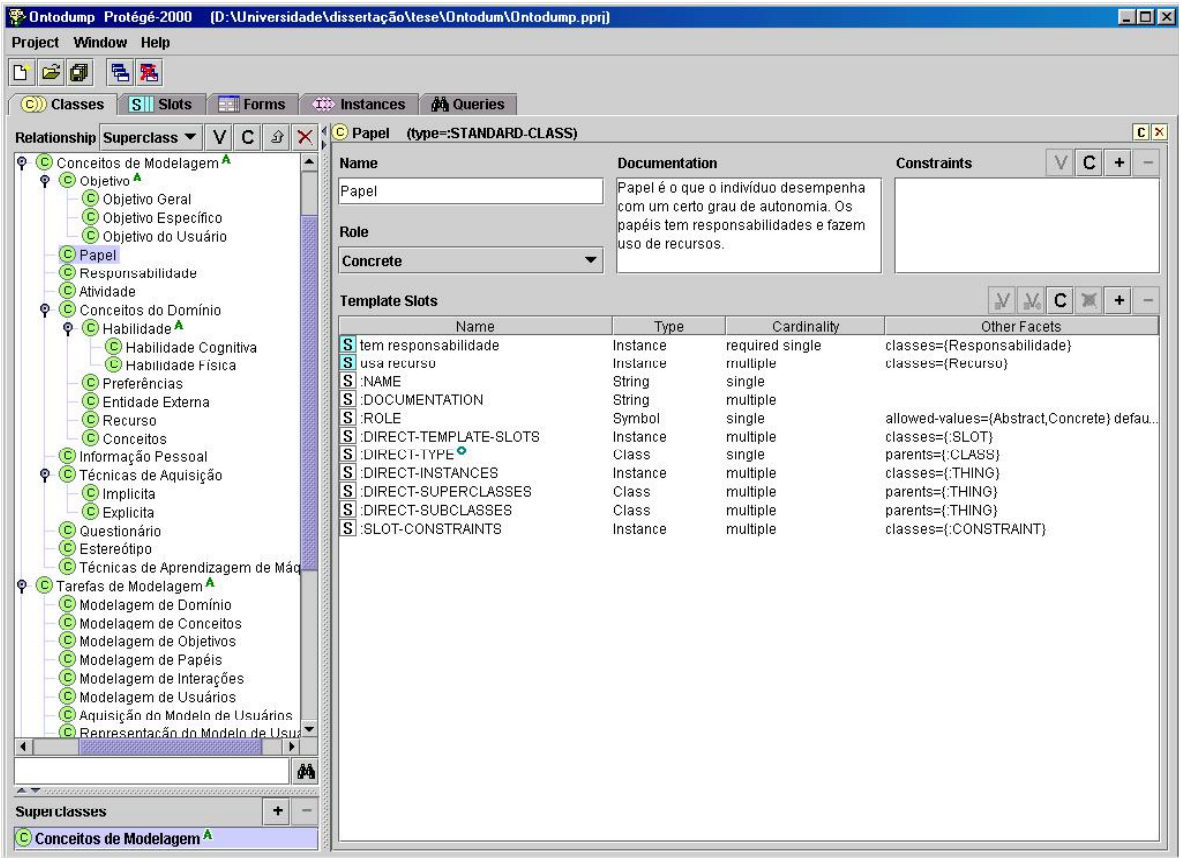


**Figura 16:** Hierarquia de meta-classes da ONTODUM e a meta-classe *Papel* [Faria, 2004]

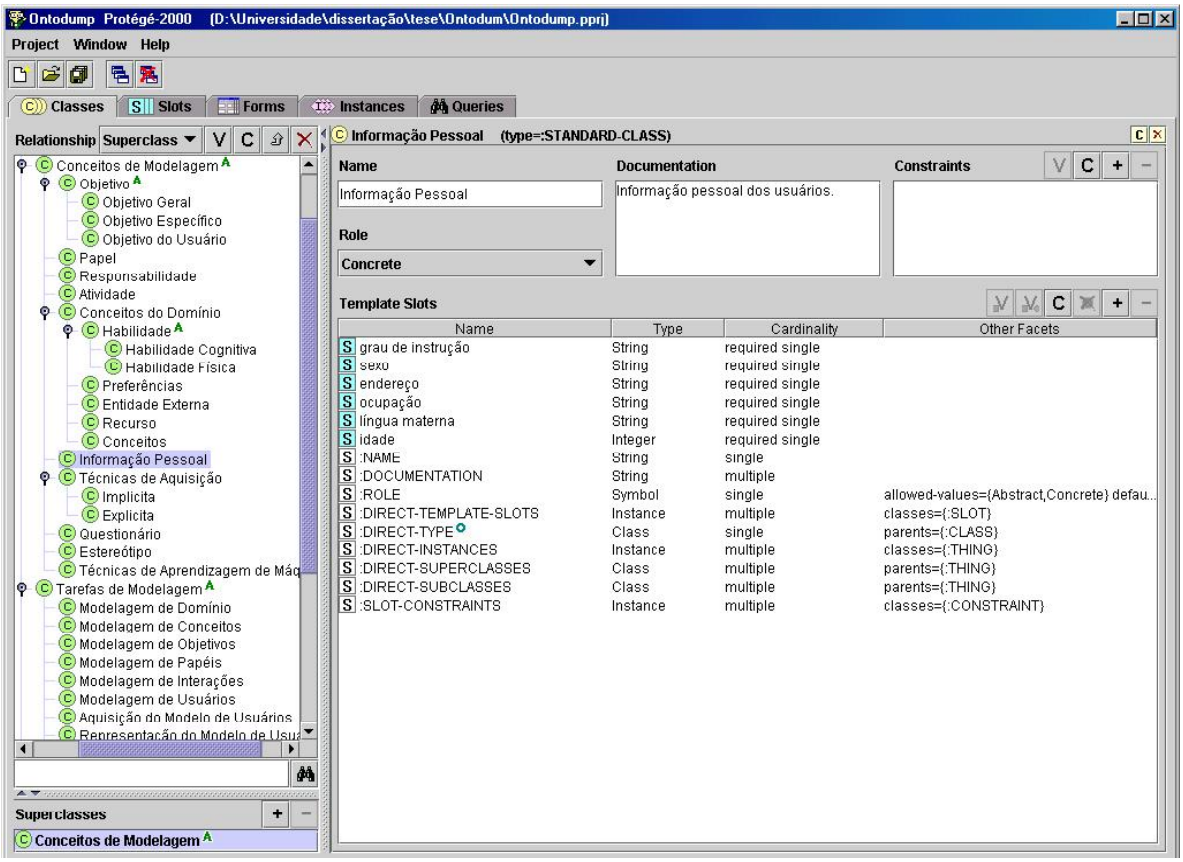


**Figura 17:** Hierarquia de meta-classes da ONTODUM e a meta-classe *Informação Pessoal* [Faria, 2004]

## 4.2 A técnica GRAMO

A técnica GRAMO (**Figura 18**) define as atividades a serem realizadas na construção de modelos de domínio e usuários. A construção de modelos de domínio é feita através da instanciação da ONTODUM com o conhecimento de um dado domínio de problema. A construção de modelos de usuários é feita através da instanciação da ONTODUM com o conhecimento sobre os usuários finais do sistema.


**Figura 18:** Insumos e produtos da GRAMO [Faria, 2004]

A técnica GRAMO consiste de duas fases: modelagem de domínio e modelagem de usuários (**Tabela 4**).

<table>
<tr><td rowspan="4">TÉCNICA GRAMO</td><td>Fases</td><td colspan="2">Atividades</td><td>Produtos</td></tr>
<tr><td rowspan="3">Modelagem de Domínio</td><td rowspan="3">Modelagem de Conceitos</td><td>Modelagem de Objetivos</td><td rowspan="3">Modelo de Domínio (Modelo de Conceitos, Modelo de Objetivos, Modelo de Papéis e Modelo de Interações)</td></tr>
<tr><td>Modelagem de Papéis</td></tr>
<tr><td>Modelagem de Interações</td></tr>
<tr><td rowspan="3">Modelagem de Usuários</td><td colspan="2">Aquisição</td><td rowspan="3">Modelo de Usuários</td></tr>
<tr><td colspan="2">Representação</td></tr>
<tr><td colspan="2">Manutenção</td></tr>
</table>

**Tabela 4:** Produtos e Atividades da técnica GRAMO [Faria, 2004]

A fase de modelagem de domínio consiste das seguintes atividades: *Modelagem de Conceitos*, *Modelagem de Objetivos*, *Modelagem de Papéis* e *Modelagem de Interações*, que geram os produtos *Modelo de Conceitos*, *Modelo de Objetivos*, *Modelo de Papéis* e *Modelo de Interações*. O produto desta fase é o *Modelo de Domínio* composto do *Modelo de Conceitos*, *de Objetivos*, *de Papéis* e *de Interações*.

A modelagem de conceitos é feita paralelamente as modelagens de objetivos, papéis e interações. Na modelagem de conceitos são identificados conceitos e relacionamentos do domínio, que são representados no *Modelo de Conceitos*. Na *Modelagem de Objetivos* são identificados o objetivo geral, os objetivos específicos e as responsabilidades, que são representados no *Modelo de Objetivos*. Na *Modelagem de Papéis* são identificados os papéis e seus respectivos atributos, que são representados no *Modelo de Papéis*. Na *Modelagem de Interações* são identificadas as interações que ocorrem entre papéis ou entre papéis e entidades externas, que são representados no *Modelo de Interações*.

A fase de *Modelagem de Usuários* consiste das seguintes atividades: *Aquisição*, *Representação* e *Manutenção* das informações dos usuários. O produto desta fase é o *Modelo de Usuários*.

Na aquisição das informações dos usuários são identificadas as informações dos usuários. Na representação das informações dos usuários é feita a construção do *Modelo de Usuários* a partir das informações coletadas na atividade de aquisição das informações dos usuários. Na manutenção das informações dos usuários é feita a aquisição das informações dos usuários e atualização do *Modelo de Usuários*. No capítulo 6 a técnica GRAMO é aplicada no desenvolvimento da ONTOINFO, um modelo de domínio para a área do acesso à informação dinâmica e não estruturada.

### 4.2.1 Modelagem de Domínio

A construção do Modelo de Domínio é feita através da instanciação da meta-classe *Modelagem de Domínio* (**Figura 19**), que cria a classe *Modelo de Domínio*, contendo a especificação dos conceitos e tarefas do domínio. A fase *Modelagem de Domínio* consiste das seguintes atividades: *modelagem de conceitos*, *modelagem de objetivos*, *modelagem de papéis* e *modelagem de interações*, que geram os produtos *modelo de conceitos*, *modelo de objetivos*, *modelo de papéis* e *modelo de interações*. O *modelo de domínio* é composto do *modelo de conceitos*, do *modelo de objetivos*, do *modelo de papéis* e do *modelo de interações*.

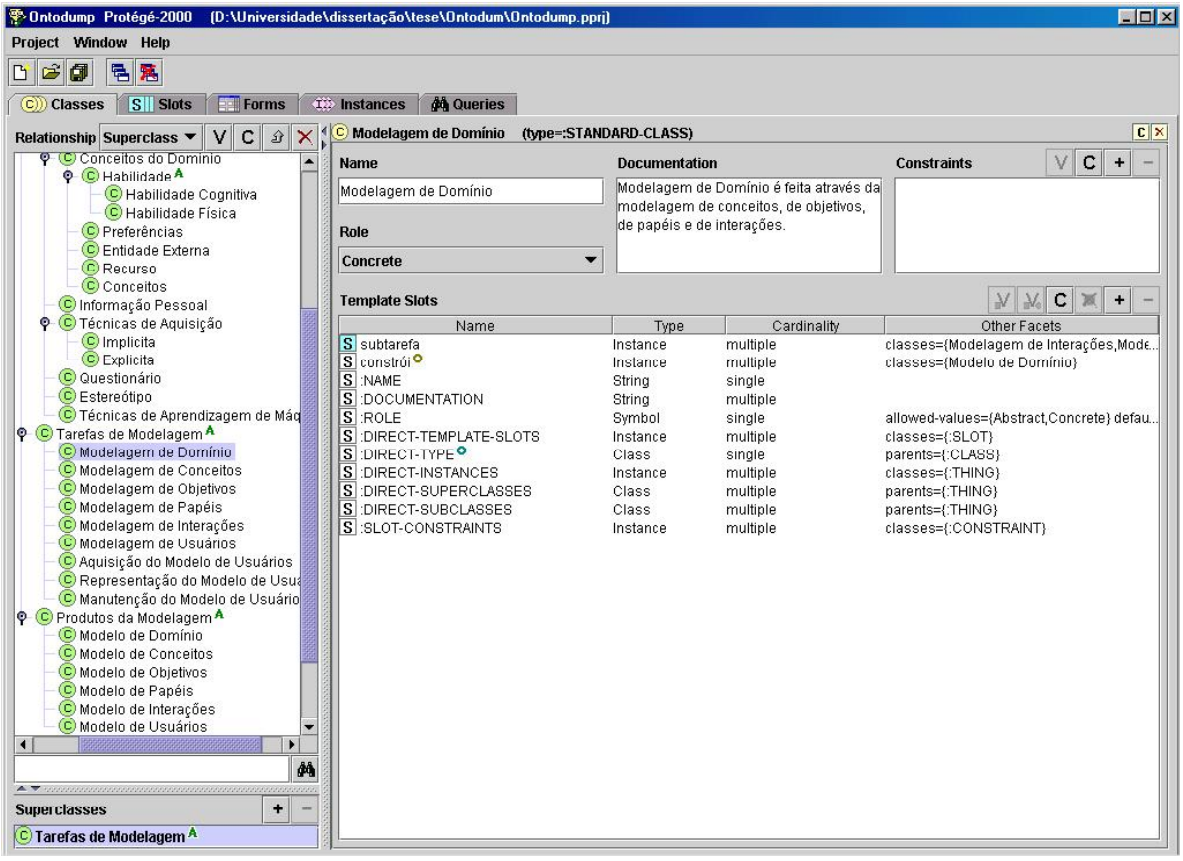


**Figura 19**: Meta-classe *Modelagem de Domínio* [Faria, 2004]

#### 4.2.1.1 Modelagem de Conceitos

A construção do *Modelo de Conceitos* é feita através da instanciação da meta-classe *Modelagem de Conceitos* (**Figura 20**), que cria a classe *Modelo de Conceitos*, contendo os conceitos e seus relacionamentos. Primeiramente são identificados conceitos e relacionamentos óbvios através de uma análise das fontes de informação relevantes no domínio como: livros, revistas, artigos, relatórios, especialistas do domínio e aplicações existentes. Posteriormente serão acrescentados conceitos do domínio identificados no decorrer das modelagens de objetivos, de papéis e de interações que se seguem. Por exemplo, alguns conceitos a serem modelados em aplicações para o acesso à informação dinâmica e não estruturada (recuperação e filtragem de informação) são:

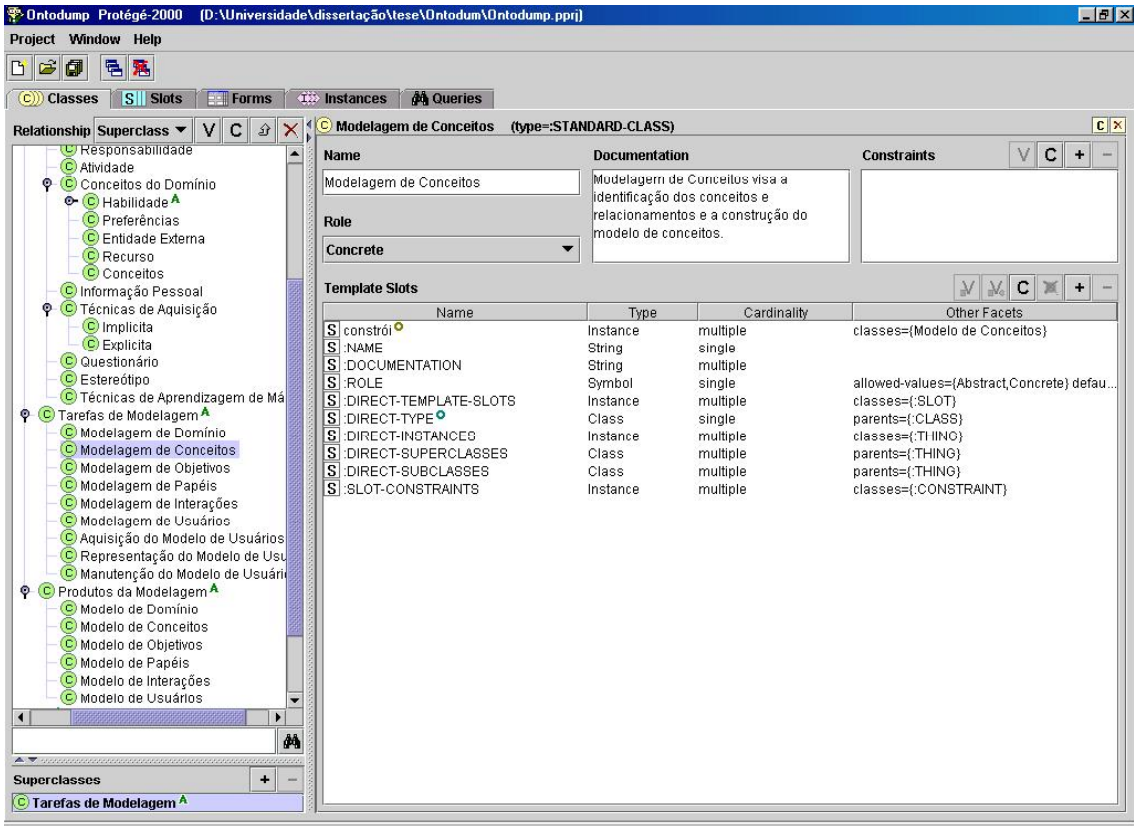


**Figura 20:** Meta-classe *Modelagem de Conceitos* [Faria, 2004]

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

A **Figura 21** mostra o exemplo do *Modelo de Conceitos* do acesso à informação dinâmica e não estruturada.

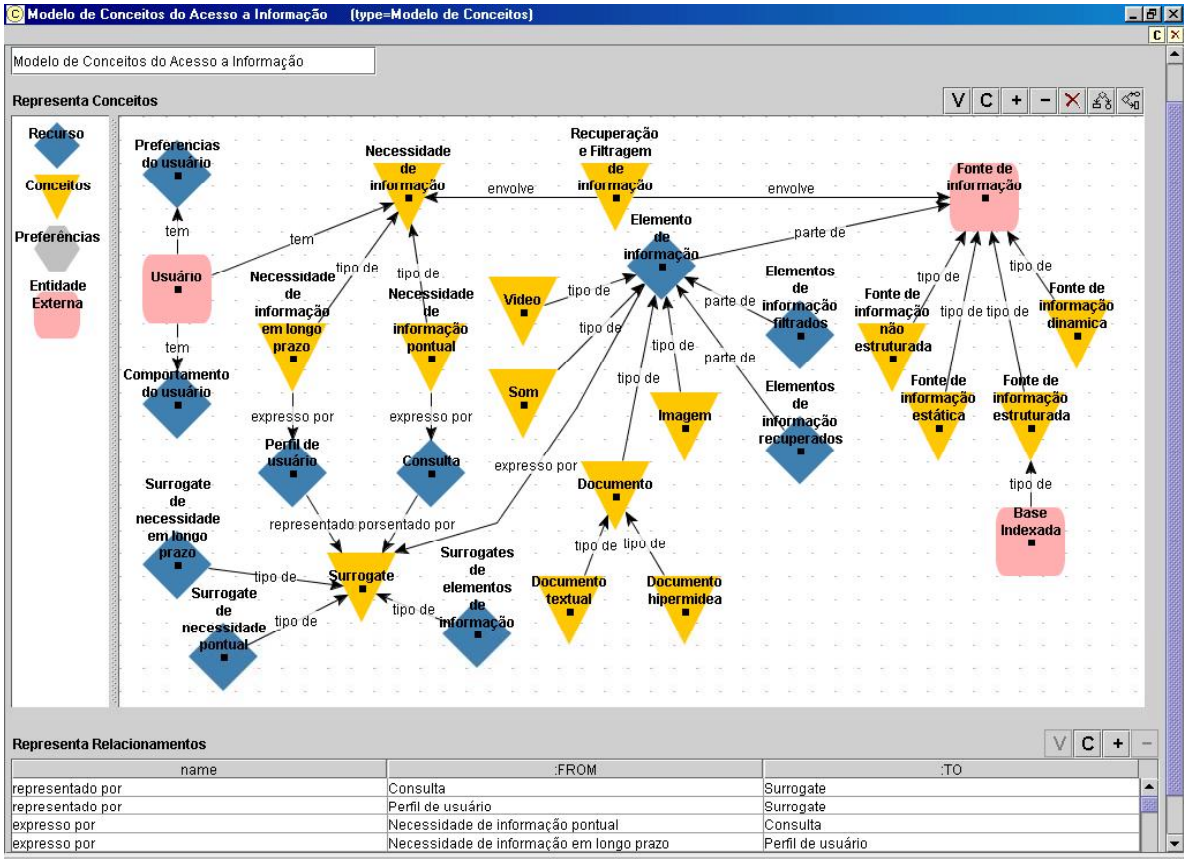


**Figura 21:** Exemplo do *Modelo de Conceitos* do Acesso à Informação

### 4.2.1.2 Modelagem de Objetivos

A construção do modelo de objetivos é feita através da instanciação da meta-classe *Modelagem de Objetivos* (**Figura 22**), que cria a classe *Modelo de Objetivos*, contendo os objetivos gerais e específicos e as responsabilidades. O *Objetivo geral* é identificado através do problema que o sistema se propõe a resolver. Os objetivos específicos são identificados através do refinamento ou especialização do *Objetivo geral*. As responsabilidades são identificadas a partir dos objetivos específicos, ou seja, as responsabilidades levam ao cumprimento dos objetivos específicos.

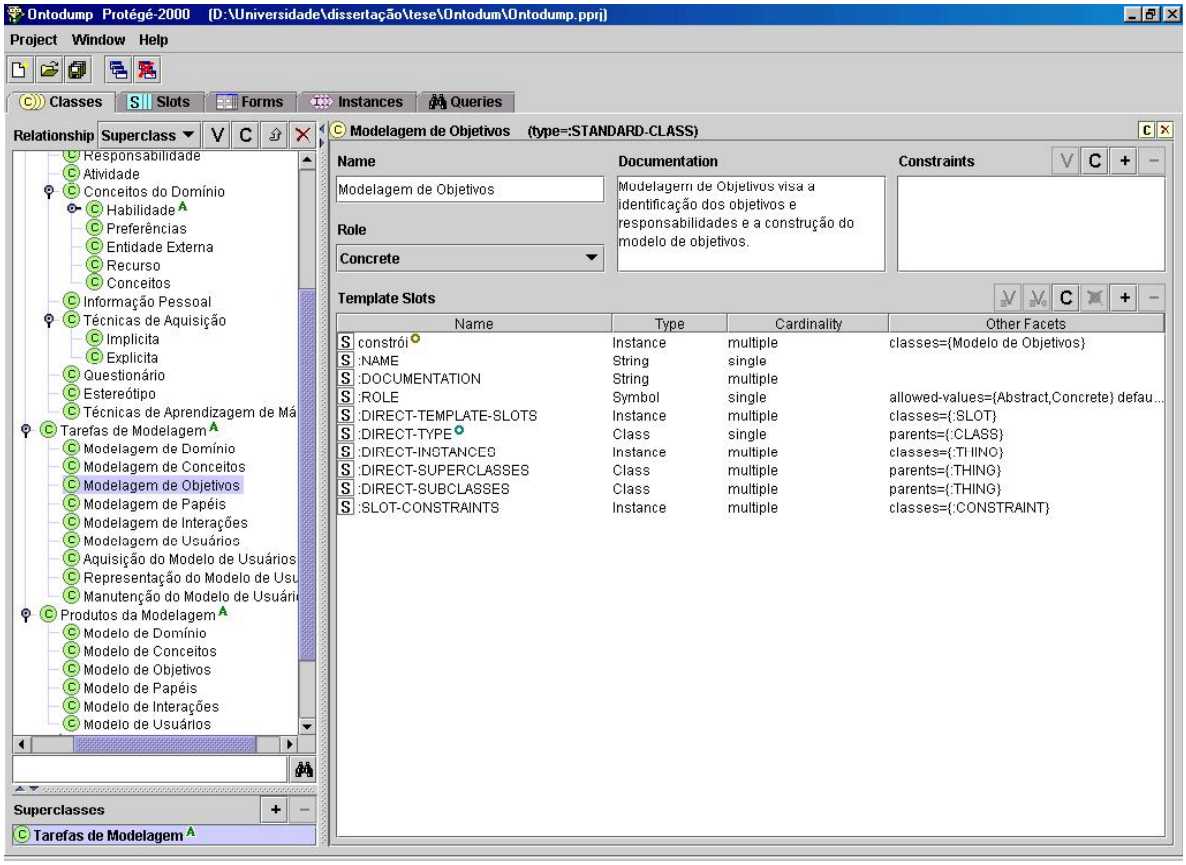


**Figura 22:** Meta-classe *Modelagem de Objetivos* [Faria, 2004]

Por exemplo, o problema atual dos sistemas para o acesso à informação é *satisfazer de forma eficiente e eficaz as necessidades de informação dos usuários do sistema*. Esse problema sugere o objetivo geral do sistema. Este objetivo geral pode ser refinado nos objetivos específicos *satisfazer a necessidade de informação em longo prazo dos usuários* e *satisfazer a necessidade de informação pontual dos usuários*. O cumprimento do objetivo específico *satisfazer a necessidade de informação em longo prazo dos usuários* requer o exercício de algumas responsabilidades, entre elas a *Modelagem de usuário*, a *Filtragem de informação*, o *Monitoramento de fontes de informação*, a *Construção de surrogate* e o *Interfaceamento com o usuário*. O cumprimento do objetivo específico *satisfazer a necessidade de informação pontual dos usuários* requer o exercício de algumas responsabilidades, entre elas o *Interfaceamento com o usuário*, a *Construção de*

*surrogate*, a *Recuperação de informação*, a *Indexação* e o *Descobrimento de novos elementos de informação* (**Figura 23**).

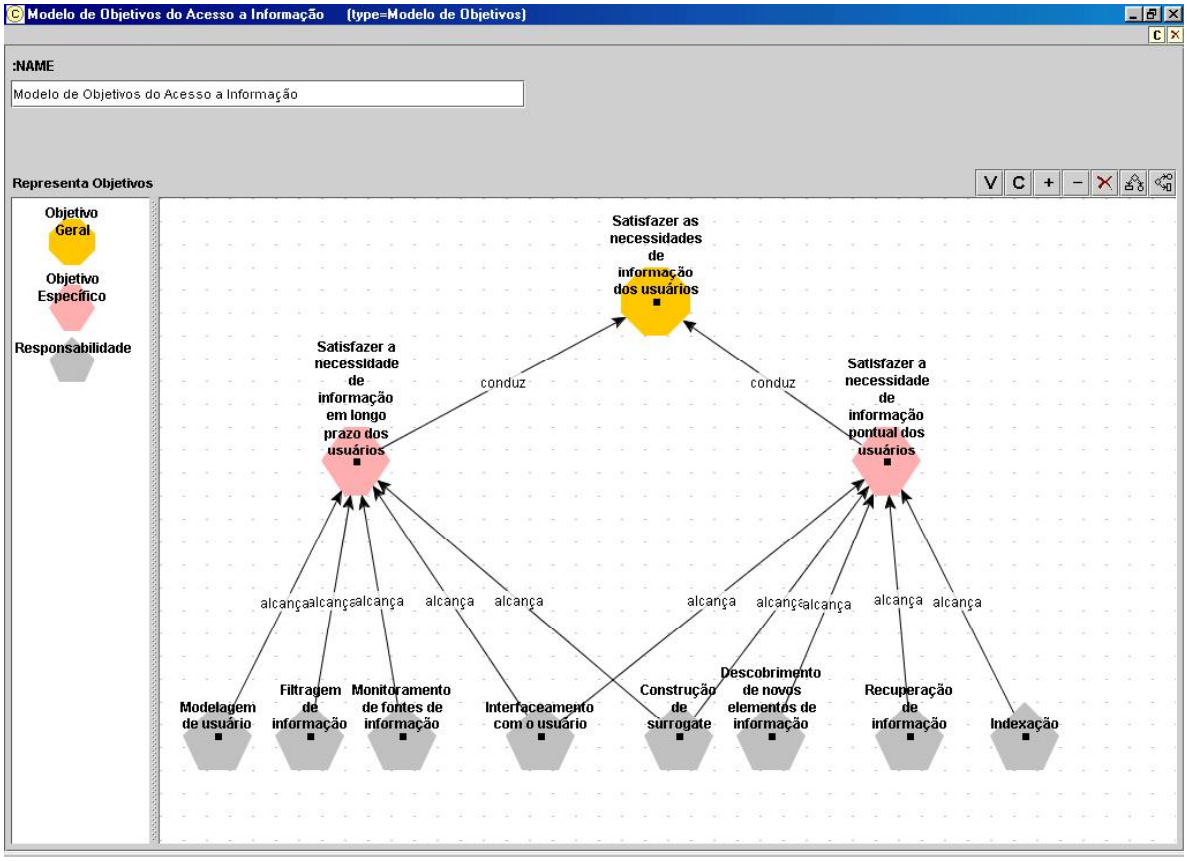


**Figura 23:** Exemplo do *Modelo de Objetivos* para o acesso à informação dinâmica e não estruturada

### 4.2.1.3 Modelagem de Papéis

A construção do modelo de papéis é feita através da instanciação da meta-classe *Modelagem de Papéis* (**Figura 24**), que cria a classe *Modelo de Papéis*, contendo papéis, responsabilidades, atividades e recursos. Cada papel é identificado a partir das responsabilidades especificadas no modelo de objetivos. As atividades de cada papel também surgem da sua responsabilidade. Ou seja, deve ser identificado um conjunto de atividades que o papel deverá realizar para cumprir sua responsabilidade. Os recursos são conceitos do domínio identificados a partir das atividades, isto é, são requeridos pelas atividades, para que essas possam ser executadas. Todos os recursos identificados serão acrescentados no *Modelo de*

*Conceitos*. São criados dois tipos de modelos de papéis: um modelo de papel geral, que contém todos os papéis do sistema com seus respectivos atributos e modelos de papel específicos, que contêm a especificação de cada papel com seus respectivos atributos.

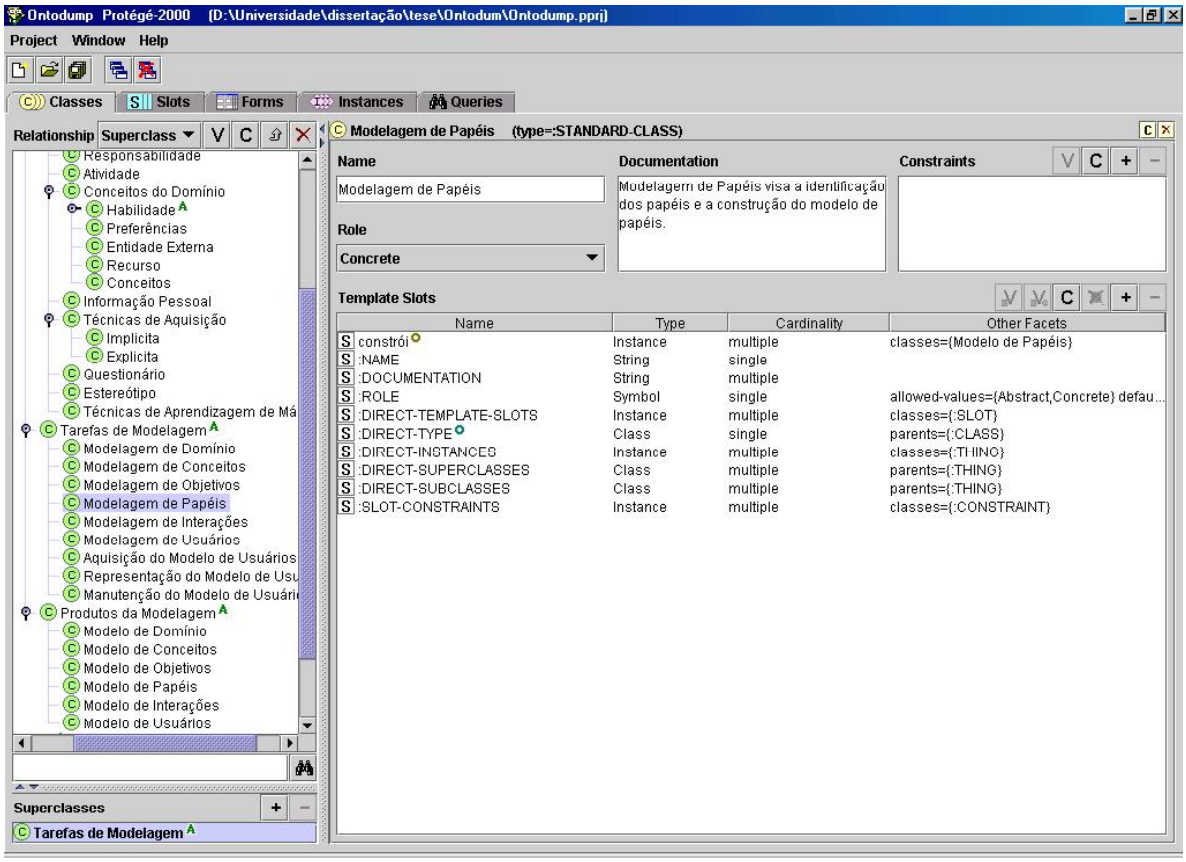


**Figura 24:** Meta-classe *Modelagem de Papéis* [Faria, 2004]

A **Figura 25** mostra um exemplo do modelo de papel específico do *Recuperador*. Os recursos *surrogate de elementos de informação*, *surrogate de necessidade pontual* e *elementos de informação recuperados* foram acrescentados ao *Modelo de Conceitos*.

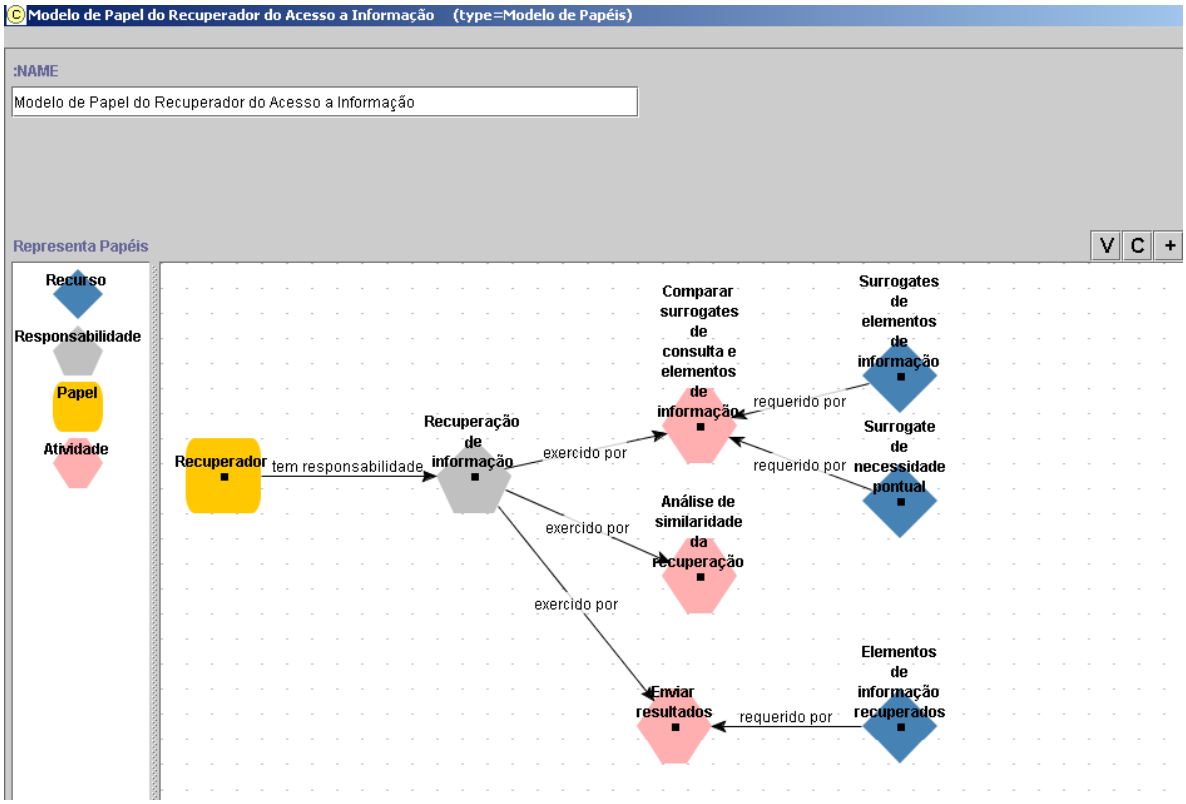


**Figura 25:** Modelo de Papel do *Recuperador* do Acesso à Informação [Serra, Girardi, 2003]

### 4.2.1.4 Modelagem de Interações

A construção do modelo de interações é feita através da instanciação da meta-classe *Modelagem de Interações* (**Figura 26**), que cria a classe *Modelo de Interações*, contendo papéis, interações e entidades externas. Um modelo de interações representa as interações entre papéis ou entre papéis e entidades externas envolvidas no alcance de um objetivo específico. As interações e as entidades externas são identificadas a partir das atividades que os papéis realizam no cumprimento de suas responsabilidades. É construído um modelo de interações para cada objetivo específico especificado no modelo de objetivos. Todas as entidades externas identificadas são acrescentadas no modelo de conceitos. Algumas interações podem dar origem a novos conceitos do domínio. Esses

conceitos, uma vez identificados, também serão acrescentados ao *Modelo de Conceitos*.

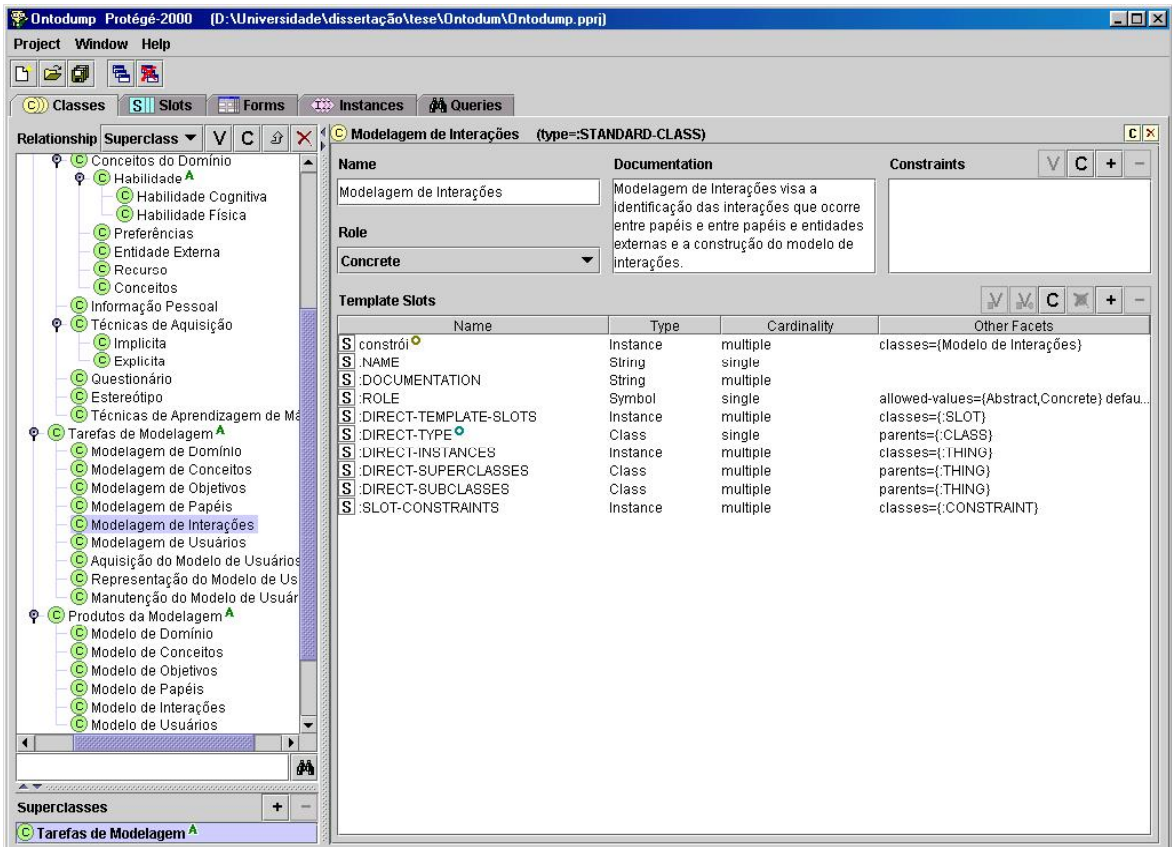


**Figura 26:** Meta-classe *Modelagem de Interações* [Faria, 2004]

Por exemplo, para o alcance do objetivo específico *satisfazer a necessidade de informação pontual dos usuários* foram identificados os seguintes papéis: *Indexador*, *Recuperador*, *Construtor de surrogate*, *Descobridor* e *Interfaceador*. Os papéis no cumprimento de suas responsabilidades realizam atividades, as atividades necessitam de recursos, o que origina as interações e as entidades externas mostradas na **Figura 27**. As entidades externas: *Fonte de informação*, *Usuário* e *Base indexada* foram acrescentadas ao *Modelo de Conceitos* (**Figura 21**).

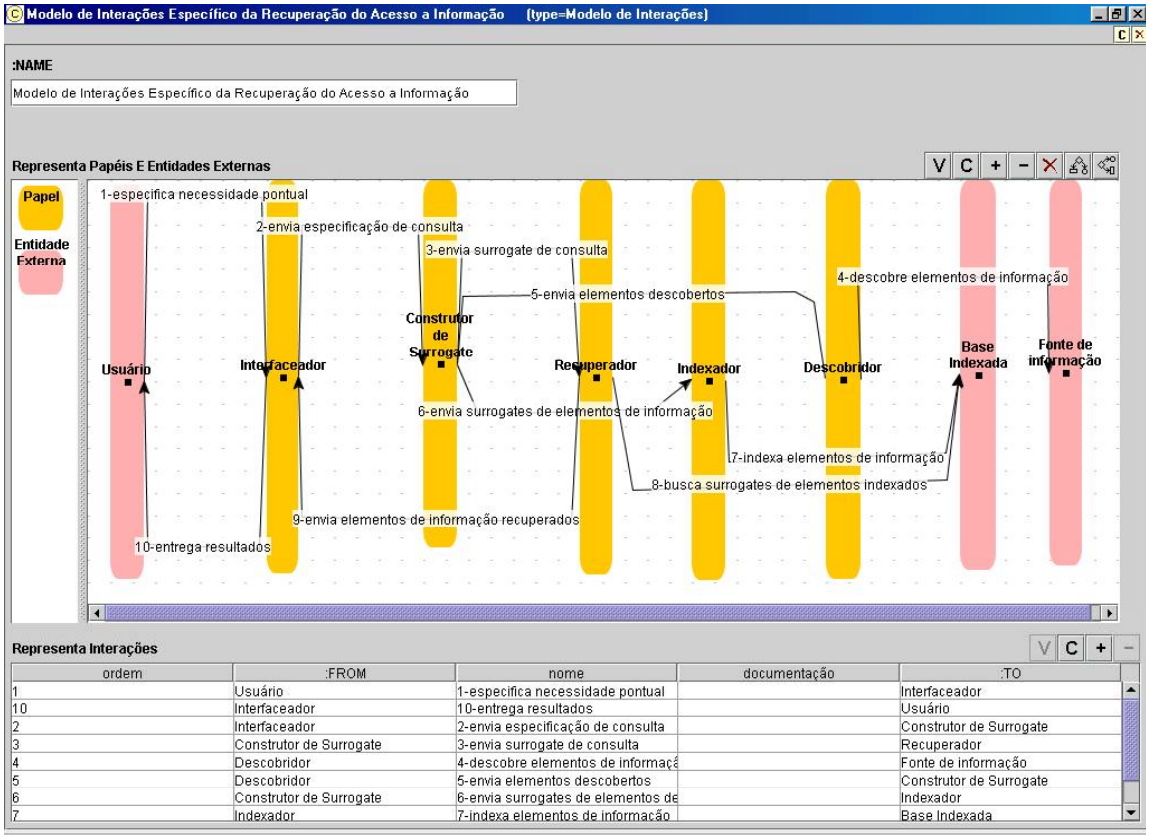


**Figura 27**: *Modelo de Interações* para o objetivo específico da Recuperação

### 4.2.2 Modelagem de Usuário

A construção do modelo de usuários é feita através da instanciação da meta-classe *Modelagem de Usuários* (**Figura 28**), que cria a classe *Modelo de Usuário*, contendo a especificação dos perfis dos usuários do domínio. A fase *Modelagem de Usuários* consiste das seguintes atividades: *Aquisição*, *Representação* e *Manutenção* das informações dos usuários, que gera o produto *Modelo de Usuários*.

#### 4.2.2.1 Aquisição das informações dos usuários

A aquisição das informações dos usuários é feita através da instanciação da meta-classe *Aquisição do Modelo de Usuários* (**Figura 29**), que utiliza a meta-classe *Técnicas de Aquisição*. É aplicada uma técnica de aquisição, que pode ser explícita ou implícita, para o recolhimento das seguintes informações dos usuários:

*informação pessoal*, *habilidade cognitiva*, *habilidade física*, *preferência* e *objetivo do usuário*. Todas as preferências identificadas serão acrescentadas no *Modelo de Conceitos*.

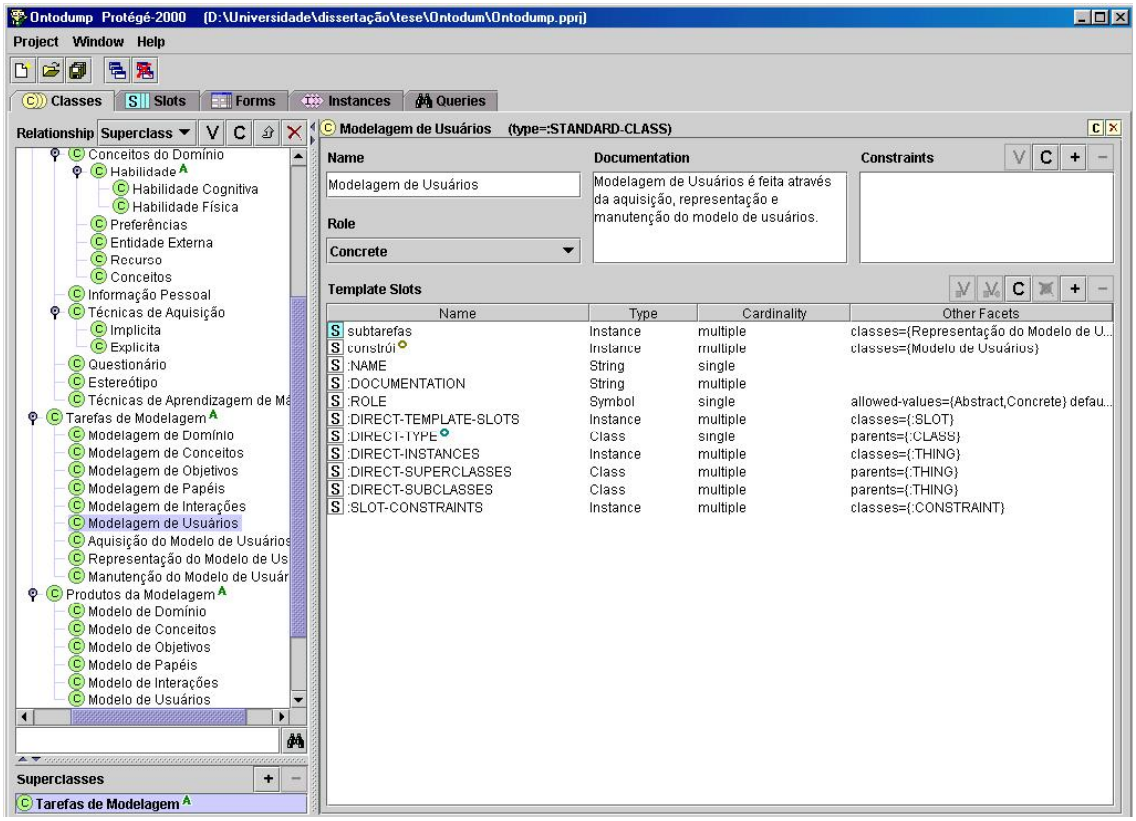


**Figura 28:** Meta-classe *Modelagem de Usuários* [Faria, 2004]

4.2.2.2 Representação das informações dos usuários

A representação das informações dos usuários é feita através da instanciação da meta-classe *Representação do Modelo de Usuários* (**Figura 30**), que cria a classe *Modelo de Usuários*. O modelo de usuários é construído de acordo com as informações adquiridas na atividade de aquisição das informações dos usuários. São criados dois tipos de modelos de usuários: um modelo de usuários geral, que contém todos os usuários do sistema com suas informações respectivas e um modelo de usuários específico, que contém a especificação de um único usuário com suas informações respectivas.

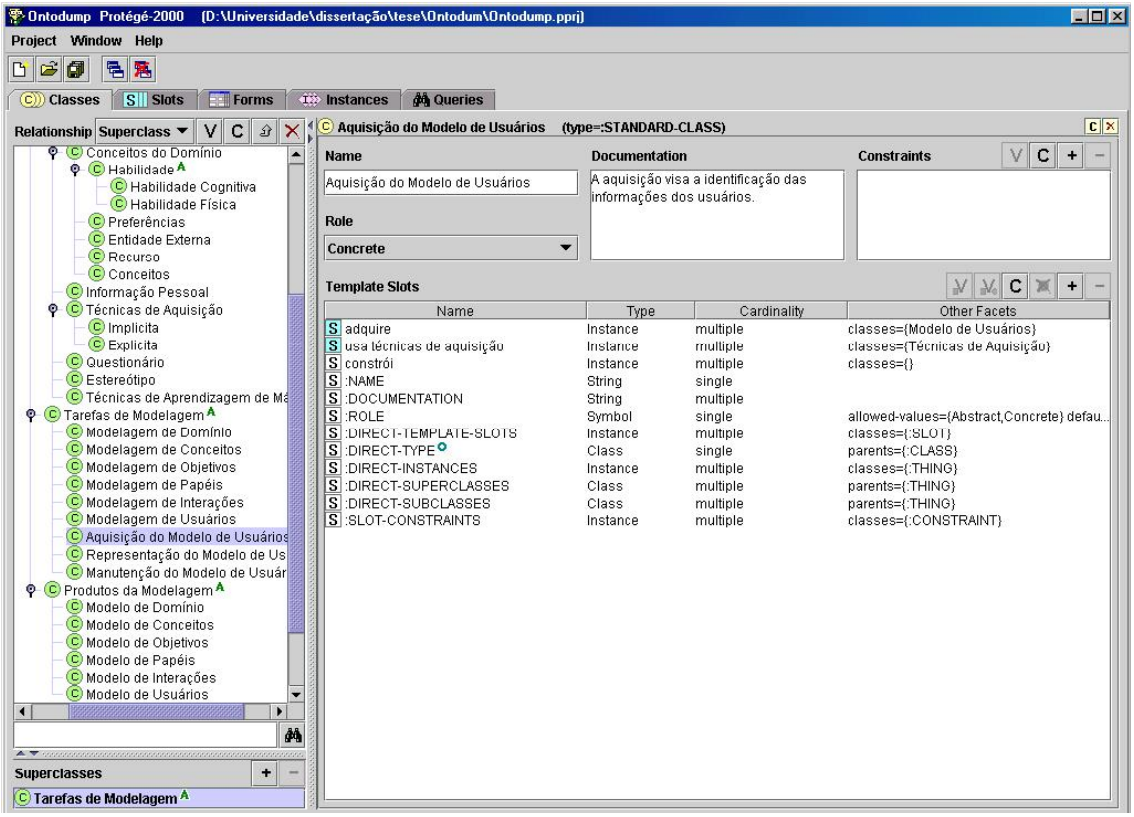


**Figura 29:** Meta-classe *Aquisição do Modelo de Usuários* [Faria, 2004]

4.2.2.3 Manutenção das informações dos usuários

A manutenção das informações dos usuários é feita através da instanciação da meta-classe *Manutenção do Modelo de Usuários* (**Figura 31**), que utiliza a meta-classe *Técnicas de Aquisição*. É aplicada uma técnica de aquisição, que pode ser explícita ou implícita, para o recolhimento das seguintes informações dos usuários: informação pessoal, habilidade cognitiva, habilidade física, preferência e objetivo do usuário. Após o recolhimento das informações dos usuários é feita a atualização do *Modelo de Usuários* com as novas informações adquiridas.

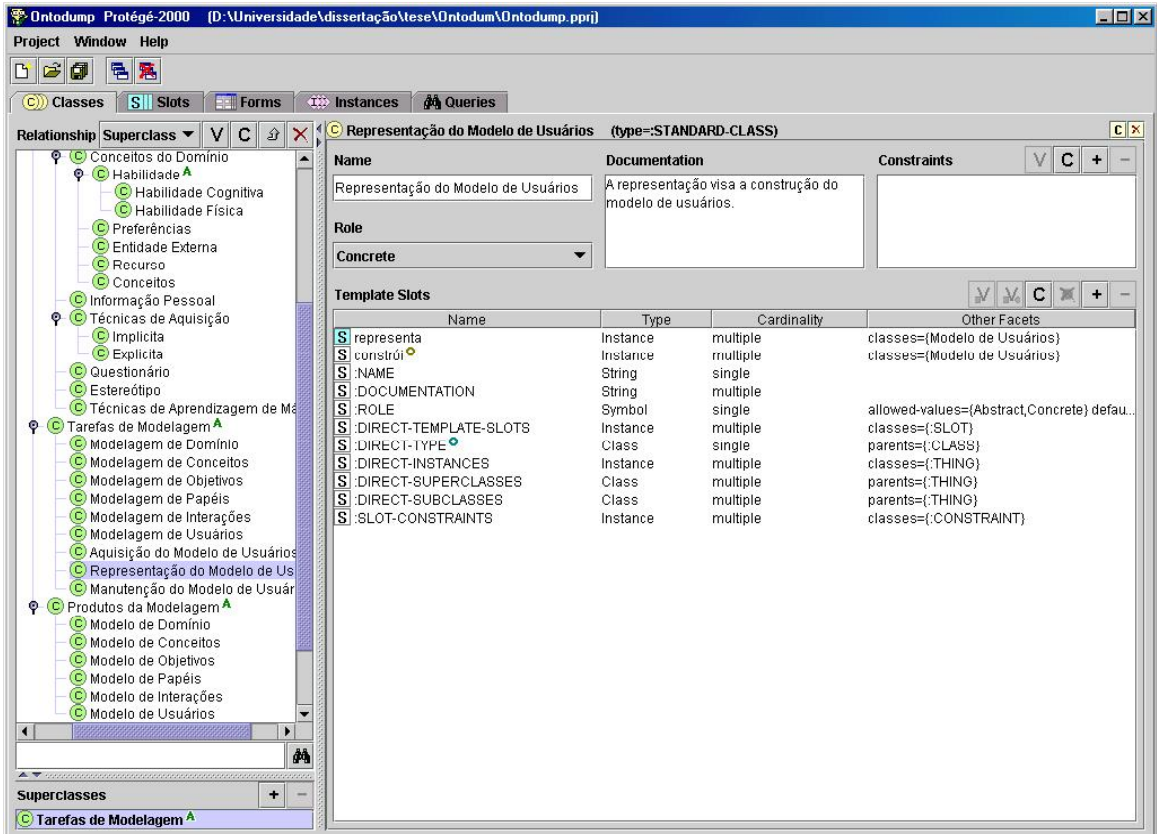


**Figura 30:** Meta-classe *Representação do Modelo de Usuários* [Faria, 2004]

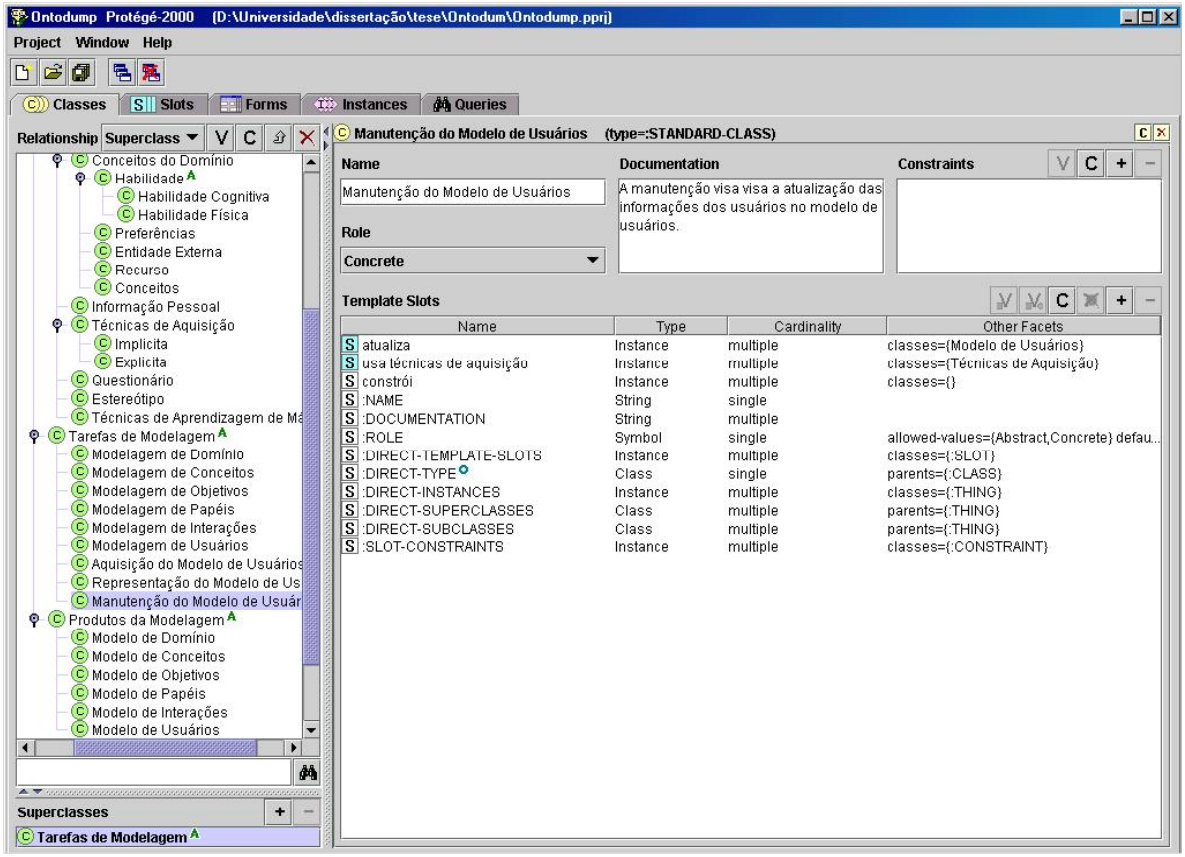


**Figura 31:** Meta-classe *Manutenção do Modelo de Usuários* [Faria, 2004]

### 4.3 Considerações finais

Neste capítulo foi apresentada a GRAMO, uma técnica para a fase de análise de domínio na Engenharia de Domínio Multiagente. A GRAMO tem como insumo o conhecimento de um domínio e constrói modelos de domínio segundo o paradigma computacional de agentes. Os modelos de domínio desenvolvidos com a GRAMO são a entrada para a TOD-LED, uma técnica para o desenvolvimento de LED's na Engenharia de Domínio Multiagente, que é apresentada no próximo capítulo.

# 5 TOD-LED – UMA TÉCNICA BASEDA EM ONTOLOGIAS PARA O DESENVOLVIMENTO DE LED'S

Neste capítulo é apresentada a TOD-LED (Técnica Baseada em Ontologias para o Desenvolvimento de LED's). A TOD-LED guia o processo de especificação de LED's, tendo como ponto de partida modelos de domínio construídos através da instanciação da ontologia genérica ONTODUM [Faria, 2004], apresentada no capítulo 4.

A TOD-LED utiliza a ONTOLED, uma ontologia que representa o conhecimento acerca do desenvolvimento de LED's na Engenharia de Domínio Multiagente. A especificação de uma LED é obtida através da instanciação da ONTOLED usando o roteiro definido pela TOD-LED.

Considerando a análise realizada da literatura relacionada, a TOD-LED se constitui 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.

Na **Tabela 5**, a técnica proposta é classificada com relação aos critérios adotados na seção 3.3. Na definição da semântica é usada uma abordagem baseada em ontologias, para tirar proveito da análise de domínio feita com a aplicação da técnica GRAMO, que tem como produto um modelo de domínio que é uma ontologia instanciada da ONTODUM. Na definição da sintaxe abstrata é usada uma gramática BNF. Com relação à aplicação da técnica, é desenvolvida a LESRF, uma LED para o domínio do acesso à informação dinâmica e não estruturada, apresentada no capítulo 6. A técnica proposta atende apenas a especificação de LED's e possui a tarefa *Especificação da Pragmática*, que define um roteiro para a documentação das LED's, o que cobre uma deficiência observada em todas as

metodologias estudadas. A TOD-LED ainda prevê a utilização da GRAMO como técnica de análise de domínio.

| | Especifi-cação da semântica | Especifi-cação da sintaxe | LED's desenvol-vidas | Especifica ção/ Proje-to/ Imple-mentação | Técnica de Análise de Domínio | Implemen-tação | Documen-tação | Analise de conceitos fixos e variáveis |
|---|---|---|---|---|---|---|---|---|
| **TOD-LED** | Máquina abstrata represen-tada com ontologias | Gramática BNF | LESRF | Especifica-ção | GRAMO | Não | Sim | Não |

**Tabela 5:** Classificação da TOD-LED com relação aos critérios de análise da tabela 3

O desenvolvimento de aplicações com as LED's especificadas através da TOD-LED é baseado em papéis. Papéis são entidades ativas que realizam um conjunto de atividades e que juntos cooperam para o alcance de um objetivo específico. Em um programa desenvolvido com alguma dessas LED's, os desenvolvedores não precisam especificar como o papel deve realizar suas atividades, mas terão apenas que incluir um papel em um programa para que esse cumpra sua responsabilidade de acordo com o que está especificado no modelo de domínio. Programas codificados segundo essas LED's consistem basicamente da declaração de papéis e de propriedades a eles relacionadas. Por esse motivo, as linguagens construídas com a TOD-LED são puramente declarativas.

Neste capítulo sempre que nos referirmos a um modelo de domínio, estaremos de fato nos referindo a um modelo de domínio desenvolvido com a técnica GRAMO, ou seja, um modelo de domínio resultante da instanciação da ontologia genérica ONTODUM [Faria, 2004].

Na seção 5.1 é apresentada uma extensão da técnica GRAMO e da ONTODUM para permitir a classificação dos conceitos do modelo de domínio em fixos e variáveis, idéias fundamentais no desenvolvimento de famílias de sistemas e no projeto de LED's. Na seção 5.2 é apresentada a ONTOLED e seu processo de construção. Na seção 5.3 é apresentada a TOD-LED.

## 5.1 Uma extensão da ONTODUM e da técnica GRAMO para a classificação dos conceitos do domínio

A técnica GRAMO para análise de requisitos para uma família de sistemas na Engenharia de Domínio Multiagente tem como produtos modelos de domínio. Um modelo de domínio identifica os objetos e relacionamentos de um domínio representando o conhecimento necessário para expressar soluções para uma família de sistemas. Entretanto, a GRAMO não leva em consideração um aspecto fundamental no desenvolvimento de famílias de sistemas e também de LED's, que é a classificação dos conceitos do domínio em fixos e variáveis. Para tanto foi necessário estender a técnica GRAMO e a ontologia genérica ONTODUM para que fossem capazes de refletir esses conceitos.

Diferentemente de outras técnicas, como a *Análise de semelhanças* do FAST [Widen, 1995], a classificação dos conceitos definida na extensão da GRAMO é mais direta, uma vez que segue regras bem definidas. Por esse motivo, a identificação dos conceitos fixos e variáveis é mais rápida e precisa, evitando possíveis discordâncias entre os desenvolvedores. Isso é possível em grande parte pela estrutura formal dos modelos de domínio desenvolvidos a partir da ONTODUM [Faria, 2004]. Por exemplo, a **Figura 32** mostra como é feita a classificação das responsabilidades no modelo de objetivos da ONTOINFO [Serra, Girardi, 2003], de acordo com as duas regras seguintes, estabelecidas pela extensão da GRAMO:

- Responsabilidades que atendem a todos os objetivos específicos são fixas, como o caso da responsabilidade *Construção de surrogate*.

- Responsabilidades que não atendem a todos os objetivos específicos são variáveis. É o caso da responsabilidade *Modelagem de usuário*.

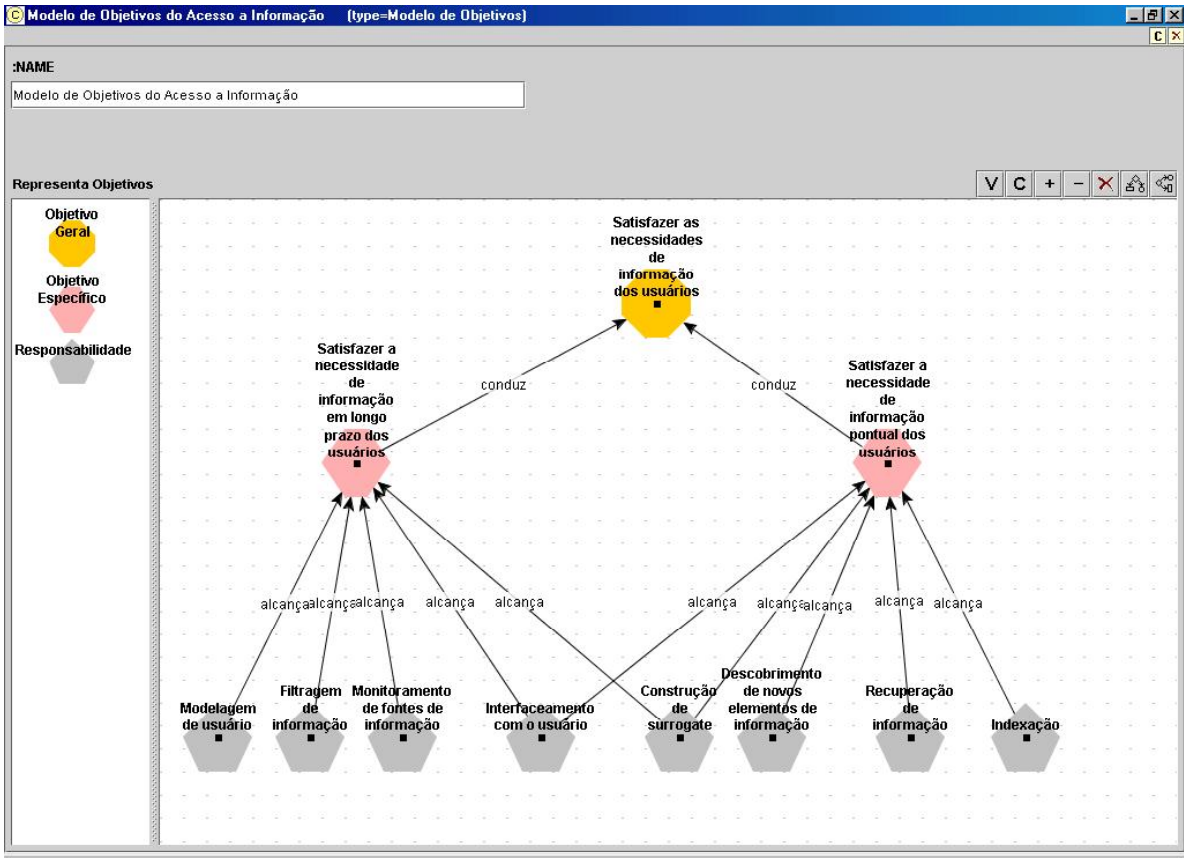


**Figura 32:** Modelo de objetivos da ONTOINFO e a classificação das responsabilidades

### 5.1.1 A extensão da ONTODUM

Para que a ONTODUM fosse capaz de representar a classificação dos conceitos em fixos e variáveis, foi acrescentado o slot *classificação* às classes *Papel*, *Responsabilidade*, *Atividade*, *Recurso*, *Objetivo geral* e *Objetivo específico*. O slot *classificação* pode assumir dois valores, *fixo* ou *variável,* representando os dois tipos de conceitos possíveis segundo essa classificação. A **Figura 33** mostra a classe *Papel* com o slot *classificação*.

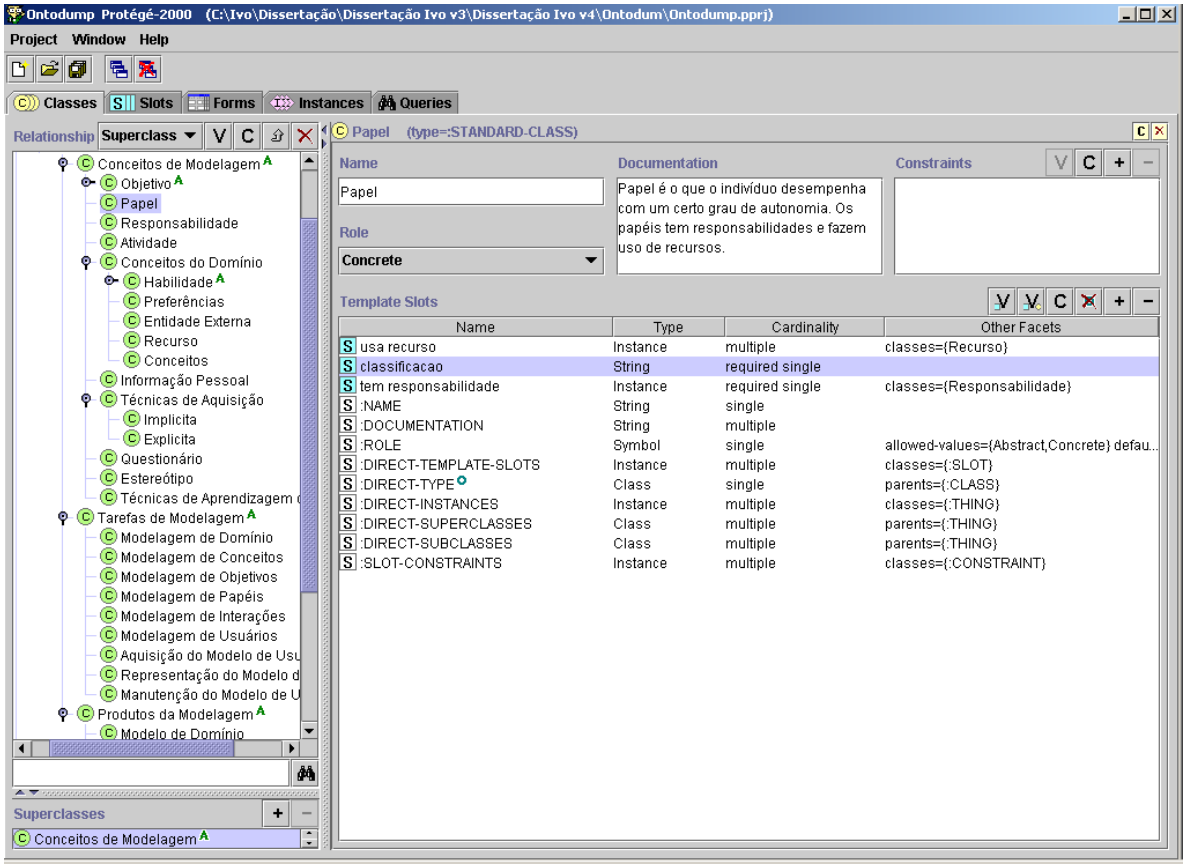


**Figura 33:** A classe *Papel* e o slot *classificação*

### 5.1.2 A extensão da técnica GRAMO

Foi acrescentada à GRAMO a atividade *Modelagem de Variabilidades* (**Tabela 6**). Esta atividade consiste em classificar as instâncias dos conceitos de modelagem: *Papel*, *Responsabilidade*, *Atividade*, *Recurso*, *Objetivo geral* e *Objetivo específico* do modelo de domínio em fixos ou variáveis. Nesta atividade, os slots *classificação,* das instâncias dos conceitos de modelagem são preenchidos com os valores *fixo* ou *variável* de acordo com as regras apresentadas nas próximas seções.

#### 5.1.2.1 Classificação das responsabilidades

- *Responsabilidades fixas*: São responsabilidades que atendem a todos os objetivos específicos. Qualquer sistema tem necessariamente que satisfazer pelo menos um objetivo específico dentre aqueles

especificados no modelo de objetivos. Sendo assim, qualquer responsabilidade que atenda a todos os objetivos específicos estará necessariamente presente em qualquer sistema pertencente a família de sistemas. Por exemplo, no modelo de objetivos da ONTOINFO [Serra, Girardi, 2003] (**Figura 32**), a responsabilidade *Construção de surrogate* é fixa.

- *Responsabilidades variáveis*: São responsabilidades que não atendem a todos os objetivos específicos. Se uma responsabilidade atende a exclusivamente um objetivo específico (conceito variável), essa responsabilidade é variável, uma vez que o objetivo específico ao qual está associado pode não estar presente em um determinado sistema pertencente a família de sistemas. Por exemplo, a responsabilidade *Filtragem de informação* é variável, uma vez que está associada exclusivamente ao objetivo específico *satisfazer necessidade de informação em longo prazo dos usuários* (**Figura 32**).

<table>
<tr><th rowspan="7">TÉCNICA GRAMO</th><th>Fases</th><th colspan="2">Atividades</th><th>Produtos</th></tr>
<tr><td rowspan="4">Modelagem de Domínio</td><td rowspan="3">Modelagem de Conceitos</td><td>Modelagem de Objetivos</td><td rowspan="4">Modelo de Domínio (Modelo de Conceitos, Modelo de Objetivos, Modelo de Papéis e Modelo de Interações)</td></tr>
<tr><td>Modelagem de Papéis</td></tr>
<tr><td>Modelagem de Interações</td></tr>
<tr><td colspan="2">Modelagem de Variabilidades</td></tr>
<tr><td rowspan="3">Modelagem de Usuários</td><td colspan="2">Aquisição</td><td rowspan="3">Modelo de Usuários</td></tr>
<tr><td colspan="2">Representação</td></tr>
<tr><td colspan="2">Manutenção</td></tr>
</table>

**Tabela 6:** A técnica GRAMO estendida

5.1.2.2 Classificação dos papéis

- *Papéis fixos*: São papéis associados às responsabilidades fixas. Cada responsabilidade tem um papel associado. Se uma responsabilidade

for fixa, o papel correspondente também o será. Por exemplo, o papel *Construtor de surrogate* é fixo, uma vez que está associado a responsabilidade *Construção de surrogate*, que é fixa (**Figura 32**).

- *Papéis variáveis*: São papéis associados às responsabilidades variáveis. Cada responsabilidade tem um papel associado. Se uma responsabilidade for variável, o papel correspondente também o será. Por exemplo, a responsabilidade *Filtragem de informação* é variável, portanto, o papel *Filtrador* também é variável (**Figura 32**).

5.1.2.3 Classificação das atividades

- *Atividades fixas*: Nem todas as atividades associada a papéis fixos são fixas. Atividades fixas são as atividades associadas a papéis fixos e que precisam necessariamente ser executadas. Por exemplo, no caso do papel *Interfaceador* (**Figura 34**), a atividade *Entregar resultados* é fixa, uma vez que sempre será executada.

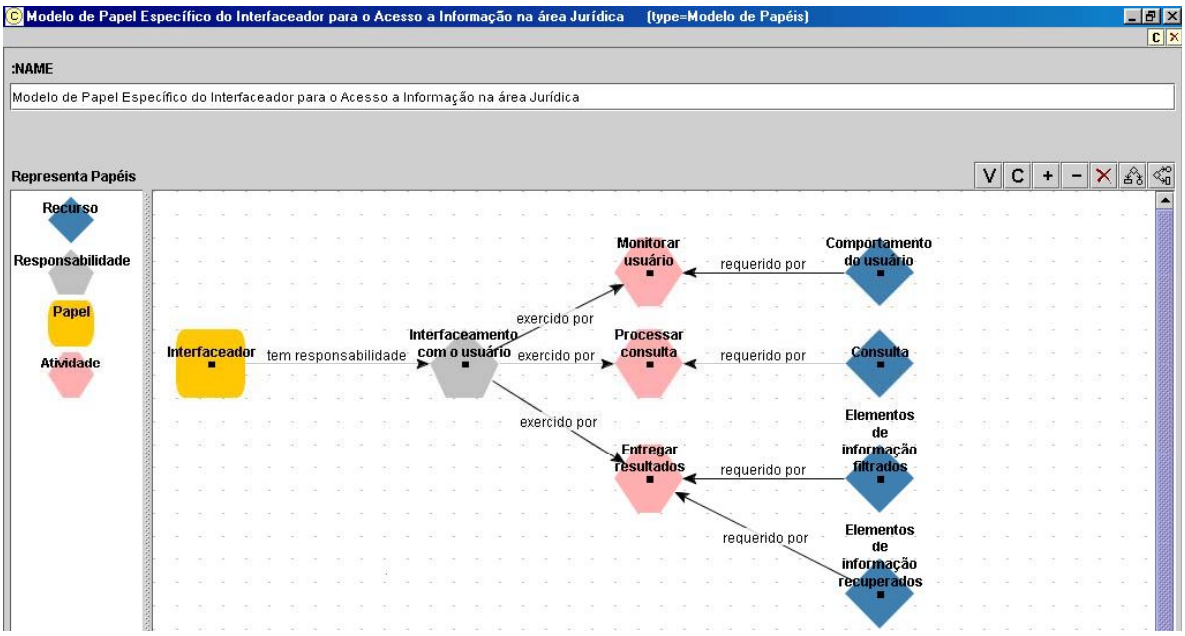


**Figura 34:** Papel *Interfaceador* e a classificação das atividades

- *Atividades variáveis*: Existem dois tipos de atividades classificadas como variáveis:
  - Atividades realizadas por papéis variáveis, uma vez que só estarão presentes em um sistema se os papéis aos quais estão associadas também o estiverem.
  - Atividades associadas a papéis fixos que não precisem ser executadas em todos os sistemas que usem determinado papel.

Por exemplo, no caso do papel *Interfaceador* (**Figura 34**) a atividade *Monitorar usuário* somente será necessária se o sistema realizar filtragem de informação. Já a **Figura 35** mostra o papel Recuperador, que é variável, portanto suas atividades *Comparar surrogates de consulta e elementos de informação*, *Análise de similaridade* e *Enviar resultados* também o são.

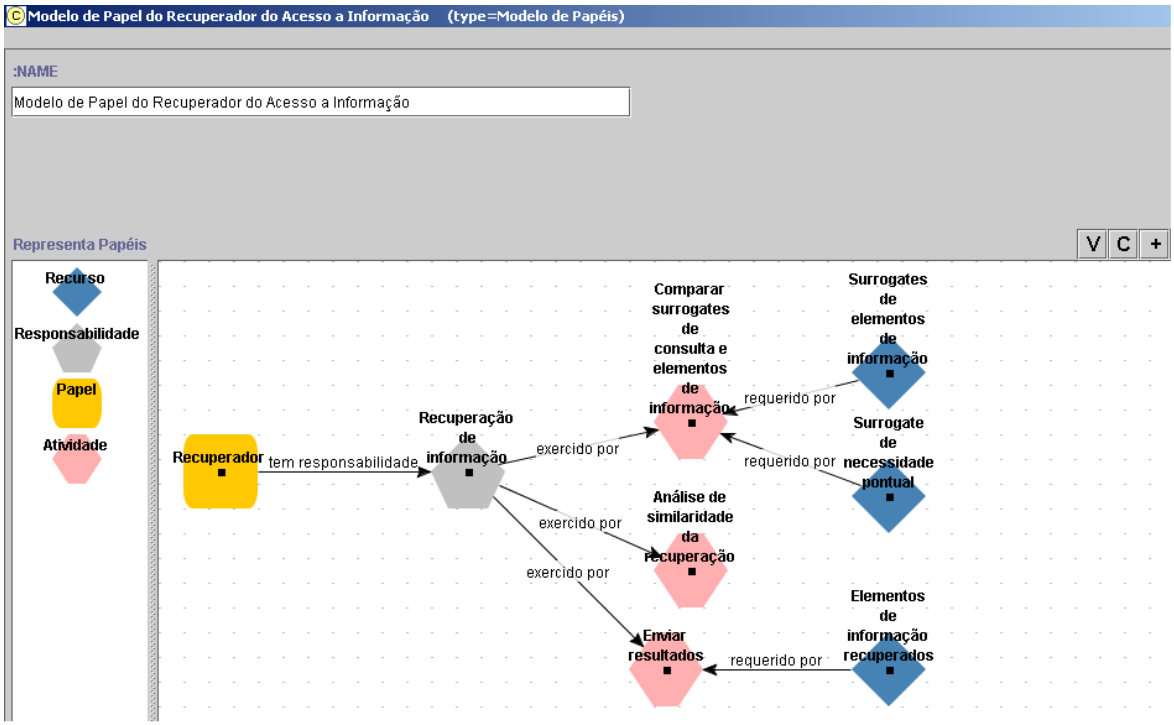


**Figura 35:** Papel *Recuperador* e a classificação das atividades

5.1.2.4 Classificação dos recursos

- *Recursos fixos*: Um recurso é fixo se em todos os sistemas que podem ser desenvolvidos em um domínio existe pelo menos um papel que sempre solicite esse recurso. Por exemplo, o recurso *Surrogates de elementos de informação* (**Figura 35**) é fixo uma vez que também é sempre requerido pelo papel *Construtor de surrogate*, que é fixo.
- *Recursos variáveis*: Um recurso é variável se nem todos os sistemas pertencentes a um domínio possuem papéis que solicitam este recurso. Por exemplo, o recurso *Surrogate de necessidade pontual* (**Figura 35**) é variável, uma vez que é requerido apenas pelo papel *Recuperador*, que é variável.

5.1.2.5 Classificação dos objetivos específicos

- *Os objetivos específicos são variáveis*: De acordo com o que é definido no modelo de objetivos, para que o objetivo geral seja alcançado, pelo menos um dos objetivos específicos tem que ser alcançado. Sendo assim, um objetivo específico é um conceito que pode estar presente ou não no desenvolvimento de cada sistema pertencente a uma família de sistemas e portanto é variável. Por exemplo, na ONTOINFO, os dois objetivos específicos: *Satisfazer necessidade de informação em longo prazo dos usuários* e *Satisfazer necessidade de informação pontual dos usuários* são conceitos variáveis. Isso porque, não necessariamente estes dois objetivos específicos precisam ser alcançados em um sistema particular da família de sistemas para o acesso à informação dinâmica e não estruturada (**Figura 32**).

5.1.2.6 Classificação do objetivo geral

- *O objetivo geral é fixo*: De acordo com o modelo de objetivos da ONTODUM, um objetivo geral deve ser definido e alcançado por todos os sistemas pertencentes a um domínio específico. Sendo assim, o objetivo geral é um conceito fixo. Por exemplo, o objetivo geral definido no modelo de objetivos da ONTOINFO, *Atender as necessidades de informação dos usuários* é fixo (**Figura 32**).

## 5.2 Descrição da TOD-LED

A TOD-LED (**Figura 36**) 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 aqueles modelos de domínio.

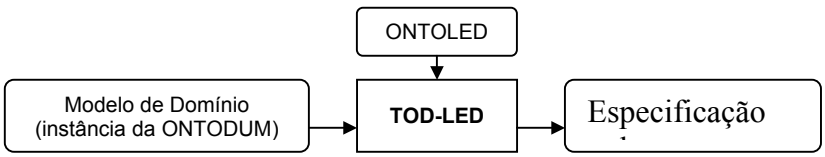


**Figura 36:** Insumos e produtos da TOD-LED

As linguagens especificadas com a TOD-LED são baseadas em papéis. Isso significa que o programa consiste na declaração de um conjunto de papéis que farão parte da solução de um problema. 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, como explicado na seção 5.4.1.2. As linguagens especificadas com a TOD-LED possuem

uma característica similar à maioria das LED's já desenvolvidas. Elas são puramente declarativas. São, portanto, linguagens que especificam o que deve ser feito em vez de como. A TOD-LED é constituída de quatro tarefas: *Especificação da Sintaxe Concreta*, *Especificação da Sintaxe Abstrata*, *Especificação da Semântica* e *Especificação da Pragmática* (**Tabela 7**).

<table>
<tr><td rowspan="5">TÉCNICA TOD-LED</td><td colspan="2">Tarefas</td><td>Produtos</td></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 7:** Tarefas e produtos da TOD-LED

A sintaxe concreta de uma linguagem corresponde ao seu vocabulário. A tarefa *Especificação da sintaxe concreta* consiste das seguintes sub-tarefas: *Especificação dos Papéis e Propriedades* e *Especificação dos Pacotes*, que geram os produtos *Lista de Papéis e Propriedades da LED* e *Lista de Pacotes da LED*.

Na sub-tarefa *Especificação dos Papéis e Propriedades,* é feita a identificação dos papéis e paralelamente a identificação das propriedades a eles associadas. Na sub-tarefa *Especificação dos Pacotes,* são identificados os pacotes e seus papéis constituintes. Pacotes são entidades que contém papéis que atendem exclusivamente a um mesmo objetivo específico.

A sintaxe abstrata de uma linguagem corresponde às suas regras de escrita. Na tarefa *Especificação da sintaxe abstrata,* é gerado como produto a *Gramática da LED*. Na tarefa *Especificação da semântica*, é gerada uma máquina

abstrata, que é um modelo computacional do domínio responsável por atribuir significado aos termos da LED. Na tarefa *Especificação da Pragmática* é gerada a documentação do usuário.

O capítulo 6 ilustra a aplicação da TOD-LED na especificação da LESRF, uma LED para a especificação de sistemas para o domínio do acesso a informação dinâmica e não estruturada.

### 5.3 ONTOLED: uma representação baseada em ontologias da TOD-LED

A ONTOLED representa o conhecimento acerca da técnica TOD-LED para a especificação de LED's na Engenharia de Domínio Multiagente. A **Figura 37** ilustra o processo de construção da ONTOLED, que foi inspirado no método proposto por Fridman e Mcguinness [Fridman, Mcguinness, 2001] para a construção de ontologias. O processo consiste de duas fases: definição e projeto da ontologia. Na fase de definição é utilizado o conhecimento da TOD-LED para a especificação de LED's de forma a gerar uma rede semântica com a representação desses conceitos. Na fase de projeto, a ONTOLED é criada através do mapeamento da rede semântica a uma ontologia baseada em *frames* representada por uma hierarquia de classes. Neste processo é também utilizada a ONTODUM para a extração dos conceitos utilizados na modelagem de um domínio.

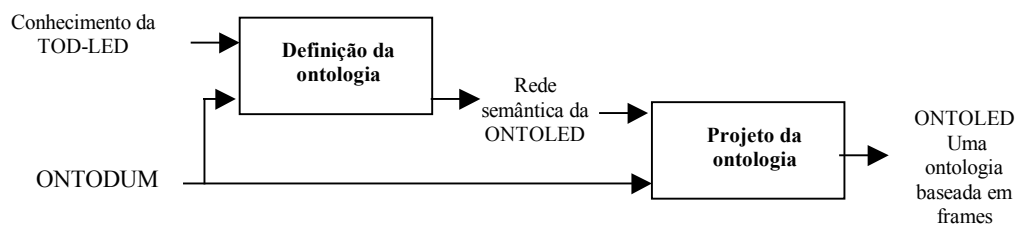


**Figura 37:** Processo de construção da ONTOLED

### 5.3.1 Definição da Ontologia

Nesta fase, o conhecimento acerca da especificação de LED's segundo a TOD-LED é representado em uma rede semântica (**Figura 38**).

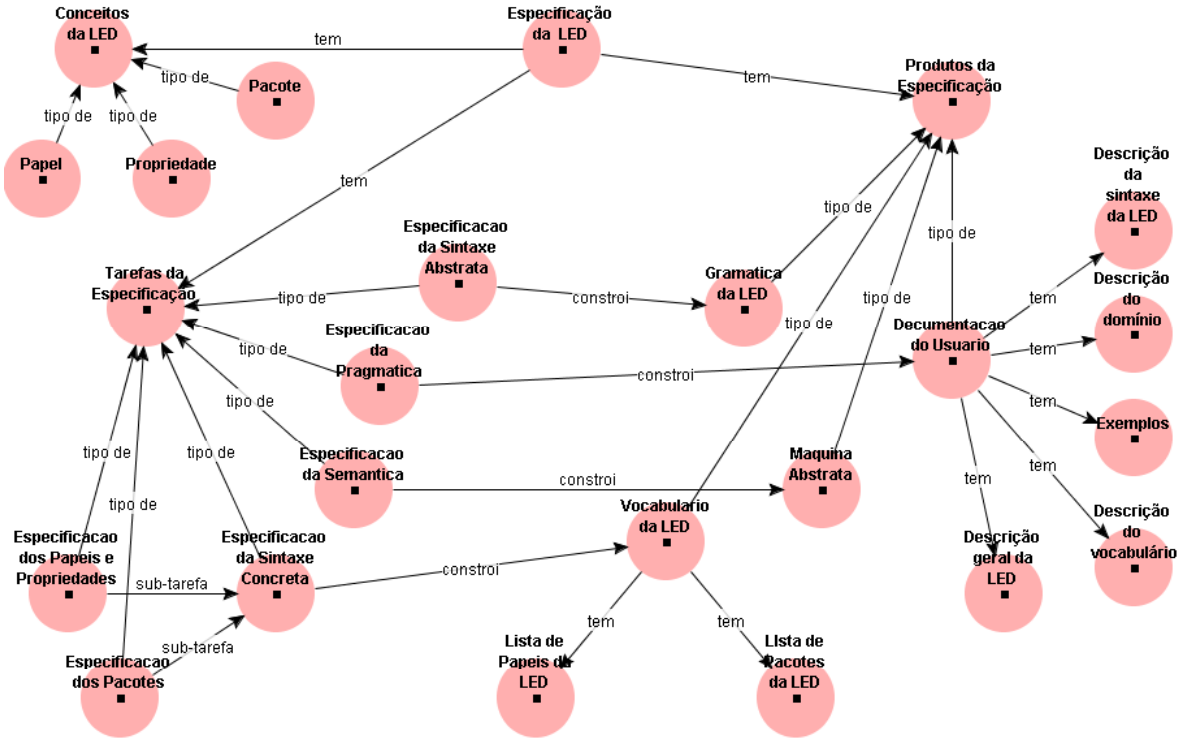


**Figura 38:** Rede semântica da ONTOLED

Os tipos de conceitos que fazem parte do vocabulário das LED's especificadas com a TOD-LED são: papéis, propriedades e pacotes. As tarefas de desenvolvimento são quatro: *Especificação da Sintaxe Concreta*, *Especificação da Sintaxe Abstrata*, *Especificação da Semântica* e *Especificação da Pragmática*. A tarefa *Especificação da Sintaxe Concreta* consiste de duas sub-tarefas: *Especificação dos Papéis e Propriedades* e *Especificação dos Pacotes*. Como resultado da tarefa *Especificação da Sintaxe Concreta* é obtido o produto *Vocabulário da LED*. Como resultado da tarefa *Especificação da sintaxe abstrata* é obtido o produto *Gramática da LED*. Como resultado da tarefa *Especificação da*

*Semântica* é obtido o produto *Máquina Abstrata da LED.* Como resultado da tarefa *Especificação da Pragmática*, é obtido o produto *Documentação do Usuário*.

### 5.3.2 Projeto da Ontologia

Nesta fase, os conceitos e relacionamentos da rede semântica são representados na ONTOLED. Os nós são mapeados para classes. Os nós relacionados por um *link* "tipo de" são mapeados em uma hierarquia de subclasses e superclasses. Outros *links* são mapeados para slots das classes correspondentes. Cada slot é associado com facetas apropriadas, como tipo e cardinalidade. A **Figura 39** dá uma visão da hierarquia de classes da ONTOLED, destacando a classe conceitos da LED que possui as subclasses: *Papel*, *Propriedade* e *Pacote*.

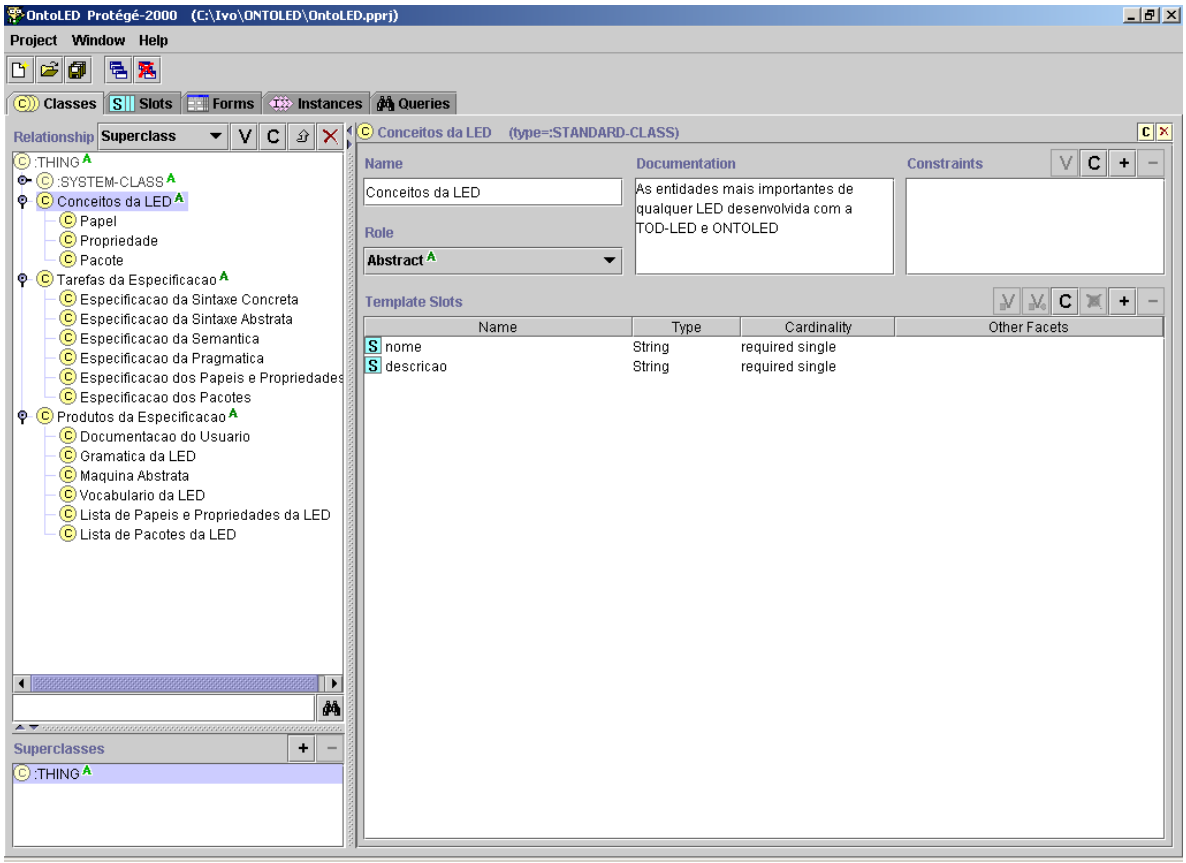


**Figura 39:** A hierarquia de classes da ONTOLED

## 5.4 Tarefas da TOD-LED

Nesta seção são descritas as tarefas da TOD-LED e como é feita a utilização da ONTOLED na especificação de LED's na Engenharia de Domínio Multiagente.

### 5.4.1 Especificação da Sintaxe Concreta

A sintaxe concreta compreende os termos ou vocabulário da linguagem. A construção do Vocabulário da LED é feito através da instanciação da classe *Especificação da Sintaxe Concreta* que cria uma instância da classe *Vocabulário da LED*, contendo os papéis, as propriedades e os pacotes da LED.

A tarefa *Especificação da Sintaxe Concreta* consiste das seguintes sub-tarefas: *Especificação dos Papéis e Propriedades* e *Especificação dos Pacotes*. A instanciação das classes correspondentes a estas tarefas cria as instâncias dos produtos *Lista de Papéis e Propriedades da LED* e *Lista de Pacotes da LED* que compõem o produto *Vocabulário da LED* (**Figura 40**).

#### 5.4.1.1 Especificação dos Papéis e Propriedades

A especificação dos papéis e propriedades é feita através da instanciação da classe *Especificação dos Papéis e Propriedades* (**Figura 41**) que cria uma instância da classe *Lista de Papéis e Propriedades da LED*. Nesta sub-tarefa são identificados os papéis que fazem parte do vocabulário da LED. Durante a identificação dos papéis são também identificadas suas propriedades.

Os papéis identificados são incluídos na *Lista de Papéis e Propriedades da LED*. Cada papel incluído nesta lista é uma instância da classe *Papel* que possui os slots: *nome,* que contém o nome do papel; *descrição,* onde é incluída uma

descrição do papel; *tem propriedade,* para listar as propriedades dos papéis; e *compõe,* que indica o pacote do qual o papel eventualmente faça parte (**Figura 42**).

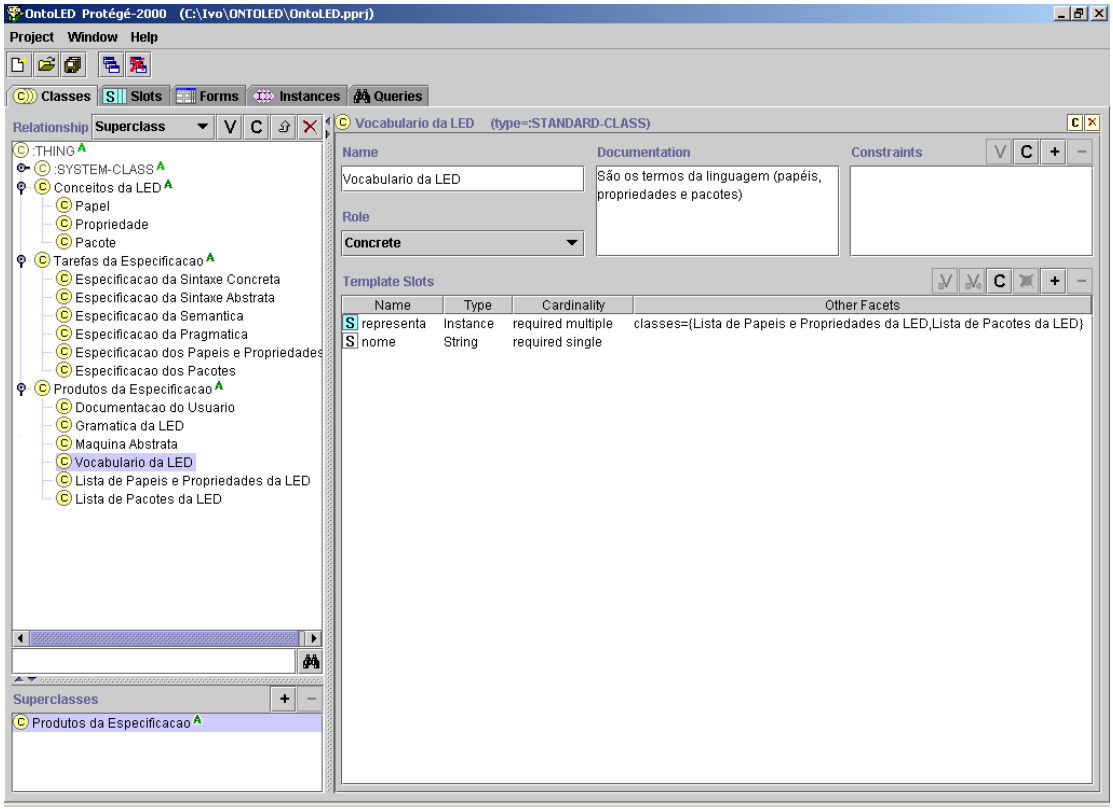


**Figura 40:** A classe *Vocabulário da LED* e o slot *representa*

*a) Identificação dos papéis da LED*

Os papéis que fazem parte do vocabulário da LED podem ser de um dos seguintes tipos:

- *Papéis variáveis*

Os papéis variáveis se enquadram nos conceitos do tipo variável, ou seja, são conceitos que podem estar presentes ou não em cada um dos sistemas pertencentes a uma família de sistemas. Os conceitos variáveis são responsáveis por especificar as características particulares ou específicas de cada sistema. Os papéis variáveis necessariamente fazem parte do vocabulário de uma LED e precisam ser explicitamente referenciados pelos desenvolvedores, quando quiserem usá-los na especificação de um novo sistema.

Para a identificação dos papéis variáveis recorre-se ao modelo de domínio estendido, que é o produto da aplicação da GRAMO estendida. Os papéis variáveis possuem o slot *classificação* marcados como *variável*.

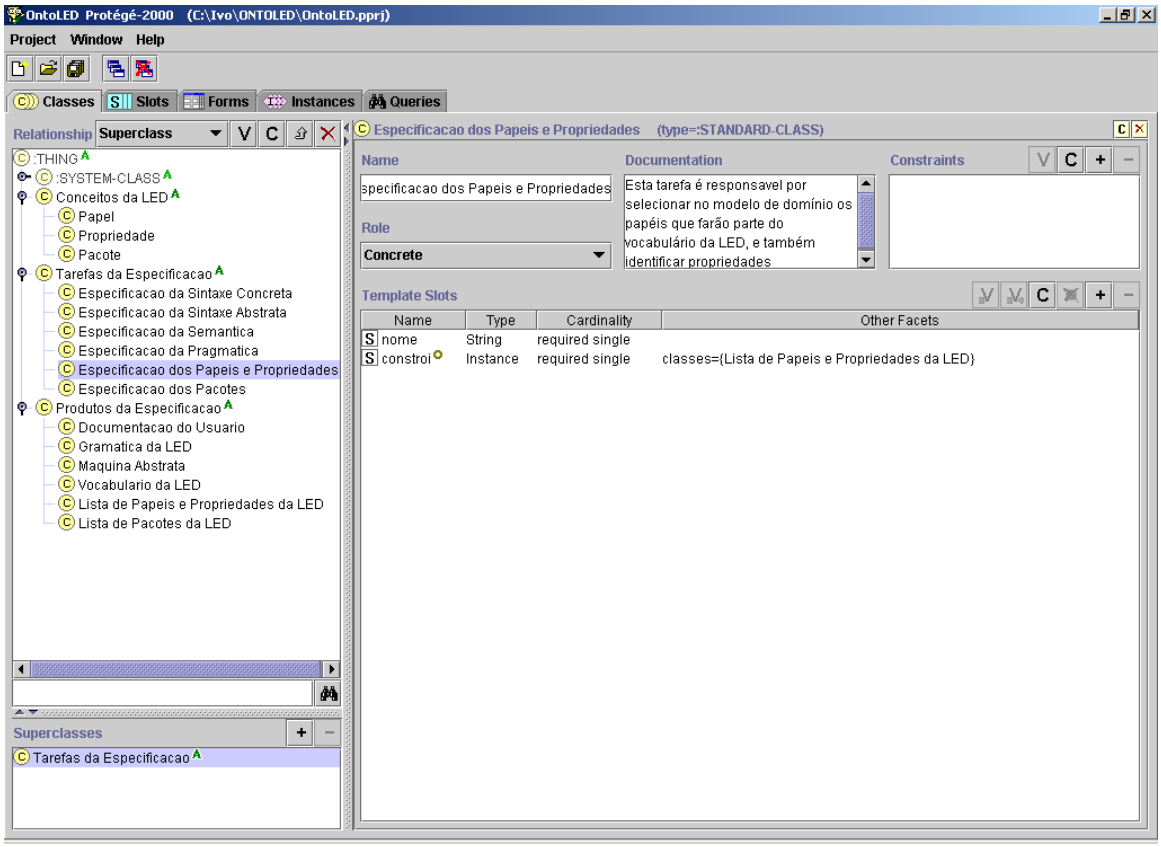


**Figura 41:** A classe *Especificação dos Papéis e Propriedades*

- *Papéis fixos que possuam propriedades*

Um papel é dito fixo quando representa uma característica que é comum, ou seja, não varia entre sistemas pertencentes a uma família de sistemas. Entretanto, certos papéis fixos possuem propriedades que podem ser ajustadas pelos programadores e que, portanto, representam pontos de variabilidade. No código de um programa, uma propriedade não pode aparecer desassociada de um papel. Por esse motivo, papéis fixos nessas condições também devem ser incluídos no vocabulário da LED. Para a identificação dos papéis fixos recorre-se ao modelo

de domínio estendido, que é o produto da aplicação da GRAMO estendida. Os papéis fixos possuem o slot *classificação* marcado como *fixo*.

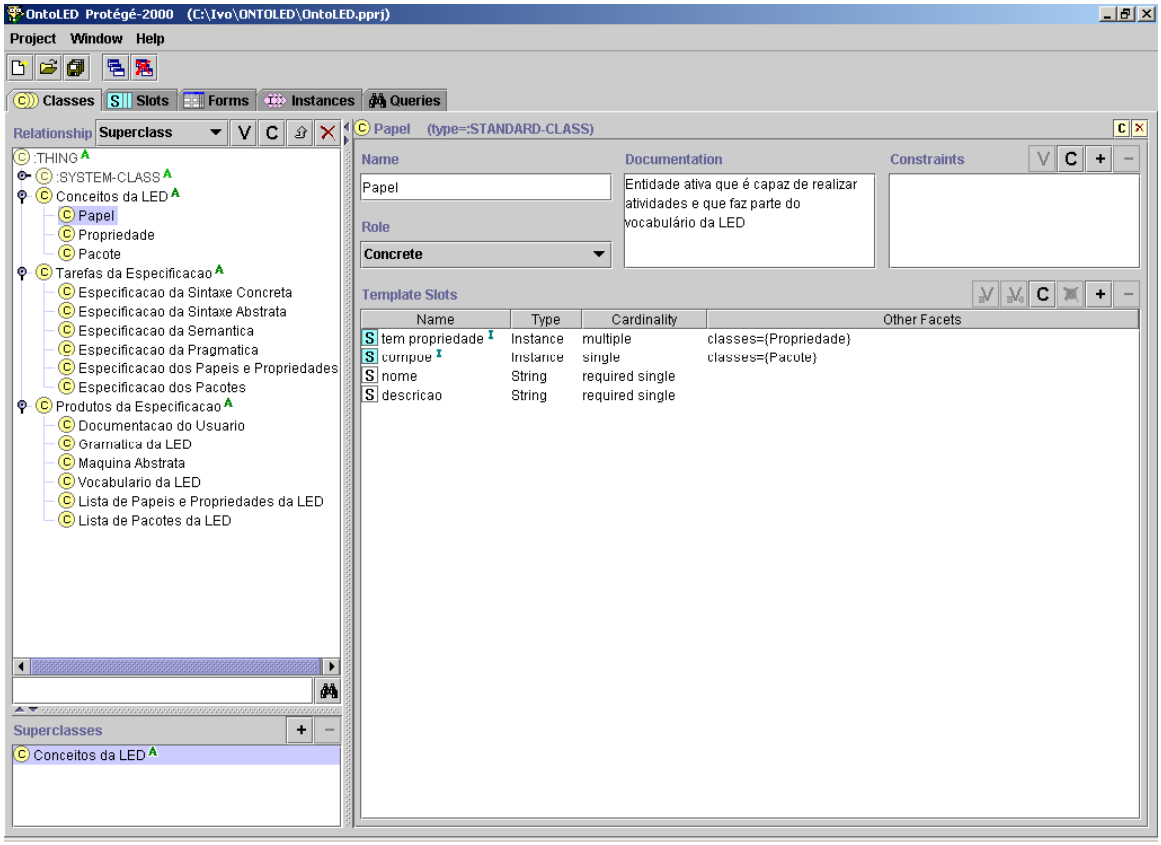


**Figura 42:** A classe *Papel* e seus slots

*b) Identificação das propriedades dos papéis*

Propriedades são aspectos relacionados aos papéis que podem ser especificados pelo programador em tempo de desenvolvimento. Seu objetivo é ajustar certos aspectos do funcionamento dos papéis.

Cada propriedade pode assumir um conjunto de valores que é denominado *domínio da propriedade*. As propriedades são conceitos variáveis uma vez que podem assumir diferentes valores.

A identificação das propriedades ocorre concorrentemente com a identificação dos papéis. O slot *tem propriedade* da classe *Papel* (**Figura 42**) cria as instâncias de propriedade associadas a cada papel. Cada propriedade incluída é

uma instância da classe *Propriedade*, que possui os slots: *nome,* que contém o nome da propriedade; *descrição,* onde é incluída uma descrição da propriedade; *domínio,* para listar os valores que a propriedade pode assumir; e *papel,* que indica o papel ao qual a propriedade é associada (**Figura 43**).

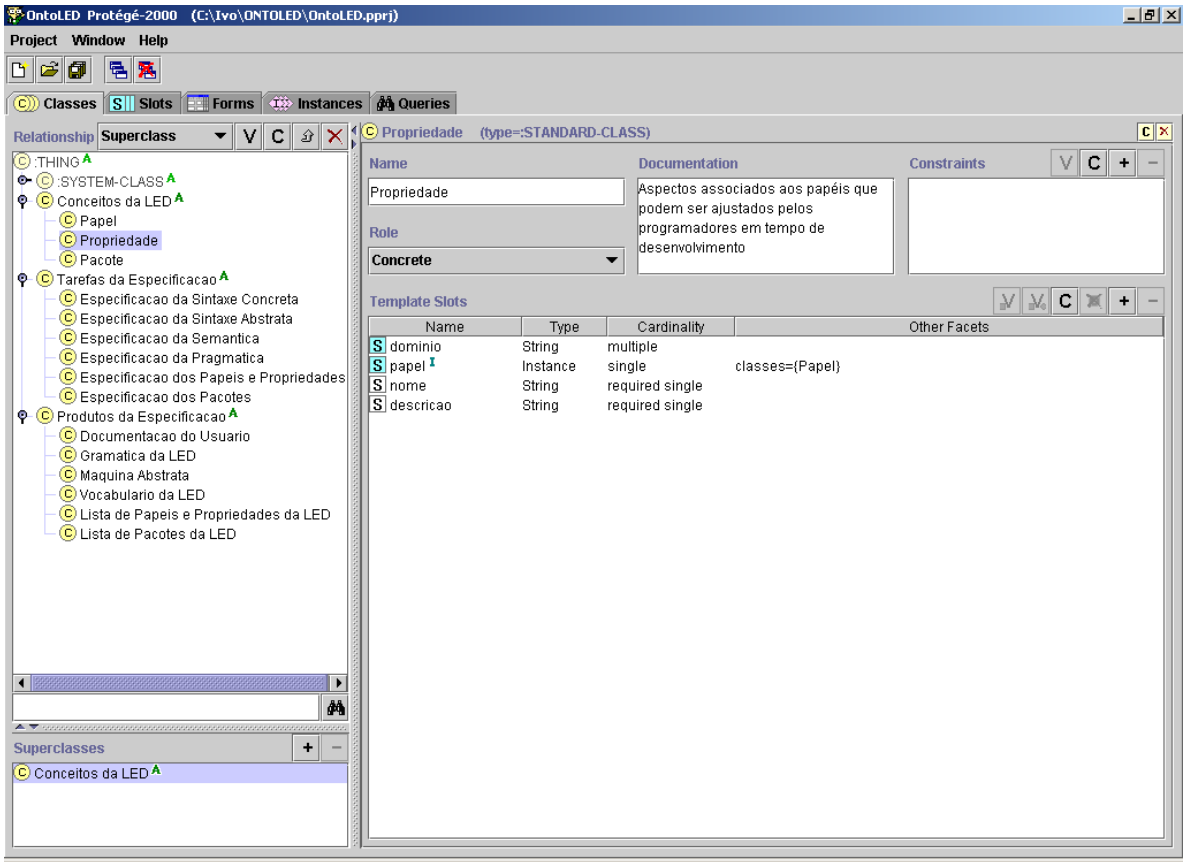


**Figura 43:** A classe *Propriedade* e seus slots

A definição das propriedades leva em consideração requisitos específicos de cada projeto e o conhecimento do domínio. A pergunta a ser feita é: que aspectos relacionados aos papéis devem poder ser ou tem que ser ajustados pelos usuários da LED? Por exemplo, no desenvolvimento da LESRF é definida a propriedade *Técnica de aquisição de perfil* que representa a forma como o perfil do usuário deve ser capturado e que tem como domínio: *implícito* (aspectos relacionados ao comportamento do usuário são observados, assim como *links* percorridos) e *explicito*

(o usuário é solicitado a fornecer ele próprio, informações sobre seus interesses). Esta propriedade é associada ao papel *Modelador de usuário* (**Figura 44**).

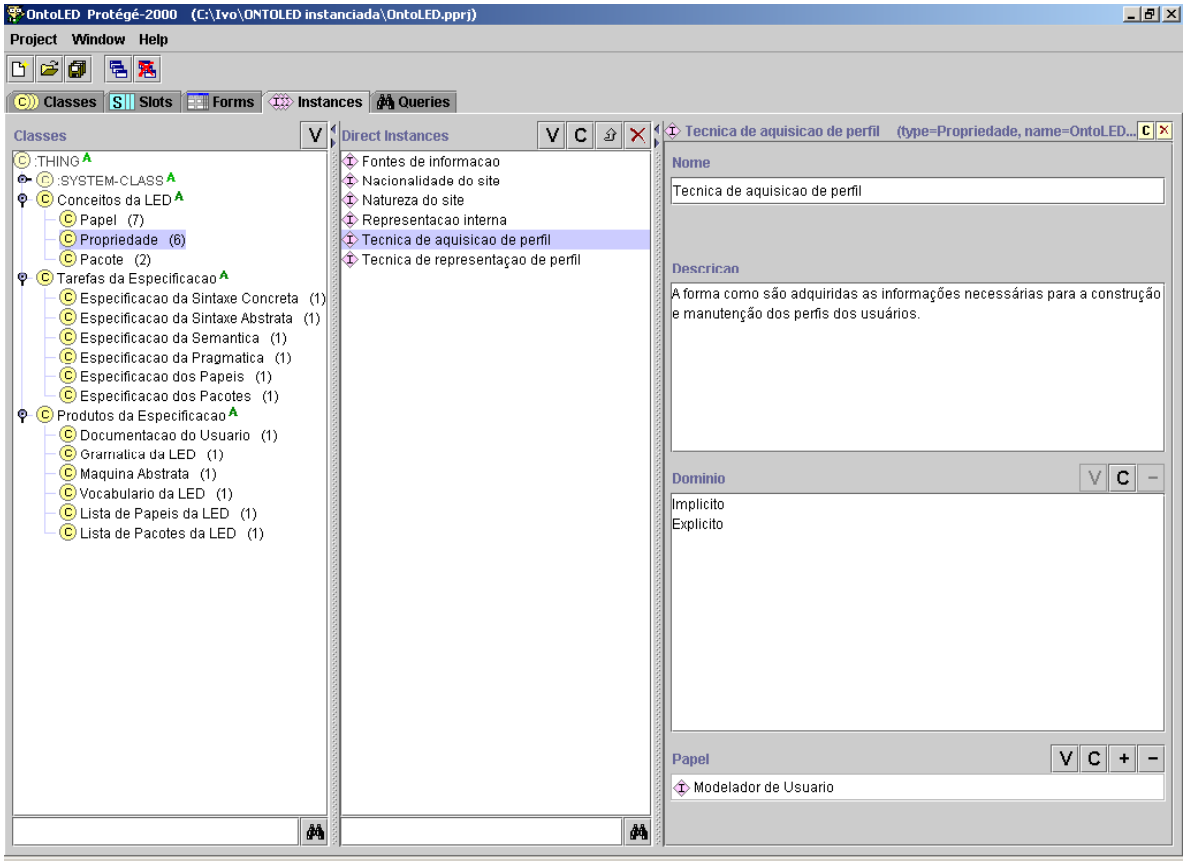


**Figura 44:** Exemplos de propriedades dos papéis da LESRF

### 5.4.1.2 Especificação dos pacotes

Pacotes são entidades compostas por papéis que atendem exclusivamente a um mesmo objetivo específico. Os pacotes são criados para facilitar o trabalho dos desenvolvedores na especificação de sistemas com o uso das LED's. Isso porque os papéis que atendem exclusivamente a um mesmo objetivo específico precisam necessariamente estar todos incluídos em um sistema para que o objetivo específico ao qual estão associados possa ser alcançado.

Quando um pacote é declarado em um programa, todos os papéis que ele contém são declarados implicitamente. Aos pacotes devem ser dados nomes representativos que reflitam a funcionalidade geral proporcionada por seus papéis.

A especificação dos pacotes é feita através da instanciação da classe *Especificação dos Pacotes* (**Figura 45**) que cria uma instância da classe *Lista de Pacotes da LED*. Cada pacote criado é uma instância da classe *Pacote* que possui os slots: *nome,* que contém o nome do pacote; *descrição,* onde é incluída uma descrição do pacote; e *componentes,* para listar os papéis que compõem o pacote (**Figura 46**).

Para a identificação dos pacotes recorre-se ao *Modelo de Objetivos* no modelo de domínio estendido. Quando mais de um papel atende exclusivamente a um mesmo objetivo específico, esses papéis podem ser agrupados em um pacote.

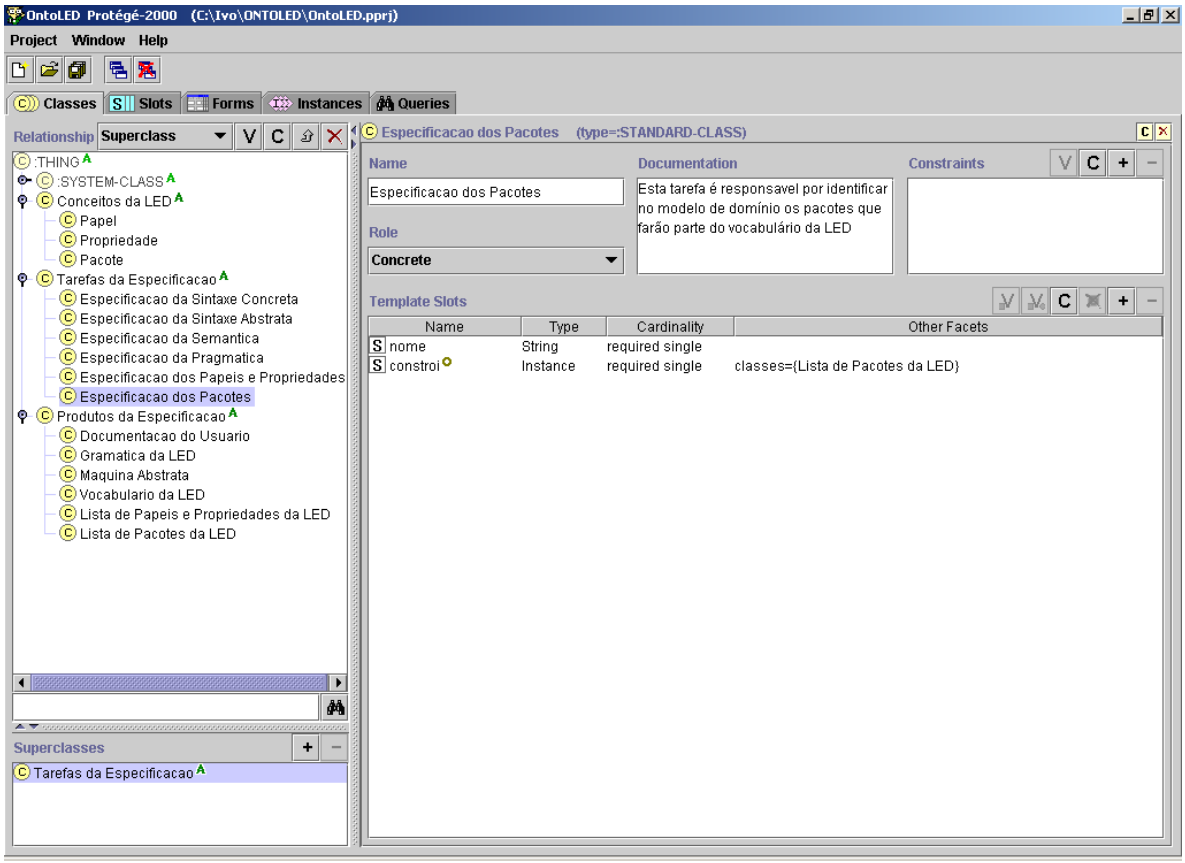


**Figura 45:** A classe *Especificação dos Pacotes*

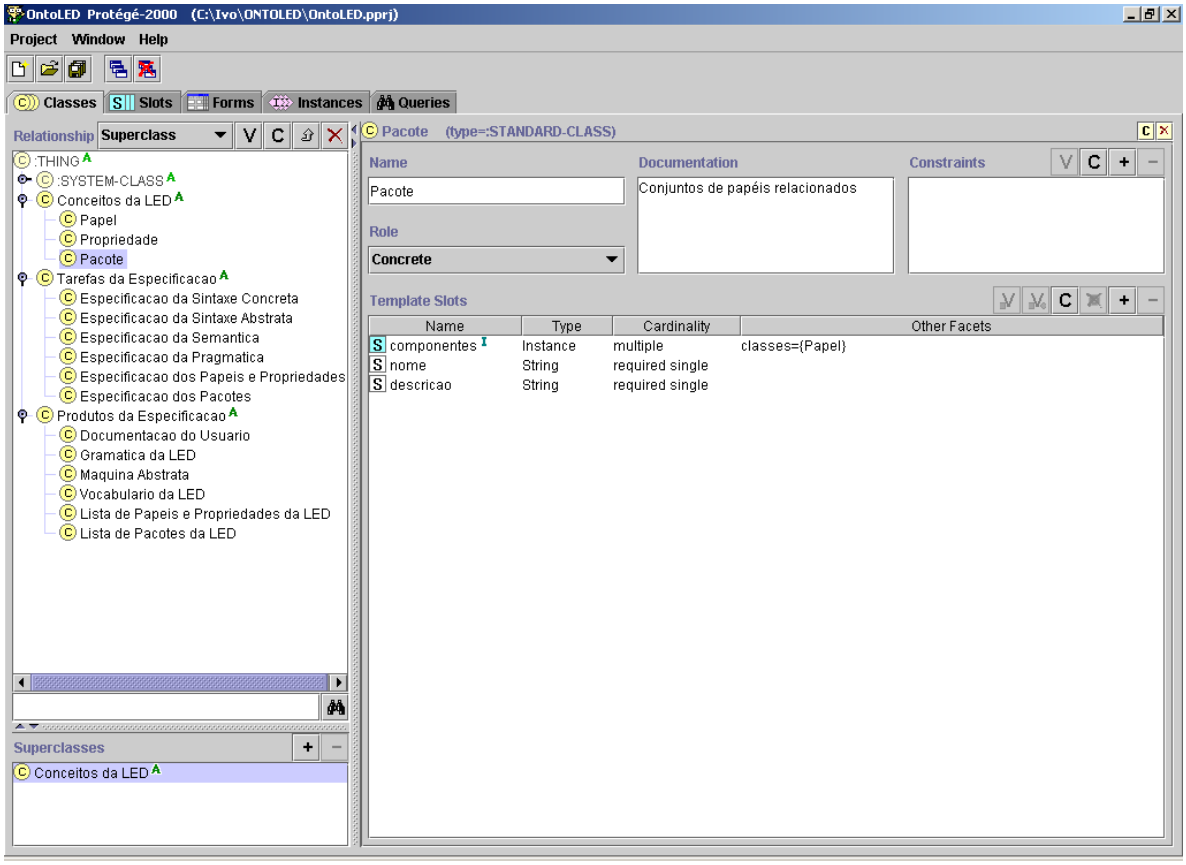


**Figura 46:** A classe *Pacote* e seus slots

Por exemplo, no desenvolvimento da LESRF, no capítulo 6 é definido o pacote *Filtragem* que contém os três papéis: *Modelador de usuário*, *Filtrador* e *Monitor*, que atendem exclusivamente ao objetivo específico *Satisfazer a necessidade de informação em longo prazo dos usuários*, e o pacote *Recuperação* que contém os papéis *Descobridor*, *Indexador* e *Recuperador*, que atendem exclusivamente ao objetivo específico *Satisfazer a necessidade de informação pontual dos usuários* (**Figura 47**).

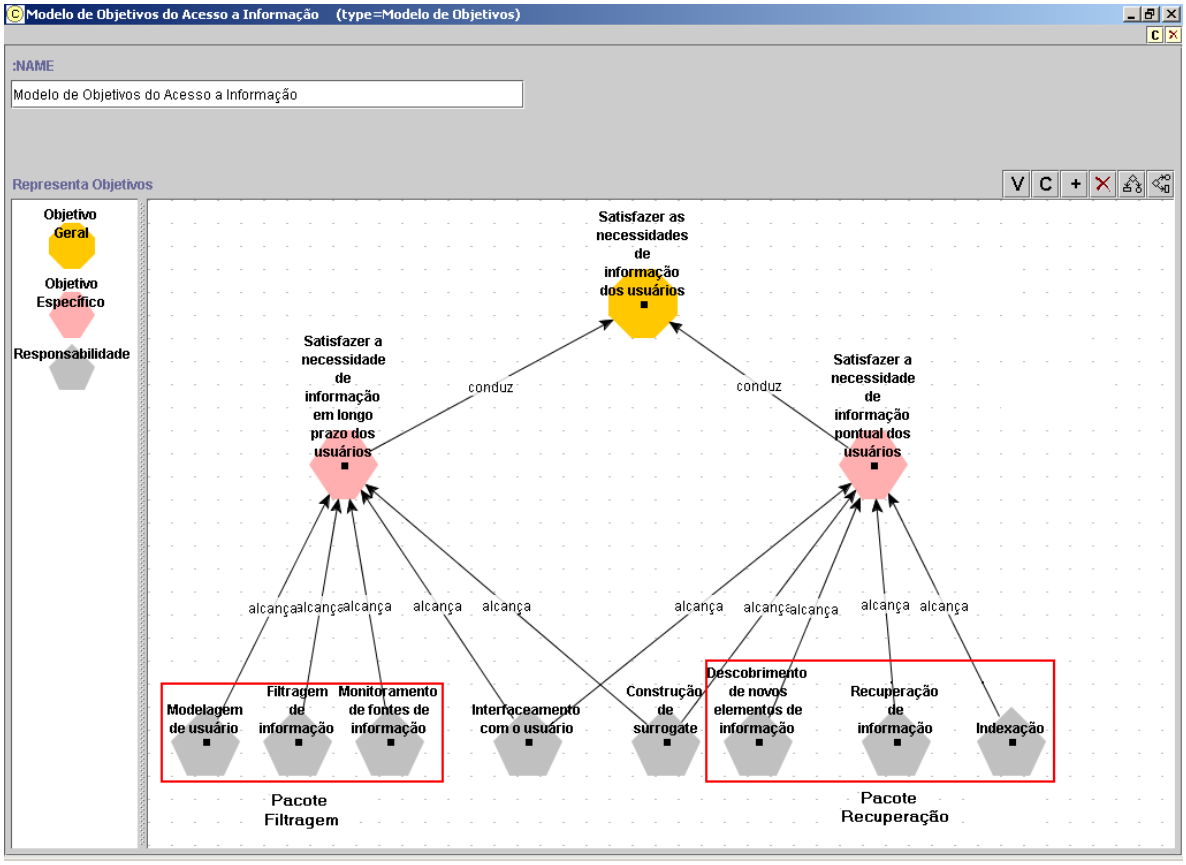


**Figura 47:** Exemplo de identificação dos pacotes na TOD-LED

### 5.4.2 Especificação da Sintaxe Abstrata

As regras de escrita ou sintaxe abstrata das linguagens desenvolvidas com a TOD-LED são simples e refletem seu aspecto puramente declarativo. A sintaxe abstrata de uma linguagem é influenciada pelo fato de ela ser imperativa ou declarativa. As linguagens declarativas tendem a ter regras de escrita mais simples.

A especificação da sintaxe abstrata é feita através da instanciação da classe *Especificação da Sintaxe Abstrata* (**Figura 48**) que cria uma instância da classe gramática da LED.

Uma instância da classe *Gramática da LED* possui um slot *nome*, para que os desenvolvedores forneçam um nome à gramática e um slot *especificação,* onde é incluída a gramática da LED (**Figura 49**).

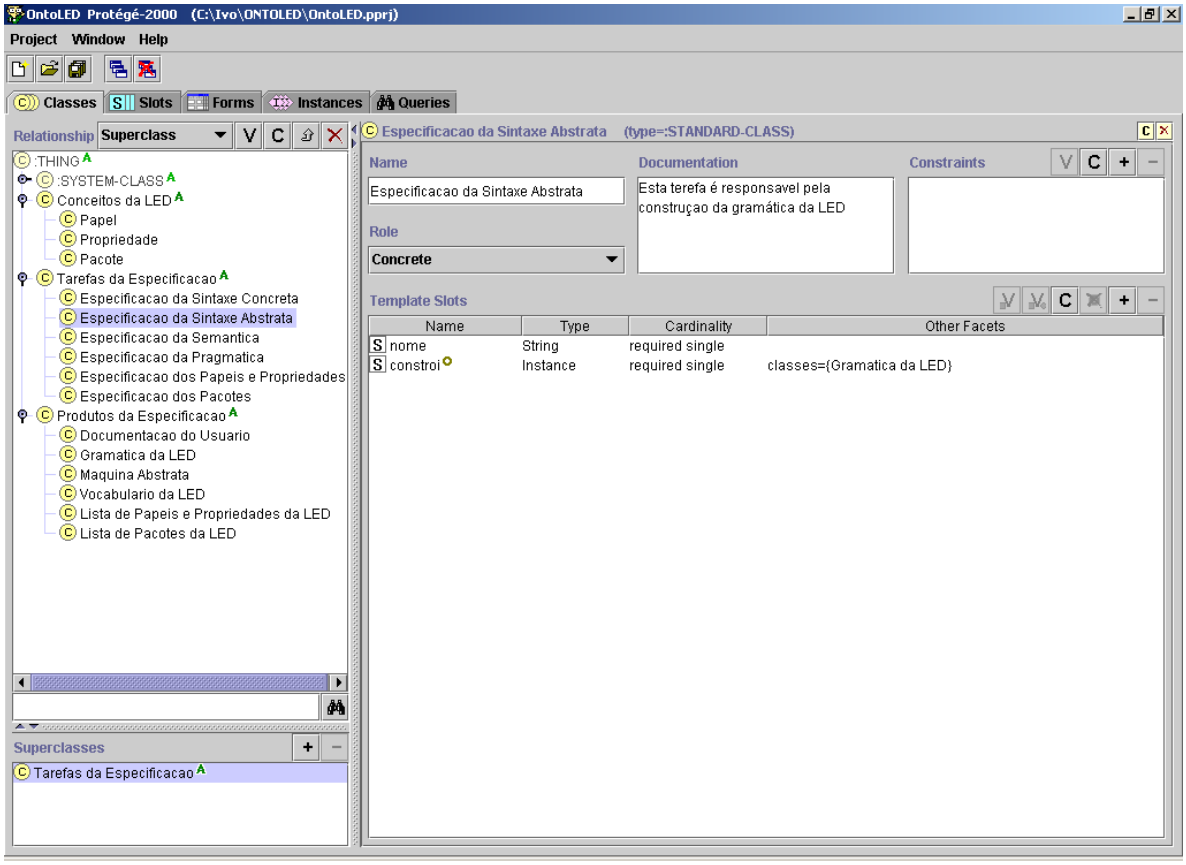


**Figura 48:** A classe Especificação da Sintaxe Abstrata

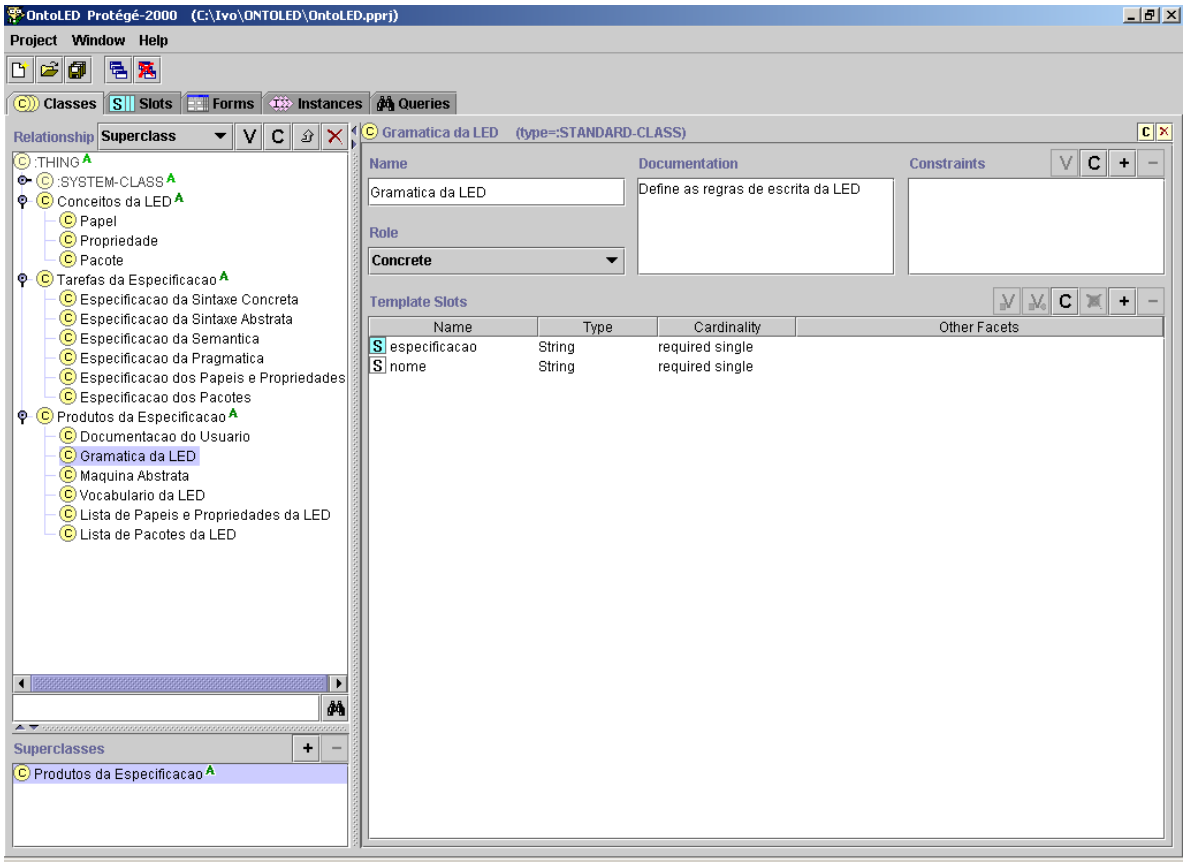


**Figura 49:** A classe Gramática da LED e seus slots

### 5.4.3 Especificação da Semântica

A semântica de uma linguagem corresponde ao significado que é dado aos termos que compõem seu vocabulário. A semântica é definida através de um domínio semântico e de um mapeamento semântico [Schmidt 1986] [Guizzardi, Ferreira e Sinderen 2002]. O domínio semântico corresponde a um conjunto de entidades do domínio e seus relacionamentos. O mapeamento semântico associa os termos da linguagem às entidades do domínio semântico. Os termos da linguagem obtêm seu significado através do relacionamento entre as entidades as quais estão associados e as outras entidades do domínio.

O mapeamento semântico das LED's desenvolvidas com a TOD-LED é simples uma vez que apenas se tem que associar os papéis e as propriedades da LED aos papéis e propriedades correspondentes do domínio semântico. Por este motivo, essa associação é deixada para a fase de implementação da LED, onde o mapeamento será feito por um interpretador ou compilador.

Na TOD-LED, o domínio semântico é definido por uma máquina abstrata [Consel, Marlet, 1998] [Thibault, 1998]. Uma máquina abstrata é um modelo computacional do domínio, que provê suporte para a implementação de todos os sistemas especificados com a LED. Ela especifica as entidades necessárias a expressar soluções para um domínio e como elas se relacionam. Na TOD-LED, a máquina abstrata é constituída do modelo de domínio estendido acrescido das propriedades dos papéis que constam do vocabulário da LED.

A especificação da semântica é feita através da instanciação da classe *Especificação da Semântica* (**Figura 50**), que cria uma instância da classe *Máquina Abstrata*. A classe *Máquina Abstrata* possui os slots *modelo de domínio,* que faz referência ao modelo de domínio estendido, e *diagrama de papéis e propriedades,*

que representa graficamente os papéis da LED e suas propriedades (**Figura 51**), que foram identificados na fase *Especificação da sintaxe concreta*. No estudo de caso apresentado no capítulo 6, a máquina abstrata é a ONTOINFO (modelo de domínio para o acesso à informação) acrescido das propriedades dos papéis da LED.

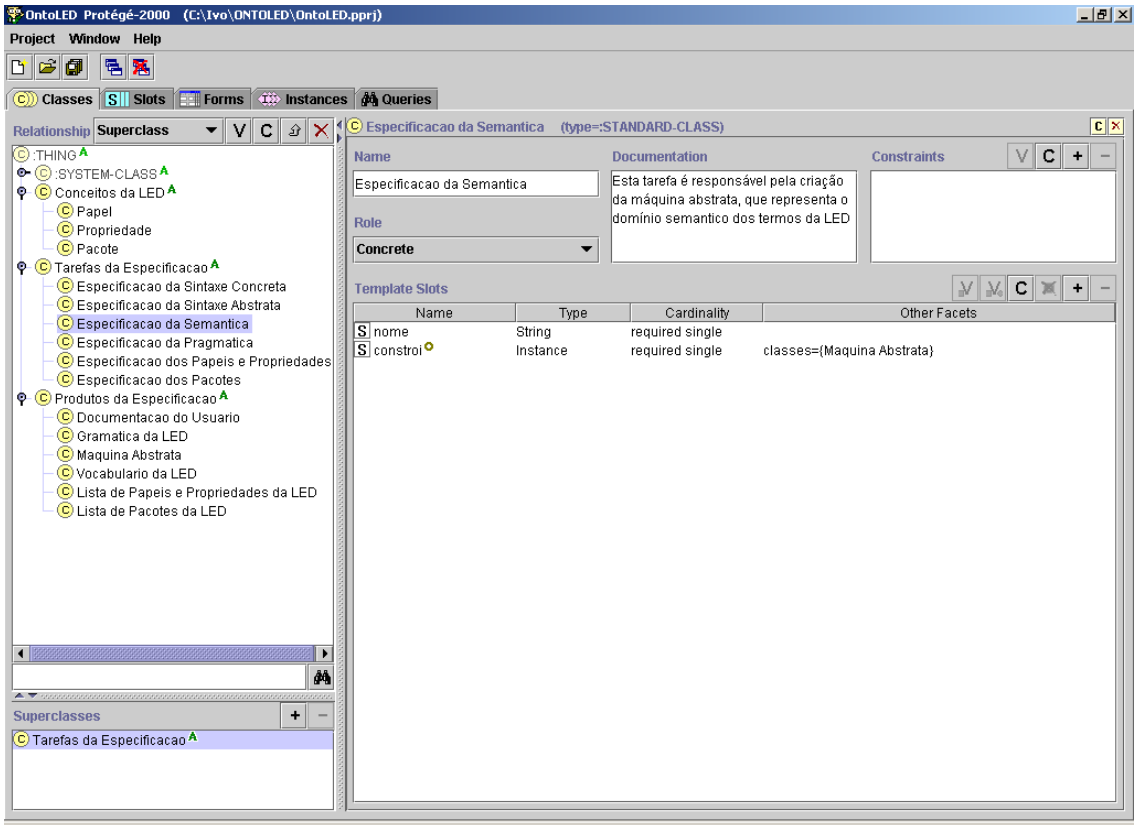


**Figura 50:** A classe Especificação da Semântica

### 5.4.4 Especificação da Pragmática

A pragmática de uma linguagem se relaciona a todos os aspectos de seu uso. O desenvolvimento de uma LED não deve ser terminado sem uma descrição de como usá-la adequadamente.

Devido ao conhecimento prévio das características mais relevantes de LED's desenvolvidas com a TOD-LED (declarativas, baseadas em papéis, propriedades associadas aos papéis, pacotes), a técnica sugere um modelo geral a

ser seguido para a descrição da pragmática. A especificação da pragmática é feita através da instanciação da classe *Especificação da Pragmática* (**Figura 52**) que cria a documentação do usuário.

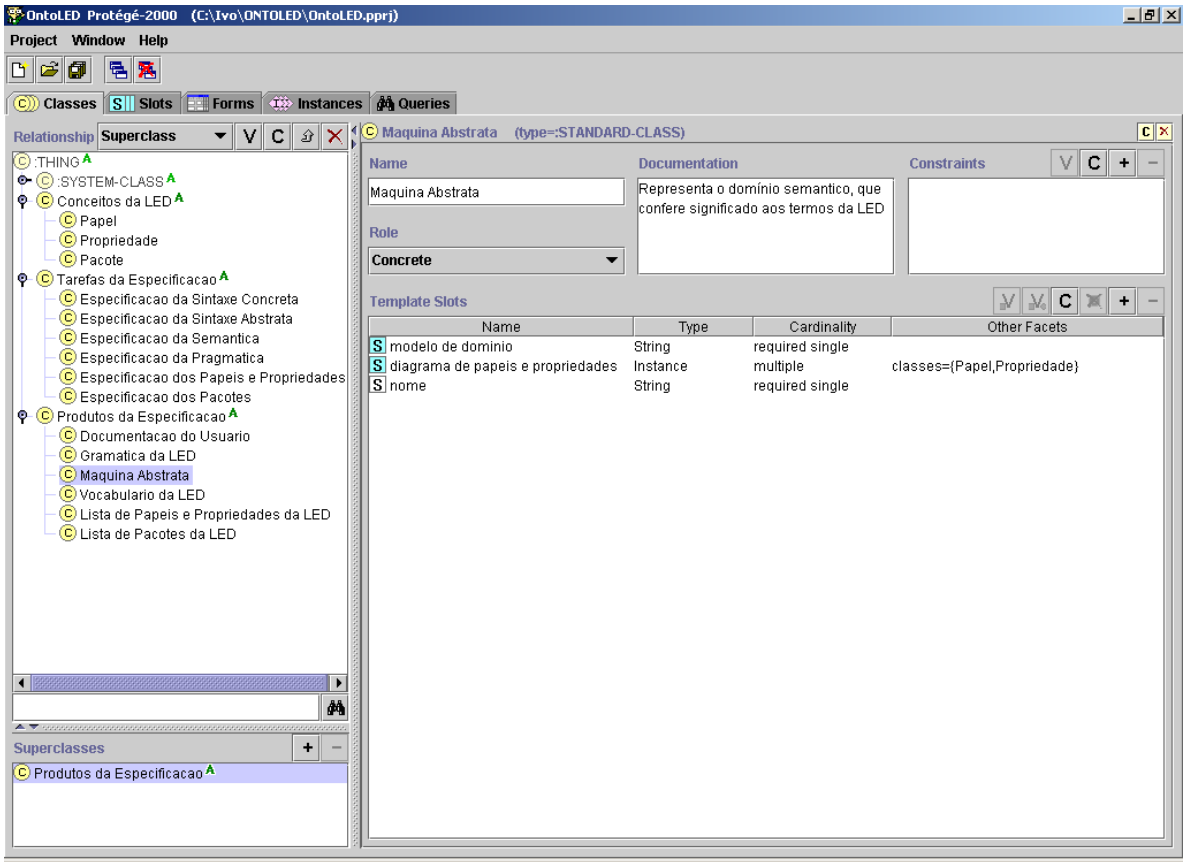


**Figura 51:** A classe Máquina Abstrata e seus slots

A classe *Documentação do Usuário* possui os slots *nome*, *descrição geral da LED*, *descrição do domínio*, *descrição do vocabulário*, *descrição da sintaxe* e *exemplos*, cujas funções são descritas a seguir:

- *Descrição geral da LED:* Uma descrição geral sobre para que a LED se destina. Existem, por exemplo, LED's que se destinam à especificação de consultas a bases de dados, como SQL, ou a estruturação de informação, assim como XML.
- *Descrição do domínio*: Uma descrição do domínio de problema para o qual a LED foi desenvolvida. Consiste em uma descrição

textual para que os usuários principalmente os novatos, possam entender o domínio para o qual a LED foi concebida. Por exemplo, existem LED's desenvolvidas para o domínio das finanças, de drivers de dispositivos gráficos (GAL) e estruturação de informação (XML).

- *Descrição do vocabulário:*
    - Listar os papéis e descrever suas responsabilidades, inclusive discriminando que papéis atendem a quais objetivos específicos. Descrever cada uma das propriedades dos papéis e seus respectivos domínios.
    - Listar os pacotes, os papéis que os compõem e descrever a funcionalidade de cada pacote.
- *Descrição da sintaxe da LED,* que consiste em:
    - Como iniciar e terminar um programa
    - Como declarar um papel
    - Como declarar um pacote
    - Como atribuir um valor a uma propriedade
- *Exemplos*: Uma lista de exemplos para dar aos usuários uma visão concreta do uso da LED.

Por exemplo, a **Figura 53** dá uma visão parcial da documentação do usuário da LESRF.

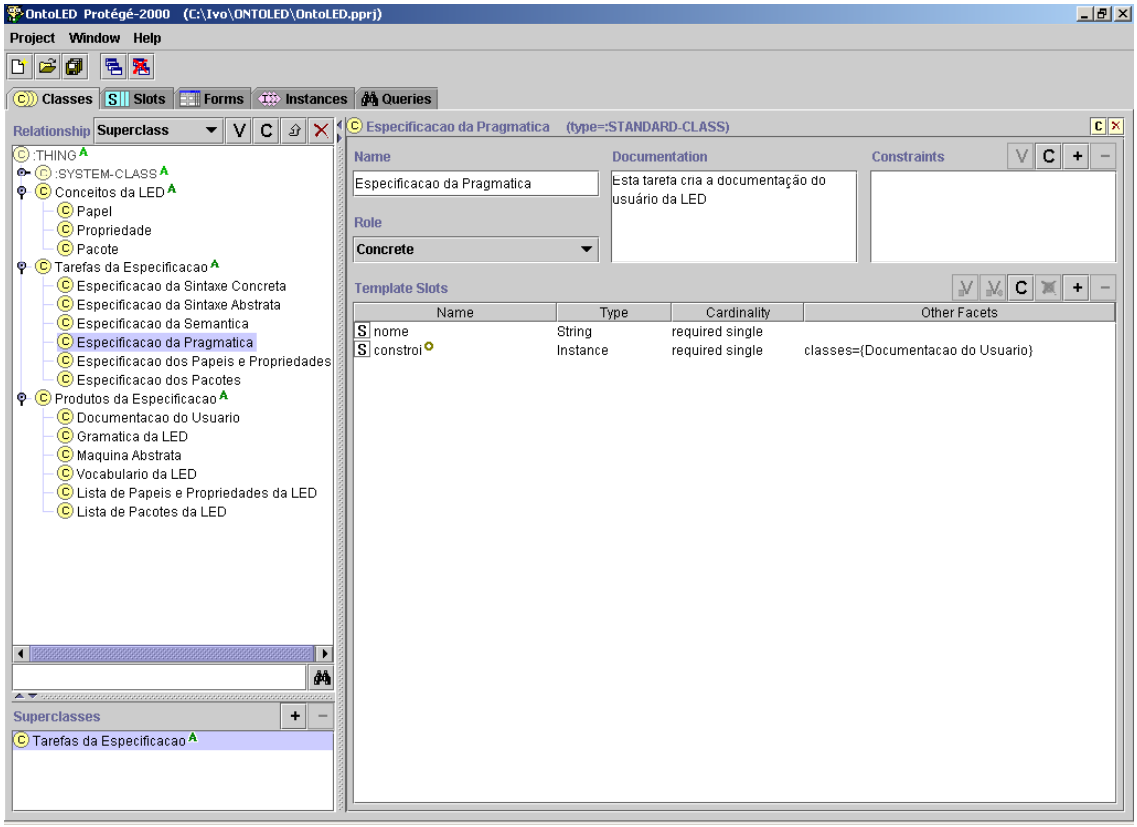


**Figura 52:** A classe Especificação da Pragmática

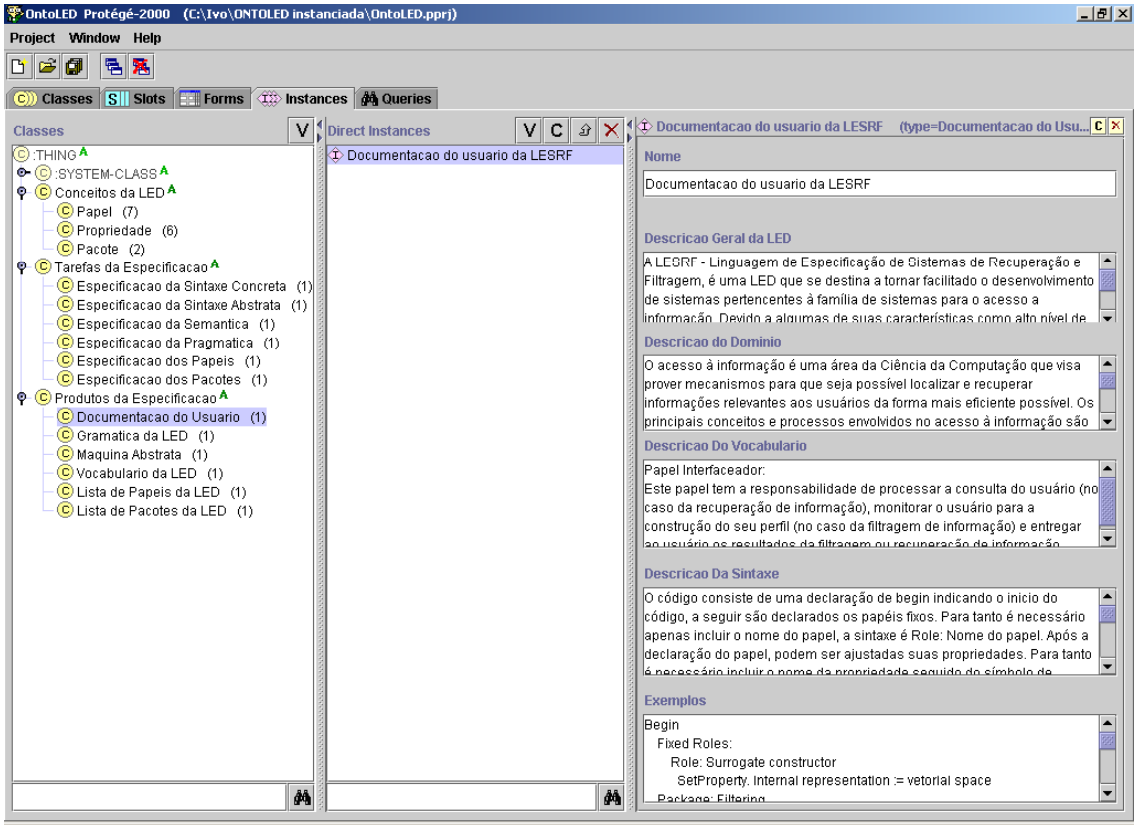


**Figura 53:** Documentação do usuário da LESRF

## 5.5 Considerações finais

Neste capítulo foi apresentada a TOD-LED, uma técnica baseada em ontologias para o desenvolvimento de LED's na Engenharia de Domínio Mutiagente. A TOD-LED tem como insumo modelos de domínio desenvolvidos com a técnica GRAMO e produz especificações de LED’s.

A técnica GRAMO não contemplava a classificação dos conceitos do domínio em fixos ou variáveis, um aspecto fundamental no desenvolvimento de LED's. Por esse motivo a técnica foi estendida através do acréscimo da atividade *Modelagem de variabilidades*.

A TOD-LED foi avaliada através de um estudo de caso que consistiu no desenvolvimento de uma LED para a área do acesso à informação, apresentada no próximo capítulo.

# 6 ESTUDO DE CASO – DESENVOLVIMENTO DA LESRF

Neste capítulo é apresentado um estudo de caso para avaliar a TOD-LED. Para tanto foi desenvolvida uma LED, a LESRF (Linguagem de Especificação de Sistemas para Recuperação e Filtragem de informação), para a geração de sistemas para o domínio da recuperação e filtragem de informação. A motivação para a escolha dessa área de aplicação se deveu principalmente aos fatos de disponibilidade de conhecimento e especialistas do domínio.

Na seção 6.1 é descrito o domínio do problema. A seção 6.2 apresenta a ONTOINFO – um modelo de domínio para a área do acesso à informação baseado em ontologias [Serra, Girardi, 2003], construído com a aplicação da técnica GRAMO estendida. A seção 6.3 mostra a aplicação da TOD-LED e a definição da LESRF.

## 6.1 O Domínio do Acesso à Informação Dinâmica e não Estruturada

O acesso à informação [Salton, 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 são descritos a seguir [Baeza-Yates, Ribeiro-Neto, 1999] [Salton, 1983] [Girardi, 1998].

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. 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, 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. 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;

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

*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;

*Perfil de usuário*: é uma representação do usuário a partir da qual o sistema filtra elementos de informação que possam ser relevantes. O perfil do usuário pode ser capturado de forma implícita (ex: preenchimento de formulário) ou explícita (observação de links percorridos);

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

*Surrogate*: é a representação interna de elementos de informação e das necessidades de informação em longo prazo e pontuais dos usuários;

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

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

## 6.2 ONTOINFO - um Modelo de Domínio para a área do Acesso à Informação

A ONTOINFO [Serra, Girardi, 2003] é um modelo de domínio resultante da aplicação da técnica GRAMO estendida ao conhecimento sobre a área do acesso a informação, reunindo os conceitos de modelagem através das tarefas de modelagem em termos da ONTODUM estendida. Isso se dá pela instanciação das meta-classes correspondentes a modelagem de domínio, que são as modelagens de conceitos, de objetivos, de papéis e de interações. Os produtos resultantes dessas modelagens são os modelos de conceitos, de objetivos, de papéis e de interações.

### 6.2.1 Modelagem de Conceitos

Na construção do modelo de conceitos são inicialmente identificados conceitos relevantes. Em seguida são instanciados os relacionamentos existentes entre esses conceitos. A **Figura 54** mostra o modelo de conceitos, com o uso do editor de ontologia Protégé [Protègè Project, 2002].

### 6.2.2 Modelagem de Objetivos

No modelo de objetivos são criadas instâncias dos objetivos gerais e específicos. Em seguida, os objetivos específicos são vinculados ao objetivo geral. São também criadas instâncias de *Responsabilidade* e vinculadas aos objetivos específicos. A **Figura 55** mostra o modelo de objetivos.

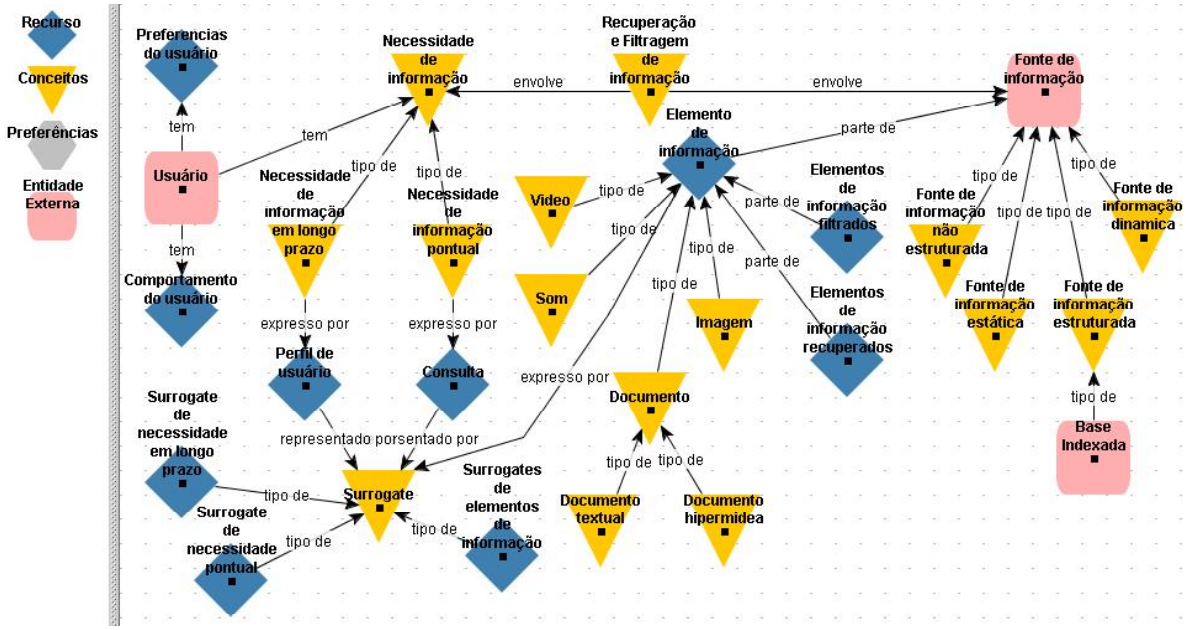


**Figura 54:** Modelo de Conceitos da ONTOINFO

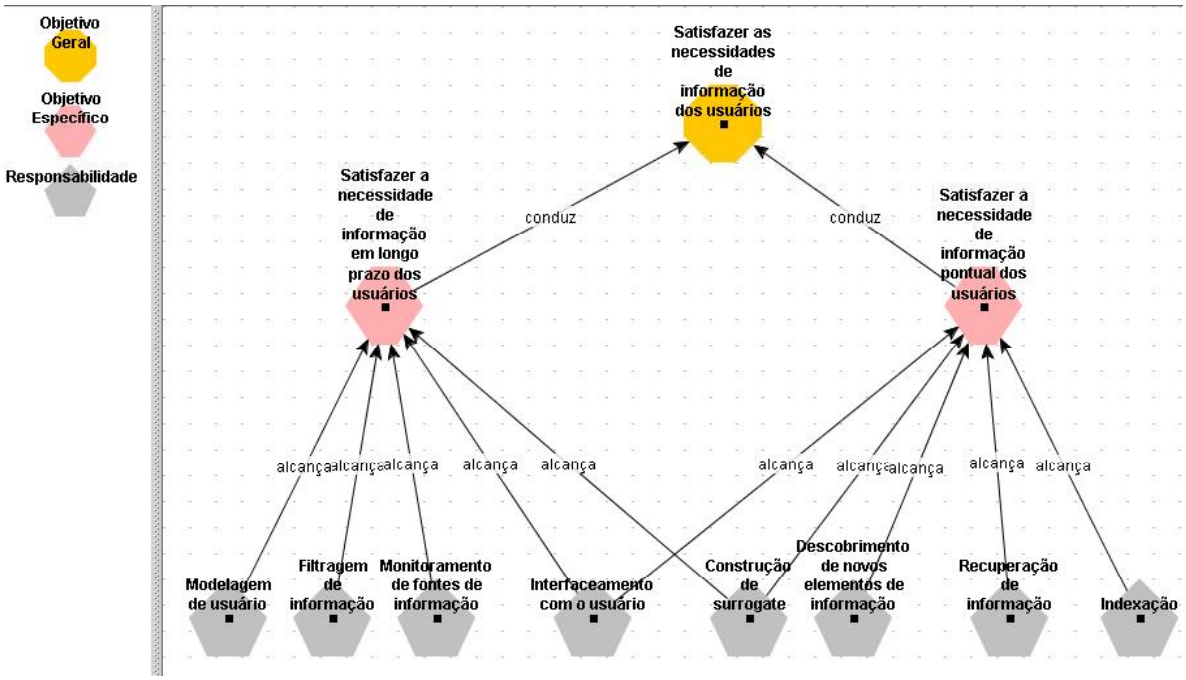


**Figura 55:** Modelo de Objetivos da ONTOINFO

### 6.2.3 Modelagem de Papéis

No modelo de papéis são criadas instâncias da meta-classe *Papel*: *Interfaceador*, *Modelador de usuário*, *Construtor de surrogate*, *Recuperador*, *Filtrador*, *Monitor*, *Descobridor* e *Indexador*.

Na criação dos modelos, primeiro os papéis são associados às respectivas responsabilidades. São criadas instâncias de atividade e vinculadas às respectivas responsabilidades, são vinculadas as atividades aos respectivos conceitos do domínio que elas manipulam, são instanciados os recursos e vinculados às atividades.

A seguir são mostrados os modelos dos oito papéis da ONTOINFO. Incluindo uma descrição textual e sua representação gráfica no editor de ontologias Protègè [Protègè Project, 2002].

- *Interfaceador*

Este papel tem a responsabilidade de processar a consulta do usuário (no caso da recuperação de informação), monitorar o usuário para a construção do seu perfil (no caso da filtragem de informação) e entregar ao usuário os resultados da filtragem ou recuperação de informação.

O papel *Interfaceador* interage com a entidade externa *Usuário* de quem recebe uma necessidade de informação pontual (expressa por uma consulta) ou uma necessidade de informação em longo prazo (expressa por um perfil) e a quem entrega os elementos de informação recuperados ou filtrados.

Através da atividade *validar consulta* o Interfaceador verifica se a consulta foi expressa corretamente pelo usuário, de acordo com as regras da linguagem de especificação de consulta. Através da atividade monitorar usuário o *Interfaceador* captura de forma implícita ou explícita informações a respeito dos usuários para a construção e manutenção de seus perfis. Através da atividade *entregar resultados* o *Interfaceador* disponibiliza aos usuários os elementos de informação provenientes da filtragem ou recuperação. A **Figura 56** mostra o modelo do papel *Interfaceador*.

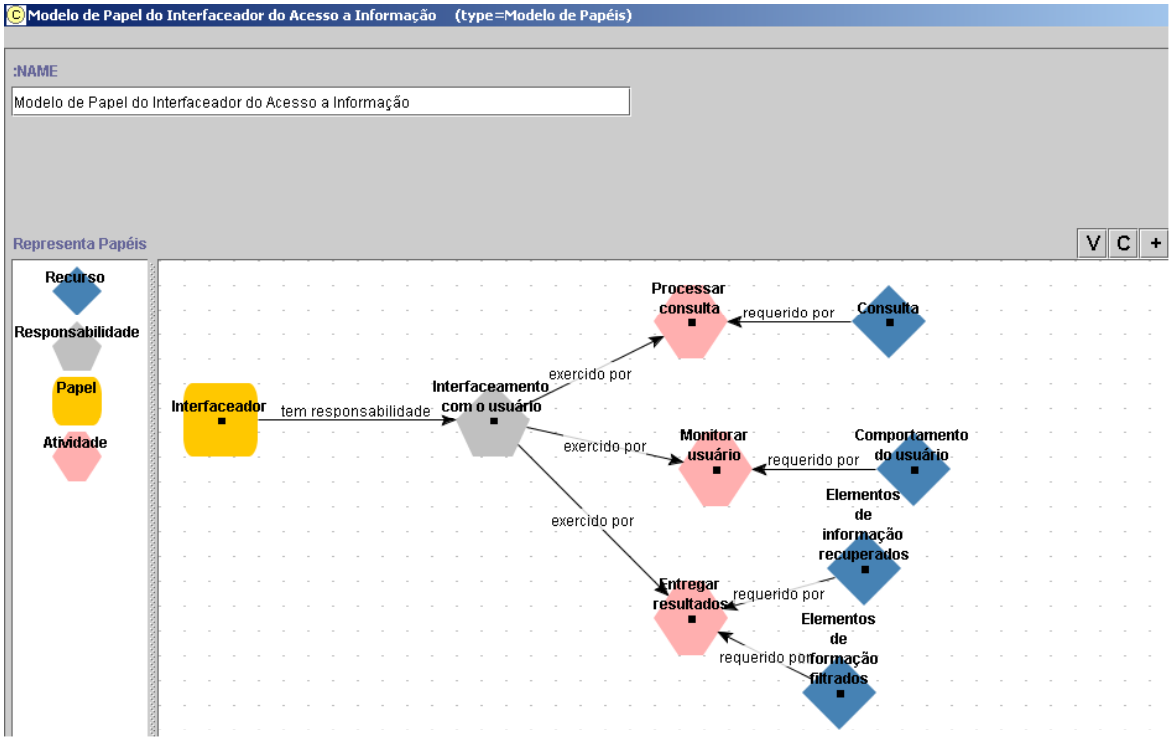


**Figura 56:** Modelo do papel *Interfaceador*

- *Modelador* de *usuário*

Este papel tem a responsabilidade gerenciar os perfis dos usuários. Para tanto o papel *Modelador de usuário* interage com o papel *Interfaceador* de quem recebe informações a respeito das preferências do usuário, através da atividade *criar e manter modelo de usuário*. O *Modelador de usuário* pode solicitar ao *Interfaceador* que capture informações do usuário de forma implícita (links percorridos, tempo despendido em um site, etc) ou explícita (através de formulário). Essas informações são utilizadas para criar e manter os perfis dos usuários. O *Modelador de usuário* também recebe do papel *Filtrador*, os elementos de informação filtrados e os armazena de acordo com os perfis para os quais são relevantes. Os elementos de informação armazenados são posteriormente enviados ao papel *Interfaceador* através da atividade *encaminhar elementos filtrados*, para que possam ser entregues ao usuário. A **Figura 57** mostra o modelo do papel *Modelador de usuário*.

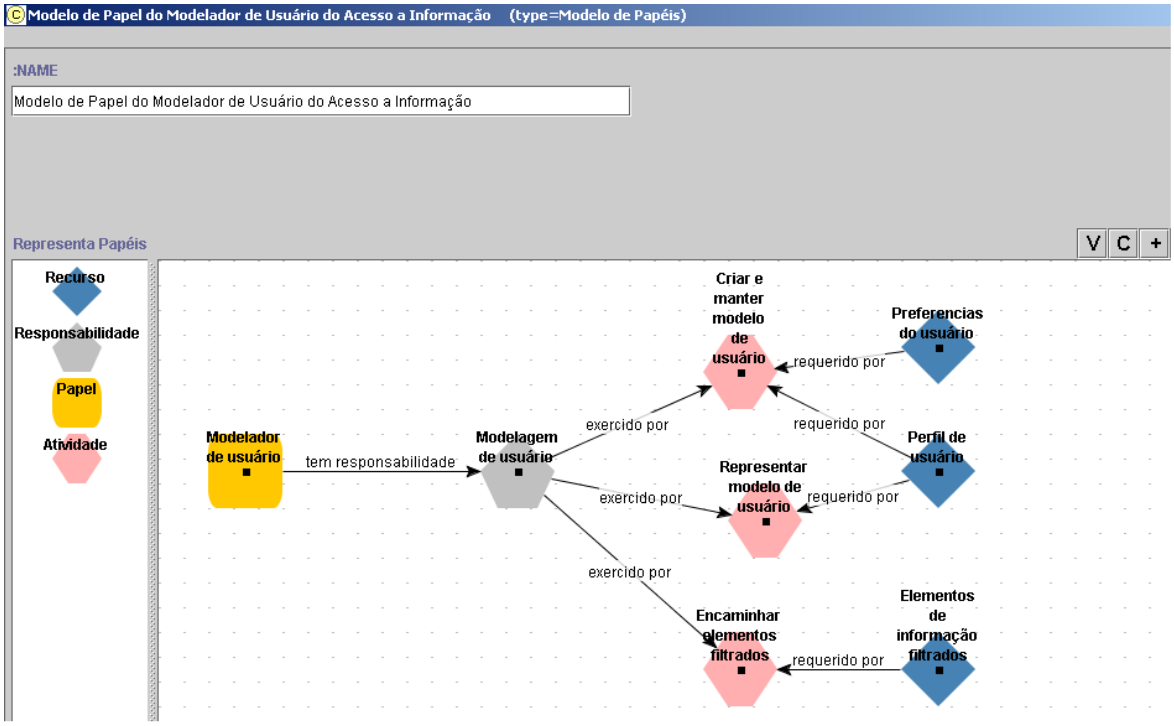


**Figura 57:** Modelo do papel *Modelador de usuário*

- *Filtrador*

Este papel tem a responsabilidade de filtrar elementos de informação de acordo com os perfis dos usuários. A filtragem ocorre sempre que o *Filtrador* recebe através do papel *Construtor de Surrogate*, elementos de informação advindos do papel *Monitor*.

Para tanto o *Filtrador* realiza duas atividades, a primeira consiste em comparar os surrogates dos elementos de informação com os surrogates das consultas inferidas dos perfis, a segunda atividade é a *análise de similaridade* na qual os elementos de informação são ordenados pelo critério de similaridade com a necessidade de informação.

Por fim o filtrador envia os elementos de informação filtrados segundo os perfis para o papel *Modelador* de usuário que se encarrega de armazena-los de

acordo com os respectivos perfis para que possam ser oportunamente entregues aos usuários. A **Figura 58** mostra o modelo do papel *Filtrador*.

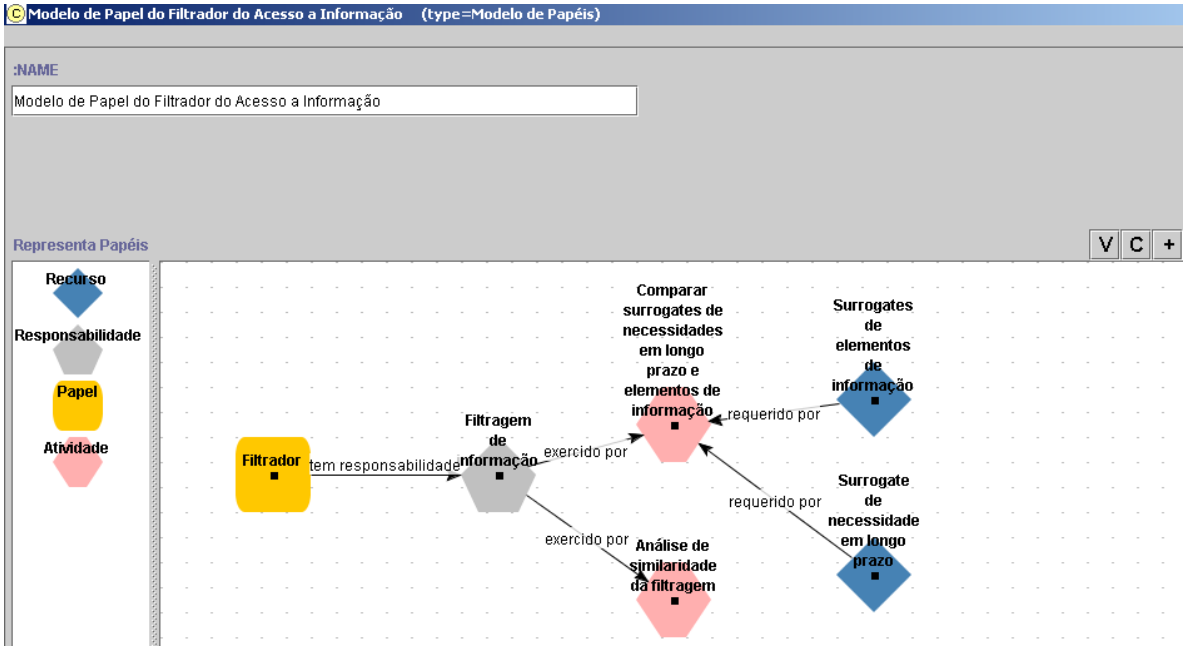


**Figura 58:** Modelo do papel *Filtrador*

- *Monitor*

Esse papel tem a responsabilidade de capturar novos elementos de informação em fontes de informação por ele monitoradas. O *Monitor* solicita ao *Construtor de Surrogate* que construa os surrogates dos novos elementos de informação para que possam em seguida ser encaminhados ao papel *Filtrador* para que possam ser filtrados. A **Figura 59** mostra o modelo do papel *Monitor*.

- *Construtor de surrogate*

Este papel tem a responsabilidade de construir os surrogates dos elementos de informação (atividade *construir surrogate de elemetos de informação*), das necessidades de informação em longo prazo (atividade *construir surrogate de perfis*) e das necessidades de informação pontuais dos usuários (atividade *construir surrogate de consulta*). Para tanto o *Construtor de Surrogate* interage com o papel *Interfaceador*, de quem recebe a especificação de uma consulta de usuário (no caso

da recuperação de informação), com o papel *Modelador de usuário* de quem recebe os perfis dos usuários (no caso da filtragem de informação) e com o papel *Descobridor* e *Monitor* de quem recebe os elementos de informação. A **Figura 60** mostra o modelo do papel *Construtor de surrogate*.

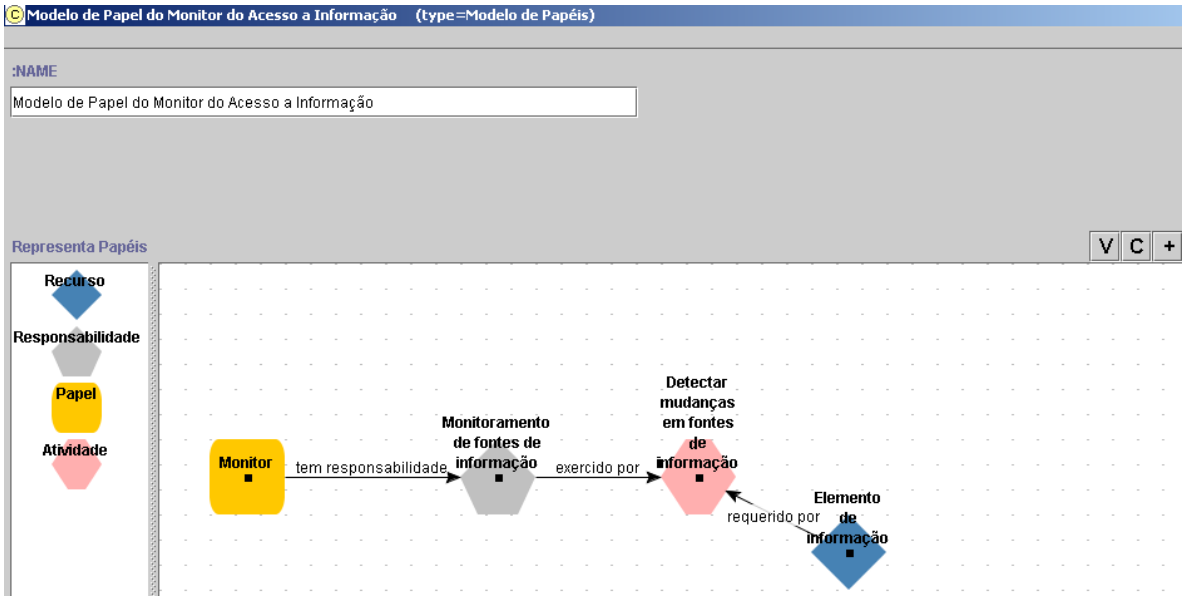


**Figura 59:** Modelo do papel *Monitor*

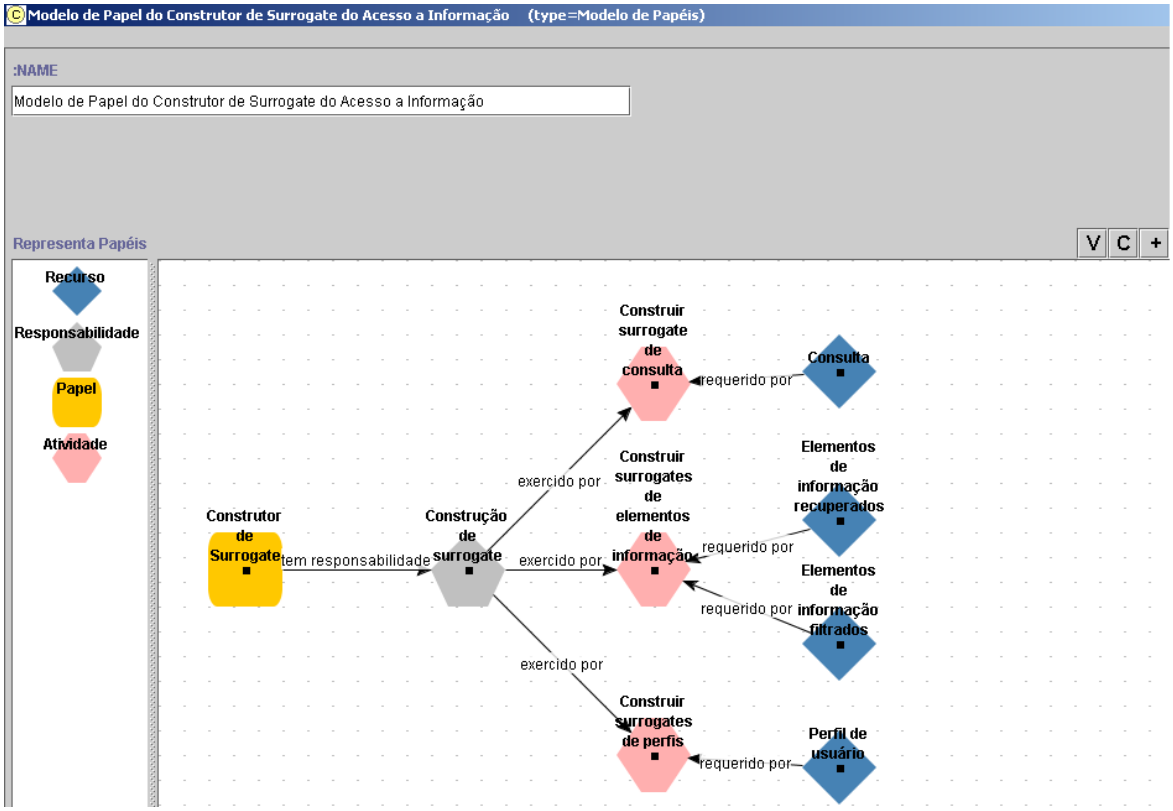


**Figura 60:** Modelo do papel *Construtor de surrogate*

- *Recuperador*

Este papel tem a responsabilidade de recuperar elementos de informação relevantes de acordo com uma necessidade de informação pontual do usuário. Para tanto o papel *Recuperador* recebe do *Construtor de Surrogate* o surrogate da consulta do usuário e busca na entidade externa *base indexada* os surrogates dos elementos de informação.

De posse desses dois recursos o *Recuperador* realiza três atividades, primeiro faz a *comparação entre os surrogates da consulta e dos elementos de informação*, em seguida realiza uma *análise de similaridade* para ordenar os elementos de informação pelo critério de relevância e por último *envia resultados* ao papel *Interfaceador* para que possam ser entregues ao usuário. A **Figura 61** mostra o modelo do papel *Recuperador*.

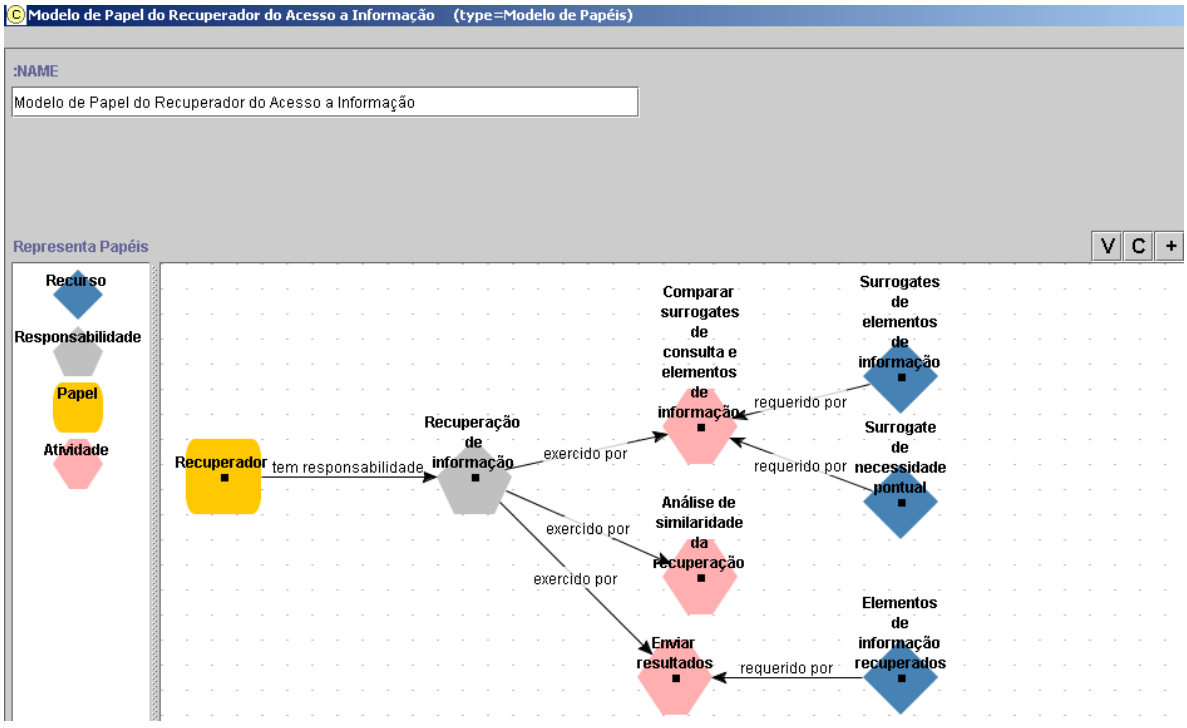


**Figura 61:** Modelo do papel Recuperador

- *Modelo do papel Indexador*

Este papel tem a responsabilidade de indexar os elementos de informação descobertos pelo papel *Descobridor*.

O papel *Descobridor* solicita ao papel *Construtor de surrogate* que construa as representações internas dos elementos de informação descobertos. Em seguida essas representações internas são repassadas ao papel *Indexador*. O *Indexador* interage também com a entidade externa *base indexada* onde armazena os elementos de informação descobertos. A base indexada contém os elementos de informação que poderão ser posteriormente recuperados.

Esse papel executa apenas uma atividade chamada, *indexar surrogates de elementos de informação* que é executada sobre a entidade externa *base indexada*. A **Figura 62** mostra o modelo do papel *Indexador*.

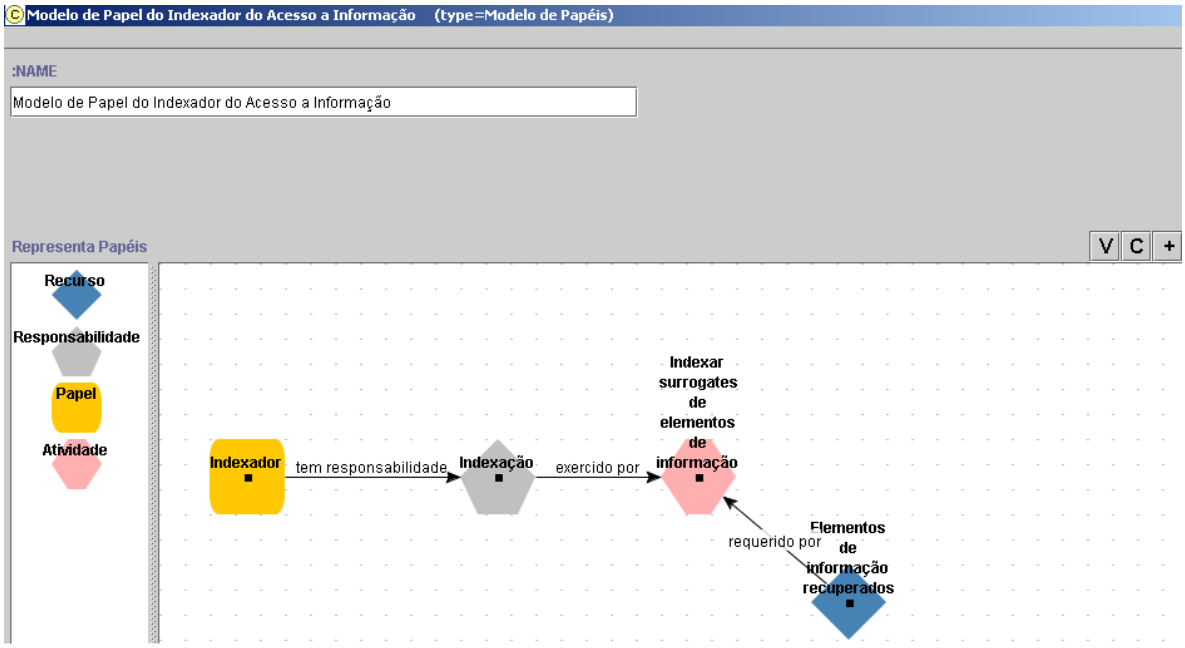


**Figura 62:** Modelo do papel Indexador

- *Modelo do papel Descobridor*

Este papel tem a responsabilidade de descobrir novos elementos de informação de acordo com as áreas de interesse do usuário. Para tanto o *Descobridor* acessa as fontes de informação e captura novas ocorrências de elementos de informação. A **Figura 63** mostra o modelo do papel *Descobridor*.

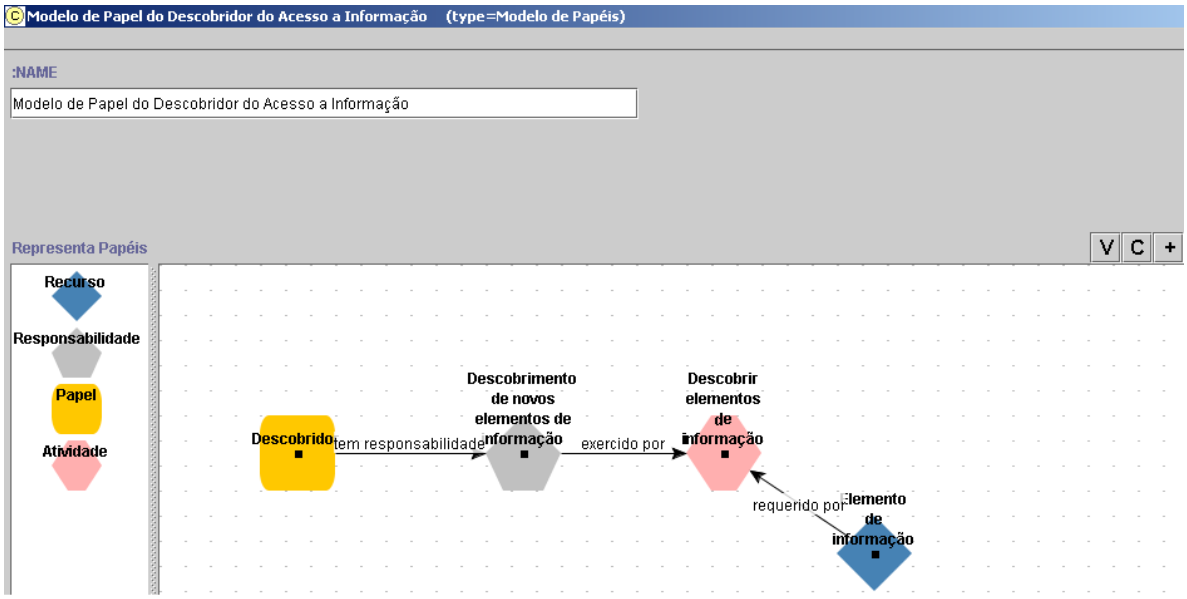


**Figura 63:** Modelo do papel Descobridor

## 6.2.4 Modelagem de Interações

No modelo de interações são criadas instâncias de entidade externa: usuário, índice, base de perfis e fonte de informação. São adicionados os papéis e criadas instâncias das interações envolvendo papéis e entidades externas. A **Figura 64** mostra o modelo de interações referente ao objetivo específico *Satisfazer a necessidade de informação pontual dos usuários* que corresponde a recuperação de informação. A **Figura 65** mostra o modelo de interações referente ao objetivo específico *Satisfazer a necessidade de informação em longo prazo dos usuários* que corresponde a filtragem de informação.

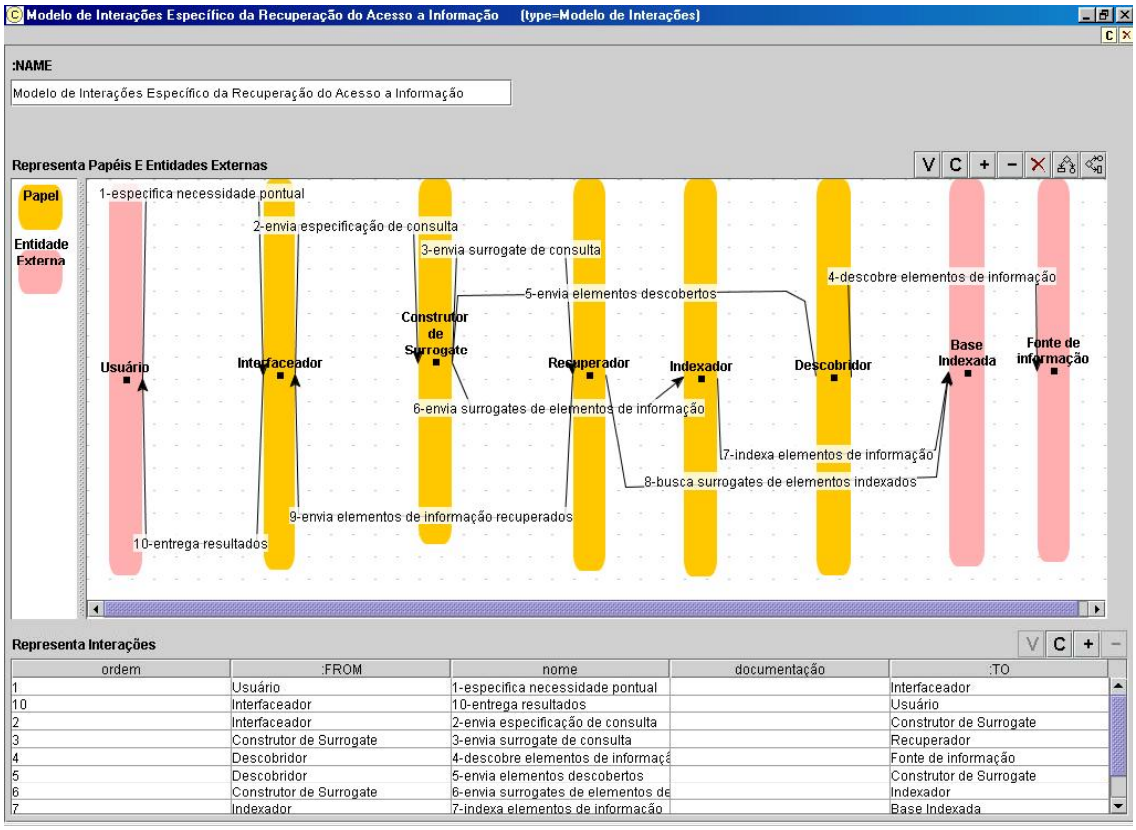


**Figura 64:** Modelo de Interações da recuperação de informação

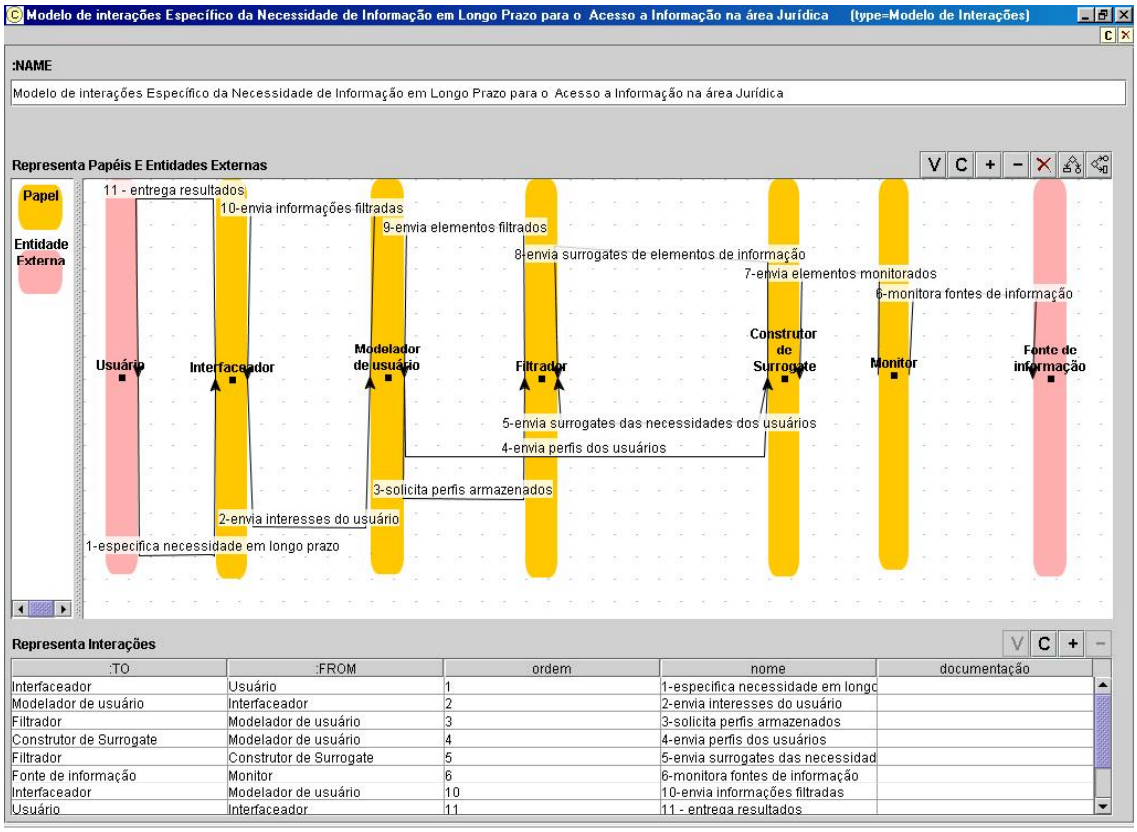


**Figura 65:** Modelo de Interações da filtragem de informação

### 6.2.5 Modelagem de Variabilidades

Nesta atividade, as instâncias dos conceitos *Responsabilidade*, *Papel*, *Atividade*, *Recurso*, *Objetivo específico* e *Objetivo geral*, são classificados em fixos ou variáveis, de acordo com as regras apresentadas na seção 5.1.2. A classificação das responsabilidades e papéis é a mesma, uma vez que existe uma associação *um para um* entre estes conceitos. A **Figura 66** mostra o papel *Construtor de surrogate* classificado como *fixo,* uma vez que atende a todos os objetivos específicos. A **Figura 67** mostra o papel *Recuperador* classificado como *variável,* uma vez que atende a apenas um dos objetivos específicos. A **Tabela 8** mostra a classificação de todos os papéis da ONTOINFO. A **Tabela 9** mostra a classificação das atividades dos papéis, e a **Tabela 10** mostra a classificação dos recursos.

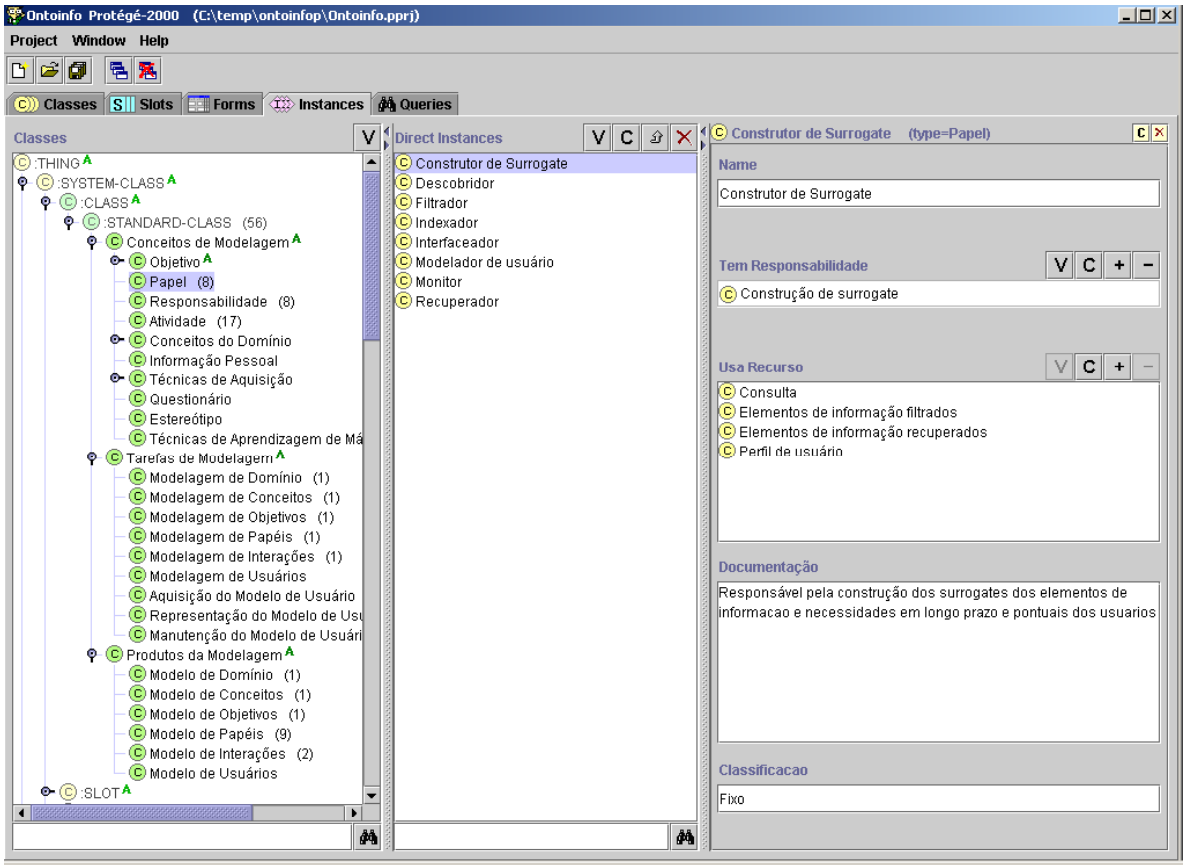


**Figura 66:** Classificação do papel *Construtor de surrogate*

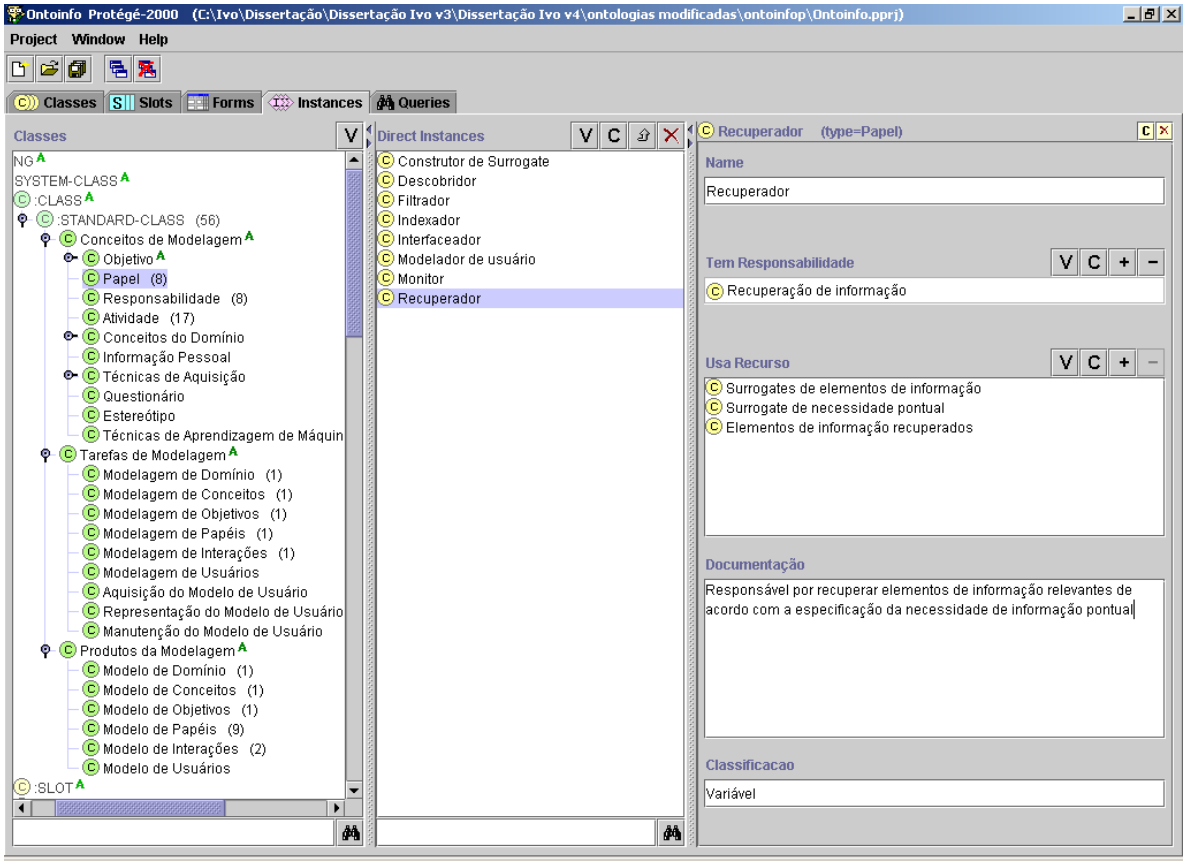


**Figura 67:** Classificação do papel *Recuperador*

| Papel | Classificação |
|---|---|
| Interfaceador | Fixo |
| Modelador de usuário | Variável |
| Filtrador | Variável |
| Descobridor | Variável |
| Recuperador | Variável |
| Construtor de surrogate | Fixo |
| Monitor | Variável |
| Indexador | Variável |

**Tabela 8:** Classificação dos papéis da ONTOINFO

| Papel | Atividade | Classificação das atividades |
|---|---|---|
| Interfaceador | Processar consulta | Variável |
| | Monitorar usuário | Variável |
| | Entregar resultados | Fixa |
| Modelador de usuário | Criar e manter modelo de usuário | Variável |
| | Representar modelo de usuário | Variável |
| | Encaminhar elementos filtrados | Variável |
| Filtrador | Fazer comparação entre surrogates de elementos de informação e surrogates de necessidade de informação em longo prazo | Variável |
| | Análise de similaridade da filtragem | Variável |
| Descobridor | Descobrir elementos de informação | Variável |
| Recuperador | Fazer comparação entre consultas e elementos de informação | Variável |
| | Análise de similaridade da recuperação | Variável |
| | Enviar resultados | Variável |
| Monitor | Detectar mudanças em fontes de informação | Variável |
| Indexador | Indexar surrogates de elementos de informação | Variável |
| Construtor de surrogate | Construir surrogate de consulta | Variável |
| | Construir surrogates de elementos de informação | Fixa |
| | Construir surrogates de perfis | Variável |

**Tabela 9:** Classificação das atividades da ONTOINFO

## 6.3 Desenvolvimento da LESRF

Nesta seção a TOD-LED é aplicada para a especificação da LESRF (Linguagem de Especificação de Sistemas para Recuperação e Filtragem de informação). A LESRF é uma LED que foi desenvolvida no intuito de avaliar a TOD-LED. Nas seções seguintes é apresentado o processo de seu desenvolvimento.

### 6.3.1 Especificação da Sintaxe Concreta

A *Especificação da Sintaxe Concreta* compreende a *Especificação dos Papéis e Propriedades* e a *Especificação dos Pacotes*.

| Recurso | Papéis | Classificação |
|---|---|---|
| Comportamento do usuário | Interfaceador | Variável |
| Consulta | Construtor de surrogate | Variável |
| | Interfaceador | |
| Elemento de informação | Descobridor | Fixo |
| | Monitor | |
| Elementos de informação filtrados | Construtor de surrogate | Variável |
| | Interfaceador | |
| | Modelador de usuário | |
| Elementos de informação recuperados | Construtor de surrogate | Variável |
| | Indexador | |
| | Interfaceador | |
| | Recuperador | |
| Perfil de usuário | Construtor de surrogate | Varável |
| | Modelador de usuário | |
| Preferencias do usuário | Modelador de usuário | Variável |
| Surrogate de necessidade em longo prazo | Filtrador | Variável |
| Surrogate de necessidade pontual | Recuperador | Variável |
| Surrogates de elementos de informação | Filtrador | Fixo |
| | Recuperador | |

**Tabela 10:** Classificação dos recursos da ONTOINFO

6.3.1.1 Especificação dos papéis e propriedades

Nesta tarefa é criada a *Lista de Papéis e Propriedades da LED*, que contém os papéis e propriedades que fazem parte do vocabulário da LED. No momento da criação de cada papel, são também especificadas suas eventuais propriedades. De acordo com o que é definido pela TOD-LED, os papéis que fazem parte do vocabulário de uma LED são: papéis variáveis e papéis fixos que possuam propriedades. Os papéis variáveis e fixos (**Tabela 8**) foram identificados na atividade *Modelagem de Variabilidades* da GRAMO estendida e constam do modelo de domínio estendido. A *Lista de Papéis e Propriedades da LED* da LESRF é mostrada na **Figura 68**. As propriedades e seus domínios foram identificados baseado no

conhecimento do domínio. A **Tabela 11** mostra os papéis da LESRF que possuem propriedades. A **Figura 69** mostra a propriedade *Técnica de aquisição de perfil* do papel *Modelador de usuário*, sua descrição e domínio.

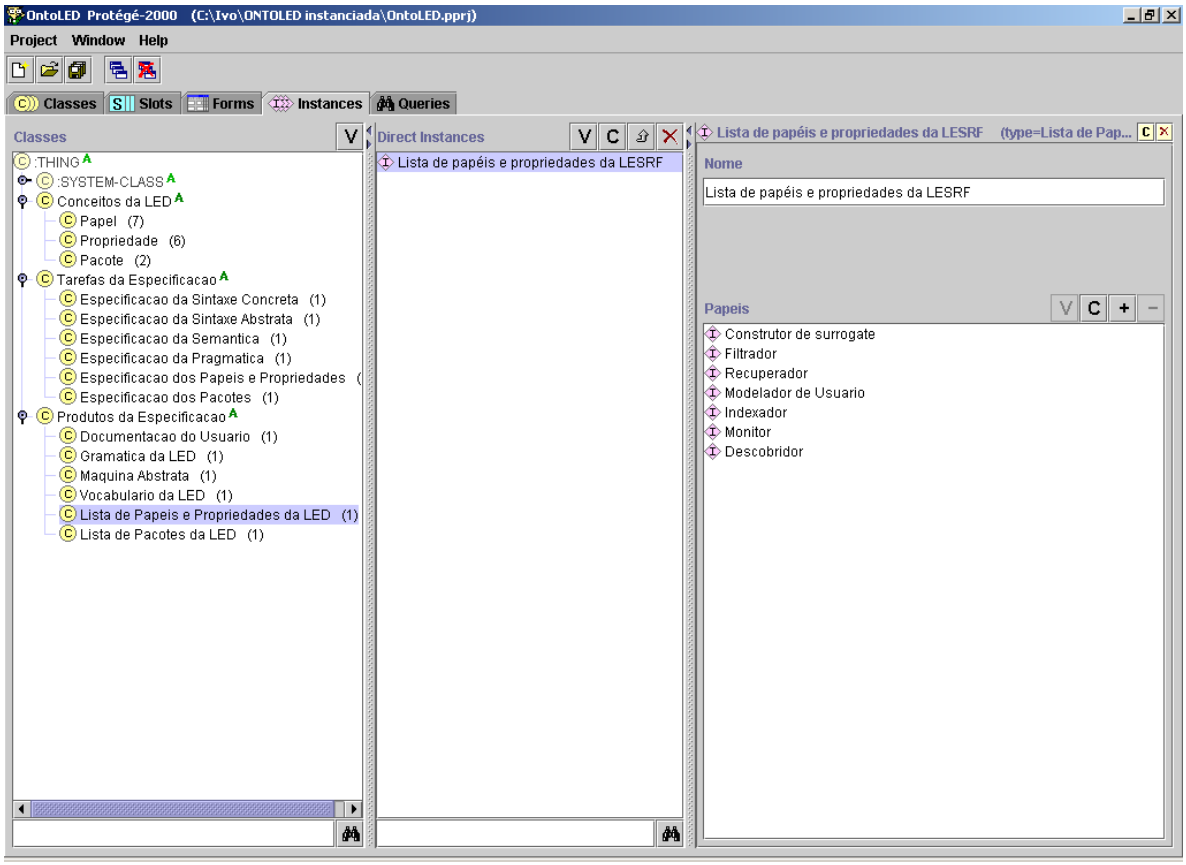


**Figura 68:** Lista de papéis e propriedades da LESRF

| Papel | Propriedade | Domínio | Descrição |
|---|---|---|---|
| Construtor de surrogate | Representa-ção interna | - Probabilístico<br>- Espaço vetorial | Especifica a forma como serão representados internamente ao sistema os elementos de informação a necessidade e as necessidades de informação em longo prazo e pontuais dos usuários |
| Modelador de usuário | Técnica de aquisição de perfil | - Implícito<br>- Explícito | A forma como são adquiridas as informações necessárias para a construção e manutenção dos perfis dos usuários |
| | Técnica de representação de perfil | - Ontologia<br>- Rede semântica<br>- Rede neural | O formalismo utilizado pelo sistema para a representação dos perfis dos usuários |
| Descobridor | Natureza do site | Qualquer domínio de segunda ordem (.com, .gov, .org) | Os tipos de sites nos quais o papel *Descobridor* deve procurar por novos elementos de informação |
| | Nacionalidade do site | Qualquer domínio de primeira ordem (.br, .uy) | A nacionalidade dos sites nos quais o papel *Descobridor* deve procurar por novos elementos de informação |

**Tabela 11:** Papéis da LESRF que possuem propriedades, e seus domínios

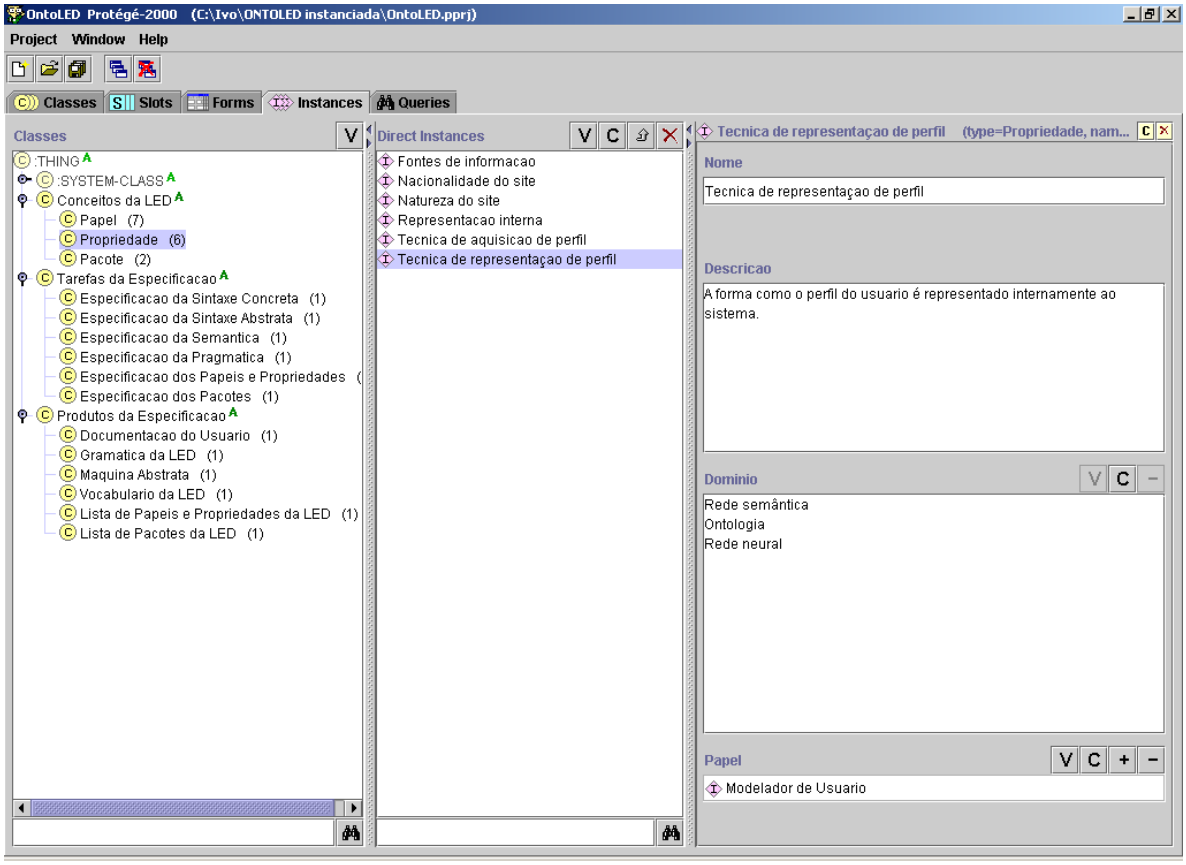


**Figura 69:** A propriedade *Técnica de aquisição de perfil* do *Modelador de usuário*

### 6.3.1.2 Especificação dos Pacotes

Nesta tarefa é criada a *Lista de Pacotes da LED*, que contém os pacotes que fazem parte do vocabulário da LED. De acordo com a TOD-LED, fazem parte de um pacote os papéis que atendem exclusivamente a um mesmo objetivo específico. A  mostra o modelo de objetivos da ONTOINFO e a identificação dos pacotes da LESRF. São definidos dois pacotes: *Filtragem* e *Recuperação*. O pacote *Filtragem* contém os seguintes papéis: *Modelador de usuário*, *Filtrador* e *Monitor*. O pacote *Recuperação* contém os seguintes: *Recuperador*, *Indexador* e *Descobridor*. A **Figura 70** mostra a lista de pacotes da LESRF.

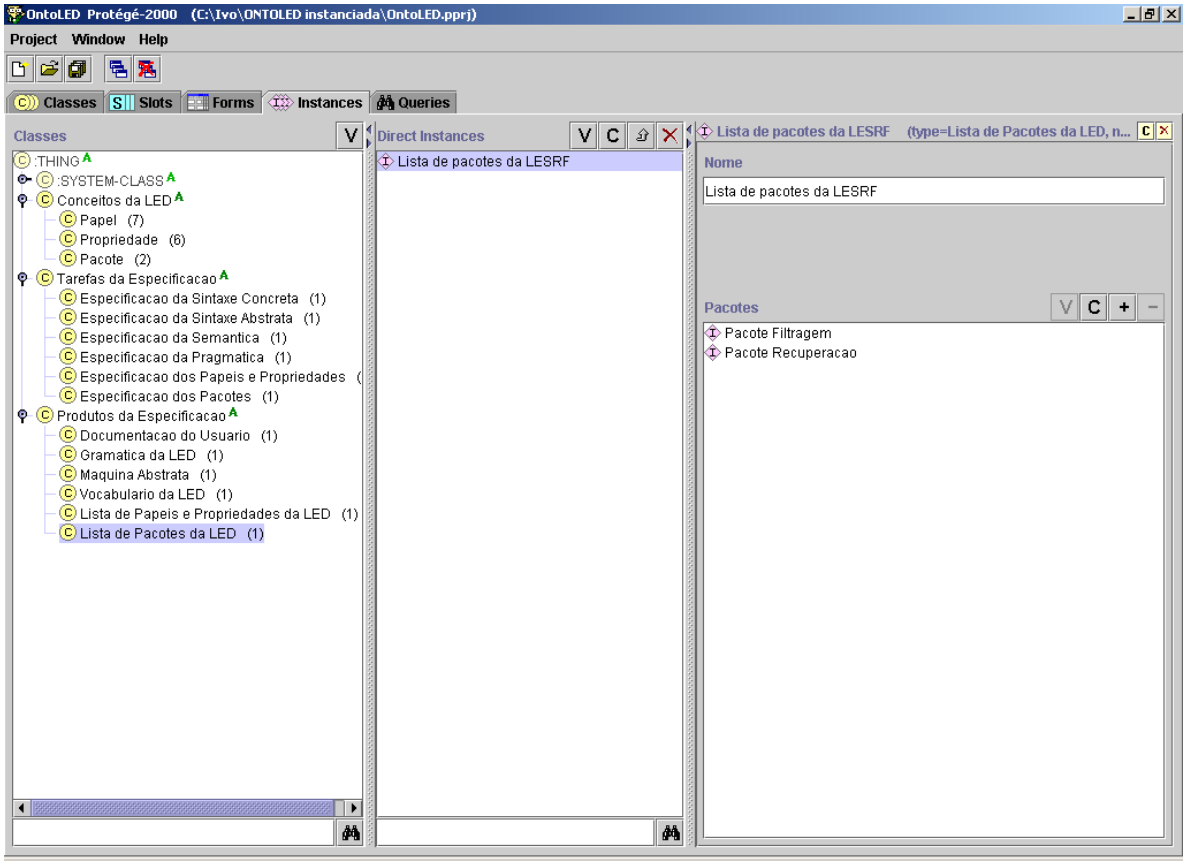


**Figura 70:** Lista de pacotes da LESRF

### 6.3.1.3 Especificação da Sintaxe Abstrata

Nesta fase é especificada a *Gramática da LESRF* (**Figura 71**) que é uma instância da classe *Gramática da LED*. A gramática da LESRF descreve um programa da seguinte forma. O código consiste de uma declaração *begin,* indicando o início do código; a seguir, uma seção de declaração de papéis fixos que possuem propriedades, identificada pela declaração *Fixed roles*. Os papéis são declarados da seguinte forma: *Role: Nome do papel*. Após a declaração do papel, podem ser ajustadas suas propriedades. Para tanto é necessário incluir o nome da propriedade, seguido do símbolo de atribuição (:=), seguido do valor que está sendo atribuído, sendo que esse valor tem que pertencer ao domínio da propriedade, a sintaxe é *Set property. Nome da propriedade := valor*.

Em seguida podem ser declarados pacotes, com as instruções *Package: nome do pacote* para iniciar um pacote e *end package* para finaliza-lo. Entre essas duas instruções serão declarados os papéis que constituem o pacote. Para finalizar um programa é necessário simplesmente incluir a instrução *End*.

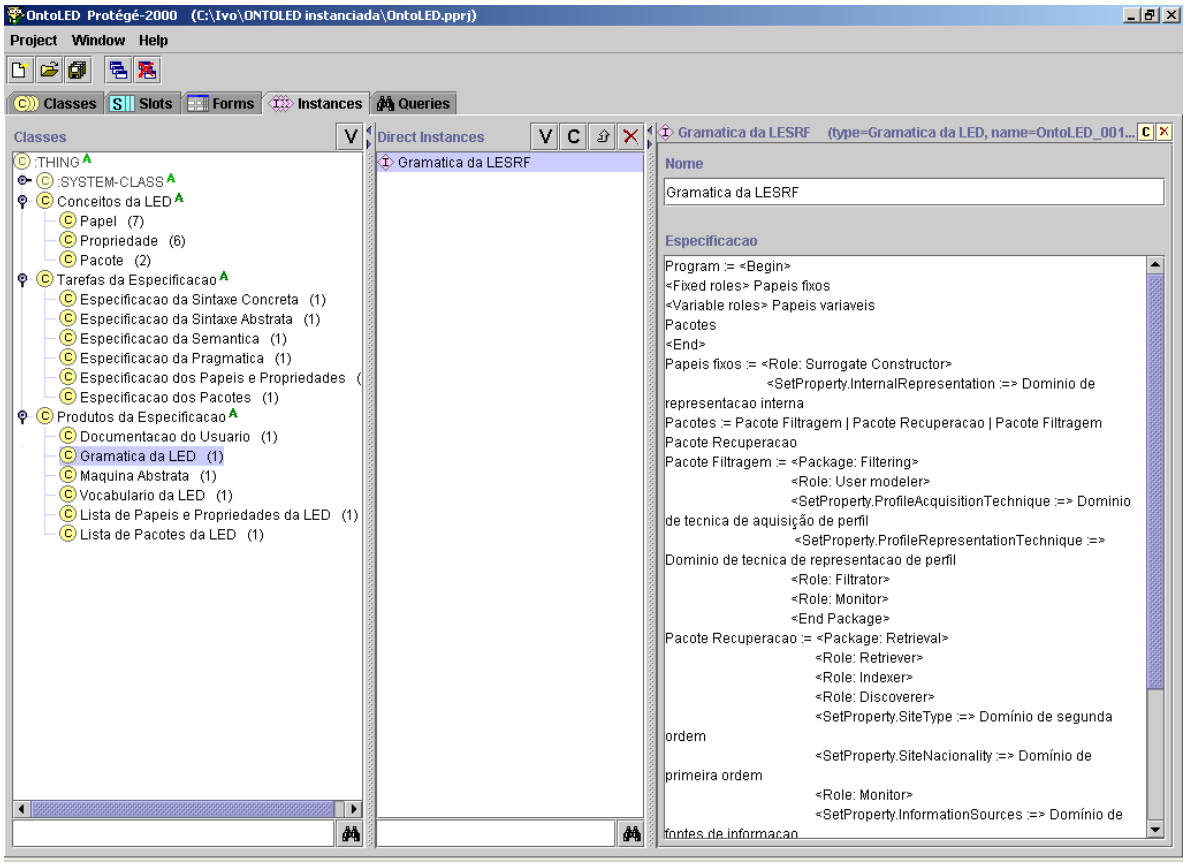


**Figura 71:** Gramática da LESRF

### 6.3.2 Especificação da Semântica

A *Especificação da Semântica* tem como produto, a *Máquina Abstrata da LED* que representa o domínio semântico que atribui significado aos termos da LED e também serve de subsídio para sua implementação. A TOD-LED define a máquina abstrata como o modelo de domínio acrescido das propriedades associadas aos papéis da LED.

Na **Figura 72**, a ONTOINFO é referenciada como o modelo de domínio utilizado, e são mostradas as propriedades, e seus respectivos papéis que completam a definição da máquina abstrata.

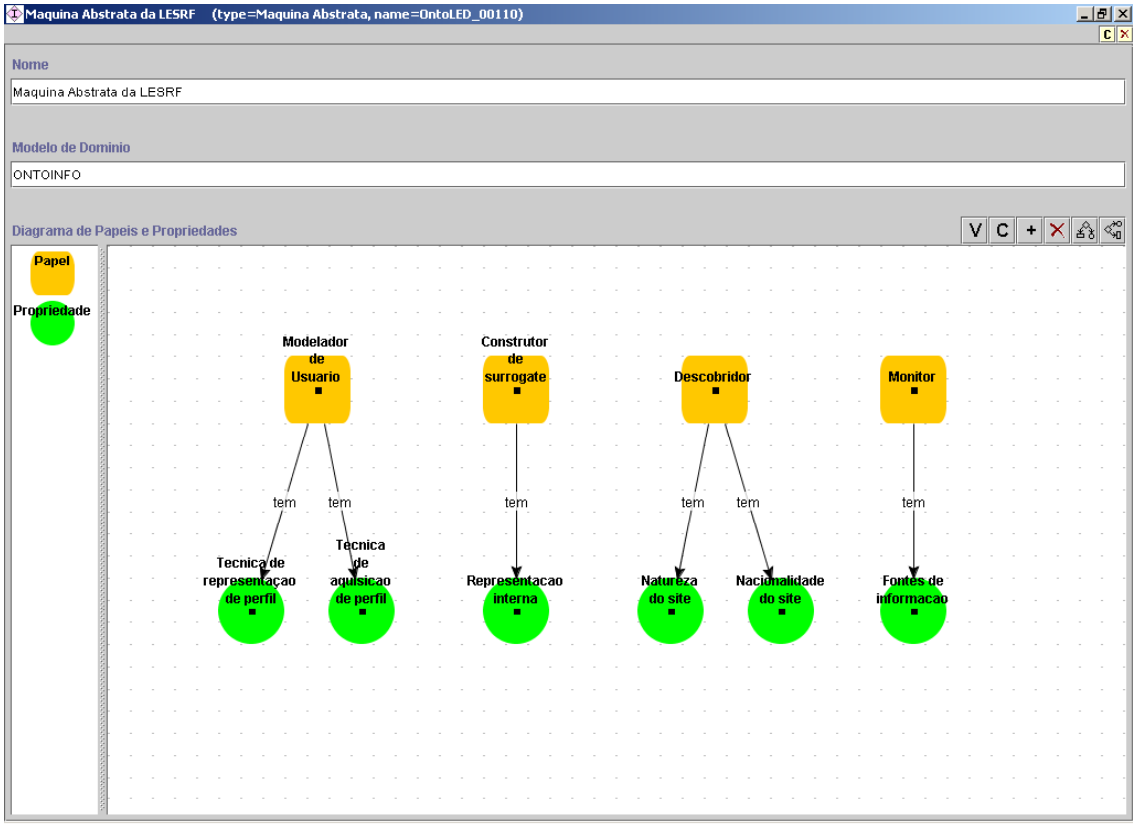


**Figura 72:** Especificação da máquina abstrata da LESRF

### 6.3.3 Especificação da Pragmática

Nesta tarefa é criada a documentação do usuário através da instanciação da classe *Documentação do usuário*. A **Figura 53** dá uma visão geral da *Documentação do usuário da LESRF*.

No slot *descrição geral da LED* tem-se: “A LESRF (Linguagem de Especificação de Sistemas de Recuperação e Filtragem de Informação) é uma LED que se destina a tornar facilitado o desenvolvimento de sistemas pertencentes à família de sistemas para o acesso a informação. Devido a algumas de suas características como alto nível de abstração, programação baseada em papéis e ser

uma linguagem puramente declarativa, a LESRF torna possível que o pré-requisito principal para utilizá-la seja tão somente o conhecimento do domínio".

No slot *descrição do domínio*, tem-se uma descrição do domínio (ver seção 6.1). No slot *descrição do vocabulário*, tem-se uma descrição dos termos que formam o vocabulário da LED, ou seja, os papéis, propriedades e pacotes. No slot *descrição da sintaxe abstrata*, tem-se uma descrição das regras de escrita da LED. No slot exemplos, tem-se a especificação de dois sistemas de acesso a informação segundo a LESRF. A **Figura 73** mostra a especificação de um sistema de acesso à informação que realiza filtragem de informação. A **Figura 74** mostra a especificação de um sistema de acesso à informação que realiza recuperação e filtragem de informação.

```
Begin
    Fixed Roles:
        Role: Surrogate constructor
          SetProperty. Internal representation := vetorial space
    Package: Filtering
        Role: User Modeler
          Set Property. Profile representation technique := ontology
          Set Property. Profile acquisition technique := implicit
        Role: Filtrator
        Role: Monitor
          Set Property. Information Sources := "www.stf.gov.br",
"www.stj.gov.br"
    End Package
End
```

**Figura 73:** Sistema de filtragem de informação segundo a LESRF

```
Begin
    Fixed Roles:
        Role: Surrogate constructor
          SetProperty. Internal representation := vetorial space
    Package: Retrieval
        Role: Retriever
        Role: Indexer
        Role: Decoverer
          SetProperty. Site type := ".gov"
          SetProperty. Site nacionality := ".br"
    End Package
    Package: Filtering
        Role: User Modeler
          Set Property. Profile representation technique := ontology
          Set Property. Profile acquisition technique := implicit
        Role: Filtrator
        Role: Monitor
          Set Property. Information sources := "www.stf.gov.br",
"www.stj.gov.br"
    End Package
End *
```

**Figura 74:** Sistema de recuperação e filtragem segundo a LESRF

## 6.4 Considerações finais

Neste capítulo foi apresentado um estudo de caso para avaliar a TOD-LED. O estudo de caso consistiu inicialmente no desenvolvimento da ONTOINFO, um modelo de domínio para a área do acesso à informação, com o uso da técnica GRAMO. Em seguida foi aplicada a TOD-LED no desenvolvimento da LESRF, uma LED para a especificação de sistemas para a recuperação e filtragem de informação.

# 7 CONCLUSÃO

Neste trabalho foi proposta a TOD-LED, uma técnica baseada em ontologias para o desenvolvimento de LED's na Engenharia de Domínio Multiagente. A técnica guia a especificação de LED's a partir de modelos de domínio resultantes da aplicação da técnica GRAMO [Faria, 2004] estendida.

A TOD-LED utiliza a ONTOLED, uma ontologia que representa o conhecimento acerca do desenvolvimento de LED's. A especificação de uma LED é representada por uma instância da ONTOLED.

A TOD-LED especifica LED's puramente declarativas, segundo o paradigma de programação de papéis. Sendo assim, programas escritos com LED's desenvolvidas segundo essa técnica consistem basicamente da declaração de papéis e do ajuste de suas eventuais propriedades.

A Engenharia de Domínio Multiagente é uma área de estudos nova e que pode ser ainda bastante desenvolvida. A TOD-LED tenta contribuir com a Engenharia de Domínio Multiagente fornecendo diretrizes para o desenvolvimento de LED’s. As LED’s são produtos da Engenharia de Domínio que permitem a reutilização gerativa, ou seja, a reutilização automática dos artefatos da Engenharia de Domínio.

## 7.1 Resultados e Contribuições da Pesquisa

As principais contribuições desta pesquisa foram:

- Análise do estado da arte de algumas das mais bem documentadas metodologias para o desenvolvimento de LED’s.
- Uma análise comparativa das metodologias estudadas, ressaltando seus pontos fortes e deficiências.

- Extensão da técnica GRAMO [Faria, 2004] e da ontologia genérica ONTODUM [Faria, 2004], para que pudessem levar em consideração a classificação dos conceitos do domínio em fixos e variáveis. A classificação dos conceitos do domínio é fundamental para o desenvolvimento de famílias de sistemas e LED's.
- Desenvolvimento da ONTOLED, uma ontologia que representa o conhecimento acerca do desenvolvimento de LED's.
- Desenvolvimento da TOD-LED, uma técnica para o desenvolvimento de LED's na Engenharia de Domínio Multiagente.
- Desenvolvimento da ONTOINFO, um modelo de domínio para a área do acesso a informação segundo o paradigma computacional de agentes.
- Aplicação da TOD-LED na especificação da LESRF, uma LED para o domínio do acesso a informação dinâmica e não estruturada.

## 7.2 Trabalhos Futuros

Vários trabalhos podem ser abordados a partir dos resultados obtidos neste:

- Extensão da técnica proposta para incluir as fases de projeto e implementação de LED's.
- Aplicação da técnica TOD-LED para a especificação de LED's em outros domínios para sua melhor avaliação.
- Especificação de uma gramática genérica, para as LED's desenvolvidas com a TOD-LED.

- Projeto de um gerador de aplicações genérico para as LED's desenvolvidas com a TOD-LED.
- Projeto da linguagem específica de domínio LESRF para o acesso à informação através da integração do framework multiagente ONTODD-INFO [Ferreira, 2004] com a especificação da LESRF.
- Protótipo de uma aplicação para a recuperação e filtragem de informação na área jurídica utilizando a linguagem LESRF.

# REFERENCIAS

Arango, G. (1988) “Domain Engineering for Software Reuse”. Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1988.

Arnold, B. R.T., Deursen, A., Res, M. (1995) “An algebraic specification of a language describing financial products”, In IEEE Workshop on Formal Methods Application in software engineering, pages 6-13.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999) “Modern Information Retrieval”, New York: ACM Press Series/Addison Wesley.

Cleaveland, J., (1988) “Building application generators”, IEEE Software July 1988.

Cohen, S., Kang, K., Hess, J., Peterson, S. (1990) “Feature Oriented Domain Analysis (FODA) Feasibility Study”, Techinical Report CMU/SEI-90-TR-21. Software Engineering Institute, Cornegie Mellon University, Pittsburgh, PA, November, 1990.

Compose project “Domain-Specific Languages – An Overview” Disponível em: http://compose.labri.fr/overview/overview.en.php3#dsl. Acessado em: 26 de fevereiro de 2003.

Consel, C., Danvy, O. (1993) “Tutorial Notes on Partial Evaluation” In ACM Symposium on Principles of Programming Languages, pages 493-501, 1993.

Consel, C., Marlet, R. (1998) “Architecturing software using a methodology for language development”, In C. Palamidessi, H. Glaser, and K. Meinke, editors, Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming, volume 1490 of Lecture Notes in Computer Science, Pisa, Italy, pages 170--194, September 1998.

Deursen, A., Klint, P., Visser, J. (2000) “Domain-specific languages: An annotated bibliografy”, ACM SIGPLAN Notices, 35(6): 26-36, June 2000.

Elliott, C. (1997) “Modelling interactive 3D and multimedia animation with an embedded language”, In Proceedings of the 1st USENIX Conference on Domain-Specific Languages, Santa Barbara, California.

Faria, C. (2004) “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.

Fernández, M., Gomez-Perez, A., Pazos, A. (1999) “Building a Chemical Ontology using METHONTOLOGY and the Ontology Design Enviroment”, IEEE Expert: Special Issue on Uses of Ontologies. January/February 1999. PP: 37:46

Ferreira, S. (2004)“Uma Ferramenta e Técnica para o Projeto Global e Detalhado de Sistemas Multiagente”. Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, 2004.

Foreman, J. (1996) “Product Line Based Software Development - Significant Results Future Chalenges”, Software Technology Conference, Salt Lake City, UT, April 23, 1996.

Fridman, N. N., Mcguinness, D. L. (2001) “Ontology Development 101: A Guide to Creating Your First Ontology”, Knowledge Systems Laboratory, March, 2001.

Girardi, R. (1998) “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).

Girardi, R. (2003) “Engenharia de Domínio Multiagente”, Anais da Terceira Jornada Iberoamericana de Engenharia de Software e Engenharia do Conhecimento (JIISIC 2003), Seção de Oportunidades de Cooperação. Valdivia, Chile. 26 a 28 de novembro de 2003.

Girardi, R., Faria, C. (2003) “A Generic Ontology for the Specification of Domain Models”, Proceedings of 1st International Workshop on Component Engineering Methodology (WCEM 2003) at Second International Conference on Generative Programming and Component Engineering, pp. 41-50, Ed. Sven Overhage and Klaus Turowski. Efurt, Germany, September, 2003.

Gruber, T. R. (1995) “Toward Principles for the Design of Ontologies used for Knowledge Sharing”, International Journal of Human-Computer Studies. Nº 43, pp. 907-928, 1995

Guarino, N. (1998) “Formal Ontology and Information Systems”, Proceedings of FOIS'98, Trento, Italy. Amsterdan, IOS Press, pp. 3-15, 6-8 June, 1998.

Guizzardi, G., Ferreira Pires L., van Sinderen, M.J. (2002) "On the role of domain ontologies in the design of domain-specific visual modeling languages." In 2nd Workshop on Domain-Specific Visual Languages, 17th ACM Conference on Object-Orientec Programming, Systems, Languages and Applications (OOPSLA 2002). Seattle, Washington, USA, 2002.

Harsu, M. (2002) “A Survey on Domain Engineering”, Report 31, Institute of Software Systems, Tampere University of Technology, December 2002, 26 pp.

Jones, N. D., Gomard C. and Sestoft, P. (1993) “Partial evaluation and automatic program generation”, Prentice Hall international series in computer science, 1993.

Kamin, S., Hyatt, D. (1997) “A special-purpose language for picture-drawing”, In proceedings of the 1st USENIX Conference on Domain-Specific Languages, Santa Barbara, California.

Kruegar, C. (1992) “Software Reuse”, ACM Computing Surveys, Vol. 24, No. 2, June 1992.

Krut, R. (1993) “Integrating 001 Tool Support into the Feature Oriented Domain Analysis Methodology”, Technical Report CMU/SEI-93-TR-11. Software Engineering Institute, Carnegie Mellon University, 1993.
Magnan, M. A. S., Murta, L. G. P., Souza, J. M., Werner, C. M. L. (2001)“Modelos de domínio e Ontologias: uma comparação através de um estudo de caso prático em hidrologia”, IV International Symposium on Knowledge Management/Document Management (ISKM/DM'2001). Curitiba, Agosto, 2001.

Mauw, S., Wiersma, W.T. and Willemse T.A.C. (2002) “Language-driven system design”, In HICSS35, Proceedings of the Hawaii International Conference on System Sciences, minitrack on Domain-Specific Languages for Software Engineering, Hawaii, January 2002.

Neighbors, J. M. (1984) “The Draco approach to constructing software from reusable components”, IEEE Transactions on software Engineering, SE-10(5):564-74, September 1984.

Parnas, D. L. (1976) “On the design and development of program families”. IEEE Transactions on Software Engineering, SE-2(1):1--9, March 1976.

Plataforma Lattes. Disponível em: http://lattes.cnpq.br.8888/plataformalattes/index.jsp. Acessado em: 03 de maio de 2003.

Protégé Project. Disponível em: http://protege.stanford.edu. Acessado em: 14 de setembro de 2002.

Salton, G. Mcgill, M. (1983) “An Introduction to Modern Information Retrieval”, New York: McGraw-Hill.

Schmidt, D. A. (1986) “Denotational Semantics: A Methodology for Language Development”, Allyn and Bacon, Inc., Newton, MA.

Serra, I. C., Girardi, R. (2003) “Uma Abordagem Gerativa para a Engenharia de Domínio Multiagente”, Anais do III Encontro de Informática do Tocantins (ENCOINFO 2003), Palmas, Tocantins, Brasil, pp. 261-270, Ed. ULBRA. 29 a 30 de outubro de 2003.

Simos, M. (1995) “Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle”, SIGSOFT Software Engineering Notes, Special Issue on the 1995 Symposium on Software Reusability, Aug 1995.

Widen, T. (1995) "Formal Language Design in the Context of Domain Engineering" Msc thesis, Oregon Graduate Institute of Science & Technology

Widen, T., Hook, J. (1998) “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, June 1998.

Thibault, S. (1998) “Domain-Specific Languages: Conception, Implementation and Application”, PhD thesis, IRISA/University of Rennes.

Thibault, S., Marlet, R., Consel, C. (1997) “A domain-specific language for video device drivers: from design to implementation”, In Conference on Domain Specific Languages, p. 11-26, Santa Barbara, Usenix.

W3 Schools. Disponível em: http://www.w3schools.com/. Acessado em: 12 de julho de 2003.

Werner, C., Braga, R. (1998) “Desenvolvimento Baseado em Componentes”, XIV Simpósio Brasileiro de Engenharia de Software, Mini-curso, João Pessoa.

Wikipedia. Disponível em: http://en.wikipedia.org/wiki/Assembly_line. Acessado em: 26 de janeiro de 2004.