Capítulo 1

Introdução

Atualmente presenciamos um número crescente de aplicações desenvolvidas para a *Web*. Essas aplicações têm um grande potencial, visto que podem ser acessadas e utilizadas por várias pessoas nos mais diferentes lugares ao mesmo tempo.

Inicialmente, a construção das aplicações voltadas para a Web se dava de forma *ad-hoc*. Entretanto, com o crescimento do número e da complexidade das *WebApps* (aplicações para a *Web*), fez-se necessário aplicar à construção desse tipo de aplicação os métodos disciplinados da Engenharia de Software, adaptados a esta nova plataforma de implementação. Nasceria, assim, a Engenharia *Web* (SOUZA et al., 2005).

Para tratar deste novo panorama, muitos métodos, *frameworks* e linguagens de modelagem para desenvolvimento de aplicações *Web* têm sido propostos. CONTE et al. (2005) fazem referência a várias dessas propostas, dentre as quais podemos citar WebML (CERI et al., 2000), WAE (CONALLEN, 2002), OOWS (FONS et al., 2003) e UWE (KOCH et al, 2000).

Em paralelo, tecnologias para codificação de *WebApps* também se desenvolveram bastante. O uso de *frameworks* ou arquiteturas baseadas em *containers* para prover uma sólida infraestrutura para desenvolvimento e implantação de aplicações para a *Web* é estado-da-prática atualmente. Essa infra-estrutura geralmente inclui uma arquitetura MVC (*Model-View-Controller*, Modelo-Visão-Controlador) (GAMMA et al., 1994), interceptadores AOP (*Aspect Oriented Programming*, Programação Orientada a Aspectos) (RESENDE et al., 2005), um mecanismo de injeção de dependências (FOWLER, 2006), mapeamento objeto/relacional automático para persistência de dados (BAUER et al., 2005) e outras facilidades. O uso desses *frameworks* reduz consideravelmente o tempo de desenvolvimento de um projeto por reutilizar código já desenvolvido, testado e documentado por terceiros.(SOUZA, 2007)

O Laboratório de Engenharia de Software da UFES desenvolveu um método para guiar a construção de *WebApps* cujo nome é FrameWeb. Esta metodologia de desenvolvimento sugere a utilização de vários *frameworks* para uma rápida e eficiente construção deste tipo de aplicação. Além disso ela nos fornece recomendações para aumentar a agilidade nas principais fases de desenvolvimento (requisitos, análise, projeto e outras), uma arquitetura básica para desenvolver esse tipo de aplicação e um perfil UML para um conjunto de modelos de projeto que trazem conceitos

usados por algumas categorias de *frameworks*.

Estima-se que hoje existam bilhões de documentos na *World Wide Web,* mas a grande maioria dessas informações não possui semântica alguma. Os dados são apresentados ao internauta de maneira que este tenha que extrair as informações que lhe interessa. O problema é que, ficando sujeito a tanta informação, torna-se complicado encontrar aquelas que lhe são úteis numa determinada situação.

Neste contexto, surge o conceito da *Web* Semântica (BERNERS-LEE, 1990), uma idéia proposta por Tim Berners-Lee que tem como grande objetivo categorizar a informação de maneira padronizada, facilitando seu acesso. A idéia é representar o conteúdo *Web* de forma a ser mais facilmente processável por máquina para que assim seja possível utilizar técnicas inteligentes para tirar vantagem desta representação.

Para fazer tal representação utilizamos o conceito de ontologia. Segundo Broekstra et al. (2001), uma ontologia é um modelo abstrato e formal que descreve os conceitos relevantes de algum fenômeno e que deve ser consensual, ou seja, difundido e aceito por uma comunidade. Para representar tais realidades ou fenômenos, uma ontologia descreve classes pertencentes ao domínio de interesse, instâncias dessas classes de domínio, relacionamentos entre classes ou entre instâncias, propriedades de classes, instâncias ou relacionamentos, e restrições e regras envolvendo esses elementos (DACONTA et al, 2003). A partir desta especificação formal, é possível fazer com que as máquinas sejam capazes de processar e fazer inferências sobre determinado domínio.

A fim de tratar a questão da Web Semântica, o método FrameWeb possui uma extensão chamada S-FrameWeb que sugere a extensão do *framework* Controlador Frontal de modo a habilitá-lo a perceber se uma aplicação está interagindo com um humano ou com uma máquina, para que, assim, possa ser apresentado o resultado corretamente.

Caso a aplicação perceba que está interagindo com máquinas, ela deve ser capaz de gerar um resultado bem estruturado, processável por máquinas. Para isso, o sistema deve ser capaz de entender o seu próprio domínio, e isto é conseguido com a exploração da ontologia do nosso domínio que foi representada na linguagem OWL. Utilizamos então um *framework* Java chamado Jena usado para construção de aplicações com Semântica. Seu papel é ser capaz de retirar as informações necessárias da nossa ontologia, por exemplo, os relacionamentos entre conceitos do nosso domínio, as propriedades das classes, etc ...

Utilizando estas informações e a característica de reflexão de Java, fomos capazes de extrair as informações que desejávamos. Assim, o método nos guiou na construção de um sistema capaz de gerar informações bem organizadas e estruturadas que foram geradas dinamicamente.

1.1. Objetivos

É objetivo deste trabalho fazer a implementação das idéias sugeridas pelo método S-FrameWeb. Podemos subdividir este objetivo em dois principais objetivos, a saber:

- Criar uma extensão para o framework Struts[2] que permitirá que as aplicações percebam a necessidade de gerar informações com semântica;
- Utilizar a API Jena para recuperar informações da ontologia gerada.

Além disso, vamos utilizar o método S-FrameWeb para guiar a construção de aplicações Web com semântica, tendo como objetivos:

- Criar os modelos como foi proposto pelo método (SOUZA, 2007) para o Portal do LabES que foi desenvolvido em (PERUCH, 2007);
- Representar a ontologia com a linguagem OWL para o Portal do LabES.

1.2.Metodologia

Este trabalho foi desenvolvido em quatro fases: revisão bibliográfica, pesquisa, extensão do framework Controlador Frontal, desenvolvimento de uma pequena aplicação e inclusão de um pequeno tratamento semântico em algumas funcionalidades no Portal do LabES.

O trabalho foi desenvolvido durante o projeto de graduação. Foi feita uma revisão bibliográfica sobre Engenharia *Web* e *Web* Semântica seguida de uma pesquisa sobre *frameworks* e linguagens de representação de ontologias. Além disso, estudamos também as metodologias FrameWeb (SOUZA, 2005) e S-FrameWeb (SOUZA et. al., 2007).

Construímos, então, uma pequena aplicação Web que é detalhada no capítulo 3 na qual aplicamos o método S-FrameWeb. Nesta etapa, criamos uma extensão de um framework Controlador Frontal e utilizamos um framework para nos dar apoio à inclusão de semântica nesta aplicação. Como fruto dessa parte do trabalho publicamos um artigo no Workshop Internacional em Modelagem de Sistemas de Informação para a Web (WISM 2007). Na parte final desse trabalho incluímos a idéia da *Web* Semântica no Portal do LabES que é mostrado no capítulo 4.

1.3.Organização da Monografia

Este documento contém, além deste capítulo, mais quatro outros.

No capítulo 2 é feito um apanhado geral dos temas relevantes a esta monografia: Engenharia

Web, *Frameworks*, FrameWeb, S-FrameWeb e o framework Jena.

No capítulo 3 temos a parte mais importante deste trabalho, que é a extensão do framework Controlador Frontal e a utilização do framework Jena que possibilita o tratamento semântico nas aplicações, também mostramos um pequeno sistema que foi construído utilizando o método S-FrameWeb. No capítulo 4, nós mostramos a inclusão de semântica no Portal do LabES.

No capítulo 5 encontram-se as conclusões e perspectivas futuras.

Capítulo 2

A Evolução da *Web*

Há algum tempo estamos presenciando uma significativa mudança de paradigma no desenvolvimento de sistemas. Aplicações *desktop*, sistemas com interfaces gráficas ricas baseadas em janelas, estão gradativamente perdendo espaço para as aplicações *Web*, as comumente chamadas *WebApps*, que executam em um servidor na *World Wide Web* (WWW) ou numa Intranet e podem ser acessadas por qualquer máquina com um navegador Internet (*browser*). O mundo tecnológico a cada dia percebe mais e mais os horizontes que a rede mundial de computadores lhe oferece, e conseqüentemente, os usuários percebem as inúmeras aplicações que a ciência da computação pode realizar nesta área.

As *WebApps* estão ocupando espaços e realizando atividades que há pouco tempo eram inimagináveis. Desde pequenas lojas virtuais, que nos oferecem a comodidade de ter produtos para ver, escolher e comprar sem ao menos sair de casa, até grandes bancos, que nos oferecem rapidez e segurança para movimentar nossa conta bancária dando apenas alguns cliques utilizando o nosso *navegador* favorito.

Em um primeiro momento, a construção de aplicações *Web* se dava de forma *ad-hoc*, sem processos de software e instrumentos formais para guiar equipes de desenvolvimento. Hoje, com o crescimento do número e da complexidade das *WebApps*, fez-se necessário aplicar à construção desse tipo de aplicação os métodos disciplinados da Engenharia de Software, adaptados a essa nova plataforma de implementação. Nasceria, assim, a Engenharia *Web* (SOUZA et al., 2005).

Com isso, o número de *WebApps* e *websites* em geral cresceu vertiginosamente e, com a popularização da Internet, o acesso às informações foi facilitado. Estima-se que hoje existam bilhões de documentos na *World Wide Web* sobre os mais variados assuntos, como ciência, religião, novela, linguagens de programação, receitas de bolo etc. Apesar desta grande quantidade de informações ser um fator positivo, visto que torna os internautas capazes de conhecer praticamente sobre tudo que desejarem navegando pela Internet, ela se torna extremamente difícil de ser organizada, acessada e até mantida.

Para tentar contornar este problema, que se agrava constantemente, há um grande esforço, inicialmente na comunidade acadêmica, de tornar a atual *Web* sintática, mais adequada ao ser humano, em uma *Web* Semântica que seja compreensível por agentes inteligentes. Desta forma,

poderemos construir programas que percorrerão a Internet executando, automaticamente, as mais variadas tarefas que, se feitas manualmente, demandariam muito tempo e esforço.

Este capítulo mostra a adaptação da engenharia de software para englobar e satisfazer necessidades das *WebApps* e um método – FrameWeb – para a construção dessas aplicações baseado no uso de *frameworks* que agilizam e facilitam o trabalho. Na seqüência, é apresentada a *Web* Semântica, que é a proposta de evolução da *Web* Sintática, e um *framework* Java – Jena – que nos fornece uma API para tratamento de semântica nas aplicações.

2.1. Engenharia *Web*

A Engenharia *Web* (*Web Engineering* ou *WebE*) está se tornando cada vez mais necessária devido ao aumento do número de aplicações e o aumento da complexidade de aplicações feitas para a *Web*. A complexidade desses sistemas exige um maior planejamento, uma melhor organização e, para que haja a continuidade no desenvolvimento desses ambientes com qualidade, é necessário que se utilize a Engenharia *Web*, que utiliza vários conceitos e princípios da Engenharia de Software, porém com algumas características que devem ser observadas devido ao meio em que esses sistemas irão operar (KAPPEL et al., 2006).

Os engenheiros *Web* devem ter atenção com algumas características dos sistemas voltados para a *Web*, por exemplo: concorrência, carga imprevisível, desempenho e várias outras características. Uma característica em especial interfere no desenvolvimento desse tipo de sistema: o imediatismo. Esta característica, que já é uma exigência também para as aplicações *desktop*, é ainda mais forte para as *WebApps*, de modo que os modelos de processo *WebE* são de desenvolvimento ágil com ciclos rápidos e quase sempre incrementais (POWELL,1998).

Como já mencionado anteriormente, a *Web* proporciona várias categorias de aplicação: informacional (conteúdo somente leitura), *downloads*, adaptável (adapta o conteúdo a necessidades específicas), interação, entrada do usuário, orientada a transação, orientada a serviços, portal, de acesso a banco de dados, armazém de dados etc. (DART, 1999).

Apesar do desenvolvimento dessas aplicações ter o fator tempo contra ele, a *WebE* adota uma postura bastante cautelosa, obedecendo a algumas etapas no desenvolvimento de *WebApps*. Tudo começa com a formulação que tem por objetivo determinar quais necessidades a aplicação deverá suprir, ou seja, quais funcionalidades deseja-se que a aplicação tenha. Para conseguir isto, são sugeridas algumas questões de formulação (POWELL, 1998):

- Qual é a principal motivação (necessidade de negócio) da *WebApp*?
- Quais são os objetivos que a *WebApp* deve atender?
- Quem vai usar a *WebApp*?

Assim, com o problema bem definido, temos os requisitos que levam ao desenvolvimento de um modelo de análise. Esse modelo pode ser subdividido em quatro tópicos, de acordo com o que está sendo analisado: análise de conteúdo identifica as classes de conteúdo e suas colaborações; análise de interação descreve elementos básicos da interação com o usuário, navegação e os comportamentos do sistema que ocorrem como conseqüência; análise de função define as funções da *WebApp* realizadas para o usuário e a seqüência de processamento que ocorre como conseqüência; e análise de configuração que identifica o(s) ambiente(s) operacional(is) no(s) qual(is) a *WebApp* residirá (PRESSMAN, 2005).

Com o propósito de determinar as estruturas navegacionais adequadas sobre os vínculos (*links)* e melhorar o entendimento dos desenvolvedores de sistema sobre o domínio da aplicação, é feita a análise de relacionamento-navegação, que deve identificar relacionamentos entre o conteúdo e elementos funcionais definidos no modelo de análise e estabelecer requisitos para definir vínculos (*links*) de navegação adequadas ao longo do sistema. Ao final do modelo de análise devem ter sido produzidos casos de uso para cada categoria de usuários, que podem ser produzidos de maneira informal para agilizar o processo, diferentemente dos casos de uso feitos para as aplicações cujo cronograma oferece um bom tempo para análise.

Terminado o modelo de análise da *WebApp*, a *WebE* propõe que seja elaborado o modelo de projeto. Apesar do desenvolvimento dessas aplicações ser ágil, a não produção desses modelos certamente resultará, proporcionalmente ao tamanho do sistema, em um sistema fraco, inseguro, ineficiente e com baixa usabilidade. Portanto, é muito importante que seja planejado e desenvolvido o modelo de projeto que, guiados pelo modelo de análise, construirá o projeto de conteúdo, o projeto de estética, o projeto arquitetônico, o projeto de interface, o projeto de navegação e o projeto de componentes (PRESSMAN, 2005).

Com um bom modelo de projeto, a *WebApp* deve ser capaz de apresentar as seguintes características de qualidade: funcionalidade, confiabilidade, usabilidade, eficiência, manutenabilidade, portabilidade, de modo que o sistema seja aceito e utilizado pelos usuários (KAPPEL et al., 2006). Devemos observar que um sistema com essas características abrange todo e qualquer tipo de usuário que a *WebApp* venha porventura ter e a idéia é que a maior parte desse tipo de aplicação estará acessível a pessoas com diferentes capacidade de lidar com o computador. Por exemplo, um sistema de banco *online* deve ser extremamente seguro, por se tratar de um canal de

movimentações financeiras, e deve ter uma interface simples e amigável, visto que todo e qualquer cliente do banco poderá ter acesso e deverá conseguir navegar na aplicação de maneira a realizar a operação que desejar.

Depois de todo o planejamento e modelagem da aplicação se inicia a fase de implementação da solução, que deve ocorrer da forma mais rápida possível. Um método que utiliza *frameworks* para facilitar e tornar ágil o desenvolvimento da aplicação será discutido na próxima seção. Os Sistemas de Informação baseado na Web possuem uma infra-estrutura arquitetônica muito similar. Conseqüentemente, pouco tempo depois que os primeiros sistemas começaram a ser construídos, foram desenvolvidos vários *frameworks* que generalizavam essa infra-estrutura e poderiam ser utilizados para seu desenvolvimento. Neste contexto, um *framework* é visto como um artefato de código que provê componentes prontos que podem ser reutilizados mediante configuração, composição ou herança. Quando combinados, esses *frameworks* permitem que sistemas *Web* de grande porte sejam construídos com arquiteturas de múltiplas camadas sem muito esforço de codificação(SOUZA, 2007).

Visto que, devido aos pequenos prazos para entrega do projeto, a fase de teste acaba sempre prejudicada. É uma boa prática da *WebE* se fazer testes ao longo do desenvolvimento da aplicação. Como não é possível fazer testes no sentido clássico das fases de análise e projeto, sugere-se que a equipe de engenharia *Web* conduza revisões técnicas formais, bem como testes executáveis (PRESSMAN, 2005). Assim como nas aplicações convencionais, o objetivo é descobrir e corrigir os erros antes de liberar seu uso.

2.2. Frameworks

Atualmente, temos um rápido avanço nas tecnologias para codificação de *WebApps.* Assim, podemos observar o desenvolvimento de *frameworks* que nos oferecem uma sólida infraestrutura para a construção de aplicações *Web*.

Há vários *frameworks* disponíveis, com as mais diversas finalidades. Entretanto, é possível separá-los em algumas categorias organizadas de acordo com o propósito de cada um. A Tabela 1 lista quatro dessas categorias: Controlador Frontal, Decorador, Mapeamento Objeto/Relacional e *Frameworks* de Injeção de Dependências. Outros tipos de frameworks incluem: Frameworks para Programação Orientada a Aspectos, Frameworks de Autenticação e Autorização, Máquinas de Busca, etc (SOUZA, 2007).

Tabela 1: Frameworks utilizados comumente como infraestrutura para a construção de WebApps

Framework	*Proposta*
Controlador Frontal	*Também conhecido como framework MVC, define uma arquitetura que separa a funcionalidade da WebApp de sua apresentação, baseado no padrão Modelo-Visão-Controlador.*
Decorador	*Baseado no padrão de projeto Decorador, automatiza a tarefa de fazer cada página Web do site ter o mesmo layout (cabeçalho, rodapé, barra de navegação, cores, imagens, etc).*
Mapeamento Objeto/Relacional	*Provê uma persistência automática e transparentes dos objetos para as tabelas do sistemas gerenciador de banco de dados relacionais usando meta-dados que descrevem o mapeamento entre os mundos.*
Injeção de Dependências	*Permite ao desenvolvedor programar para interfaces e especifica as dependências concretas em um arquivo de configuração. A idéia é que as classes que dependem de serviços de diferentes camadas declarem uma associação com a interface ao invés das implementações concretas.*

O *framework* mais importante no contexto deste trabalho é o Controlador Frontal, pois no Struts[2], que é um exemplo deste tipo de *framework*, foi inserida uma nova extensão a fim de possibilitar a utilização da *Web* Semântica. Um framework Controlador Frontal implementa o padrão MVC que é a abreviatura de Modelo-Visão-Controlador (*Model-View-Controller*) (GAMMA et al., 1994), uma arquitetura de software desenvolvida pelo Centro de Pesquisas da Xerox de Palo Alto (Xerox PARC) para a linguagem Smalltalk em 1979 (REENSKAUG, 1979). Desde então, a arquitetura se desenvolveu e ganhou aceitação em diversas áreas da Engenharia de Software e hoje é possivelmente a arquitetura mais utilizada para construção de aplicações *Web*. O

esquema da Figura 1 mostra como funciona esse padrão arquitetural.

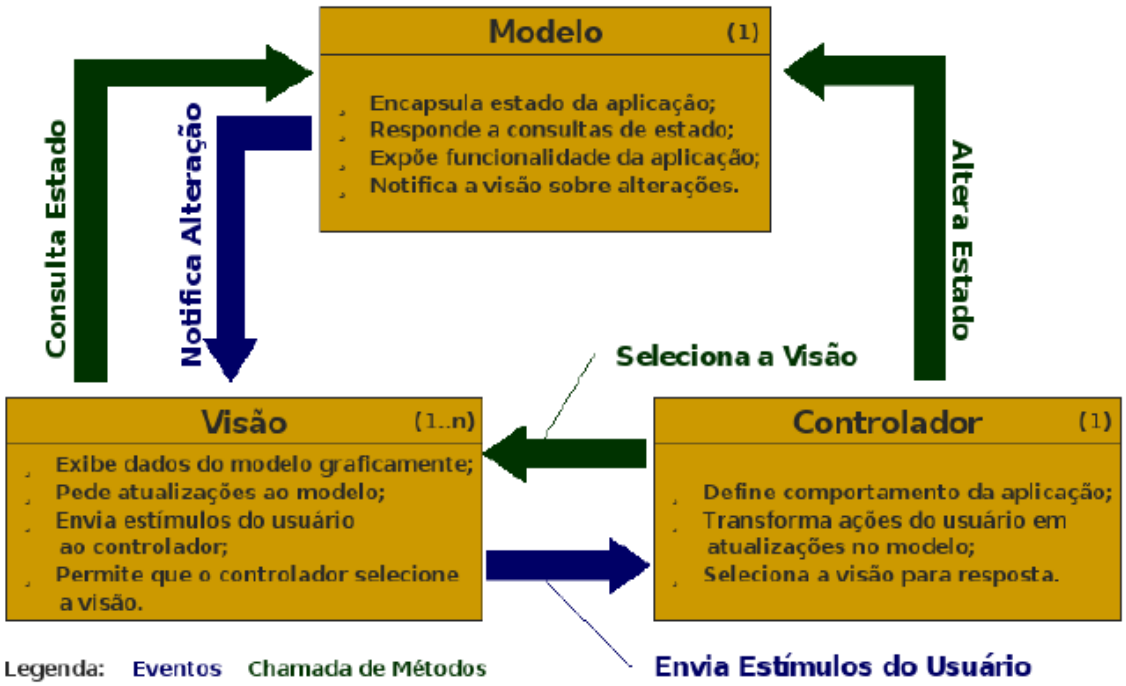

Figura 1: Funcionamento do padrão Modelo-Visão-Controlador

Elementos da visão representam informações de modelo e as exibem ao usuário, que pode enviar, por meio deles, estímulos ao sistema, que são tratados pelo controlador. Este último altera o estado dos elementos do modelo e pode, se apropriado, selecionar outros elementos de visão a serem exibidos ao usuário. O modelo, por sua vez, notifica alterações em sua estrutura à visão para que esta consulte novamente os dados e se atualize automaticamente. (SOUZA, 2007)

Essa arquitetura veio ao encontro das necessidades dos aplicativos *Web*. No entanto, se formos primar pelo purismo, veremos que a arquitetura MVC não se encaixa perfeitamente nessa plataforma, visto que a camada de modelo, situada no servidor *Web*, não pode notificar a camada de visão sobre alterações, já que esta encontra-se no navegador do lado do cliente e a comunicação é sempre iniciada no sentido cliente – servidor. Portanto, apesar de MVC ser um nome muito difundido, o nome correto para esse padrão arquitetônico, quando aplicado à *Web*, seria "Controlador Frontal" (*Front Controller*) (ALUR et al., 2003, p. 166). Neste trabalho, usaremos ambos os nomes indistintamente. O padrão Controlador Frontal funciona como mostra a Figura 2.

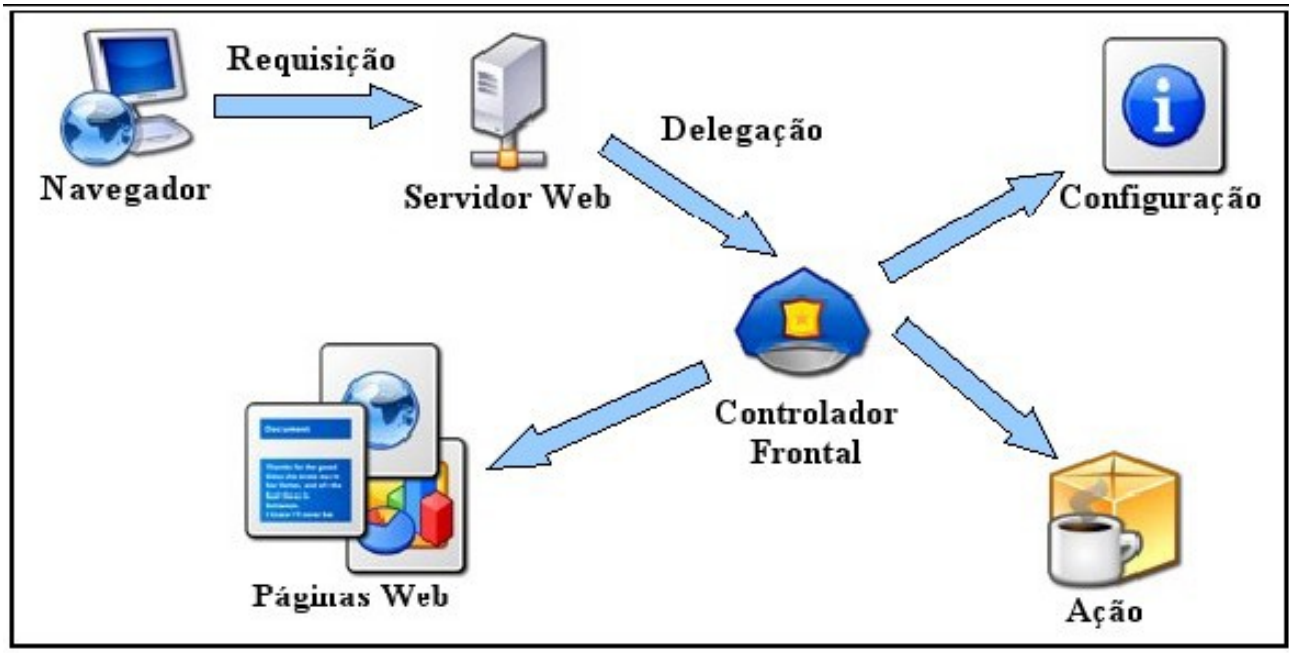

Figura 2: Funcionamento do padrão arquitetônico "Controlador Frontal"

O navegador do lado do cliente efetua uma requisição HTTP ao servidor *Web*, que pode ser tanto um pedido de leitura de uma determinada página quanto o envio de dados para processamento. O servidor *Web*, então, delega essa requisição para o controlador frontal, que passa a gerenciar todo o processo. Com base em uma configuração pré-determinada, o controlador cria uma instância de uma classe de ação específica e pede a esse objeto que execute seu serviço, similar ao que acontece com padrão de projeto "Comando" (*Command*) (GAMMA et al., 1994). Esse objeto deve retornar uma chave representando um dos resultados possíveis da execução e o controlador, novamente se referindo à sua configuração, escolhe um tipo de resultado, que pode ser a renderização de uma página, redirecionamento a outra página, montagem e exibição de um relatório, dentre várias possibilidades. *Frameworks* MVC implementam exatamente essa arquitetura, fornecendo o controlador, uma superclasse base ou interface para as ações, tipos de resultado e uma sintaxe bem definida para o arquivo de configuração (geralmente escrito em XML), deixando para o desenvolvedor a tarefa de configurar o *framework*, escrever as classes de ação e as páginas *Web*. Via de regra, o *framework* também fornece uma série de facilidades, como conversão automática de tipos, validação de dados de entrada, internacionalização de páginas *Web*, etc. (SOUZA, 2007)

Nós escolhemos o Struts[2], cujo fluxo de requisição é exibido na Figura 3. Podemos observar que uma requisição inicialmente passa através de uma cadeia padrão de filtros. A cadeia inclui o filtro `Limpador do Contexto de Ação` (opcional), que é útil na integração de tecnologias. Em seguida, o `Filtro Despachante` requisitado é chamado, que consultará o `Mapeador de Ação` para determinar se a requisição deve invocar uma ação. Se o `Mapeador de Ação` determinar que

uma Ação deve ser invocada, o Filtro Despachante delega o controle ao `Proxy de Ação`, que consultará os arquivos de configuração (a partir do arquivo `struts.xml`). Depois, o Proxy de Ação cria uma Invocação da Ação, que é responsável pela implementação do comando padrão. Isto inclui primeiramente, invocar os Interceptadores e depois, invocar a própria ação. Uma vez que a ação retornou, a `Invocação da Ação` é responsável por procurar o resultado correto associado com aquele tipo de resultado da ação que está mapeado no arquivo de configuração.

Os tipos de resultados que são gerados e retornados para o usuário de uma ação não precisam ser do mesmo tipo. O tipo de resultado SUCESSO pode fazer com que seja construída uma página JSP enquanto o o resultado ERRO pode precisar enviar um cabeçalho HTTP de volta para o browser. Na maioria das vezes, os tipos usados são os oferecidos pelo framework, mas é possível fazer implementações customizadas de novos tipos.

O resultado é, então, executado, o que freqüentemente (mas não sempre, como no caso de ações encadeadas) envolve um *template* escrito em JSP ou *FreeMarker* para ser renderizado. Então, interceptadores são executados novamente (na ordem reversa). Finalmente, a resposta retorna através dos filtros configurados para toda a aplicação *Web* (`web.xml`) (HUSTED, 2007).

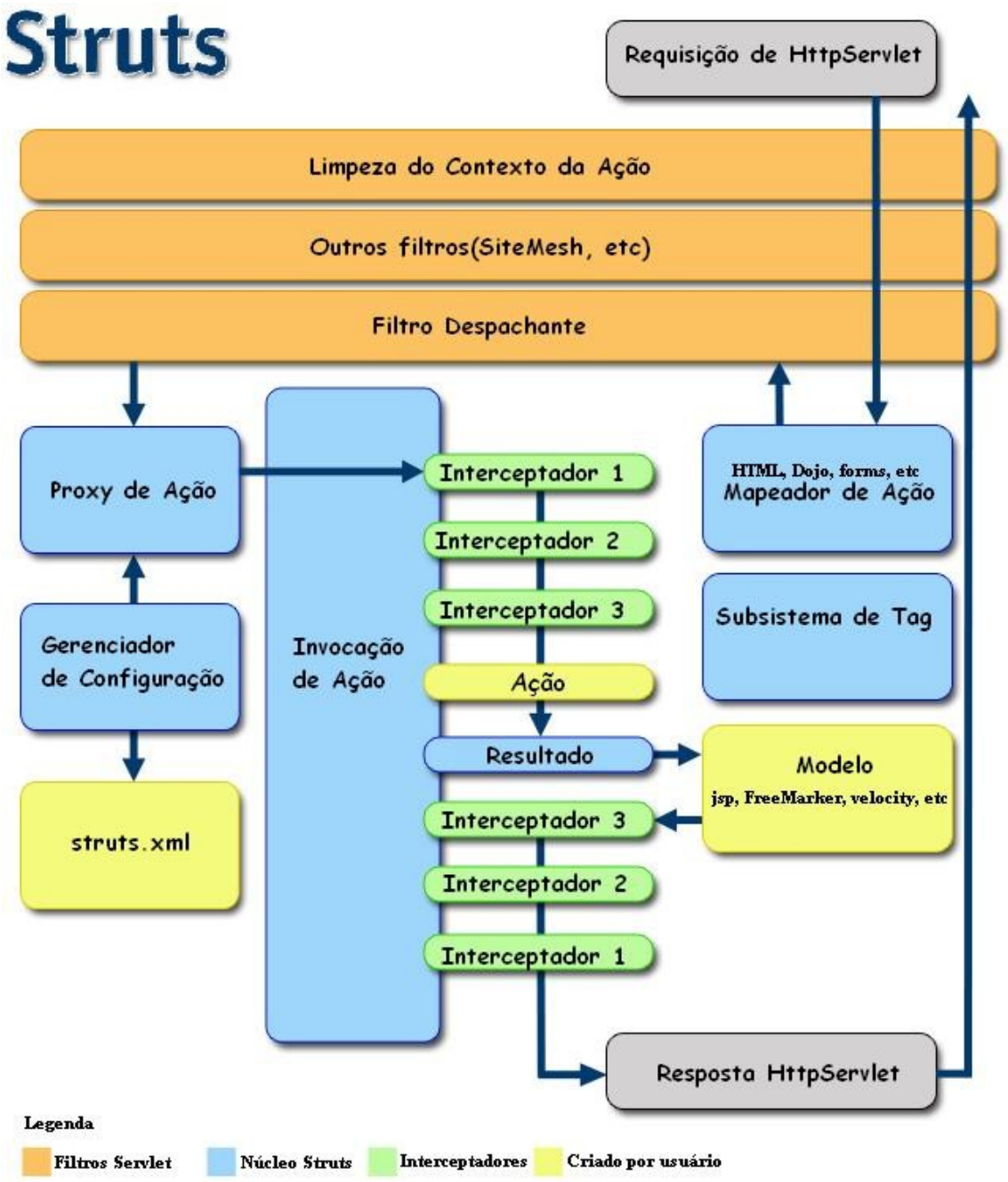

Figura 3: Fluxo de Requisição do Struts 2 (HUSTED, 2007).

Esta arquitetura provida pelo *framework* muda significativamente a forma que as *WebApps* são desenvolvidas e, portanto, tem um impacto também significativo na fase de projeto de sistemas. Para adequar esta fase do desenvolvimento de software ao uso deste e de outros *frameworks*, foi proposto em (SOUZA, 2007) um método para projeto de aplicações *Web* baseadas em *frameworks*, que apresentamos na próxima seção.

2.3. FrameWeb

A Engenharia *Web* tem como forte característica a agilidade nos processos. Com o propósito de atingir essa agilidade estão sendo propostos diversos métodos, *frameworks* e linguagens de modelagem. O Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo desenvolveu um método para guiar a construção de WebApps. Esse método se chama FrameWeb – *Framework-based Design Method for Web Engineering* (SOUZA, 2007).

FrameWeb, além de propor a utilização dos diversos tipos de *frameworks* que existem para se conseguir a agilidade principalmente na fase de codificação da aplicação, nos apresenta (SOUZA, et al, 2005):

- Recomendações para aumentar a agilidade nas principais fases de desenvolvimento (requisitos, análise, projeto e outras);
- Uma arquitetura básica para desenvolvimento de *WebApps*;
- Um perfil UML para um conjunto de modelos de projeto que trazem conceitos usados por algumas categorias de *frameworks*.

O processo de software proposto utiliza UML desde a primeira atividade, quando o processo sugere a utilização de modelos de casos de uso para captura de requisitos funcionais. Apesar de não ser um processo rígido, se espera que alguns modelos sejam construídos a fim de facilitar o processo como um todo. Para se descrever os casos de uso CRUD (*Create, Retrieve, Update and Delete* – Criar, Recuperar, Atualizar e Excluir – comum em situações de cadastro), é recomendado o uso de uma representação comum usando uma tabela que indica quando cada cenário acontece e quais atores têm acesso a eles.

Após a conclusão da primeira atividade, passa-se à fase de análise. Nesta fase, FrameWeb espera que o desenvolvedor crie, usando UML, diagrama de classes para modelagem dos conceitos envolvidos no domínio do problema. Esse modelo é criado em um alto nível de abstração sem levar em consideração a plataforma de implementação. Outros modelos UML podem ser produzidos caso o desenvolvedor entenda que seja necessário. Ao término dessa fase serão realizadas atividades de projeto.

O método dá uma maior importância à fase de projeto, apresentando uma arquitetura de software padrão, mostrada na Figura 4, e um conjunto de modelos de projetos que trazem conceitos usados por algumas categorias de *frameworks,* usando um perfil UML desenvolvido para tornar esses diagramas o mais próximo de suas implementações.

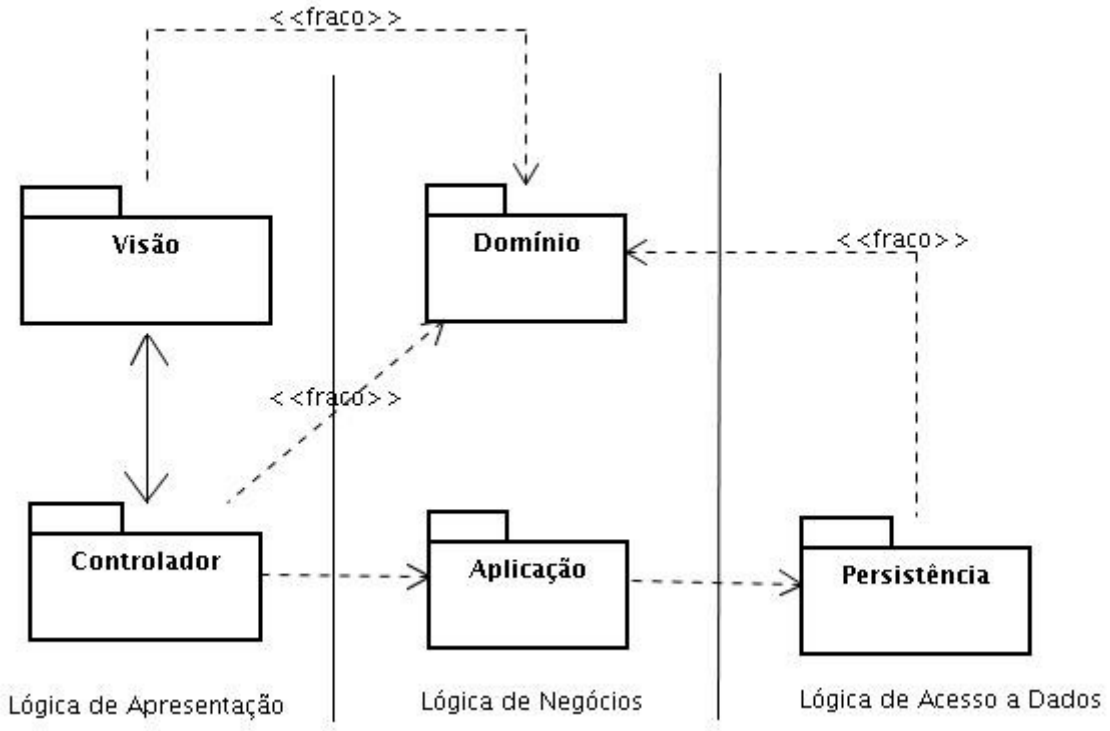

Figura 4: Modelo em três camadas proposto pelo FrameWeb (SOUZA et al., 2005)

A arquitetura proposta divide o sistema em três camadas sendo uma para lógica de apresentação, uma para lógica de negócio e uma para lógica de acesso a dados. Como visto na Figura 4, a camada de lógica de apresentação contém dois pacotes, o de visão contém os arquivos relacionados com a exibição de informação para o usuário, enquanto o pacote de controle compreende classes de ação e outros arquivos relacionados ao *framework* MVC, usado para estruturar a camada *Web*. A camada de lógica de negócio engloba os pacotes de domínio, que incluem os conceitos do domínio identificados e modelados no diagrama de classes, e aplicação, que implementam os casos de uso definidos. Por fim, a camada de lógica de acesso a dados contém o pacote de persistência, que conterá as classes responsáveis por fazer a persistência dos dados em mídias de longa duração(SOUZA, 2007).

O método apresenta, ainda, uma linguagem de modelagem baseada em UML para modelar as classes e outros artefatos, para integração com os *frameworks*. FrameWeb define extensões para o meta-modelo UML para um típico modelo *Web* e componentes relacionados com o *frameworks*. A idéia é criar quatro diagramas: modelo de domínio, modelo de persistência, modelo de navegação e modelo de aplicação.

Uma descrição mais completa do método pode ser encontrada em (SOUZA, 2007).

2.4. *Web* Semântica

Desde que Tim Berners-Lee apresentou sua proposta para a *World Wide Web* em 1990 (BERNERS-LEE, 1990), a sociedade vem passando por uma revolução cultural. Isto porque a interatividade vinda com a Internet nos possibilitou facilidades que afetaram diretamente o nosso cotidiano, porque a Internet nos permite comunicar quase que instantaneamente com pessoas em outros lugares do mundo, com diferentes culturas, diferentes idiomas, tudo isto por meio do computador. Sobretudo, o acesso à informação e a facilidade de disseminar e recuperar informações fazem com que os usuários da grande rede possam encontrar o conhecimento sobre quase tudo o que desejarem.

Atualmente nós temos uma *Web* Sintática. Assim, o computador apresenta informações e os usuários interpretam. Com a finalidade de obter informação sobre determinado assunto, nós recorremos às máquinas de busca baseadas em palavra chave, mas o aumento exagerado de informação que circula na rede torna os resultados encontrados por essas máquinas bastante insatisfatórios (DAVIES, 2003). Na maioria das vezes temos um grande retorno de resultados, mas com uma baixa precisão, outras vezes as palavras-chave escolhidas na consulta não trazem os resultados que nós desejamos, pois os documentos usam uma terminologia diferente da escolhida. Porém, na verdade consultas semanticamente similares deveriam retornar resultados similares.

O problema é que as páginas da Internet não contêm informações sobre si mesmas, e isto dificulta não só a criação de relacionamentos entre documentos do mesmo assunto, mas a busca por documentos em geral (BREITMAN, 2005). Para tentar contornar esta situação o *World Wide Web Consortium* (W3C) propõe a *Web* Semântica, que tem como grande objetivo categorizar a informação de maneira padronizada, facilitando seu acesso. A idéia é representar o conteúdo *Web* de forma a ser mais facilmente processável por máquina e que use técnicas inteligentes para tirar vantagem dessa representação.

Tim Berners-Lee propôs um modelo em camadas, mostrado na Figura 5, para a arquitetura *Web* do futuro (BERNERS-LEE, 2001). Na base dessa arquitetura está o XML, que é uma linguagem que proporciona a escrita de documentos *Web* estruturados. Ele é muito eficaz para enviar documentos pela *Web*. Acima da camada do XML, se encontra o RDF que fornece um modelo formal de dados e sintaxe para codificar meta-dados que podem ser processados por máquinas. Essa camada contém ainda o *RDF Schema* que é baseado no RDF e provê primitivas de modelagem, tais como classes e propriedades, para organizar objetos *Web* em hierarquias.

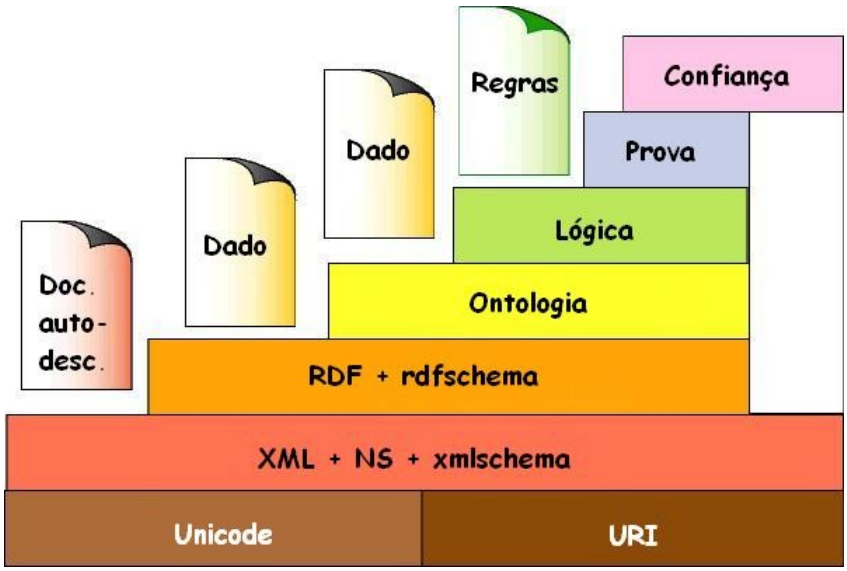

Figura 5: Modelo em camadas proposto para suportar a Web Semântica (BERNERS-LEE, 2000)

Apesar do *RDF Schema* conter conceitos de classes e propriedades, era necessário a representação de relacionamentos mais complexos entre objetos *Web*. Com o intuito de suprir esta necessidade, foi acrescentada a camada de ontologia, que a W3C define como a definição dos termos utilizados na descrição e na representação de uma área do conhecimento. A ontologia deve prover descrições para os seguintes tipos de conceito:

- Classes ou "coisas" nos vários domínio de interesse;
- Relacionamentos entre essas coisas;
- Propriedades ou atributos que essas "coisas" devem possuir.

Com o RDF e RDFS é possível representar algum conhecimento ontológico, porém, isto é feito, por limitações da linguagem, de forma inexpressiva. Segundo a W3C, a linguagem para representar ontologias deve principalmente ter os seguintes requisitos: **uma sintaxe bem definida** para que seja possível haver um processamento da informação pelas máquinas, **uma forma semântica** para descrever o significado do conhecimento precisamente, de forma que não seja sujeito a diferentes interpretações por diferentes pessoas ou máquinas, o que torna possível outro requisito, o **suporte a raciocínio eficiente**. Além disso, a linguagem deve possuir **um grande poder de expressão** (ANTONIOU, 2004).

Visto isto, a W3C observou as linguagens de ontologia existentes a fim de determinar um padrão. As linguagens mais conhecidas eram: SHOE(LUKE, 2000), OIL(FENSEL, 2000), DAML (STEIN, 2000) e DAML+OIL (CONNOLLY, 2001) . A linguagem SHOE foi uma extensão da HTML e é utilizada para anotar o conteúdo de páginas da *Web*. Além disso, fornecia etiquetas

específicas para representar ontologias, contudo essa linguagem era menos expressiva que a RDF. A linguagem OIL (*Ontology Inference Layer*) apresenta uma semântica formal e o mecanismo de inferência baseados na lógica de descrição. É uma linguagem baseada na sintaxe do XML e RDF, que organiza o conhecimento de uma ontologia em três níveis: o recipiente que é o responsável por armazenar informações sobre a própria ontologia, a definição da ontologia que define conceitos como classes, e o objeto onde instância são armazenadas. A linguagem DAML foi desenvolvida pela DARPA (*Defense Advanced Research Projects Agency*) com o objetivo de facilitar a interação de agentes de software autônomos na *Web.* Acabou herdando muitos aspectos presentes em OIL e é composta por elementos de classe, expressões de classe e propriedades. A linguagem DAML+OIL foi uma combinação das duas linguagens anteriores, sendo ela dividida em duas partes:

- Domínio dos objetos: consiste nos objetos que são membros das classes definidas na ontologia DAML;
- Domínio dos tipos de dados: consiste nos valores importados do modelo XML.

Porém, a linguagem que, segundo a W3C, melhor se adequou às necessidades para representação do conhecimento ontológico foi a linguagem OWL (*Ontology Web Language*) que é uma revisão da linguagem DAML+OIL. A OWL vem atender às necessidades da *Web* semântica:

- Construção de ontologias;
- Explicitar fatos sobre um determinado domínio;
- Raciocionar sobre ontologias e fatos.

Esta rica linguagem possui três diferentes sub-linguagens, cada uma para preencher os diferentes aspectos do conjunto de requisitos que ela deve atender. As sub-linguagens são (BREITMAN, 2005):

- OWL *Full*: que usa todas as primitivas da linguagem OWL além de permitir a combinação dessas primitivas com RDF e RDF *Schema*. Uma vantagem dessa sub-linguagem é a completa compatibilidade com RDF, tanto sintática como semanticamente;
- OWL DL: que restringe como os construtores de OWL e RDF podem ser usados, permitindo suporte a um eficiente raciocínio. Em contrapartida, perde a total compatibilidade com o RDF;
- OWL *Lite*: que limita a OWL DL a um subconjunto dos construtores da linguagem, excluindo, por exemplo, classes enumeradas, declarações disjuntas e cardinalidade arbitrária. Isto, obviamente, faz com que essa sub-linguagem tenha uma restrita

expressividade, contudo, é uma forma mais fácil de compreender e implementar.

Com uma ontologia definida e representada em OWL, a idéia é fazer com que agentes de software concordem com os significados de certos termos que estarão na ontologia compartilhada. Assim, eles poderão inferir conclusões e, de fato, será possível incluir semântica nas aplicações *Web* que deixarão de ser somente sintáticas, o que permitirá uma imensa lista de possibilidade de utilização.

2.5. S – FrameWeb

O S-FrameWeb é uma extensão de FrameWeb para o desenvolvimento de aplicações voltadas para a *Web* Semântica (SOUZA et al., 2007). Esse método estende FrameWeb, apresentado na seção 2.4, e tem como principal objetivo fazer com que Sistemas de Informação para *Web* venham incorporados com semântica seguindo a proposta da *Web* Semântica.

A fim de conseguir alcançar seu objetivo, devemos descrever formalmente o domínio da *WebApp, o* que, como visto na seção 2.4, podemos fazer a partir da construção de ontologias. S-FrameWeb acrescenta três novos passos no processo de software: a análise de domínio, o projeto da ontologia e a implementação da ontologia. Assim, o processo sugerido por FrameWeb incorporado com estas três novas etapas fica como mostrado na Figura 6. Nas próximas seções, explicitaremos o desenvolvimento da aplicação com as fases indicadas pelo FrameWeb juntamente com os novos passos incluídos pelo S-FrameWeb.

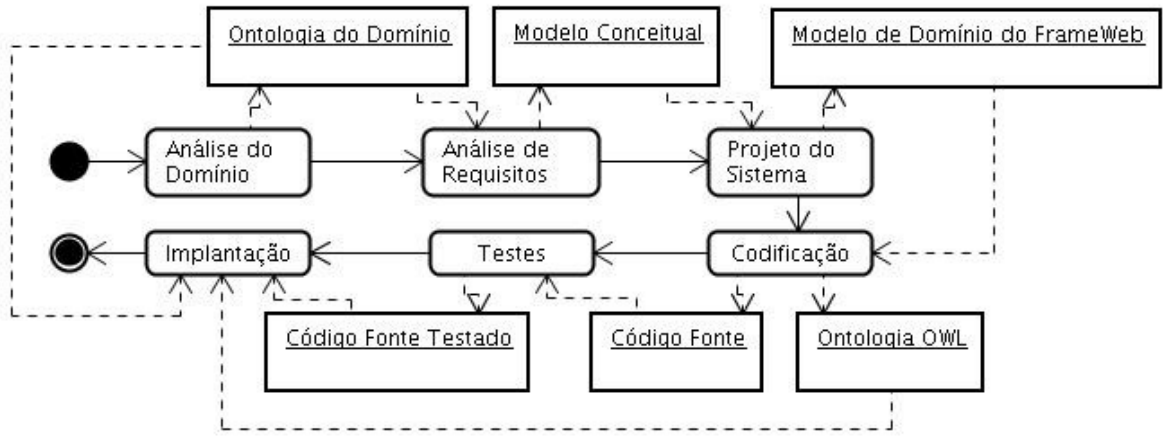

Figura 6: Processo de Software Sugerido pelo S-FrameWeb

Inicialmente, desenvolvemos a Análise do Domínio. Esta engloba a identificação, entendimento e análise do conhecimento do domínio para ser reutilizado na construção e

especificação do software. A idéia é produzir um modelo do domínio do problema, devendo portanto servir como (FALBO et al., 2002):

- uma fonte única de referência a ser utilizada quando aparecem ambiguidade na análise de problemas ou posteriormente na implementação de componentes reusáveis;
- um repositório do conhecimento compartilhado para ensino e comunicação;
- uma especificação para o desenvolvedor de componentes reusáveis.

Com o domínio do problema bem definido e formalmente representado, teremos condições de representar as informações presentes no nosso sistema de maneira com que agentes inteligentes possam receber a nossa ontologia e assim fazer inferências sobre as informações contidas nela. Assim, S-FrameWeb propõe que a *WebApp* identifique se uma ação está sendo requisitada por humanos ou por um agente inteligente para que, assim, ela possa responder da forma adequada. Caso seja um humano, uma página *Web* é apresentada, caso seja um agente, a aplicação retorna um documento OWL. Na seção 3.5, mostraremos em detalhes como esta identificação é feita e como esta resposta é dada pelo *framework*.

Para representar a ontologia, S-FrameWeb indica o uso do *Ontology Definition Metamodel* (ODM), da OMG[1], que é uma linguagem para modelar ontologias para a Web Semântica no contexto do MDA (*Model Driven Architecture*). A especificação ODM oferece vários benefícios a seus potenciais usuários *(IBM, 2005*):

- Opções no nível de expressividade, complexidade e formas disponíveis para projetar e implementar modelos conceituais, abrangendo desde a familiar UML e metodologias de ER até ontologias formais representada em lógica de descrição ou lógica de primeira ordem;
- Fundamentação na lógica formal, com base em padrões, modelos semânticos téoricos para representação de conhecimento suportados por linguagem, suficiente para que máquinas de inferência sejam capazes de entender, validar e aplicar ontologias desenvolvidas usando ODM;
- Perfis e mapeamentos suficientes para suportar não somente a troca de modelo desenvolvidos independentemente em vários formalismos mas também para habilitar checagem e validação consistentes de maneiras que não são possíveis atualmente;
- A base para uma família de especificações que envolvem MDA e tecnologias de Web Semântica para suportar *Web Services* Semânticos, ontologias, políticas baseadas em comunicações e interoperabilidade e políticas baseadas em aplicações em geral.

1 Object Management Group – http://www.omg.org

A especificação define uma família de metamodelos independentes, perfis relacionados e mapeamentos entre estes metamodelos correspondendo a vários padrões internacionais para ontologia, bem como a capacidade de suportar paradigmas de modelagem convencionais para captura conceitual de conhecimento como, por exemplo, a modelagem entidade-relacionamento.

O ODM é aplicável à representação de conhecimento, modelagem conceitual, desenvolvimento de taxonomia formal e definição de ontologia, e torna possível o uso de uma variedade de modelos como ponto de partida para o desenvolvimento de ontologias através de mapeamentos para UML.

Seguindo o método S-FrameWeb, nós utilizamos um perfil UML de ontologia. Este perfil nos permite representar ontologias em um diagrama de classe UML.

2.6. Jena

A *Web* Semântica pode usar ontologias para representar conhecimento, contudo devemos ter formas de manipular essas ontologias para conseguirmos extrair ou incluir nelas as informações que forem necessárias. Com o objetivo de suprir esta necessidade, o *HP Labs Semantic Web Programme* juntamente com colaboradores, criou o Jena.

Jena é uma API para a linguagem de programação Java usada na criação e manipulação de grafos RDF. A API Jena possui objetos, classes para representar os modelos, recursos, propriedades e literais do RDF. As interfaces representando recursos, propriedades e literais são chamadas *Resource*, *Property* e *Literal*, respectivamente. Em Jena, um grafo RDF é chamado de modelo e é representado pela interface *Model*. A primeira versão da API Jena possui métodos para a manipulação de ontologias em DAML+OIL, enquanto que na segunda versão foram desenvolvidos métodos que permitem a manipulação em ontologias RDFS e OWL. A API DAML da Jena teve muito pouco suporte para desenvolvimento de ontologias com a semântica da linguagem DAML (Gonçalves et al. 2004).

Como mostrado nas seção anterior, existem várias linguagens de ontologias disponíveis para representar conhecimento ontológico, contudo a API de ontologia do Jena provê uma interface de programação consistente para o desenvolvedor de aplicações na *Web* Semântica, independente da linguagem de ontologia que está sendo processada. Isto é possível porque no Jena cada linguagem de ontologia tem um perfil que lista os construtores permitidos e as URI's (*Uniform Resource Identifier*), que é um conjunto compacto de caracteres usados para identificar um recurso, das classes e propriedades.

O Jena permite construir modelos de ontologia, importar ontologias que estão em documentos

para um modelo de ontologia por meio de um gerenciador de documentos, tratar componentes básicos e complexos da ontologia, como classes, propriedades, restrições, expressões de classes booleanas, expressões de listas, expressões de classes de intersecção, união, complemento, classes enumeradas, listar classes e, por fim, ainda permite criar e manipular indivíduos (DICKINSON, 2007).

Para o suporte a manipulação de ontologias, a segunda versão da API Jena possui um pacote específico, chamado: API Jena 2 *Ontology*. Nesse pacote existem classes para a manipulação de ontologias em RDFS, DAML+OIL e OWL. Para o suporte a essas linguagens de ontologias a API possui as seguintes classes: *OntClass* e *ObjectProperty*.(Gonçalves et al. 2004).

Jena também oferece modelos persistentes. Assim, os modelos de ontologia podem ser gravados em arquivos ou mesmo bancos de dados. Isso é de grande utilidade, visto que, se usar um modelo em memória, será necessário inserir os indivíduos cada vez que a aplicação for iniciada.

Por fim, Jena provê vários tipos de raciocinadores para trabalhar com diferentes tipos de documentos. Assim, dados uma ontologia e um modelo, o motor de inferência do Jena pode derivar declarações adicionais que o modelo não expressa explicitamente.

Capítulo 3

A extensão do FrameWork Struts²

Esste capítulo mostra a implementação que fizemos baseado nas idéias do método S-FrameWeb sobre incorporar ao *framework* Controlador Frontal a capacidade de distinção de acesso por agentes inteligentes e humanos.

Para alcançar nosso objetivo, estendemos o *framework* Struts 2, que é o nosso Controlador Frontal, para que ele fosse capaz de reconhecer requisições de agentes de software e responder com um documento OWL, contendo as mesmas informações que seriam retornadas pela página, mas codificada com instâncias OWL. Juntamente com a ontologia de domínio e o modelo de domínio da aplicação, agentes de software podem raciocinar sobre os dados que resultaram da requisição (SOUZA et al., 2007).

Além de estender o Controlador frontal utilizamos o *framework* Jena para manipular o nosso arquivo OWL que estava representando a nossa ontologia. Na Figura 7, é mostrado o diagrama de classes que foram desenvolvidas. As classes *SemanticWebInterceptor* e *SemanticWebPreResultListener* tratam da estensão do *framework* Controlador Frontal enquanto a classe *OWLOntologyResult* foi a classe desenvolvida para extrair do arquivo OWL e das classes da aplicação, utilizando a característica de reflexão da linguagem Java, as informações necessárias para gerar anotações dinâmicas.

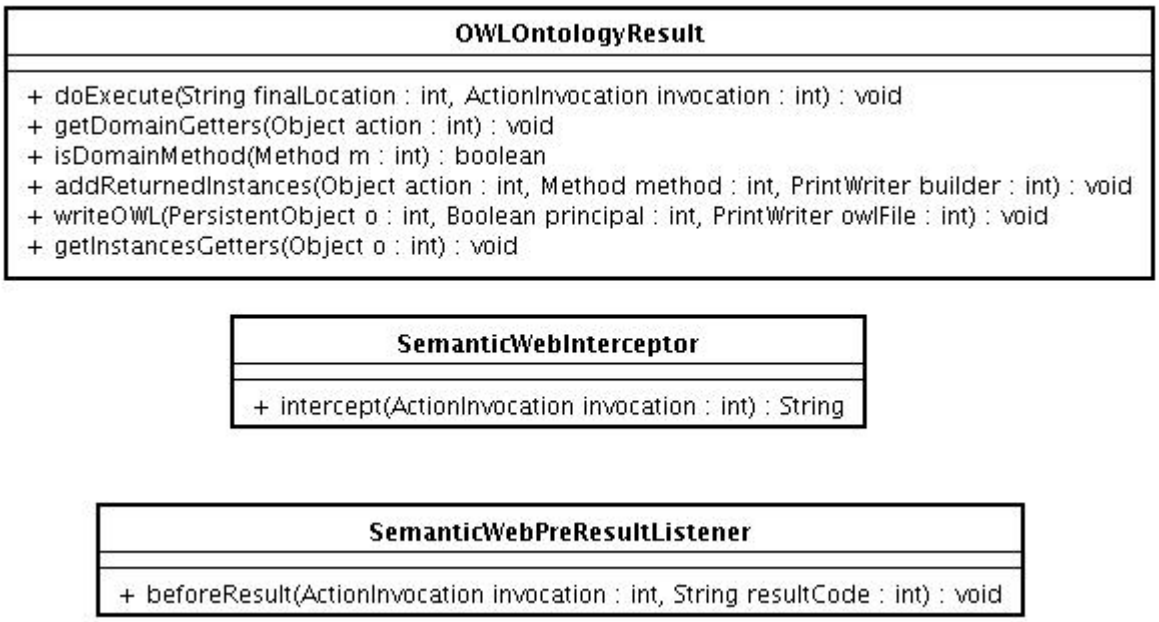

Figura 7: Diagrama de classes da extensão do Controlador Frontal e utilização do Jena

Utilizamos então o FrameWeb e S-FrameWeb na construção de uma pequena aplicação. Para desenvolvê-la utilizamos diversas sugestões do FrameWeb, por exemplo, a utilização de *frameworks* e posteriormente criamos os modelos propostos pelo S-FrameWeb e aplicamos a parte principal desse trabalho que é apresentada nas próximas seções.

3.1. O Interceptador

A Figura 8 ilustra a nossa idéia, o navegador do internauta faz uma requisição de ação para o *framework*. Antes que a ação seja executada, o Controlador faz a requisição passar por uma pilha de interceptadores. O fluxo de requisição do Struts 2 é melhor detalhado no capítulo 2.

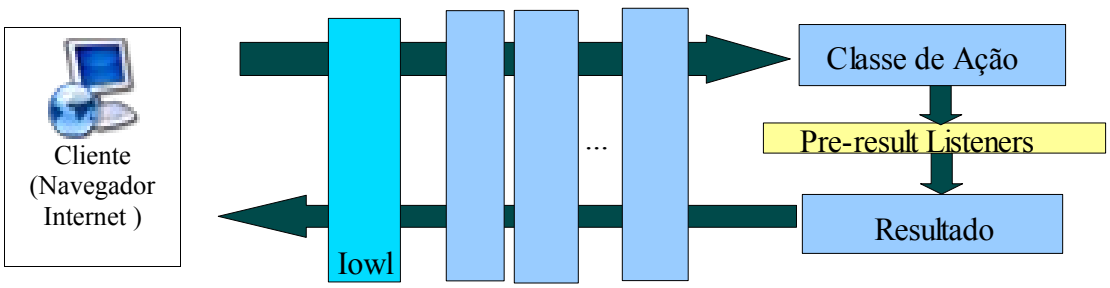

Figura 8: Extensão proposta pelo S-FrameWeb (SOUZA et al., 2007)

No arquivo de configuração `struts.xml`, estão presentes todos os interceptadores da

aplicação. Assim, definimos no arquivo `struts.xml` o Interceptador OWL (I_{owl}) como o primeiro interceptador da pilha. Esse interceptador é definido na classe *SemanticWebInterceptor* no diagrama de classes apresentado na figura 7.

Quando a requisição está passando pela pilha de interceptadores, o Interceptador OWL, verifica a existência de um parâmetro específico, que definimos ser o conjunto de caracteres "owl", indicando que a ação deva retornar um resultado OWL. Note que neste momento sabemos se a requisição foi feita por um humano ou por um agente inteligente, sendo assim capazes de retornar um resultado compreensível por humano ou processável por máquinas.

3.2. Tratamento do arquivo OWL utilizando o Jena

Caso seja detectado a presença do argumento que indica uma requisição feita por agente inteligente, o interceptador cria um *PreResultListener* que será responsável por desviar a requisição para a classe responsável por criar o resultado que é o documento OWL. O nome dessa classe no diagrama de classes é *OWLOntologyResult* e ela é responsável por carregar os objetos que estão persistentes, gerando assim os indivíduos no nosso documento OWL. Nesta seção vamos descrever mais detalhadamente essa classe.

Para que isso fosse possível, a nossa ontologia escrita em OWL e ainda sem os indivíduos foi colocada na nossa aplicação. Assim, a *WebApp* tem à sua disposição a estrutura ontológica do domínio definida em uma linguagem que pode ser processável, ou seja, um arquivo que possui basicamente as classes, propriedades e seus relacionamentos definidos. Para processar estas informações utilizamos o Jena, que é também um *framework* Java que oferece um ambiente programático para muitas linguagens de ontologia e que foi descrito no capítulo 2.

Assim, conseguimos extrair as classes do domínio, criando um conjunto de classes do domínio, para isso usamos a Classe OntModel do Jena que é capaz de recuperar dados da ontologia, sendo assim, utilizamos esta classe para recuperar as classes definidas no nosso arquivo:

```
Conjunto classesDomínio
OntModel ontologia = lêEstruturaOntológica(estruturaOntologia.owl)
enquanto ontologia.Classes.temClasse()
        classesDomínio.adicionaClasse(ontologia.Classes.recuperaPróxima)
fimEnquanto
```

Então, tendo conhecimento das classes do domínio definidos na ontologia, somos capazes de

usar reflexão. Esta característica, que algumas linguagens possuem, permite um programa examinar ou fazer introspecção em suas próprias classes. Daí, poderemos manipular todas as informações que temos e conseguir recuperar os objetos persistentes que são de nosso interesse.

Inicialmente, recuperamos todos os métodos que retornam objetos que sejam de classes de domínio. Fazemos isto através dos métodos cujo pseudocódigo é exibido abaixo:

```
//Método que define se um método retorna objetos cuja classes estão presente em classesDomínio
éMetodoDomínio(Metodo m)
  Classe classeCandidata = m.tipoRetornado()
  retorna classesDomínio.contémClasse(classeCandidata.recuperaNome())
fim do método

//Método que recupera os métodos de recuperação que recuperam objetos do domínio
Conjunto<Métodos> recuperaMétodosDeRecuperação (Objeto o)
  Conjunto <Métodos> recuperadores
  Classe classeAtual = o.recuperaClasse()

  //Percorre toda a hierarquia da classeAtual, exceto as classes de frameworks e da classe Objeto
  enquanto classeAtual diferente das classes de framework e da classe Objeto
        Métodos métodos = classeAtual.recuperaMétodosDeclarados()
        enquanto métodos.temMétodo()
              Método m = métodos.recuperaPróximo()
              se (m é método Público e inicia com get e possui a quarta letra maiúscula e é método de
                    domínio e não possui parâmetros)
                    recuperadores.adicionaMétodo(m)
              fim se
        fim enquanto
  fim enquanto
  retorna recuperadores
fim do método
```

Recuperamos os objetos através da invocação destes métodos de recuperação retornados e, para cada um destes objetos, aplicamos o método `escreveOWL`, este método é responsável por navegar na estrutura do objeto e, de acordo com sua classe, descobrir suas propriedades a partir da interação com o documento de estrutura da ontologia para assim escrever o documento OWL. Foi definido que as propriedades deveriam possuir comentários, algo análogo às anotações de Java, que identificasse qual o tipo relacionado àquela propriedade. Por exemplo, Receita é *feitaDe* um ou

vários Ingredientes, portanto a propriedade *feitaDe* deve possuir um comentário indicando que ela está relacionada com Ingredientes. Assim, ao encontrar a propriedade *feitaDe*, usaremos o método `recuperaIngredientes` para recuperar tais objetos.

O método `escreveOWL` deve ser chamado recursivamente para que todas as propriedades de todos os objetos sejam relacionados. O pseudocódigo do método é mostrado abaixo:

```
Mapa mapaInstâncias
Mapa mapaPropriedadesMétodos
escreveOWL(Objeto o, Booleano éPrincipal)
  //verifica se o objeto não está no mapa de instâncias
  se (não mapaInstâncias.contém(o.IdentificadorÚnico))
    mapaInstâncias.adicionaObjeto(o)
    Classe classeObjeto = o.recuperaClasse()
    se éPrincipal
      imprime(o.IdentificadorÚnico)
    fim se
    //recupera a classe da ontologia para posteriormente recuperar as propriedades da classe
    OntClass classeOntologica = ontologia.recuperaClasseOntologia(classeObjeto.recuperaNome())

    enquanto classeOntológica.propriedades.temPropriedade()
      Propriedade p = classeOntológica.propriedades.recuperaPróximo()
      //recupera somente as propriedades que não são funcionais inversas
      se  não p.éPropriedadeFuncionalInversa()
        //verifica se o comentário possui a variável atributo que contém a classe de objetos que devem ser
            retornados
        se p.recuperaComentário().contém(@atributo)
          //coloca no mapa de propriedades e métodos o nome da propriedade e o método que será responsável
              por recuperar os objetos, este método é iniciado com os caracteres "recupera" seguido dos
              caracteres que estão no comentário
          mapaPropriedadesMétodos.insere(p.recuperaNome(), "recupera"+p.recuperaComentário())
        fim se
      fim se
    fim enquanto
    Conjunto<Métodos> métodosRecuperadoresObjetos = o.recuperaClasse().recuperaMétodos();
    enquanto métodosRecuperadoresObjetos.temMétodo()
      Método m = métodosRecuperadoresObjetos.recuperaPréximo()
      Objeto objetoRetornado = m.executa(o)
      Classe tipoRetornado = m.recuperaTipoRetorno()
      se tipoRetornado = Conjunto
        enquanto objetoRetornado.temObjeto()
          se mapaPropriedadesMétodos.contém(m.recuperaNome())
            imprime("<" + mapaPropriedadesMétodos.recuperaPropriedade() + ">")
          imprime(objetoRetornado.recuperaPróximo().recuperaIdentificador())
          imprime("</" + mapaPropriedadesMétodos.recuperaPropriedade() + ">")
          escreveOWL(objetoRetornado.recuperaPróximo(), Falso)
        fim se
      fim enquanto
    fim se
    senão
      se mapaPropriedadesMétodos.contém(m.recuperaNome())
        imprime("<" + mapaPropriedadesMétodos.recuperaPropriedade() + ">")
      imprime(objetoRetornado.recuperaIdentificador())
      imprime("</" + mapaPropriedadesMétodos.recuperaPropriedade() + ">")
      escreveOWL(objetoRetornado.recuperaPróximo(), Falso)
      fim se
    fim senão
  fim enquanto
```

fim se
fim do método

Podemos perceber que este método utiliza um parâmetro para indicar o objeto, e um parâmetro para indicar se é um objeto principal. Isto deve ser indicado pelo responsável pela invocação do método. Quando ele é chamado recursivamente, observe que este parâmetro sempre recebe valor falso. Continuando a explicação, nós primeiramente verificamos se o objeto está presente no `mapaInstancias` e, caso esteja, não continuamos a execução do método, de forma que o mesmo objeto não seja processado duas vezes. Usando a reflexão, descobrimos qual a classe do objeto e carregamos a classe da ontologia que possui o nome da classe do objeto. Com a classe da ontologia, recuperamos todas as propriedades declaradas para aquela classe, com exceção das propriedades funcionais inversas, pois estas somente sobrecarregariam a ontologia. Todas as propriedades recuperadas são colocadas no `mapaPropriedadesMetodos`. Então nós colocamos em um conjunto, `metodosRecuperadoresObjeto`, todos os métodos capazes de recuperar objetos, para fazer isso utilizamos uma idéia semelhante a do método `recuperaMetodosDeRecuperacao` mostrado anteriormente, sendo a única diferença que, desta vez, não recuperamos somente métodos que retornam objetos do domíno da ontologia, pois o nosso documento OWL deve possuir todos os objetos ligados ao objeto em questão, seja caracteres, inteiro, etc. Em seguida, invocamos estes métodos e fazemos o tratamento dos objetos retornados por eles, verificando como estes objetos se relacionam com o objeto inicial através das propriedades. Estes objetos possivelmente farão uma nova chamada ao método `escreveOWL`, de modo a buscar os outros objetos que se relacionam com eles.

1

3.3. Cookbook

O Cookbook foi a nossa aplicação inicial para o desenvolvimento de um sistema utilizando o método FrameWebvisando uma maneira para desenvolvimento de aplicações para a *Web* Semântica. A proposta do Cookbook é bem simples: fazer o gerenciamento de receitas. O usuário poderá armazenar receitas, das quais é importante saber: o nome, o modo de preparo, as porções resultantes e os ingredientes que a compõem, bem como a medida de cada ingrediente utilizado na receita. De um ingrediente é importante saber o nome e a medida padrão do ingrediente. De uma medida deve-se saber apenas o nome (ex.: litro, kg, pitada, etc.).

As receitas armazenadas ficarão disponíveis para pesquisa dos usuários, sendo possível informar o nome da receita ou de algum ingrediente que compõem a receita para realizar a pesquisa. Como sugerido por FrameWeb, elaboramos casos de uso para capturar os requisitos funcionais, mostrado na Figura 9.

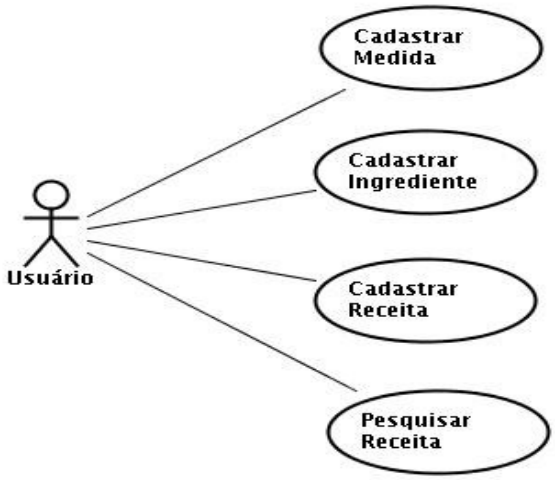

Figura 9: Diagrama de casos de uso do Cookbook

A seguir descrevemos os casos de uso. Devido à simplicidade e ao objetivo desta aplicação de servir apenas de protótipo para inserção da *Web* Semântica, a descrição dos casos de uso será feita de maneira simples e informal. No próximo capítulo, ao tratarmos de uma aplicação real, apresentaremos estas etapas com um maior rigor:

- Cadastrar Receita: este caso de uso trata da inserção de uma nova receita. O usuário informa o nome da receita, o modo de preparo, quantidade de porções geradas pela receita, e os ingredientes que compõem a receita. Para cada ingrediente da receita, o usuário também informa a quantidade e a medida;
- Cadastrar Ingrediente: este caso de uso abrange a inclusão, exclusão e alteração de um ingrediente. Para inserir ou alterar um ingrediente, o usuário deve informar o nome e a medida padrão do ingrediente que se deseja inserir/alterar;
- Cadastrar Medida: este caso de uso abrange a inclusão, exclusão e alteração de um medida. Para inserir ou alterar um medida, o usuário deve informar o nome da medida que se deseja inserir/alterar;
- Pesquisar Receitas: divide-se em dois cenários: “Procurar Receita pelo Nome” e “Procurar Receita por Ingredientes”. O usuário informa o nome da receita a ser pesquisada

ou nome do ingrediente contido na receita, de acordo com o tipo de pesquisa que se deseja fazer.

3.4. S-FrameWeb

O Cookbook foi desenvolvido utilizando S-FrameWeb, uma extensão de FrameWeb para odesenvolvimento de aplicações voltadas para a *Web* Semântica (SOUZA et al., 2007). Esse método estende FrameWeb, apresentado no capítulo dois, e tem como principal objetivo fazer com que Sistemas de Informação para *Web* venham incorporados com semântica seguindo a proposta da *Web* Semântica.

A fim de conseguir alcançar seu objetivo, devemos descrever formalmente o domínio da *WebApp, o* que, como visto na seção 2, podemos fazer a partir da construção de ontologias. S-FrameWeb acrescenta três novos passos no processo de software: a análise de domínio, o projeto da ontologia e a implementação da ontologia. Assim, o processo sugerido por FrameWeb incorporado com estas três novas etapas fica como mostrado na Figura 10. Nas próximas seções, explicitaremos o desenvolvimento da aplicação com as fases indicadas pelo FrameWeb juntamente com os novos passos incluídos pelo S-FrameWeb.

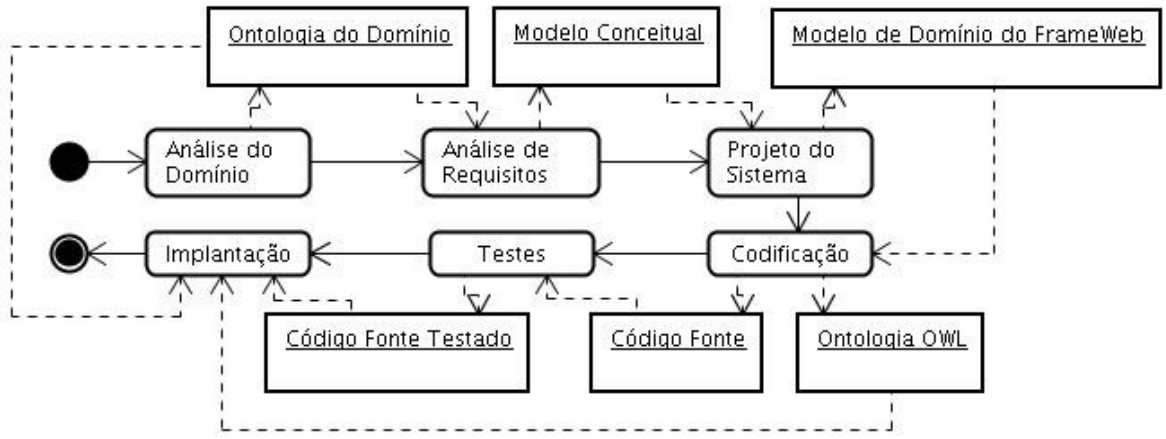

Figura 10: Processo de Software Sugerido pelo S-FrameWeb

Inicialmente, desenvolvemos a Análise do Domínio. Esta engloba a identificação, entendimento e análise do conhecimento do domínio para ser reutilizado na construção e especificação do software. A idéia é produzir um modelo do domínio do problema, devendo portanto servir como (FALBO et al., 2002):

- uma fonte única de referência a ser utilizada quando aparecem ambiguidade na análise

de problemas ou posteriormente na implementação de componentes reusáveis;

- um repositório do conhecimento compartilhado para ensino e comunicação;
- uma especificação para o desenvolvedor de componentes reusáveis.

Com o domínio do problema bem definido e formalmente representado, teremos condições de representar as informações presentes no nosso sistema de maneira com que agentes inteligentes possam receber a nossa ontologia e assim fazer inferências sobre as informações contidas nela. Assim, S-FrameWeb propõe que a *WebApp* identifique se uma ação está sendo requisitada por humanos ou por um agente inteligente para que, assim, ela possa responder da forma adequada. Caso seja um humano, uma página *Web* é apresentada, caso seja um agente, a aplicação retorna um documento OWL. Na seção 3.5, mostraremos em detalhes como esta identificação é feita e como esta resposta é dada pelo *framework*.

Para representar a ontologia, S-FrameWeb indica o uso do *Ontology Definition Metamodel* (ODM), da OMG[2], que é uma linguagem para modelar ontologias para a Web Semântica no contexto do MDA (*Model Driven Architecture*). A especificação ODM oferece vários benefícios a seus potenciais usuários *(IBM, 2005*):

- Opções no nível de expressividade, complexidade e formas disponíveis para projetar e implementar modelos conceituais, abrangendo desde a familiar UML e metodologias de ER até ontologias formais representada em lógica de descrição ou lógica de primeira ordem;
- Fundamentação na lógica formal, com base em padrões, modelos semânticos téoricos para representação de conhecimento suportados por linguagem, suficiente para que máquinas de inferência sejam capazes de entender, validar e aplicar ontologias desenvolvidas usando ODM;
- Perfis e mapeamentos suficientes para suportar não somente a troca de modelo desenvolvidos independentemente em vários formalismos mas também para habilitar checagem e validação consistentes de maneiras que não são possíveis atualmente;
- A base para uma família de especificações que envolvem MDA e tecnologias de Web Semântica para suportar *Web Services* Semânticos, ontologias, políticas baseadas em comunicações e interoperabilidade e políticas baseadas em aplicações em geral.

A especificação define uma família de metamodelos independentes, perfis relacionados e mapeamentos entre estes metamodelos correspondendo a vários padrões internacionais para ontologia, bem como a capacidade de suportar paradigmas de modelagem convencionais para

2 Object Management Group – http://www.omg.org

captura conceitual de conhecimento como, por exemplo, a modelagem entidade-relacionamento.

O ODM é aplicável à representação de conhecimento, modelagem conceitual, desenvolvimento de taxonomia formal e definição de ontologia, e torna possível o uso de uma variedade de modelos como ponto de partida para o desenvolvimento de ontologias através de mapeamentos para UML.

Seguindo o método S-FrameWeb, nós utilizamos um perfil UML de ontologia. Este perfil nos permite representar ontologias em um diagrama de classe UML. Na próxima seção, será apresentado o modelo conceitual do Cookbook, que foi construído utilizando este perfil definido pelo ODM.

3.5. Análise

FrameWeb nos sugere a criação de um diagrama de classes na fase de análise da aplicação. Este modelo deve ser feito com um alto nível de abstração sendo, assim, independente de plataforma. Ele deve modelar os conceitos envolvidos no domínio do problema facilitando as fases seguintes como criação do modelo do banco de dados. De acordo com os requisitos que puderam ser observados a partir do diagrama de casos de uso gerado, construímos o diagrama de classe mostrado na Figura 11. Este diagrama representa o modelo de domínio que é definido pelo FrameWeb.

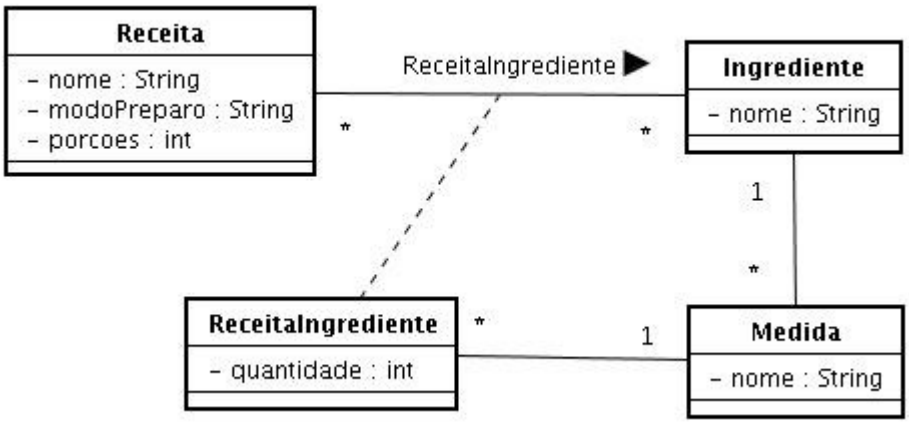

Figura 11: Diagrama de Classes

Como já mencionado anteriormente, algumas etapas foram incluídas no processo proposto pelo FrameWeb a fim de possibilitar o suporte à *Web* Semântica. Com o objetivo de descrever formalmente o domínio da aplicação seguimos o método S-FrameWeb, descrito na seção 3.2, e construímos uma ontologia para o nosso domínio. Deve ficar claro que, no desenvolvimento de um

ambiente real e de maiores proporções, teríamos uma ontologia mais complexa. Um caso real será tratado no próximo capítulo, quando mostraremos a inserção de suporte à *Web* Semântica no Portal do LabES.

Na Figura 12, é mostrado o modelo conceitual do Cookbook, que foi construído utilizando o perfil UML proposto pelo ODM. É importante observar que o estereótipo `<<owlClass>>` indica uma classe de domínio, `<<owlObjectProperty>>` indica associações entre classes de domínio, `<<owlDataRange>>` representa tipos de dados e `<<owlDataTypeProperty>>` modela associações entre classes e tipos de dados.

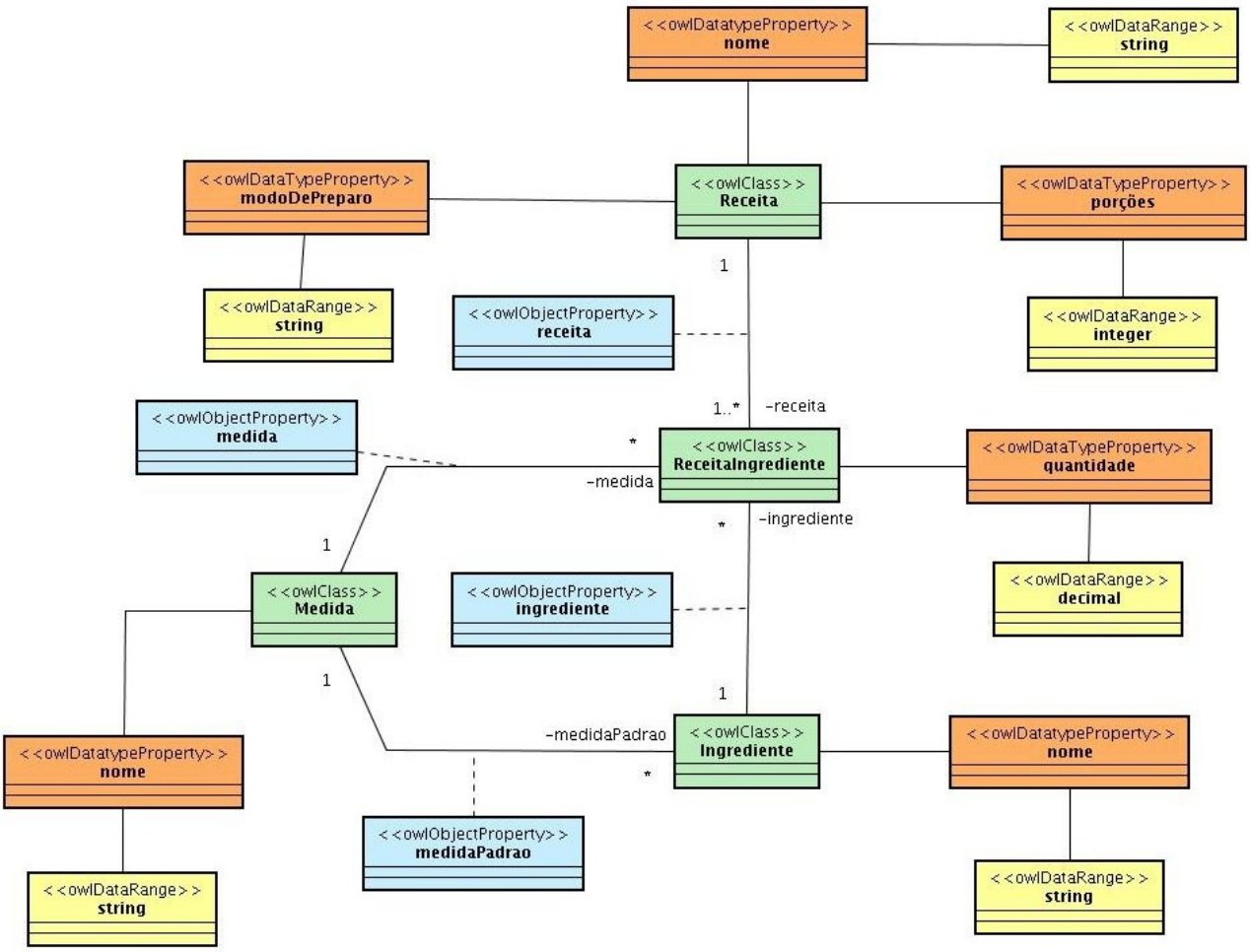

Figura 12: Modelo Conceitual para o Cookbook

S-FrameWeb propõe uma extensão para este diagrama misturando os perfis UML do FrameWeb e do ODM para a construção do modelo de domínio. A idéia é simplificar a sintaxe do ODM e adicionar mapeamentos objetos/relacionais. Na Figura 13, podemos ver o diagrama simplificado, e notamos que os elementos com estereótipo `<<owlDataTypeProperty>>` foram transformados em atributos de classe.

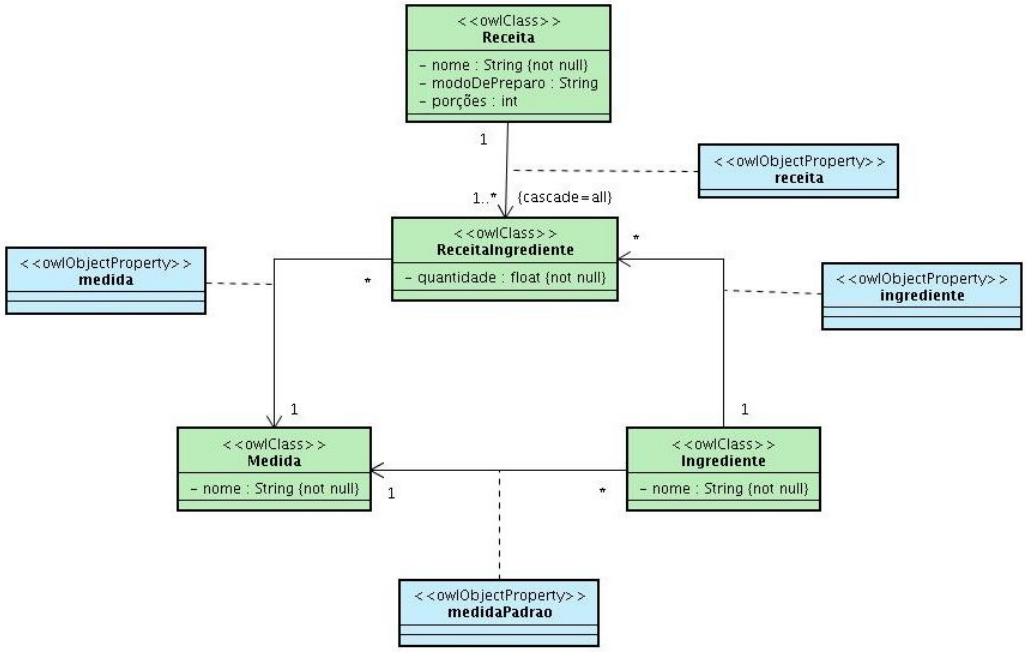

Figura 13: Modelo de Domínio proposto pelo S-FrameWeb

Na fase de codificação, a ontologia de domínio deve ser codificada em OWL. O arquivo gerado é colocado então em uma localização pré-determinada para que assim a *WebApp* saiba em que local deverá procurar por ele. Estes passos serão melhor descritos na seção 3.5, na qual comentaremos a implementação do Cookbook.

3.6. Projeto

Seguindo o processo de software, iniciamos a fase de projeto. Nesta fase, especificaremos alguns dos modelos sugeridos pelo método FrameWeb, a saber: Modelo de Domínio, Modelo de Persistência, Modelo de Navegação e Modelo de Aplicação. Estes modelos serão de grande utilidade na próxima fase do processo, visto que, eles oferecem uma boa visão de como será a implementação da *WebApp*.

Iniciamos nossa fase de projeto pelo modelo de domínio que representa os objetos do domínio do problema. As Figuras 12 e 13 nos mostram as representações diagramáticas do nosso domínio. Podemos observar a utilização de algum mapeamentos sugeridos pelo método seguido. Alguns atributos de algumas classes contém o mapeamento `{not null}` indicando que não é possível que ele seja nulo, além disso existe um mapeamento de associação de cascateamento total,

os demais mapeamentos não aparecem pois possuem o valor padrão especificado pelo método. A seguir, definimos o modelo de persistência, que apresenta os objetos responsáveis pela persistência dos dados em banco de dados relacional. Note que para cada classe persistente que precisa de lógica de persistência, o modelo deve apresentar uma interface DAO e uma ou mais classes DAO, sendo que estas devem implementar a interface DAO que definem os métodos disponíveis para a persistência de uma determinada classe. Na Figura 14, mostramos o modelo de persistência desenvolvido para o Cookbook.

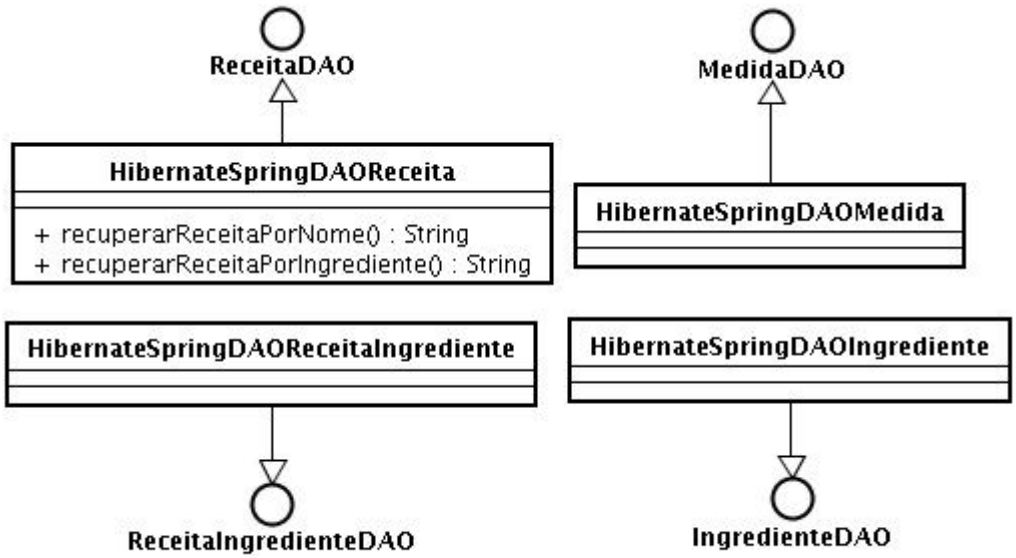

Figura 14: Modelo de Persistência do Cookbook

Uma importante observação é que todos estas interfaces DAO estendem a interface BaseDAO, conforme sugerido pelo método. Esta interface, mostrada na Figura 15, define os métodos básicos de persistência que todos os DAOs devem implementar.

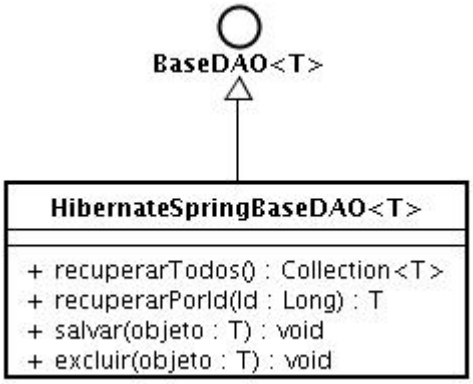

Figura 15: Interface BaseDAO sugerido pelo método FrameWeb

Passamos então ao próximo modelo, o de Navegação. Este modelo é um diagrama de classe

UML que descreve os diferentes componentes que forma a camada de apresentação para um dado caso de uso ou cenário e os relacionamentos entre eles (SOUZA et al., 2005). Na Figura 16, temos o modelo de navegação para o caso de uso Cadastrar Receita. Não exibiremos todos os modelos de navegação do Cookbook, um projeto mais bem elaborado e completo será mostrado no próximo capítulo.

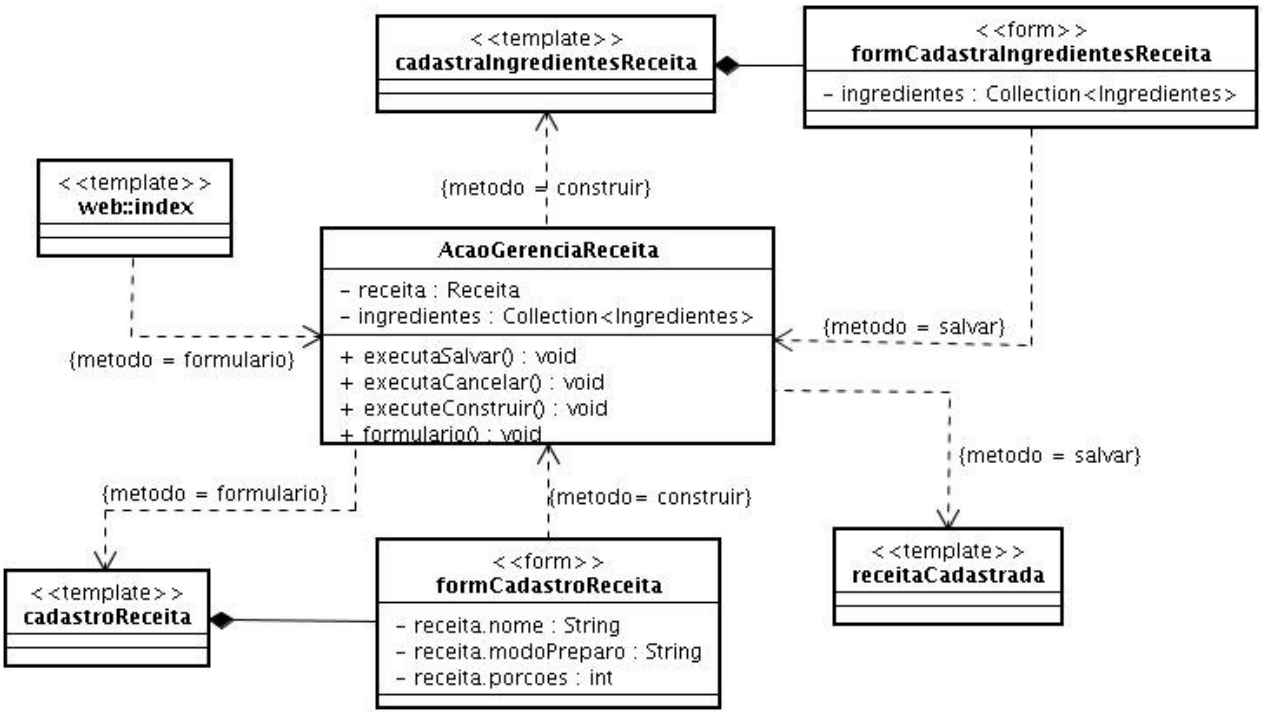

Figura 16: Modelo de Navegação para o caso de uso Cadastrar Receita do Cookbook

Por fim, construímos o modelo de Aplicação, que é um diagrama de classe UML que mostra as classes do pacote de Aplicação e os relacionamentos dele com as outras classes do pacote de Controle e Persistência. Além de guiar a construção dos pacotes da camada de negócio, este modelo demonstra como integrar as diferentes camadas para prover a solução desejada na arquitetura proposta pelo FrameWeb (SOUZA et al., 2005). A Figura 17, mostra o modelo de aplicação do Cookbook, podemos observar que para cada classe modelada, há um interface e uma classe concreta, o que reforça a prática de "programação por interfaces".

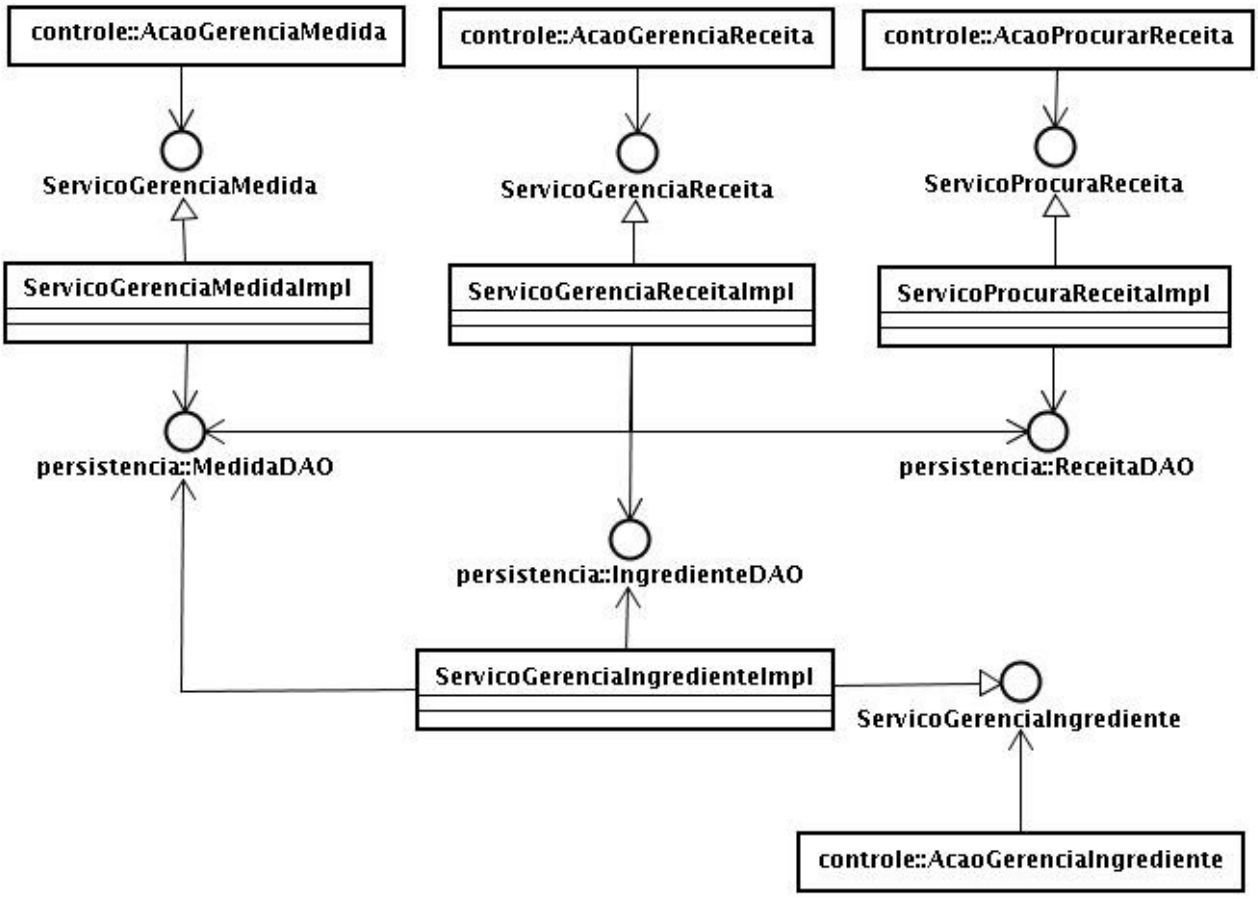

Figura 17: Modelo de Aplicação do Cookbook

Aplicando as implementações do interceptador e utilização do framework Jena mostrados nas primeiras seções, conseguimos atingir nosso objetivo de incluir a semântica no Cookbook. No próximo capítulo estaremos demonstrando as etapas do desenvolvimento de uma aplicação real utilizando o S-FrameWeb. Incluiremos a *Web* Semântica no Portal do do Laboratório de Engenharia de Software da UFES.

Capítulo 4

Portal do LabES - Uma Aplicação Real

O Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo construiu um portal educacional com o objetivo de organizar as informações de diversas publicações e projetos e melhorar a interação com o público interessado na área de pesquisa de Engenharia de Software. O portal armazena informações sobre toda a estrutura pessoal da instituição, como professores, pesquisadores, alunos, etc., e os trabalhos realizados por essas pessoas.

Ressaltamos que o objetivo maior deste capítulo é mostrar a incorporação da *Web* Semântica em uma aplicação real. Sendo assim, utilizamos este portal para aplicar o método S-FrameWeb, discutido no capítulo 3, mostrando desde os modelos ontológicos criados para esta *WebApp* até a implementação da geração do documento OWL que representa conceitos da ontologia.

No decorrer deste capítulo detalhamos toda a construção do Portal do LabES, mostrando análise, projeto e implementação, usando o método S-FrameWeb. Assim, mostramos os modelos sugeridos pelo método que provêem um melhor entendimento dessa *WebApp*. Na seção 4.1, discutimos quais são os objetivos e funcionalidades do Portal do LabES, ambientando o leitor com os requisitos e explicitando os casos de uso dessa aplicação. Seguindo no desenvolvimento da aplicação, mostramos a Análise de Domínio, a Elicitação e Análise de Requisitos e o Projeto nas seções 4.2, 4.3 e 4.4 respectivamente. Por fim, mostramos detalhes da implementação da ontologia dessa aplicação na seção 4.5.

4.1. Entendendo o Domínio

Com o objetivo de oferecer uma maior interação com a comunidade em geral, o LabES desenvolveu um portal educacional para disponibilizar informações e materiais desenvolvidos na área de Engenharia de Software. A idéia é criar um grande acervo abrangendo esta área que vem ganhando cada vez mais relevância para a construção das mais diversas aplicações.

O Portal conta com o cadastro de usuários que terão acesso às informações contidas no banco de dados da aplicação. De um usuário é importante saber o nome, data de nascimento, sexo, instituição, profissão, endereço, telefone de contato, e-mail, login, senha, escolaridade, tipo e áreas

de interesse. De acordo com o tipo dele, será permitido o acesso às funcionalidades do sistema. As áreas podem estar organizadas em uma hierarquia, de maneira que uma área possa ter super áreas e sub áreas.

Como já foi dito, o portal disponibilizará várias informações e materiais que chamamos de itens. Estes incluem informações sobre projetos, publicações e materiais. Esses dois últimos itens podem ou não estar associados a um projeto, que, por sua vez, pode possuir vários materiais e publicações.

Um projeto também pode possuir sub-projetos. Um item está associado a uma ou mais áreas de interesse de pesquisa, que podem possuir sub-áreas. Uma publicação pode ser de vários tipos, incluindo: projeto de graduação, dissertação de mestrado, tese de doutorado, livro ou capítulo de livro, artigo publicado em evento e artigo publicado em periódico. Um capítulo de livro está sempre associado a um livro, que, por sua vez, está associado a uma editora. Um artigo em evento está sempre associado a um evento. Há, ainda, a necessidade de se definir um serviço de busca de itens.

Modelo de Casos de Uso

O gerenciamento e o acesso ao Portal do LabES são feitos de acordo com o tipo de usuário. Para tal, os usuários são agrupados nas seguintes categorias: Internauta, Usuário Padrão, Membro LabES, Colaborador, Professor e Administrador, como mostrado na Figura 18.

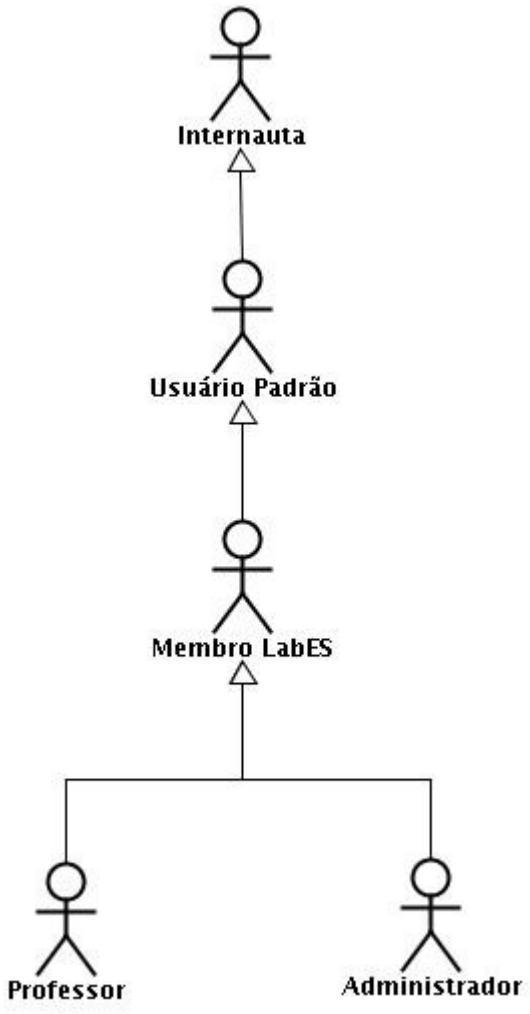

Figura 18: Tipos de Usuário do Portal do LabES

Os tipos de usuários estão dispostos segundo as suas funcionalidades da seguinte forma:

- Internauta: representa qualquer pessoa que esteja navegando na Internet. Esse tipo de usuário pode se cadastrar como um Usuário Padrão, tem acesso às funcionalidades de visualização de qualquer material ou informação disponível, pode efetuar *download* de publicações e materiais e pode utilizar o mecanismo de busca do portal;
- Usuário Padrão: representa usuários cadastrados no LabES. Esse tipo de usuário poderá alterar seus dados e sua senha e terá de se autenticar no sistema para ter acesso às funcionalidades específicas dessa classe de usuário.
- Membro LabES: agrupa todos os usuários que atuam dentro do laboratório. Esse tipo de usuário pode consultar os dados de qualquer outro membro do LabES e pode registrar materiais a serem disponibilizados no portal;
- Professor: representa os professores associados ao LabES. Esse tipo de usuário

pode disponibilizar publicações e projetos, cadastrar áreas de pesquisa, além de poder consultar os dados dos membros do LabES e registrar materiais, como qualquer membro do LabES;

- Administrador: representa os administradores do sistema, que têm permissão para cadastrar novas funcionalidades, tipo de usuário, membros do LabES, professores e outros administradores. De fato, um administrador tem acesso a todas as funcionalidades do sistema.

Deve-se observar que essa descrição é uma contextualização do sistema e, portanto, os dados de cada um dos tipos de usuários a serem cadastrados são apresentados nas descrições dos casos de uso.

A figura 19 mostra o diagrama de pacotes do sistema, subdividindo-o em dois subsistemas, a saber:

- **Controle de Usuário**: envolve toda a funcionalidade relacionada com o controle de usuários do Portal LabES, abrangendo controle de funcionalidades, tipos de usuários e usuários.
- **Controle de Itens**: disponibiliza todas as funcionalidades do sistema aos seus usuários, abrangendo todo o controle de publicação, material, projeto e área.

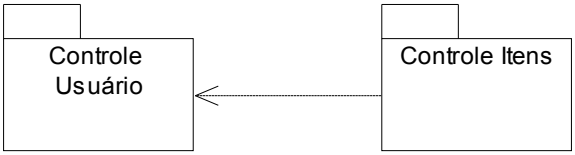

Figura 19 - Diagrama de Pacotes

Assim mostramos abaixo os diagramas de caso de uso desses pacotes nas Figuras 20 e 21.

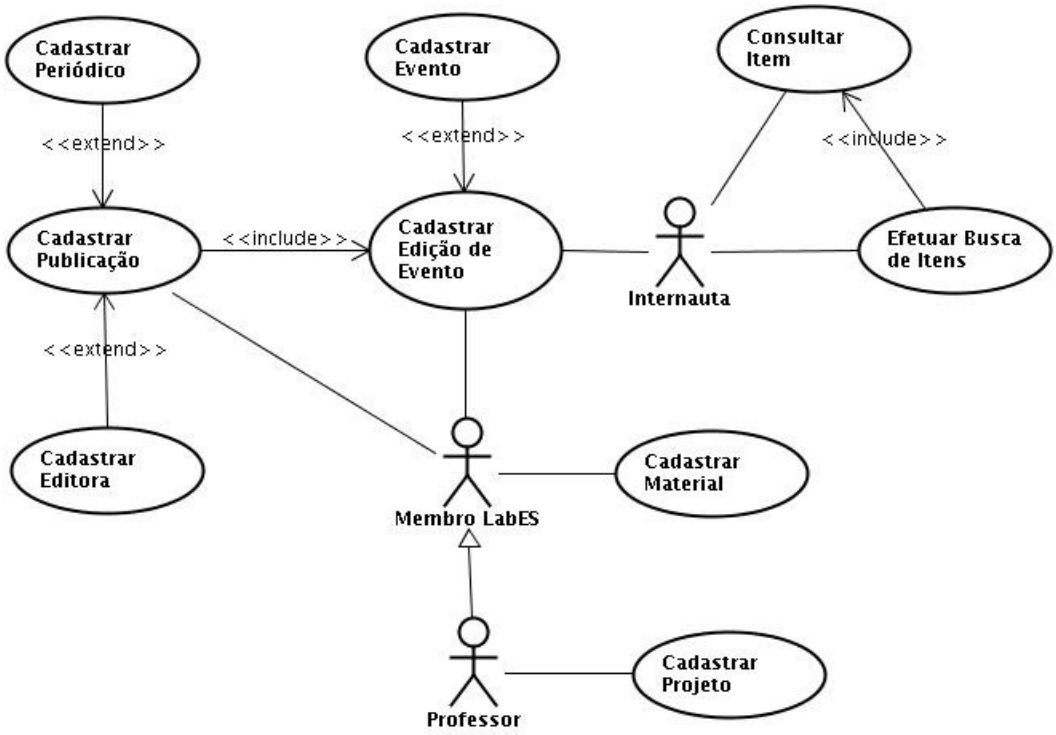

Figura 20: Diagrama de casos de uso do pacote Controle de Itens

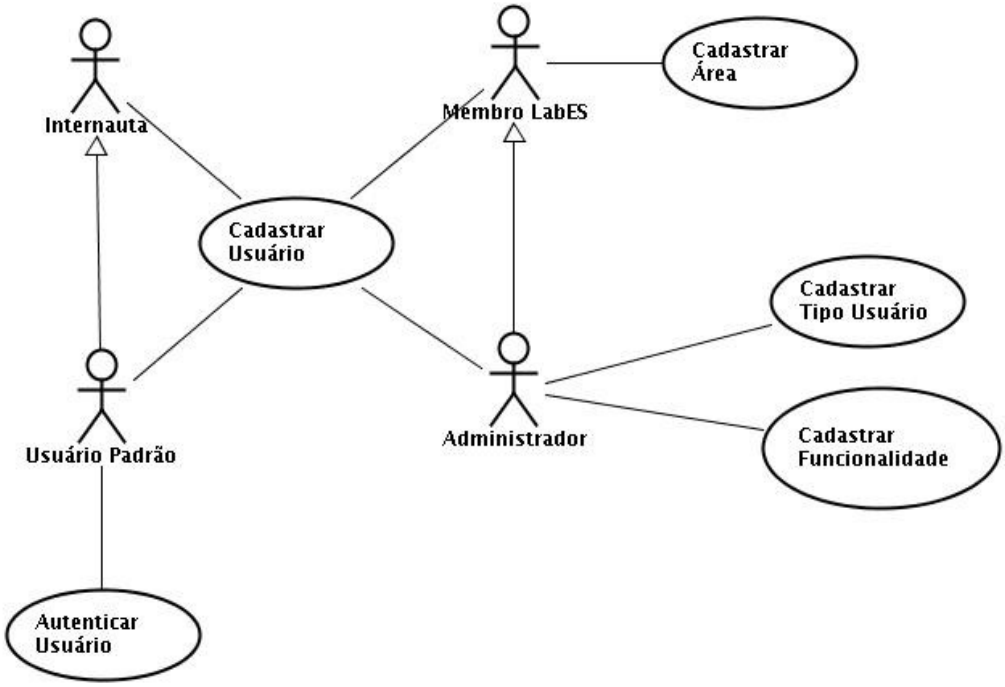

Figura 21: Diagrama de casos de uso para o pacote Controle de Usuários

4.2. Análise de Domínio

Como visto no diagrama de pacotes exibido na seção anterior, o Portal do LabES foi dividido

em dois subsistemas, seguindo esta idéia nós construímos dois modelos conceituais para representar o domínio no qual esta *WebApp* encontra-se inserida: um para representar a estrutura de portais educacionais e outra para abordar os tipos de publicação acadêmica. A ontologia foi desenvolvida com o objetivo de guiar a construção do Portal do Laboratório de Engenharia de Software da UFES e procurou reutilizar conceitos da Ontologia de Organizações de Software (RUY, 2006) desenvolvida na UFES.

A Ontologia do Portal do LabES deve tratar das seguintes questões de competência:

1. Quais as pessoas relacionadas com a instituição em questão?
2. Que papéis essas pessoas desempenham na instituição?
3. Quais as áreas de pesquisa de interesse da organização?
4. Como essas áreas estão organizadas?
5. Quais as áreas de interesse das pessoas da organização?
6. Como as pessoas estão organizadas?
7. Quais os projetos em desenvolvimento na instituição?
8. Em que áreas esses projetos se enquadram?
9. Quais as instituições e grupos envolvidos nos projetos?
10. Quais as publicações produzidas?
11. Quais tipos de publicações produzidas?

A partir dessas questões de competência, nós desenvolvemos a ontologia do Portal que foi dividida em duas sub-ontologias com a finalidade de simplificar o entendimento do sistema. A Figura 22 mostra a ontologia que trata da estrutura de um portal educacional:

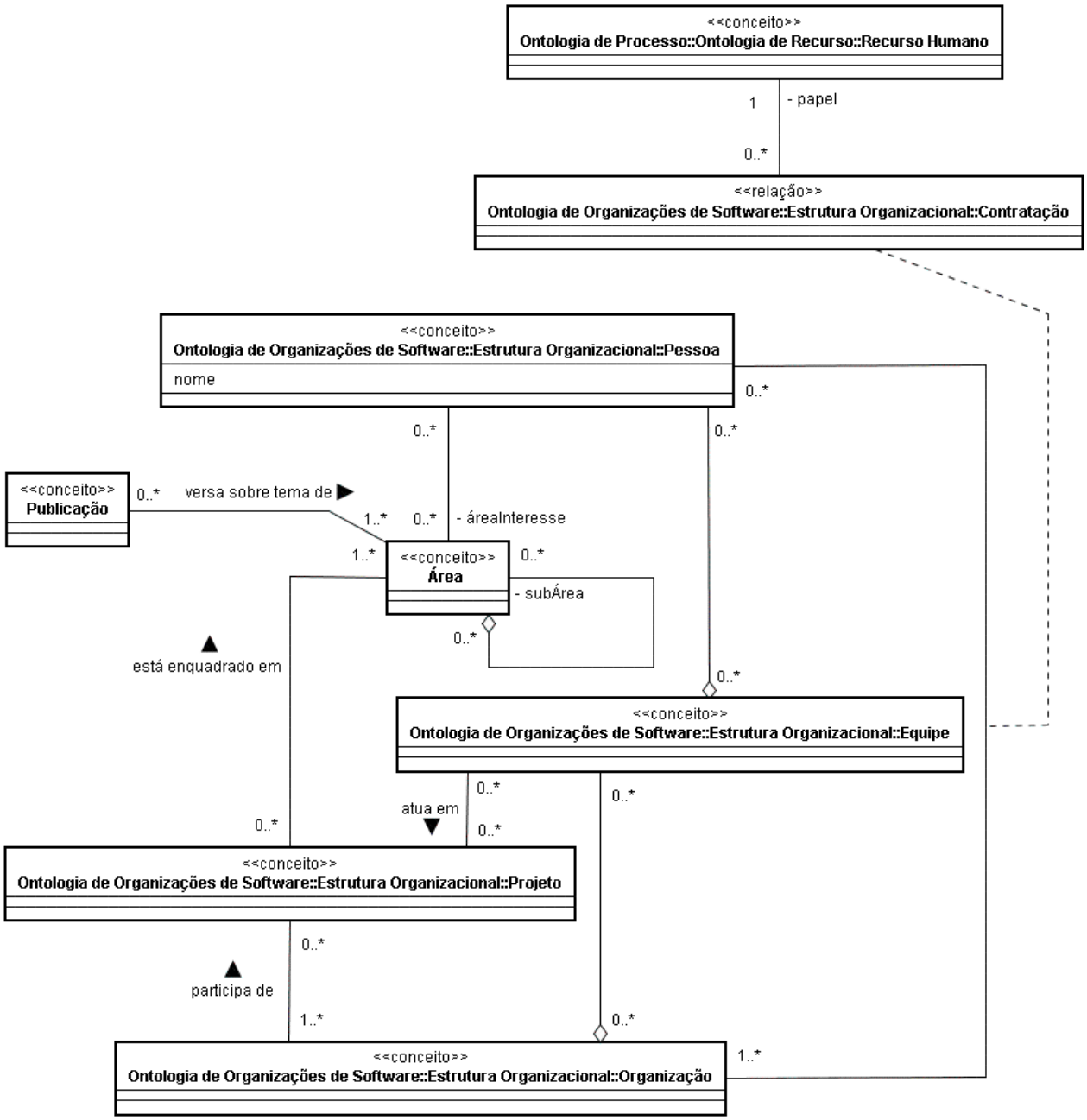

Figura 22: Modelo da sub-ontologia de Estrutura.

Como mostra a Figura 22, pessoas estão vinculadas a organizações (relação *Contratação*), exercendo papéis (*Recurso Humano*) tais como professor, pesquisador, aluno etc (QC1 e QC2). Como é possível notar, essa conceituação foi reutilizada da Ontologia de Organizações de Software proposta em (RUY, 2006) e cabem alguns comentários:

- No contexto de portais educacionais, organizações são geralmente referenciadas por instituições e, portanto, *Instituição* é um termo sinônimo a *Organização* nesta ontologia.
- De maneira análoga, normalmente diz-se que pessoas estão vinculadas a instituições. Assim, o termo *Vínculo* é um termo sinônimo de *Contratação* nessa ontologia.

Uma instituição atua em diversas áreas (conceito *Área*) que podem ser sub-divididas em sub-

áreas (QC3 e QC4).

Pessoas têm áreas de interesse e se organizam em grupos de interesse (QC5 e QC6). Tendo em vista que para tratar essas questões de competência foi reutilizada a conceituação correspondente da Ontologia de Organizações de Software (RUY, 2006), vale ressaltar que o conceito de *Equipe* é melhor tratado por *Grupo de Interesse* e, portanto, este último deve ser visto como um sinônimo do primeiro.

Instituições e *Grupos de Interesse* estão envolvidos em *Projetos*, que são enquadrados em áreas de pesquisa (QC7, QC8 e QC9).

Por fim, publicações (conceito *Publicação*) são produzidas, versando sobre temas relacionados às áreas de pesquisa.

A Figura 23 mostra o modelo da porção da ontologia que trata de tipos de publicações, mostrando uma taxonomia típica de publicaçõese conceitos relacionados (QC11).

Como mostra essa figura, publicações são produzidas por pessoas (os autores da publicação), possuem um título, estão escritas em um idioma, foram publicadas em uma data e versam sobre temas de uma ou mais áreas. Há quatro tipos principais de publicações : artigos, livros, capítulos de livros e trabalhos acadêmicos.

Artigos podem apresentar trabalhos completos, resumos ou pôsteres. A propriedade *indicador de completeza* indica qual a natureza de um artigo quanto a esse aspecto. Artigos podem ser publicados em edições de eventos, tais como conferências, simpósios etc, ou podem ser publicados em periódicos.

Capítulos de livros fazem parte de livros, que por sua vez são publicados por editoras.

Por fim, trabalhos acadêmicos podem ser de dois tipos: relatórios técnicos e trabalhos de conclusão de curso. Uma tese é um trabalho de conclusão de doutorado. Uma dissertação é um trabalho de conclusão de mestrado. Já uma monografia é um trabalho de conclusão de graduação ou pós-graduação lato-senso. Todos os trabalhos de conclusão de curso têm pelo menos um orientador e foram produzidos no contexto de uma instituição.

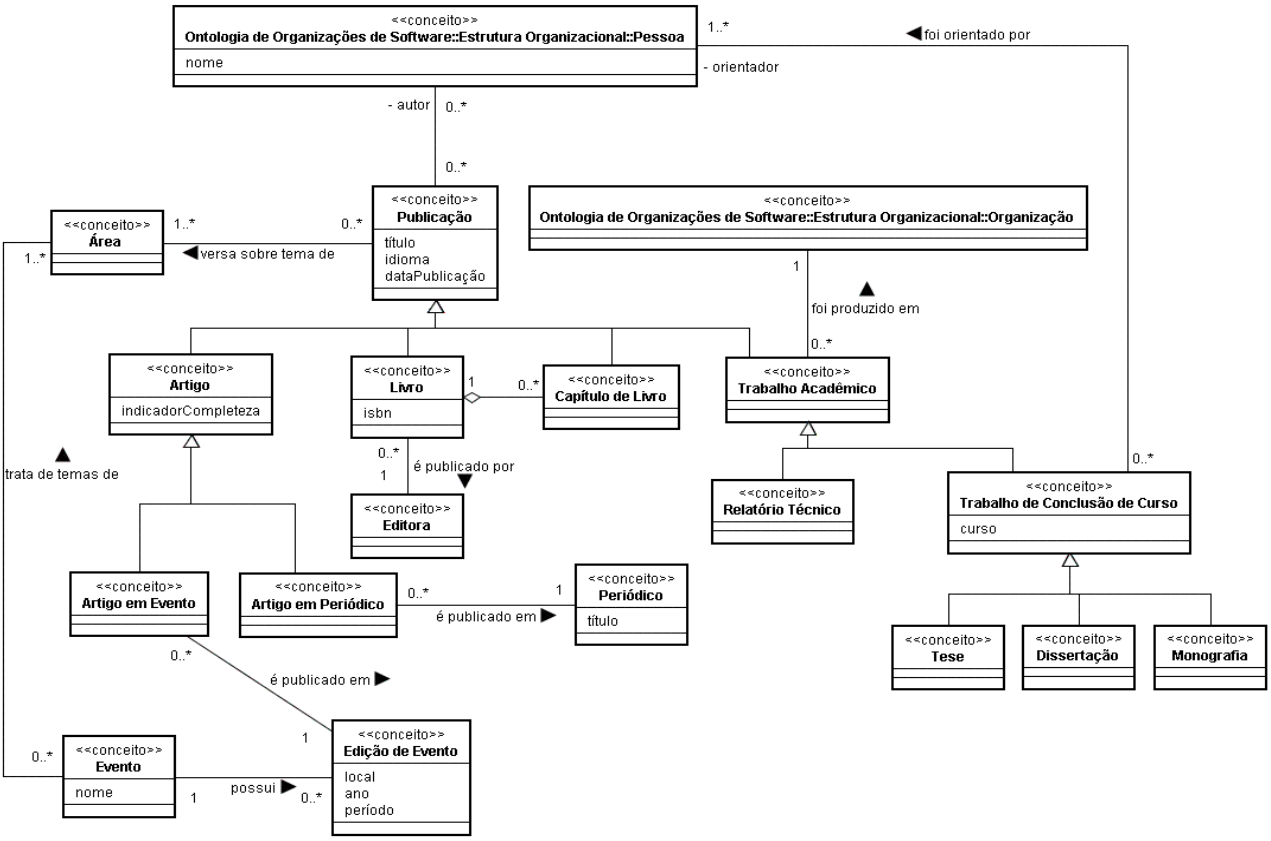

Figura 23: Modelo da sub-ontologia de Publicações

A partir da ontologia apresentada, nós fizemos uma representação em diagramática usando o Metamodelo de Definição de Ontologia (ODM), conforme sugerido pelo método *S-FrameWeb* para representar a ontologia.

4.3. Especificação e Análise de Requisitos

Construímos o modelo conceitual do Portal do LabES em ODM, tal como fizemos no capítulo anterior com o Cookbook. Podemos notar que, desta vez, utilizamos atributos simplificados de modo a ter uma melhor visualização do modelos. A Figura 24 mostra o modelo desenvolvido para o módulo controle de usuários. Esse modelo é baseado na ontologia e as necessidades já explicitadas anteriormente.

De um usuário é importante saber seu nome, profissão, instituição, endereço, telefone de contato, uma senha de acesso ao sistema, o e-mail, o login, o sexo, a data de nascimento. Além disso, devemos saber qual o nível de escolaridade do usuário. Um Usuário terá Áreas de Interesse vinculadas a ele, tal como descreve a sub-ontologia da estrutura. De uma área é importante saber

seu nome e sua descrição. Além disso, o modelo abrange a necessidade da hierarquia de usuários, sendo que um Usuário deve ser de um Tipo que determinará as Funcionalidades que o Usuário estará apto a realizar. De um tipo de usuário e de uma funcionalidade é importante saber o nome e a descrição.

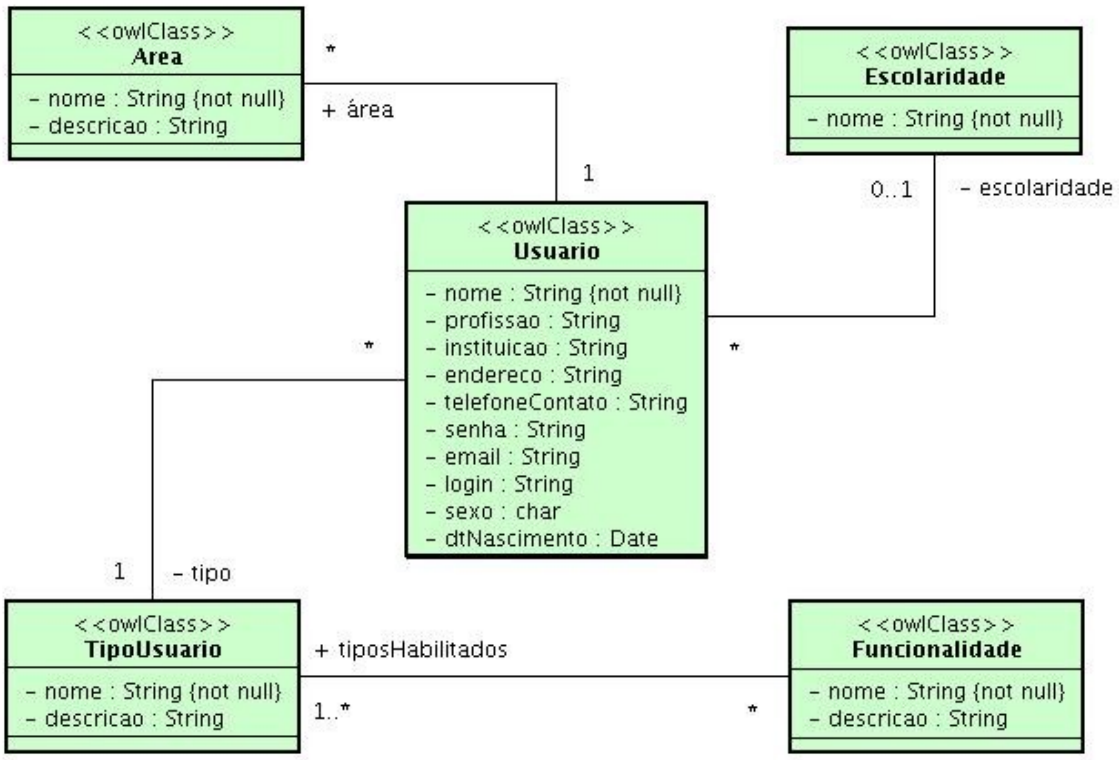

Figura 24: Modelo conceitual do Portal do LabES (módulo de controle de usuários)

A partir da sub-ontologia de Publicações, desenvolvemos também o modelo conceitual para o módulo de controle de itens. Deve ficar claro que as sub-ontologias tiveram que ser adaptadas para o contexto específico da *WebApp*.

O Usuário do módulo de controle de usuários é incluído neste modelo podendo ser responsável por Itens e/ou participante de Projetos. Um Item possui Áreas de Interesse e deve pertencer a um Projeto. Uma Publicação é uma subclasse de um Item e deve conter um Idioma e um Arquivo. Uma Publicação pode ser um Artigo que por sua vez pode ser um Artigo de Periódico devendo pertencer a um Periódico, ou um Artigo de Evento devendo pertencer a uma Edição de Evento. Uma publicação pode ser também de um Livro ou Capítulo de Livro que pertence a uma determinada Editora. Por fim uma Publicação pode ser um Trabalho Acadêmico que foi feito em uma Instituição e é de determinado Tipo. Outra subclasse de Item é Material que tem um Arquivo Documento.

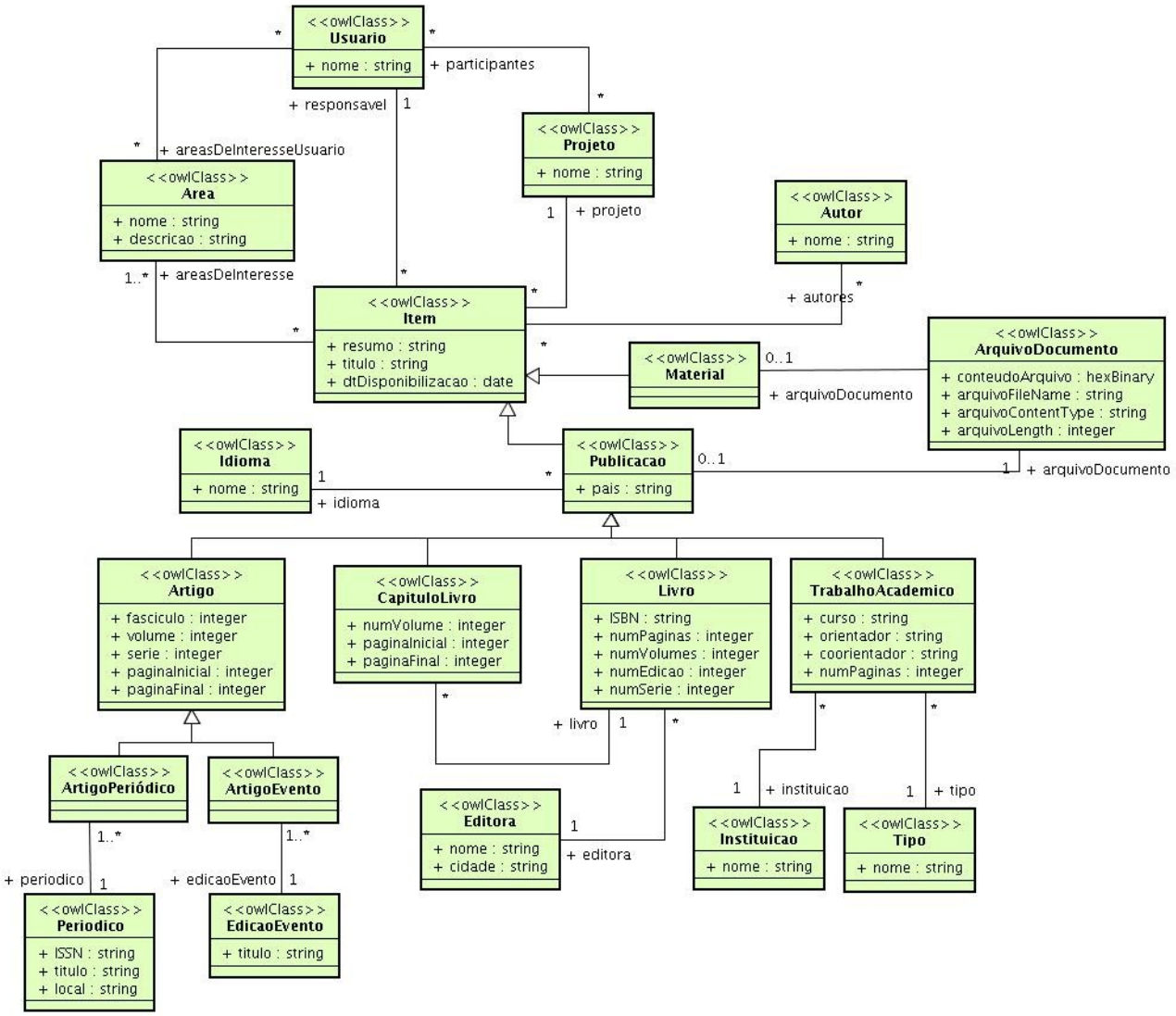

Figura 25: Modelo Conceitual do Portal do LabES (módulo de controle de itens)

Nesses dois modelos podemos perceber que não há mais classes com estereótipos <<owlDataTypeProperty>>, <<owlDataRange>> e nem mesmo <<owlObjectProperty>> como havia nos modelos conceituais do CookBook. Assim, a fim de simplificarmos o modelo, colocamos as classes com estereótipo <<owlDataTypeProperty>> como atributos de nossas classes que possuem estereótipo owlClass e as classes com estereótipo <<owlDataRange>> tornaram-se o tipo destes atributos. Percebemos que a não simplificação deste modelo resultaria em uma alta poluição visual. Além disso, não perdemos informações importantes ao fazermos esta simplificação.

Estes modelos serão refinados na próxima fase, a fase de Projetos que abordaremos na

próxima seção. Deste modo, são incluídos novos elementos ao modelo a fim de representar aspectos da arquitetura, específicas da plataforma de implementação.

4.4. Projeto

Na fase de projeto o método que estamos seguindo sugere alguns modelos, a saber: Modelo de Domínio, Modelo de Persistência, Modelo de Navegação e Modelo de Aplicação. Mostramos alguns destes modelos no capítulo anterior quando abordamos a fase de projeto do Cookbook. PERUCH (2007) aborda esses modelos para o Portal do LabES. No entanto, neste trabalho daremos um maior foco no modelo de domínio, pois S-FrameWeb sugere um novo perfil UML para esse modelo. Esse perfil é uma mistura do perfil definido por ODM com o perfil definido por FrameWeb.

Na Figura 26, mostramos o Modelo de Domínio de S-FrameWeb para o módulo do controle de usuários. Podemos notar os elementos sugeridos pelo método e que foram inseridos nesse modelo (SOUZA, 2007):

- Inclusão da navegabilidade das associações;
- Adição dos mapeamentos Objeto/Relacionais para configuração do framework ORM;
- Utilização dos tipos de dados da plataforma de implementação.

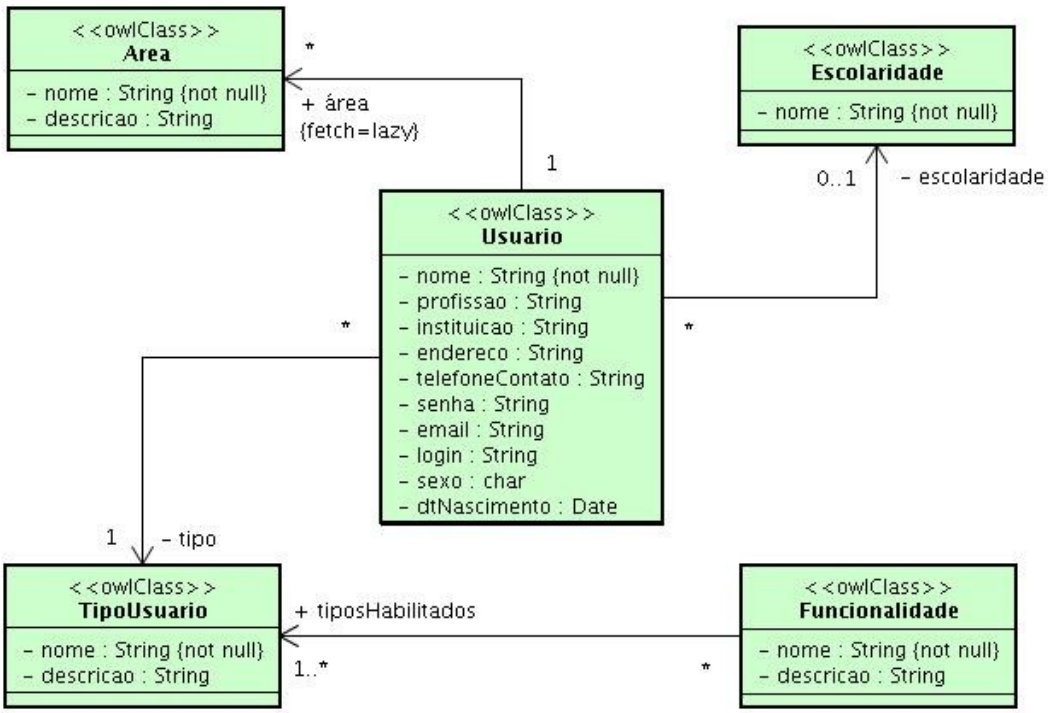

Figura 26: Modelo de Domínio de S-FrameWeb para o Portal do LabES (módulo de controle de usuários).

A Figura 27 mostra o Modelo de Domínio de S-FrameWeb para o módulo de Controle de Itens. Nesse modelo notamos mais claramente a mudança de tipos de dados que foi adequado à plataforma de implementação. No modelo conceitual os tipos que podem ser usados são somente os tipos definido pelo padrão XSD (XML Schema Definition), por exemplo na classe ArquivoDocumento do modelo conceitual colocamos o atributo conteudoArquivo com tipo hexBinary. Agora, neste modelo podemos definir este atributo como sendo byte[], que é um tipo específico da linguagem Java, utilizada na implementação do portal.

Em algumas classes observamos também a utilização de notações sugeridas pelo método. É o caso de classes como Item que possuem o estereótipo <<mapped>> e do atributo nome presente em algumas classes que não permitem valores nulo, o que é denotado pelo texto {not null} após o tipo deste atributo.

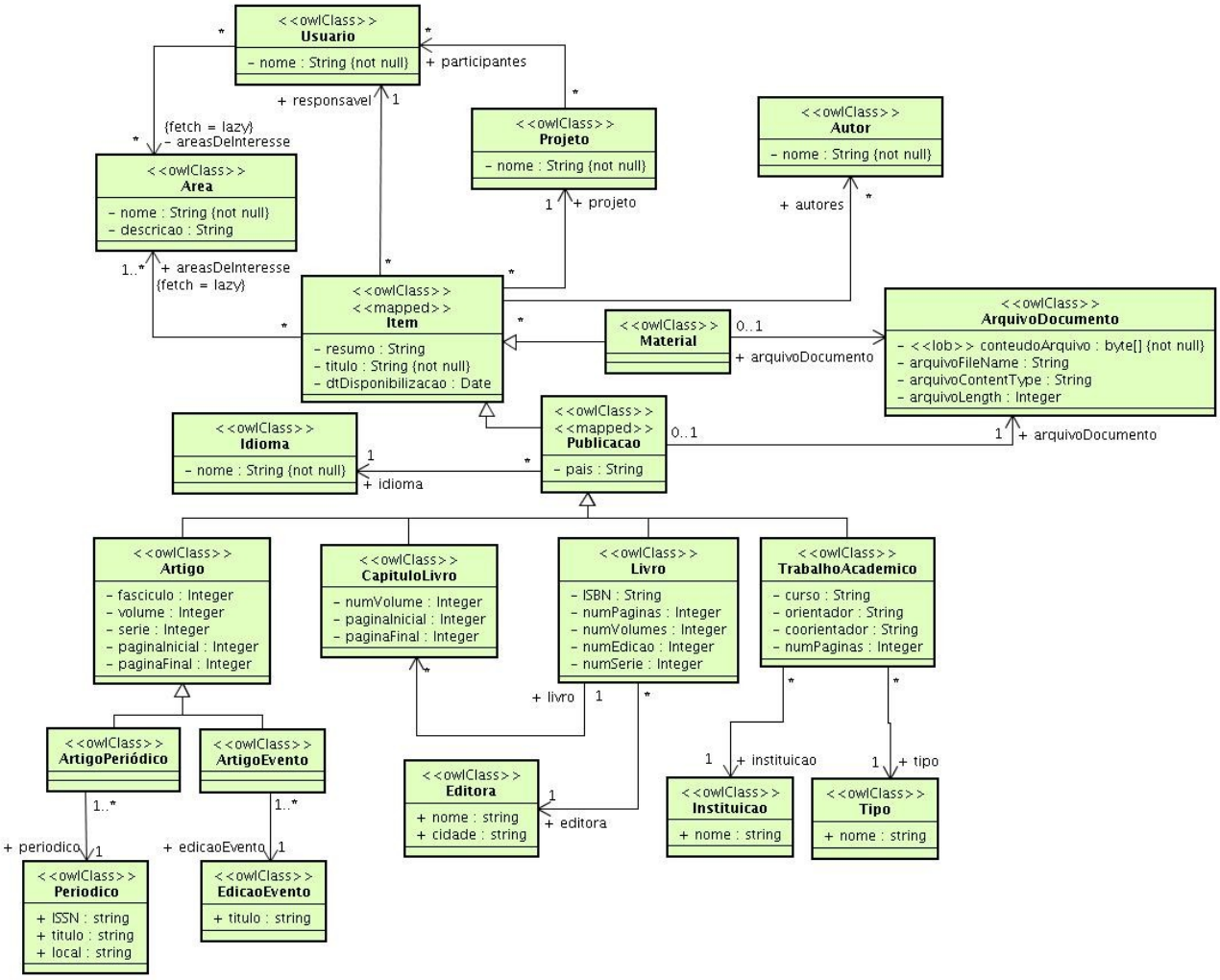

Figura 27: Modelo de Domínio de S-FrameWeb para o Portal do LabES (módulo controle de itens).

4.5. Implementação

S-FrameWeb adicionou a esta fase a codificação da ontologia de domínio e do modelo conceitual específico da aplicação em OWL. Existem diversas ferramentas que auxiliam a construção destas representações do domínio, por exemplo OilEd (BECHHOFER, 2001) e Protégé (KNUBLAUCH, 2004). Optamos por utilizar o Protégé para gerar nosso arquivo que representa o nosso modelo conceitual em OWL.

O Protégé é um editor que suporta a OWL, sendo capaz de carregar e salvar representações de domínios OWL e RDF, editar e visualizar classes, propriedades, entre outras coisas. Na Figura 28,

podemos observar a interface do Protégé. Na aba mostrada na figura, nós configuramos as classes que compõem a nossa ontologia. Do lado esquerdo estão as classes ontológicas, colocadas hierarquicamente tal como definido pelos modelos apresentados anteriormente. Para cada classe, verificamos quais condições devem ser satisfeitas e as incluímos no quadro central (Condições Declaradas). Por exemplo, para um Item nós temos uma condição que é ter um responsável que é um Objeto da classe Usuário.

As propriedades dos objetos são definidas na aba de Propriedades. Nela, definimos as Propriedades do Objeto, o seu Domínio e a sua Imagem de acordo com os modelos. Um exemplo já citado é a Propriedade responsável, que tem como Domínio a classe Usuário e como Imagem a classe Item. Nesta aba também são definidas as Propriedades dos Tipos de Dados, que nos nossos modelos passaram a ser atributos nas classes da ontologia.

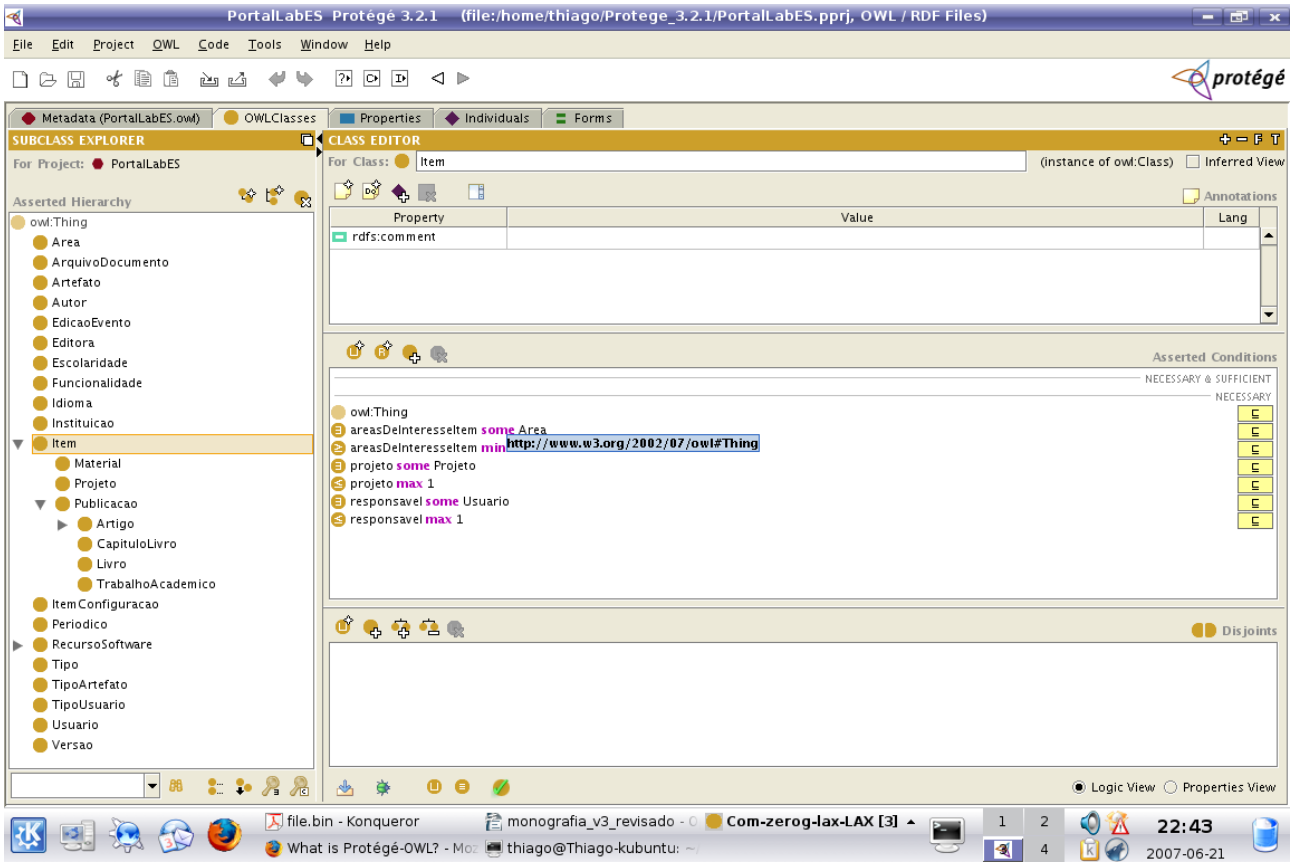

Figura 28: Interface do Protégé

Esta representação da ontologia é convertida pelo Protégé em um arquivo OWL. No Quadro 1 podemos visualizar um trecho do arquivo no formato OWL do Portal do LabES que foi gerado pelo Protégé. Note na linha 1 do quadro a definição da classe Item que foi mostrada na Figura anterior. Observe as condições que estabelecemos. Por exemplo, na linha 7 é definido que essa classe deve ter um responsável e que sua cardinalidade máxima, mostrada na linha 5, é um. Na linha 16, é

definida uma outra propriedade que é o projeto e que possui cardinalidade um, conforme definido na linha 14.

Enfim, é notório que o Protégé gera esse arquivo a partir das classes e propriedades que definimos na sua interface de um modo bem estruturado e organizado. Sendo assim, é possível explorar esse arquivo de forma a obter informações e fazer inferências sobre o domínio da aplicação a partir destas informações.

A construção da ontologia no Protégé foi extremamente fácil e intuitiva a partir dos modelos que construímos. Visto isto, observamos que um trabalho futuro é a construção de um gerador de arquivo OWL a partir dos modelos construídos, o que poderá agilizar o processo da fase de implementação da ontologia.

```
1 <owl:Class rdf:about="#Item">
2  <rdfs:subClassOf>
3    <owl:Restriction>
4      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
5      >1</owl:maxCardinality>
6      <owl:onProperty>
7        <owl:FunctionalProperty rdf:ID="responsavel"/>
8      </owl:onProperty>
9    </owl:Restriction>
10  </rdfs:subClassOf>
11  <rdfs:subClassOf>
12    <owl:Restriction>
13      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
14      >1</owl:maxCardinality>
15      <owl:onProperty>
16        <owl:ObjectProperty rdf:ID="projeto"/>
17      </owl:onProperty>
18    </owl:Restriction>
19  </rdfs:subClassOf>
20  <rdfs:subClassOf>
21    <owl:Restriction>
22      <owl:onProperty>
23        <owl:FunctionalProperty rdf:about="#responsavel"/>
24      </owl:onProperty>
25      <owl:someValuesFrom>
26        <owl:Class rdf:ID="Usuario"/>
27      </owl:someValuesFrom>
28    </owl:Restriction>
29  </rdfs:subClassOf>
```

Quadro 1: Arquivo OWL gerado pelo Protegé.

Para explorar as informações contidas na nossa ontologia que agora está codificada em OWL, nós utilizamos a API do Jena, como já citado e mostrado no capítulo anterior. O algoritmo utilizado no Cookbook para recuperação dos indivíduos, integrando o Jena com a aplicação, é genérico. Assim, foi utilizado o mesmo algoritmo no Portal do LabES, sendo necessária apenas a modificação do arquivo OWL para o arquivo que nós geramos para esta aplicação.

Na Figura 29 é mostrada a tela de Busca de Itens do Portal do LabES. Aplicamos a *Web* Semântica nesta funcionalidade. Como explicado no capítulo anterior, nós criamos um interceptador que verifica a existência do parâmetro OWL na requisição feita ao Controlador Frontal. Caso haja esse parâmetro, a *WebApp* retorna um resultado processável por máquinas, caso

contrário retorna um resultado para humanos.

Essa tela apresenta um filtro pelo qual faremos a nossa busca. Além disso, é possível escolher o tipo de item pelo qual desejamos buscar. De maneira experimental, inserimos um *checkbox* chamado OWL, que explicamos mais tarde.

Como visto, utilizamos como Filtro a palavra "FrameWeb", assim o sistema devolverá os itens que contiverem esta palavra em seu título. Na Figura 30 é exibido o resultado dessa busca e como podemos notar o sistema nos retornou o Item "Aplicação do S-FrameWeb". Ao clicarmos no ícone do lado direito do nosso Item, podemos ver mais detalhes sobre o nosso Item. Esses detalhes são mostrados na Figura 31.

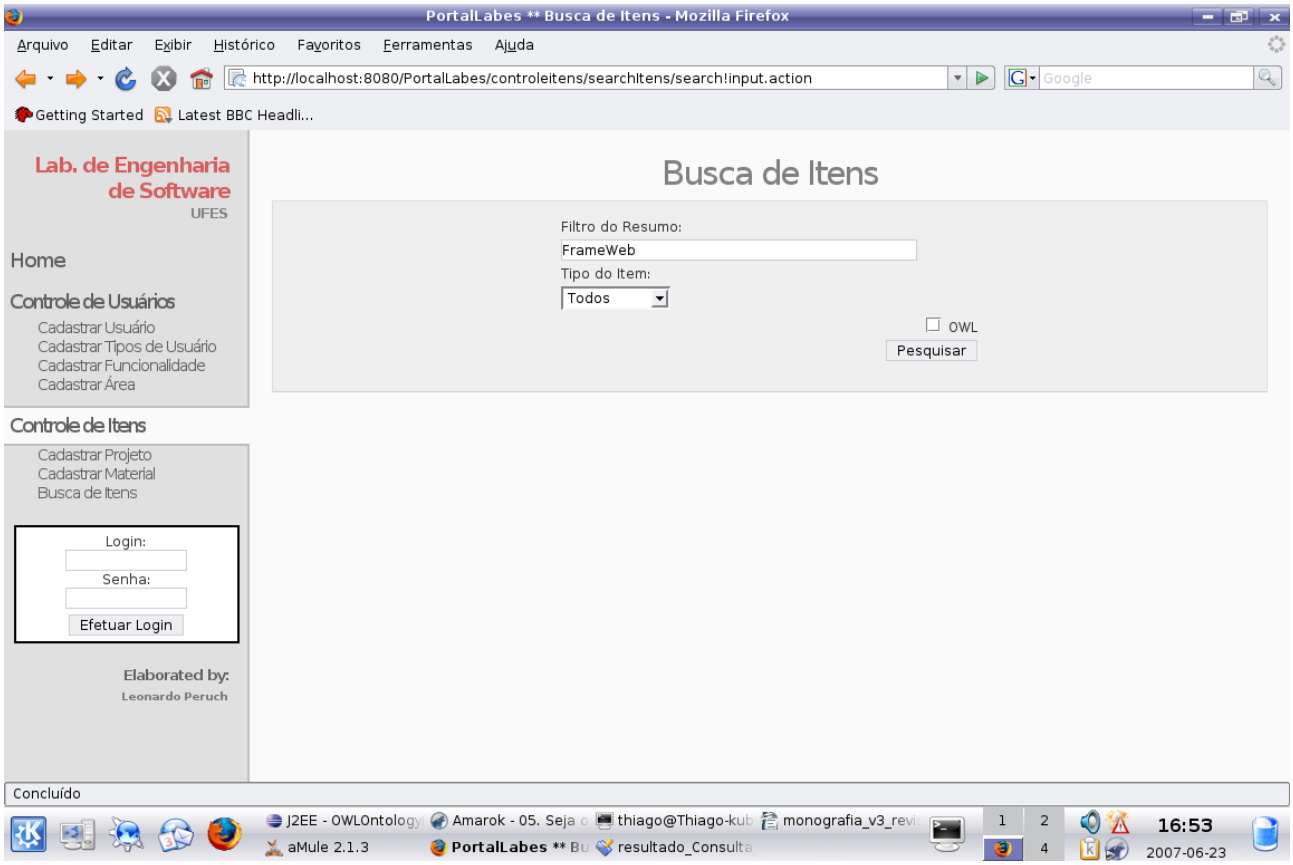

Figura 29: Tela de Busca de Itens do Portal do LabES

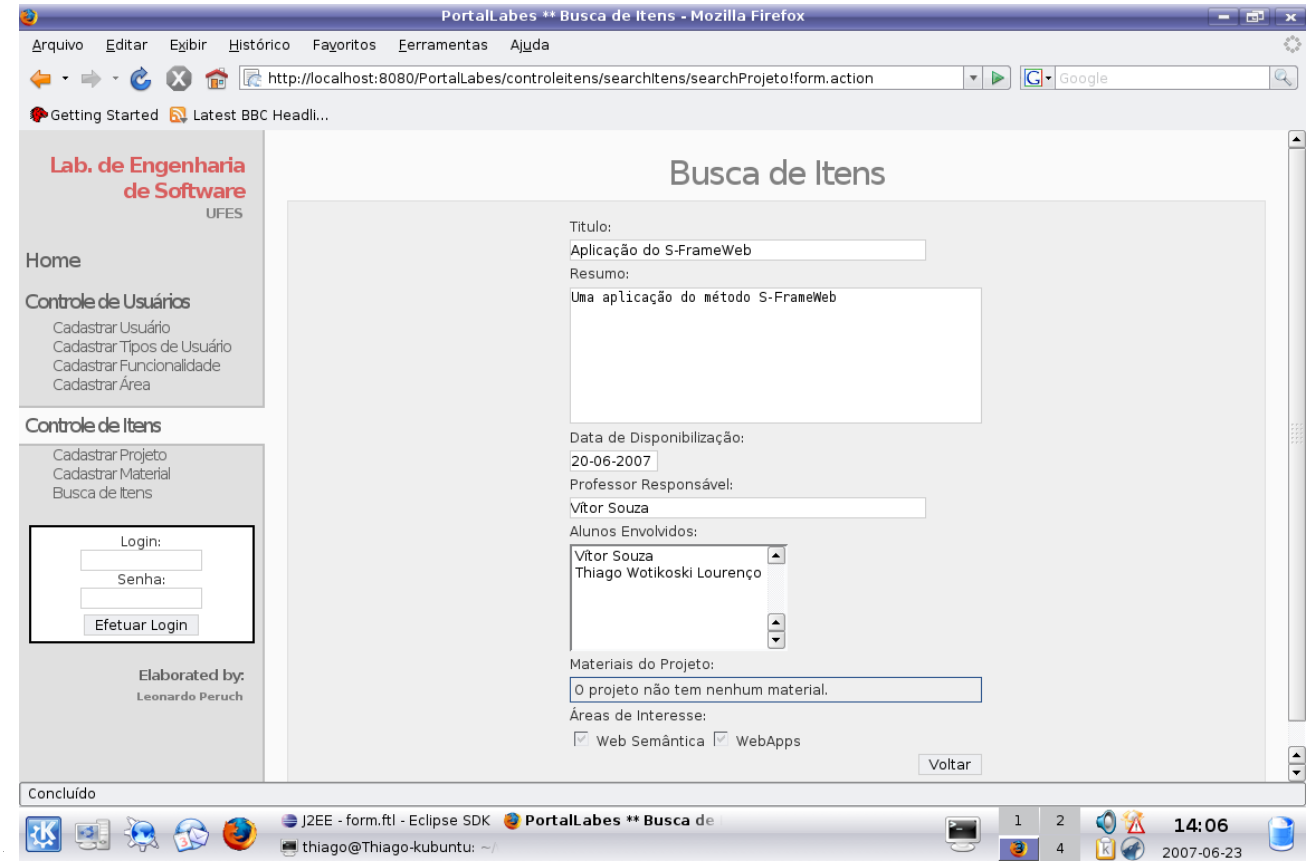

Figura 30: Detalhes da tela busca de itens do Portal do Labes

O *checkbox* OWL inserido na tela de Busca de Itens tem a função de incluir o parâmetro OWL na requisição feita ao Controlador Frontal. Ao marcarmos esse *checkbox* e realizarmos a busca com os mesmos parâmetros, obteremos agora um resultado bem organizado e estruturado de forma que um agente inteligente possa processá-lo e fazer inferências a partir das informações obtidas.

No Quadro 2 é mostrado este resultado. Podemos observar o resultado da nossa busca entre os delimitadores <result> e </result>. Dentro desses delimitadores temos a instância principal resultante da consulta entre os delimitadores <instance> e </instance>, este está sendo indicado pelo seu identificador único universal (UUID). O mapeamento entre o UUID e a referência da instância se encontra na lista de instâncias, que fica entre os delimitadores <instanceList> e </instanceList>. Cada objeto recuperado no método recursivo que utilizamos está referenciado nesta listagem. Temos ainda no nosso resultado, todas as propriedades inerentes ao objeto que foram recuperadas, por exemplo, áreas de interesse, responsável, participantes etc.

```
<result>
<instance>2a6304f5-34c9-4356-a1ce-baa1e7b99e04</instance>
<areasDeInteresseItem>130c2f70-5a37-4ad3-815b-841922584cd9</areasDeInteresseItem>
<areasDeInteresseItem>93fcbf36-cdfe-4fd3-be23-a0d6ab3b45e8</areasDeInteresseItem>
<dtDisponibilizacao>2007-06-20</dtDisponibilizacao>
<resumo>Uma aplicação do método S-FrameWeb</resumo>
<participantes>e5242491-b2be-4a34-b7b4-b0d9b7537517</participantes>
<dtNascimento>1979-06-05</dtNascimento>
<sexo>M</sexo>
<profissao>Professor</profissao>
<instituicaoUsuario>UFES</instituicaoUsuario>
<tipo>2e9c5b6e-0d0f-4da0-b99b-24178ca6873a</tipo>
<descricao>Professor de ensino superior</descricao>
<senha>vitor</senha>
<endereco>Rua da casa dele</endereco>
<telefoneContato>3333333</telefoneContato>
<email>vitor@teste.com.br</email>
<login>vitor</login>
<participantes>52796989-8658-4dec-ae6e-6af639747c1e</participantes>
<dtNascimento>2007-06-12</dtNascimento>
<sexo>M</sexo>
<profissao>Estudante</profissao>
<instituicaoUsuario>UFES</instituicaoUsuario>
<tipo>15b05e5a-5e82-4e60-8858-caa76829a1af</tipo>
<descricao>Aluno de Ensino Superior</descricao>
<senha>thiago</senha>
<endereco>Rua das Palmeiras</endereco>
<telefoneContato>33395752</telefoneContato>
<email>thiago@teste.com.br</email>
<login>thiago</login>
<responsavel>e5242491-b2be-4a34-b7b4-b0d9b7537517</responsavel>
</result>

<instancesList>
<instance>
<name>Professor Universitário</name>
<uuid>2e9c5b6e-0d0f-4da0-b99b-24178ca6873a</uuid>
</instance>
<instance>
<name>Vítor Souza</name>
<uuid>e5242491-b2be-4a34-b7b4-b0d9b7537517</uuid>
</instance>
<instance>
<name>Web Semântica</name>
<uuid>130c2f70-5a37-4ad3-815b-841922584cd9</uuid>
</instance>
<instance>
<name>Aluno Universitário</name>
<uuid>15b05e5a-5e82-4e60-8858-caa76829a1af</uuid>
</instance>
<instance>
<name>Aplicação do S-FrameWeb</name>
<uuid>2a6304f5-34c9-4356-a1ce-baa1e7b99e04</uuid>
</instance>
<instance>
<name>WebApps</name>
<uuid>93fcbf36-cdfe-4fd3-be23-a0d6ab3b45e8</uuid>
</instance>
<instance>
<name>Thiago Wotikoski Lourenço</name>
<uuid>52796989-8658-4dec-ae6e-6af639747c1e</uuid>
</instance>
</instancesList>
```

Quadro 2: Resultado processável por máquina da busca utilizando o Filtro S-FrameWeb

Neste capítulo, introduzimos de maneira simples a semântica em uma aplicação real. O maior trabalho foi necessário na construção da nossa ontologia em ODM e na implementação da ontologia a partir dos modelos desenvolvidos. Contudo, voltamos a lembrar que pode ser possível criar uma ferramenta capaz de gerar a ontologia no formato OWL a partir dos modelos construídos. Isto reduziria o tempo gasto nesta importante fase do método.

Capítulo 5

Considerações Finais

Neste capítulo são apresentadas as conclusões a respeito do projeto desenvolvido ao longo deste trabalho (seção 5.1) e as perspectivas de trabalhos futuros (seção 5.2).

5.1.Conclusões

O mundo está percebendo a necessidade de se ter um maior dinamismo e eficiência na busca de informações, mas acaba se perdendo na imensa quantidade de informações a qual está exposto. Devemos, então, criar formas que agilizem este processo de busca a informações úteis, por exemplo, deixando a cargo das máquinas fazerem este trabalho por nós.

Contudo, para isto precisamos fazer com que nossos documentos sejam organizados e estruturados de tal forma que as máquinas possam processá-los, tornando-se, assim, capazes de fazer inferências sobre eles. Ao conseguirmos inserir semântica nas nossas informações, seremos capazes de não só fazer buscas mais eficientes, mas automatizar diversas atividades do nosso cotidiano, como por exemplo, marcar uma consulta com um médico que precisamos e que está disponível em um horário que nossa agenda está sem compromissos e que atende em uma clínica localizada a 1 km da nossa casa.

Neste trabalho desenvolvemos aplicações utilizando a metodologia S-FrameWeb, conseguindo gerar anotações semânticas em páginas *Web* dinâmicas. A partir dos modelos criados, a saber, modelo conceitual e modelo do S-FrameWeb, conseguimos facilmente gerar nossa ontologia em OWL utilizando o software Protégé. Conseguimos explorar a ontologia utilizando a API do Jena e com as informações que obtivemos, usamos a característica de reflexão computacional de Java para recuperar instâncias que estavam persistentes, gerando, assim, um resultado bem estruturado das informações buscadas.

O nosso trabalho ficou restrito ao *framework* Struts[2], no qual fizemos uma extensão criando um novo interceptador. Sendo assim, não é uma solução genérica, visto que outros *frameworks* não possuem tal característica, ou seja, é necessário que, para outros *frameworks* que não utilizem o conceito de interceptadores, sejam criadas novas soluções. Além disso, percebemos que uma necessidade é a criação de uma maneira de especificar que tipo de resultado deve ser considerado

bem-sucedido, pois no nosso trabalho, o interceptador criado considera apenas o tipo de resultado SUCESSO como bem sucedido, dificultando a geração de resultados para máquinas nas outras situações, por exemplo, quando recebemos o resultado de uma listagem que retorna o tipo LISTA.

5.2. Trabalhos Futuros

Diversos trabalhos futuros foram percebidos durante o desenvolvimento deste trabalho, dentre eles:

- Devemos encontrar uma maneira para que os agentes de software saibam como encontrar as páginas *Web*. Atualmente, as máquinas de busca procuram por páginas que são ligadas por outras, mas este não é o caso quando estamos tratando da requisição para um serviço. Pesquisas na área de *Web Services* poderão oferecer um tratamento em relação a isto;
- Agentes devem ter uma linguagem comum para entender as páginas *Web*. Existem palavras que podem ter vários significados de acordo com o contexto no qual estão inseridas. Por exemplo, se uma instância de "monitor" é retornada para um agente, como ele saberá se está tratando de um dispositivo de saída do computador ou se é um aluno que ajuda no ensino, ou qualquer outra coisa?
- O protótipo da infra-estrutura foi desenvolvido somente para o *framework* Struts 2. Nosso trabalho foi desenvolvido com base nos interceptadores que estão presentes neste *framework.* Contudo há muitos outros *frameworks* para desenvolvimento *Web* que não possuem tal característica. Para estes, deve-se encontrar uma solução que se adeque às características do framework;
- Criação de uma ferramenta para gerar a ontologia em OWL a partir dos modelos contruídos em ODM. Uma tarefa que demanda um certo tempo que pode ser poupado é a construção da ontologia num ambiente capaz de gerar a ontologia no formato OWL. Contudo fomos capazes de observar durante o desenvolvimento deste trabalho que os modelos possuem informações suficientes para que uma ferramenta seja capaz de gerar a ontologia utilizando estes modelos;
- Inclusão, no protótipo da infra-estrutura, de uma maneira de especificar qual retorno da ação deve ser considerado como bem sucedido. Atualmente, o protótipo considera apenas o retorno SUCCESS. Isto dificulta a inclusão do tratamento semântico em outras situações,

que retornem outros tipos de resultados.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTONIOU, G.; van Harmelen, F.. **A Web Semantic Primer.** Cambridge, Massachussets. The MIT Press, 2004.

BAUER, C.; KING, G. **Hibernate em Ação**. 1. ed. Editora Ciência Moderna, ISBN 8573934042, 2005.

BECHHOFER,S; Horrocks, I; Goble C; Stevens R. **OilEd: a Reason-able Ontology Editor for the Semantic Web**. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

BERNERS-LEE, T. **Semantic Web – XML 2000**. Disponível em: <http://www.w3.org/2000/Talks/1206-xml2k-tbl/>. Acesso em: 13 abr. 2007.

BERNERS-LEE, T.; Hendler, J.; Lassila, O. **The Semantic Web. Scientific American**, n. 284 (maio), p. 34-43, 2001.

BERNERS-LEE, **WorldWideWeb: Proposal for a HyperText Project 1990.** Disponível em: <http://www.w3.org/Proposal.html>. Acesso em: 13 abr. 2007.

BREITMAN, K. **Web Semântica – A Internet do Futuro.** 1 ed. São Paulo: LTC, 2005.

BROEKSTRA, J.; KLEIN, M.; DECKER, S.; FENSEL, D.; VAN HARMELEN, F.; HORROCKS, I. **Enabling knowledge representation on the Web by Extending RDF Schema**. Proceedings of the Tenth International World Wide Web Conference (WWW10 – Hong Kong), maio 2001.

CERI, S.; FRATERNALI, P.; BONGIO, A. **Web Modeling Language (WebML): a modeling language for designing Web sites**. Computer Networks, v. 33, n. 1-6 (junho), p.

137-157, Elsevier, 2000.

CONALLEN, J. **Building Web Applications with UML**, 2nd ed. Addison-Wesley, ISBN 0201730383, outubro 2002.

CONNOLLY D.; van Harmelen, F.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; Stein, L. A. **"DAML+OIL (March 2001) Reference Description"**, December. 2001.

CONTE, T.; TRAVASSOS, G. H.; MENDES E. **Revisão Sistemática sobre Processos de Desenvolvimento para Aplicações Web**. Relatório Técnico ESE/PESC – COPPE/UFRJ,
2005.

DACONTA, M. C.; OBRST, L. J.; SMITH, K. B. **The Semantic Web**: A Guide to the Future of XML, Web Services, and Knowledge Management, 1st ed. Wiley, 2003.

DART, S. **Containing the Web Crisis Using Configuration Management**, Proc. First ICSE Workshop on Web Engineering, ACM, Los Angeles, 1999.

DAVIES, J.; Fensel, D.; Van Harmelen, F. **Towards The Semantic Web – Ontology-driven Knowledge Management***,* England. JohnWiley & Sons Ltd., 2003.

DICKINSON, I. **Jena 2 Ontology API 2007**. Disponível em: <http://jena.sourceforge.net/ontology/index.html>. Acesso em: 31 maio 2007.

FALBO, R. A., Guizzardi, G., Duarte, K. C. : **An Ontological Approach to Domain Engineering**. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002). Ischia, Italy (2002).

FENSEL, D.; Horrocks, I.; Van Harmelen, F.; Decker, S.; Erdmann, M.; Klein, M. **Oil in a Nutshell**. Knowledge Acquisition, Modeling, and Management,

Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), Juan-les-Pins, French Riviera, 2000.

FONS, J.; VALDERAS, P.; RUIZ, M.; ROJAS, G.; PASTOR, O. **OOWS: A Method to** 103 **Develop Web Applications from Web-Oriented Conceptual Models**. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, FL - USA, julho, 2003.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection pattern**.
Chicago, IL, EUA, 1994. Disponível em:
<http://www.martinfowler.com/articles/injection.html>. Acesso em: 23 mar. 2006.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, ISBN 0201633612, outubro 1994.

GONÇALVES, P.; Brito, P.; **Manipulação de uma ontologia desenvolvida em OWL através da utilização da API JENA 2 Ontology,** 6º Encontro de Estudantes de Informática (ENCOINFO -2004), Tocantins, 2004.

HUSTED, Ted. **Apache Struts 2 Documentation** – Big Picture 2007. Disponível em:
<http://struts.apache.org/2.x/docs/big-picture.html>. Acesso em: 31 maio 2007.

IBM. **Ontology Definition Metamodel 2005**. Disponível em:
<www.omg.org/docs/ad/05-08-01.pdf>. Acesso em 31 maio 2007.

KAPPEL, G.; Proll, B.; Reich, S.; Retschitzegger, W. **Web Engineering – The Discipline of Systematic Development of Web Applications**. 1st edition.
Heidelberg, Germany, John Wiley & Sons Ltd, 2006.

KNUBLAUCH, H.; Fergerson R. W. ; Noy, N. F. ; Musen M. A. **The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications**. Third International Semantic Web Conference, Hiroshima, Japan, 2004.

KOCH, N.; BAUMEISTER, H.; HENNICKER, R.; MANDEL, L. **Extending UML to Model Navigation and Presentation in Web Applications**. Proceedings of Modelling Web
Applications in the UML Workshop (UML'2000), outubro, 2000.

LUKE, S.; Heflin, J. **SHOE 1.01 – Proposed Specification 2000**. Disponível em: <http://www.cs.umd.edu/projects/plus/SHOE/spec.html>. Acesso em: 27 maio 2007.

PERUCH, L. **Aplicação e Análise do Método FrameWeb com Diferentes Frameworks Web**. 2007. Monografia (Ciência da Computação) – Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2007.

POWELL, T.; Jones, D.; Cutts, D. **Web Site Engineering: Beyound Web Page Design**. Prentice Hall, 1998.

PRESSMAN, R. *S.* **Software Engineering: A Practitioner's Approach**. 6th edition. New York, USA, McGraw Hill, 2005.

RESENDE, A.; SILVA, C. **Programação Orientada a Aspectos em Java**, 1. ed. Brasport,
ISBN 8574522120, 2005.

SOUZA, V. E. S. **FrameWeb**: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web. 2007. Dissertação (Mestrado em Informática), Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2007.

SOUZA, V.; Lourenço, T.W; FALBO, R.A; Guizzardi, G. **S-FrameWeb***: a Framework-Based Design Method for Engineering with Semantic Web Support*,

International Workshop on Web Information System Modeling(WISM), Trondheim, Norway, 2007.

SOUZA, V.; Falbo, R.A. **An Agile Approach for Web Systems Engineering**, XI Simpósio Brasileiro de Sistemas Multimídia e Web(WebMedia), MG, Brasil, 2005.

STEIN, L.; Connolly D.; McGuinness D. **DAML Ontology Language Specification**, 2000. Disponível em <http://www.daml.org/2000/10/daml-ont.html>. Acesso em: 27 maio 2007.