UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM INFORMÁTICA

VÍTOR ESTÊVÃO SILVA SOUZA

FrameWeb: um Método baseado em *Frameworks* para o Projeto de Sistemas de Informação *Web*

VITÓRIA
2007

VÍTOR ESTÊVÃO SILVA SOUZA

FrameWeb: um Método baseado em *Frameworks* para o Projeto de Sistemas de Informação *Web*

Dissertação apresentada ao Mestrado em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Orientador: Prof. Dr. Ricardo de Almeida Falbo.

Co-orientador: Prof. Dr. Giancarlo Guizzardi.

VITÓRIA

2007

VÍTOR ESTÊVÃO SILVA SOUZA

FrameWeb: um Método baseado em *Frameworks* para o Projeto de Sistemas de Informação *Web*

COMISSÃO EXAMINADORA

Prof. Ricardo de Almeida Falbo, D. Sc.
Orientador

Prof. Giancarlo Guizzardi, Ph.D.
Co-orientador

Prof. Davidson Cury, D.Sc.
(UFES)

Profa. Fernanda Lima, D.Sc.
(UCB - Universidade Católica de Brasília)

Vitória, 16 de julho de 2007.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus por todas as oportunidades que tive até hoje, e por me permitir dedicar o tempo e esforço necessários para alcançar este momento tão importante.

Aos meus pais, Eloi Corveto de Souza e Dalva Silva Souza, por todo o investimento em minha educação intelectual, moral e emocional. Aos meus irmãos, Renata, Luiz Gustavo e Marcelo Henrique, pelo companheirismo ao longo de toda vida. Às minhas avós, Penha Corveto de Souza e Lydia Mendes Silva (in memorian), pelo carinho que sempre demonstraram. A Gian e Lauri, pela maravilhosa adição à família que vocês são. À toda minha família, meu muito obrigado por tudo.

À minha amada esposa, Renata Oliveira Lacerda, por estar ao meu lado em todos os momentos, pela compreensão necessária e pelo amor incondicional. Aos seus pais e irmãs, Marcelo, Vanete, Anita e Patrícia, por serem minha segunda família e por me acolherem e apoiarem durante todo este processo.

Obrigado, Ricardo, pela orientação indispensável: sua participação foi fundamental para o sucesso deste empreendimento. Sem sombra de dúvida, você é o melhor orientador que eu conheço. Obrigado, Giancarlo, pelas idéias e reflexões proporcionadas em nossas conversas. Obrigado, Dedê, pelo ponto de partida para todo este trabalho: sem o “MinAmCorA”, não sei se teria sugerido ao Ricardo trabalhar na área de Engenharia Web. Obrigado, Rosane, por compreender minha sobrecarga de trabalho na hora de designar minhas disciplinas, mesmo sendo um problema NP-Completo. Agradeço a todos os professores do Departamento de Informática da UFES, pelo apoio ao longo da Graduação e do Mestrado.

Não poderia deixar de agradecer a todos os meus amigos, aos meus colegas de graduação e de mestrado, aos companheiros do Laboratório de Engenharia de Software, à turma de Engenharia Web que desenvolveu o Portal do LabES, à equipe de desenvolvimento do JSchool (em especial, Lucas Arantes), a Thiago e Leonardo por contribuirem com seus trabalhos de graduação, dentre muitos outros que, de uma forma ou de outra, contribuíram para que eu chegasse até aqui.

Muito obrigado a todos!

Vítor Estêvão Silva Souza

RESUMO

As primeiras gerações de aplicações para a *Web* foram construídas de maneira *ad-hoc*, sem uma metodologia ou processo de software para dar apoio à equipe de desenvolvimento. Para atender à necessidade de abordagens disciplinadas e sistemáticas, uma nova disciplina foi criada: a Engenharia *Web* (*Web Engineering* ou *WebE*). Nessa nova área de pesquisa, muitos métodos têm sido propostos para análise, projeto e desenvolvimento de Sistemas de Informação baseados na *Web* (*Web-based Information Systems* – WISs). Juntamente com essas pesquisas, tecnologias para codificação de aplicações *Web* também se desenvolveram. A utilização de *frameworks* ou arquiteturas baseadas em *containers* para prover uma infraestrutura sólida como base para as aplicações é estado-da-prática. No entanto, não foi encontrado na literatura um método de Engenharia *Web* que tire vantagens do uso desses *frameworks* durante a fase de projeto de sistema. Este trabalho apresenta um método para o projeto de WISs baseados em *frameworks* chamado FrameWeb. O método propõe uma arquitetura básica para desenvolvimento de WISs, um conjunto de atividades e um perfil UML para quatro tipos de modelos de projeto que trazem conceitos utilizados por algumas categorias de *frameworks*. A idéia é que o uso de FrameWeb favoreça um aumento da produtividade da equipe de desenvolvimento por utilizar uma linguagem de modelagem que permite aos projetistas produzir diagramas que representam conceitos dos *frameworks* e aos desenvolvedores ou ferramentas CASE diretamente traduzir esses diagramas para código. Considerando os investimentos em pesquisas na área da *Web* Semântica nos últimos anos e que a visão que é proposta para essa evolução da *Web* não se tornará uma realidade enquanto os autores de *websites* não adicionarem semântica às suas páginas, achamos importante incluir diretrizes sobre como desenvolver WISs com semântica associada usando FrameWeb. Portanto, este trabalho também propõe S-FrameWeb, que estende FrameWeb com o intuito de construir WISs Semânticos baseados em *frameworks*.

ABSTRACT

First-generation Web Applications (WebApps) were constructed in an ad-hoc manner, without a methodology or a software process to support the development team. To address the need for more systematic and disciplined approaches, a new discipline and research field was born: Web Engineering (WebE). In this field of research, many methods have been proposed for the analysis, design and development of Web-based Information Systems (WISs). Along with these researches, technologies for codifying WebApps have also evolved. The use of frameworks or container-based architectures to provide a solid Web infrastructure for the application to be built upon is state-of-the-practice. However, we didn't find a WebE method that takes advantage of the use of these frameworks as early as during system design phase. This work presents a method for the design of framework-based WISs called FrameWeb. This method proposes a basic architecture for developing WISs, a set of activities and a UML profile for four kinds of design models that brings concepts used by some categories of frameworks. The idea is that the use of FrameWeb would further improve team productivity by using a modeling language that would allow designers to produce diagrams that represent framework concepts, and developers or CASE tools to directly translate these diagrams to code. Considering that research on the Semantic Web is gaining momentum in the last few years and that what is envisioned for it will not become a reality until authors of websites attach semantic annotations to their pages, we found it important to include directions on how to develop WISs with semantics associated to them using FrameWeb. Thus, we also propose S-FrameWeb, which extends FrameWeb with the intent of building framework-based Semantic WISs.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1: Introdução

No início da *World Wide Web* (WWW), a infra-estrutura de software por trás dos servidores dava suporte somente a páginas estáticas, ou seja, documentos hipertextos entregues diretamente ao navegador do cliente como resposta a requisições HTTP (*HyperText Transfer Protocol*, protocolo utilizado na WWW). A partir de 1993, com o surgimento da tecnologia CGI (*Common Gateway Interface*) e de linguagens de programação para a *Web*, tais como PHP (em 1994), Microsoft ASP (em 1995) e Java Servlets (em 1996) / JSP (em 1999), os servidores se tornaram mais poderosos, permitindo o desenvolvimento de aplicações de negócio. Em pouco tempo, grandes sistemas B2C (*Business-to-Consumer* ou De-Empresa-para-Consumidor, como lojas virtuais) e B2B (*Business-to-Business* ou De-Empresa-para-Empresa, como sistemas de controle de fornecimento) começaram a ser desenvolvidos para a *Web*.

Nesse momento surge o conceito de aplicação *Web*, ou *WebApp*, que consiste em um conjunto de páginas *Web* que interagem com o visitante, provendo, armazenando e processando informações. Exemplos *são websites* informativos, interativos, aplicações transacionais ou baseadas em fluxos que executam na *Web*, ambientes colaborativos de trabalho, comunidades *online* e portais (GINIGE e MURUGESAN, 2001). Neste trabalho, no entanto, focamos em uma categoria específica de aplicações *Web*, a qual chamamos Sistemas de Informação Baseados na *Web* (*Web-based Information Systems* - WISs), que são como sistemas de informação tradicionais, porém disponíveis na Internet, ou em uma Intranet, como é o caso de lojas virtuais, ambientes cooperativos, sistemas de gerência empresarial, dentre muitos outros.

As aplicações *Web* (*WebApps*) da primeira geração eram, geralmente, desenvolvidas de maneira *ad hoc*, sem levar em conta princípios de Engenharia de Software. Entretanto, para o desenvolvimento das atuais *WebApps*, e em especial dos WISs, é imprescindível a adoção de uma abordagem de engenharia. A Engenharia *Web* (*WebE*) pode ser definida como o estabelecimento e uso de princípios científicos, de engenharia e de gerência, além de abordagens sistemáticas, para o bem sucedido desenvolvimento, implantação e manutenção de aplicações e sistemas *Web* de alta qualidade (MURUGESAN et al., 1999). Pressman (2005) complementa essa definição afirmando que a *WebE* utiliza-se de conceitos e princípios da Engenharia de Software convencional, mas incorpora modelos de processo especializados,

métodos adaptados às características dessas aplicações e um conjunto de importantes tecnologias capazes de permitir seu desenvolvimento.

Neste novo campo de pesquisa, 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 infra-estrutura 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 e SILVA, 2005), um mecanismo de injeção de dependências (FOWLER, 2006), mapeamento objeto/relacional automático para persistência de dados (BAUER e KING, 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.

Este contexto nos motivou a iniciar pesquisas com o intuito de propor um método para Engenharia *Web* baseado em *frameworks*. Este método deve promover a modelagem dos componentes dos *frameworks* nos diagramas de projeto com vistas a fazer com que estes modelos estejam mais próximos de implementação em código e, conseqüentemente, atribuir a responsabilidade de definir a arquitetura ao projetista e não aos programadores. Ao permitir a modelagem dos *frameworks* na fase de projeto, o método incentiva a utilização de uma arquitetura robusta e o aumento da produtividade no desenvolvimento de WISs.

Dentro dessa proposta, consideramos interessante adicionar também ao método uma proposta voltada à *Web* Semântica. A *Web* Semântica é considerada o futuro da Internet, mais especificamente uma evolução da *World Wide Web* atual, referida pelo termo "*Web* Sintática". Nesta última, "os computadores fazem apenas a apresentação da informação, porém o processo de interpretação fica a cabo dos seres humanos mesmo" (BREITMAN, 2006). Os objetivos da *Web* Semântica são permitir que softwares (agentes) instalados nos computadores sejam capazes de interpretar o conteúdo de páginas da *Web* para auxiliar humanos a realizarem suas tarefas diárias na rede.

Considerando que a *Web* Semântica será uma realidade somente quando os autores de páginas *Web* adicionarem semântica às suas páginas, consideramos importante incluir na

proposta diretrizes para construção de aplicações *Web* com semântica associada, de forma a auxiliar na construção desse novo paradigma da Internet. Novas atividades, como análise de domínio, codificação e e implantação de ontologias etc., são necessárias para a construção de WISs semânticos.

1.1. Objetivos da Dissertação

O objetivo principal deste trabalho é propor um método de projeto (*design*) de WISs para Engenharia *Web*, focado no uso de *frameworks*, batizado de FrameWeb (*Framework-based Design Method for Web Engineering*), que possua as seguintes características:

- Ser baseado em metodologias de Engenharia de Software Orientada a Objetos e linguagens de modelagem bem difundidas entre acadêmicos e profissionais da área, facilitando seu aprendizado e aceitação;
- Ser direcionado à construção de aplicações que tenham sua infra-estrutura arquitetônica baseada no uso de *frameworks* ou *containers*;
- Incorporar idéias de desenvolvimento ágil;
- Incluir diretrizes para construção de aplicações prontas para a *Web* Semântica como uma opção, caso a equipe de desenvolvimento considere importante adicionar semântica às páginas *Web* de sua aplicação;
- Não ser restritiva em nenhuma de suas proposições, permitindo que organizações utilizem processos que já possuem familiaridade, adaptando-os para aplicações *Web* por meio das diretrizes de FrameWeb.

FrameWeb é baseada em metodologias e linguagens de modelagens bastante difundidas na área de Engenharia de Software sem, no entanto, impor nenhum processo de desenvolvimento específico. É esperado, apenas, que determinadas atividades comuns à maioria dos processos de software, como levantamento de requisitos, análise, projeto, codificação, testes e implantação, sejam conduzidas pela equipe de desenvolvimento.

O método, portanto, sugere um processo de software orientado a objetos contendo as fases descritas anteriormente, mas podendo ser estendido e adaptado pela equipe de desenvolvimento. A linguagem de modelagem UML (*Unified Modeling Language*) (BOOCH et al., 2005) é utilizada durante todo o processo.

Para o levantamento de requisitos, é proposta a utilização de modelos de casos de uso. Para a análise, a proposta é utilizar diagramas de classes com alto nível de abstração,

construindo um modelo conceitual do domínio do problema. Tais propostas são apenas para uma melhor integração com as diretrizes específicas da fase de projeto, não sendo de forma alguma obrigatórias.

As fases de projeto e implementação constituem o núcleo da proposta de FrameWeb. É durante a fase de projeto que a plataforma de desenvolvimento/implantação – neste caso, aplicação *Web* baseada em *frameworks* – é levada em consideração. São feitas, portanto, as seguintes propostas:

- Uma arquitetura de software padrão que divide o sistema em camadas e integra-se bem com os diferentes tipos de *frameworks*;
- Um conjunto de modelos de projeto que podem ser construídos para representar elementos comuns em *WebApps* baseadas em *frameworks*;
- Uma extensão da UML para construção desses modelos.

Concluindo o processo de software sugerido, as fases de teste e implantação não foram incluídas no escopo deste trabalho, devendo ser objeto de investigação em trabalhos futuros. Em paralelo a todo o processo, a última, porém não menos importante, proposta deste trabalho consiste em um conjunto de diretrizes para que a aplicação *Web*, produto final de um processo de desenvolvimento no qual FrameWeb foi utilizado, contenha anotações semânticas de forma a promover a construção da *Web* Semântica.

1.2. Metodologia

Este trabalho iniciou-se com uma discussão sobre o desenvolvimento de aplicações para a *Web*, no contexto da disciplina “Ambientes Inteligentes”, ministrada no Programa de Pós-Graduação em Informática (PPGI) da Universidade Federal do Espírito Santo (UFES). Durante as discussões sobre a metodologia proposta por Conallen (CONALLEN, 2002), surgiram as primeiras idéias para a criação de um método que trouxesse para a fase de projeto os elementos corriqueiramente utilizados na implementação de WISs. Tais discussões promoveram a definição das bases para esta dissertação, resumidas em (SOUZA e FALBO, 2005).

Definida a área de interesse, o trabalho prosseguiu com uma revisão bibliográfica sobre a mesma. Foram avaliados e discutidos artigos científicos, relatórios técnicos e trabalhos acadêmicos que tratam sobre Engenharia *Web* (em especial, métodos de projeto de aplicações *Web*), processos ágeis, *Web* Semântica, *Web Services* Semânticos, dentre outros. Tal trabalho

sistemático permitiu uma extensa revisão dos métodos e linguagens propostos na literatura, do qual foram extraídas idéias para refinamento da proposta.

Elaborada uma primeira versão do método, o próximo passo consistiu da experimentação junto a um grupo de alunos do PPGI e do curso de graduação em Ciência da Computação da UFES, no contexto da disciplina de "Engenharia *Web*". Nessa disciplina, foram conduzidas diversas discussões sobre *WebE*, além de um rápido treinamento sobre alguns *frameworks* e o método FrameWeb para finalizar com o desenvolvimento de um WIS, o Portal do LabES, utilizando FrameWeb. Tal experiência permitiu uma avaliação do método e posteriores melhorias em suas diretrizes.

Um segundo experimento foi iniciado junto ao Grupo de Usuários de Java do Estado do Espírito Santo (ESJUG[1]) para desenvolvimento da aplicação *Web* modelada na disciplina "Ambientes Inteligentes": um ambiente cooperativo de aprendizagem chamado JSchool[2]. Foram conduzidas reuniões de treinamento e o projeto encontra-se em fase de implementação.

Essa fase do trabalho, a definição de uma versão experimentada e discutida de FrameWeb, rendeu duas publicações (SOUZA e FALBO, 2007) e (SOUZA et al., 2007b) e uma orientação de projeto final de graduação em Ciência da Computação na UFES (PERUCH, 2007). Consideramos, então, que atingimos um estágio maduro dessa proposta e iniciamos a discussão da segunda fase, a definição de S-FrameWeb.

Tal definição envolveu estudos mais aprofundados na área da *Web* Semântica, incluindo materiais sobre análise de domínio, metodologias para construção de ontologias, linguagens de modelagem e codificação de ontologias, agentes da *Web* Semântica etc. Após experimentações com o Portal do LabES, chegamos a um conjunto de diretrizes que propomos aqui. Essa segunda parte do trabalho teve como resultado uma publicação (SOUZA et al., 2007c) e mais uma orientação de projeto final de graduação (LOURENÇO, 2007).

Esta dissertação é o produto final de todo esse processo e descreve o método para projeto de sistemas de informação *Web* baseados em *frameworks*, FrameWeb, e sua extensão para a *Web* Semântica, S-FrameWeb.

1.3. Organização da Dissertação

Além desta introdução, esta dissertação possui outros cinco capítulos.

O Capítulo 2, "Engenharia *Web*", faz um resumo do estado-da-arte e do estado-da-prática do desenvolvimento de aplicações para a plataforma *Web*, resumindo e analisando

1 http://esjug.dev.java.net

2 http://jschool.dev.java.net

idéias propostas nos diversos trabalhos científicos publicados na área, além de tecer reflexões sobre as tecnologias mais utilizadas para formação de uma fundação arquitetônica para WISs baseada em *frameworks*.

De forma análoga, o Capítulo 3, "A *Web* Semântica", trata das pesquisas em torno dessa evolução da *World Wide Web*, na qual agentes de software poderão interpretar o conteúdo das páginas da Internet e conduzir diversas tarefas por nós consideradas dispendiosas e tediosas. O capítulo aborda temas relacionados, como ontologias, agentes e linguagens da *Web* Semântica.

No Capítulo 4, "O Método FrameWeb", temos a descrição detalhada do método de projeto de aplicações *Web* baseadas em *frameworks*, o principal objetivo desta dissertação. Além de apresentar a arquitetura básica para WISs baseados em *frameworks*, a linguagem de modelagem e as diretrizes para condução de projetos com FrameWeb, o capítulo descreve o Portal do LabES e mostra diversos exemplos de diagramas produzidos durante seu desenvolvimento.

Na seqüência, o Capítulo 5, "S-FrameWeb", complementa o capítulo anterior com as diretrizes a serem seguidas pelos utilizadores de FrameWeb para adição de semântica às suas aplicações, com vistas a prepará-las para este possível cenário futuro que é a *Web* Semântica.

Finalmente, o Capítulo 6, "Considerações Finais", conclui esta dissertação com uma avaliação geral do trabalho e as perspectivas de trabalhos futuros abertas pela pesquisa de FrameWeb.

Capítulo 2: Engenharia *Web*

Com o advento da *World Wide Web*, ou simplesmente *Web*, páginas e aplicativos começaram a popular o universo de *sites* disponíveis. Nesta época, *websites* eram desenvolvidos de maneira *ad-hoc*, sem uma metodologia ou processo para apoiar seus criadores. No entanto, à medida que eles cresceram em complexidade e tornaram-se verdadeiras aplicações na *Web,* tornou-se necessário aplicar métodos disciplinados de Engenharia de Software, adaptados para essa nova plataforma de implementação.

Características do ambiente *Web*, como concorrência, carga imprevisível, disponibilidade, sensibilidade ao conteúdo, evolução dinâmica, imediatismo, segurança e estética (PRESSMAN, 2005) imprimiram uma nova dinâmica aos processos já existentes, dando origem a uma nova sub-área da Engenharia de Software, a Engenharia *Web*. Murugesan et al. (1999) a definem como o estabelecimento e uso de princípios de engenharia e abordagens disciplinadas para o desenvolvimento, implantação e manutenção de aplicações baseadas na *Web*.

Existem diversos tipos de *website*. Há páginas que são estritamente informativas (ex.: página de um professor da universidade), algumas vezes com uma grande carga de elementos multimídia (ex.: *website* de um museu ou enciclopédia *online*). Outras, no entanto, são verdadeiros sistemas de informação baseados na *Web*, como é o caso de lojas virtuais, ambientes cooperativos, sistemas de gerência empresarial, dentre muitos outros. Neste trabalho, nos referimos a esse tipo de *website* como Sistemas de Informação *Web* (*Web-based Information Systems* - WISs) e ao conjunto amplo de todas as aplicações *Web* como *WebApps* (abreviação de *Web Applications*).

Em se tratando de WISs, tecnologias para implementação deste tipo de aplicação *Web* vêm se desenvolvendo de forma rápida e consistente. Em 1994, com o advento do *Common Gateway Interface* (CGI), programas em linguagens como C ou PERL poderiam ser ativados por requisições de navegadores *Web*, fazendo com que um servidor *Web* pudesse retornar ao cliente o que fosse impresso pelo programa em sua saída padrão. Hoje, existem *containers* que gerenciam automaticamente componentes criados em linguagens orientadas a objetos, provendo uma série de serviços automáticos, como gerenciamento de persistência, controle de transações etc.

Em particular, destaca-se para nós o surgimento de diversos *frameworks* que provêem

uma sólida infra-estrutura para o desenvolvimento de WISs. O uso desses *frameworks*, com o passar do tempo, tornou-se estado-da-prática e até mesmo influenciou na definição de padrões para desenvolvimento de aplicações distribuídas. Seu uso (ou reúso) auxilia a equipe de desenvolvimento a construir software mais rapidamente (vários componentes já estão prontos) e com maior qualidade (os *frameworks* já foram extensivamente testados por seus criadores).

Este capítulo apresenta uma revisão geral das metodologias e *frameworks* propostos pela comunidade de Engenharia *Web*. A partir dessa revisão, foi possível observar práticas bem sucedidas e extrair idéias que poderiam ser reproduzidas no contexto da proposta de um novo método de projeto.

Tal revisão foi feita a partir de diversas fontes, sendo a principal delas o relatório técnico da revisão sistemática conduzida por Conte et al. (2005), no qual diversos artigos extraídos dos acervos digitais da Elsevier[3] e IEEE[4] são referenciados e brevemente descritos, no contexto de uma pesquisa por metodologias *Web* que incluam atividades de garantia da qualidade. Outras fontes foram as bibliotecas digitais da ACM[5] e Kluwer[6], além de mecanismos de busca regulares da Internet (ex.: Google), trocas de experiência em congressos da área e conhecimento prévio do autor deste trabalho e de seus orientadores.

Para uma melhor organização do capítulo, dividimos a apresentação dos trabalhos pesquisados em quatro seções: metodologias (Seção 2.1), linguagens de modelagem (Seção 2.2), *frameworks* (Seção 2.3) e outros trabalhos relacionados (Seção 2.4). Na Seção 2.5 concluímos com uma breve análise dos trabalhos citados.

2.1. Metodologias para Engenharia *Web*

Nesta seção são apresentadas algumas metodologias propostas para a Engenharia *Web*. Foram consideradas metodologias as propostas que definem um processo, obrigatório ou sugerido, listando atividades a serem executadas e artefatos a serem produzidos e apresentando ou indicando a notação a ser utilizada na construção de tais artefatos.

São apresentadas a seguir as seguintes propostas: a Metodologia Baseada em Processos de Negócio, a Metodologia de Prototipação Rápida Dirigida a Modelos em Escala Real, a metodologia proposta por Lee & Shirani, o Método de Desenvolvimento Ariadne, a Metodologia de Desenvolvimento de Comércio na Internet, a metodologia proposta por Conallen, a Solução *Web* Orientada a Objetos, a Engenharia *Web* Baseada em UML, a

3 http://www.sciencedirect.com

4 http://www.ieee.org/web/publications/home

5 http://portal.acm.org/dl.cfm

6 http://www.springerlink.com

metodologia de apoio ao uso do *Framework* JAFAR2 e o método OOHDM.

2.1.1. Metodologia de Prototipação Rápida Dirigida a Modelos em Escala Real

A Metodologia de Prototipação Rápida Dirigida a Modelos em Escala Real (*Mockup-Driven Fast-Prototyping Methodology* – MODFM) (ZHANG et al., 2003a) (ZHANG e CHUNG, 2003b) é uma metodologia baseada em prototipação que tem como objetivo ajudar no levantamento e elaboração dos requisitos, além de facilitar o ajuste a mudanças de requisitos, agilizando, assim, o desenvolvimento de aplicações *Web*. Seus componentes essenciais são uma arquitetura de duas camadas (*front-end* e *back-end*) organizadas segundo um padrão Modelo - Visão - Controlador (MVC) (GAMMA et al., 1994) e geradores automáticos de código. MODFM incorpora tecnologias apropriadas para a *Web*, tais como J2EE (SHANNON, 2003) e XML (HAROLD e MEANS, 2004).

A metodologia baseia-se na crença de que qualquer módulo a ser desenvolvido para uma aplicação *Web* pode ser implementado pela composição de elementos de sete categorias de artefatos de código: páginas JSP (*JavaServer Pages*), *beans* de formulários (*form beans*), pré-ações, pós-ações, métodos de serviço, componentes EJB (*Enterprise JavaBeans*) e esquemas de banco de dados. Tal estruturação meticulosa permite a aplicação de geradores automáticos de código. MODFM conta com um gerador de códigos (*WGenerator*) e um gerador de menus para ligação entre as páginas. Ambos estão integrados num ambiente de suporte, chamado *WGWeb*. O processo é dividido em oito etapas, mostradas na Figura 2.1.

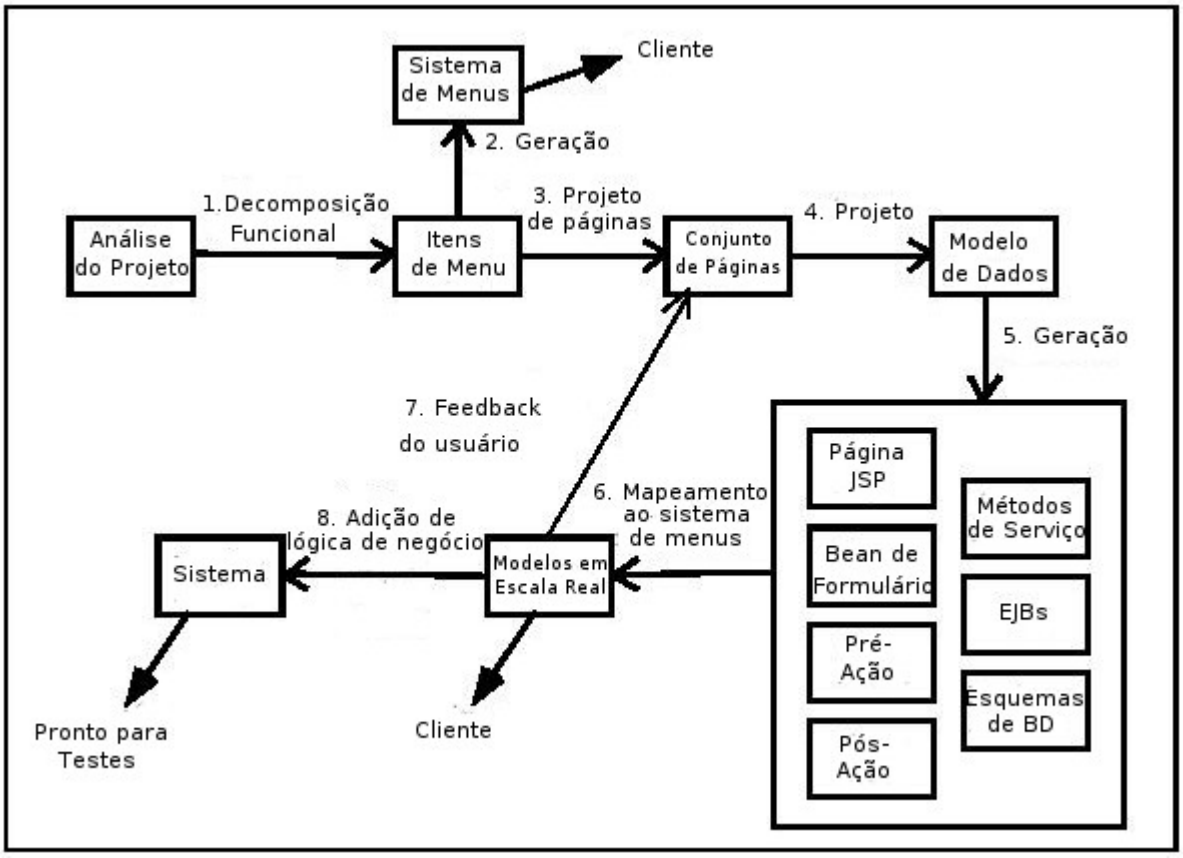

Figura 2.1: Processo de Análise e Projeto do MODFM (ZHANG e CHUNG, 2003b)

2.1.2. A metodologia de Lee & Shirani

Seung Lee e Ashraf Shirani (LEE e SHIRANI, 2004) propõem uma metodologia baseada em componentes para o desenvolvimento de aplicações *Web*. Os autores defendem a idéia de que a natureza das aplicações *Web* facilita a componentização e propõem métodos para análise de requisitos e projeto de sistema utilizando tanto técnicas estruturadas quanto orientadas a objetos.

Nessa metodologia, as páginas *Web* são os blocos de construção fundamentais da aplicação. Elas podem ser visíveis ou invisíveis para o usuário e são divididas em diversos tipos de página. Além disso, podem ou não envolver a execução de lógica de negócio do lado do servidor (*server-side*). Elas são conectadas entre si por *hiperlinks* ou outros tipos de associação. Páginas relacionadas podem ser agrupadas em compêndios, que são pequenos, porém completos, conjuntos de itens relacionados a um assunto particular (ex.: o processamento de um pedido numa loja virtual).

A metodologia divide-se em duas fases principais: análise de requisitos de componentes e especificação de componentes. A análise inicia com a identificação das funções da aplicação

(em abstrações de alto-nível e detalhamentos de nível mais baixo) e segue para a determinação do máximo denominador comum entre as funções requeridas e os componentes disponíveis. Finalizando, modelos de análise dos compêndios identificados são construídos, utilizando uma notação própria da metodologia, mostrada na Figura 2.2.

Figura 2.2: Notação para modelos de requisitos de componentes de baixo-nível (LEE e SHIRANI, 2004)

A fase de especificação de componentes possui três atividades. Na primeira - especificação de representação (*rendering*) - as páginas invisíveis, que são responsáveis pela renderização das páginas visíveis, são modeladas utilizando a notação mostrada na Figura 2.3. A segunda atividade - especificação de integração - identifica relacionamentos de um compêndio com outros e com componentes externos. A última fase - especificação de interface - reúne as interfaces dos compêndios para localização dos componentes em um ambiente distribuído.

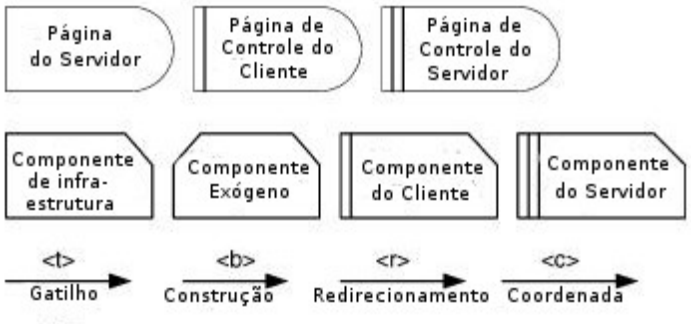

Figura 2.3: Notação para modelos de especificação de componentes (LEE e SHIRANI, 2004)

Apesar de propor sua própria notação para análise de requisitos e especificação de componentes, a metodologia aceita a utilização da notação de casos de uso para a modelagem dos compêndios e de diagramas de classes para a modelagem das páginas.

2.1.3. Método de Desenvolvimento Ariadne

O Método de Desenvolvimento Ariadne (*Ariadne Development Method* – ADM) (DÍAZ

et al., 2004) propõe um processo sistemático, flexível, integrador e independente de plataforma para especificação e avaliação de sistemas hipermídia e aplicações *Web*. ADM considera seis visões de projeto distintas: estrutura, navegação, apresentação, comportamento, processo e segurança.

Seu processo é composto de três fases: projeto conceitual, focado na identificação de tipos abstratos de componentes, relacionamentos e funções; projeto detalhado, referente à especificação de funcionalidades, processos e comportamentos do sistema em detalhe suficiente para que o mesmo seja gerado automaticamente em seguida; e avaliação, que se preocupa com o uso de protótipos e especificações para avaliar a usabilidade do sistema em relação a uma série de critérios bem definidos. Cada uma dessas fases é dividida em atividades, gerando artefatos. A Tabela 2.1 resume as atividades e artefatos produzidos.

Tabela 2.1: Fases, atividades e produtos de ADM

Fase	***Atividade***	***Produto***
Projeto Conceitual	Definição da estrutura lógica	Diagrama Estrutural
	Estudo das funções do sistema	Diagrama Navegacional Especificações Funcionais
	Especificação das entidades	Diagramas Internos Catálogo de Atributos Catálogo de Eventos
	Modelagem de usuários	Diagrama de Usuários
	Definição da política de segurança	Catálogo de Categorias Tabela de Acesso
Projeto Detalhado	Identificação das instâncias	Diagrama de Instâncias de Nós Diagrama de Instâncias de Usuários
	Especificação de funções	Especificação de Estruturas de Acesso Especificação Detalhada de Funções
	Especificação de instâncias	Diagramas Internos Detalhados Regras de Autorização Delegação de Usuários
	Definição de requisitos de apresentação	Especificação de Apresentação
Avaliação	Desenvolvimento de um protótipo	Protótipo
	Avaliação	Documento de Avaliação Relatório de Conclusão

O método *Ariadne* não impõe nenhuma seqüência rígida para as fases ou suas

atividades, deixando a cargo do desenvolvedor escolher a melhor forma de conduzir seus projetos.

2.1.4. A metodologia proposta por Conallen

Jim Conallen, em seu livro "Desenvolvendo Aplicações Web com UML" (CONALLEN, 2002), descreve um processo de desenvolvimento de software iterativo, incremental e centrado em casos de uso, baseado no Processo Unificado Rational (*Rational Unified Process* – RUP) (KRUTCHEN, 2000) e no Processo Unificado ICONIX (*ICONIX Unified Process* – ICONIX-UP) (ROSENBERG e SCOTT, 1999, apud CONALLEN, 2002).

Apesar de enfatizar que não há uma receita pronta para um bom processo de software, Conallen indica um processo que poderia ser aplicado no desenvolvimento de *WebApps*, definindo seu fluxo de trabalho, atividades, artefatos e operadores. As atividades do processo são mostradas na Figura 2.4.

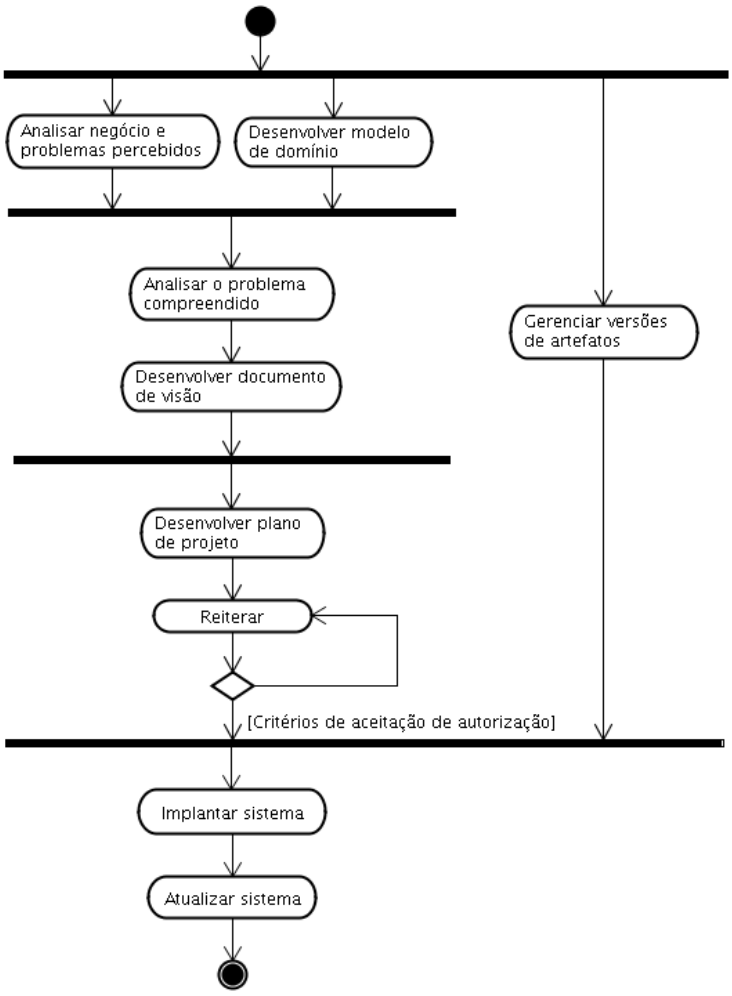

Figura 2.4: Processo de desenvolvimento de software proposto por CONALLEN (2002)

O processo começa com uma análise do negócio, que acontece em paralelo com o desenvolvimento de um modelo do domínio. Em seguida, é feita uma análise mais profunda do problema e é construído o documento de visão, o qual expressa o escopo e a finalidade de todo o projeto e a partir do qual todos os outros artefatos são derivados. Então, o plano de projeto é elaborado, esboçando as atividades do processo e definindo os eventos principais e documentos padrão, incluindo o plano de gerência de configuração. A atividade “Reiterar” representa um sub-processo iterativo de construção de software, expandido na Figura 2.5. Após terminada a execução desse sub-processo, o software é implantado e inicia-se a atividade de manutenção (atualizar sistema). A atividade “Gerenciar versões de artefatos” ocorre em paralelo ao processo e corresponde à gerência de alterações e ao uso de um sistema de controle de versão.

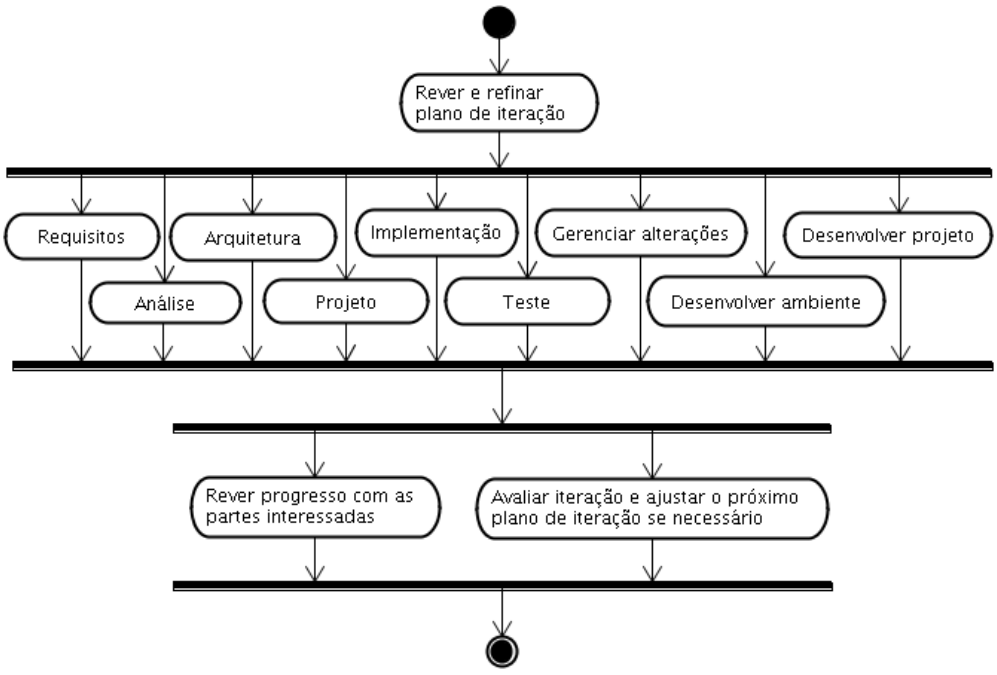

Figura 2.5: Detalhamento da atividade "Reiterar" (CONALLEN, 2002)

O sub-processo “Reiterar” inicia com a revisão e refinamento do plano da iteração. A seguir, as seguintes atividades são executadas em paralelo: levantamento de requisitos, análise, projeto da arquitetura, projeto detalhado, implementação, testes, gerência de alterações, desenvolvimento do ambiente e desenvolvimento do projeto. Finalizada a iteração, o progresso alcançado até o momento é apresentado às partes interessadas (*stakeholders*) e a iteração atual é avaliada, ajustando o plano da próxima iteração, se necessário.

No que tange ao levantamento e à análise de requisitos, o grande diferencial da metodologia de Conallen em relação ao desenvolvimento convencional é a definição de um

novo modelo, chamado Modelo de Experiência do Usuário (*User Experience* – UX), que tem como objetivo definir diretrizes para a construção de páginas *Web*, especificando regras de aparência e navegação que visam manter toda a aplicação num mesmo estilo visual e fornecer uma interface mais amigável para os futuros usuários da *WebApp*.

A modelagem da experiência do usuário é realizada pela equipe de UX, liderada pelo Arquiteto da Informação, um novo papel em processos de software, específico para a Engenharia *Web*. O objetivo é a criação do documento de diretrizes de UX, que começa a ser construído logo na fase de análise e descreve tudo que um desenvolvedor de páginas *Web* precisaria saber para especificar e criar uma nova tela que, quando adicionada ao restante do sistema, pareça ser uma parte integrante do mesmo. A equipe de UX tem o desafio de manter a interface com o usuário consistente com os paradigmas atuais e, principalmente, adequada ao contexto no qual se espera que o sistema seja executado.

A modelagem UX complementa os casos de uso, mostrando de forma abstrata como as informações serão trocadas entre o sistema *Web* e seus usuários. Seus artefatos principais são mapas de navegação e roteiros. Como a modelagem UX e a análise são feitas em paralelo por duas equipes distintas, Conallen sugere que se faça um mapeamento entre as classes limítrofes (*boundary*) do modelo de análise e as telas do modelo UX. Assim, garante-se que ambas as equipes estão trabalhando na solução do mesmo problema.

Conallen retrata a importância do modelo UX ao dizer que "formalizar a experiência do usuário em um modelo UML é uma maneira de estabelecer esse acordo de modo que seja compreensível pelas equipes criativas que tendem a ser aparentemente menos treinadas em ciência da computação. A UML é visual o bastante para que muitos membros não técnicos a compreendam e formal o suficiente para ter valor semântico significativo. A outra vantagem [...] é que é possível para as equipes de engenharia estabelecer e manter mapeamentos diretamente entre um caso de uso e modelos de projeto para o modelo UX" (CONALLEN, 2002).

O projeto, segundo a metodologia de Conallen, é dividido em duas visões: lógica e de componentes. A primeira possui um nível de abstração mais alto (classes e associações), enquanto a segunda trata de arquivos que efetivamente comporão o sistema implementado (páginas e diretórios).

Finalmente, Conallen propõe a utilização de um perfil UML para modelar diversos artefatos sugeridos por sua metodologia. Esse perfil é descrito com mais detalhes na Seção 2.2.2.

2.1.5. Solução *Web* Orientada a Objetos

A Solução *Web* Orientada a Objetos (*Object Oriented Web Solution* – OOWS) é uma extensão da metodologia *OO-Method* (PASTOR et al., 2001, apud FONS et al., 2003) que tem por objetivo permitir a especificação de uma aplicação *Web* em um *framework* integrado, capturando os requisitos em modelos conceituais e gerando protótipos operacionais a partir das especificações (FONS et al., 2003).

Seguindo a metodologia definida por *OO-Method*, existem dois passos principais para a construção de um sistema orientado a objetos: Especificação do Problema e Desenvolvimento da Solução. Na primeira fase devem-se capturar as peculiaridades e o comportamento que o sistema deve oferecer para satisfazer os requisitos identificados. Esse passo inclui a atividade de especificação de requisitos utilizando a abordagem de casos de uso e posteriormente as atividades de modelagem conceitual, nas quais derivam-se abstrações do problema em termos de classes com estrutura, comportamento e funcionalidade definidas. Em OOWS, os seguintes modelos são construídos: de Objetos, Dinâmico, Funcional, de Navegação e de Apresentação (FONS et al., 2002).

O Modelo de Navegação captura os requisitos de navegação da aplicação, definindo um mapa de navegação para cada tipo de usuário do sistema. Sua construção é dividida em duas etapas: Identificação e Categorização do Usuário e Especificação do Diagrama de Navegação. O mapa de navegação, que é criado para cada usuário na segunda etapa, é representado por um grafo dirigido, cujos vértices representam as unidades básicas de interação – os contextos de navegação (graficamente representados por pacotes da UML) – e os arcos denotam os caminhos de navegação válidos pré-definidos.

Cada contexto de navegação é posteriormente modelado na forma de um diagrama de classes, exibindo as classes de navegação que dele fazem parte. O modelo de apresentação usa os contextos de navegação como principais insumos para a definição dos requisitos de apresentação, que são especificados por meio de padrões de apresentação (*presentation patterns*) que podem ser associados aos elementos do contexto de navegação.

Na segunda fase, Desenvolvimento da Solução, propõe-se uma estratégia de geração de código baseada em componentes para implantar um protótipo operacional da aplicação *Web*, com funcionalidade equivalente à especificação inicial. Essa fase se dá de maneira completamente automática por um compilador de modelos conceituais, facilitando as tarefas de manutenção e evolução (FONS et al., 2002). A aplicação *Web* gerada é dividida em uma arquitetura de três camadas: camada de apresentação (interface com o usuário), camada de

aplicação (lógica de negócio) e camada de persistência (acesso a repositórios de dados).

2.1.6. Engenharia *Web* Baseada em UML

A metodologia Engenharia *Web* Baseada em UML (*UML-based Web Engineering* – UWE) (KOCH et al., 2001) (KOCH e KRAUS, 2002) define um conjunto de modelos a serem construídos no desenvolvimento de uma aplicação *Web*, uma linguagem de modelagem para a elaboração desses artefatos (discutida na Seção 2.2.3) e um processo para construção dos modelos.

Segundo o processo proposto, as atividades principais de modelagem são: análise de requisitos, projeto conceitual, projeto de navegação, projeto de apresentação, modelagem de tarefas e modelagem de implantação.

A abordagem UWE apóia o desenvolvimento de aplicações *Web* com foco especial na sistematização, personalização e geração automática de código. Ela indica a utilização de casos de uso para a especificação de requisitos e, baseado nos casos de uso especificados, o projeto conceitual produz um modelo conceitual do problema, definido em termos de classes e associações entre classes relevantes do domínio. O projeto de navegação utiliza o modelo conceitual como base e é definido em duas etapas: definição do modelo de espaço de navegação e construção do modelo de estrutura de navegação. O primeiro modelo mostra quais objetos podem ser vistos através da navegação pela aplicação *Web* e é, portanto, um modelo de análise. O segundo modelo é baseado no primeiro e define como os objetos são visitados, sendo construído na fase de projeto.

O projeto de apresentação consiste na modelagem de interfaces com o usuário abstratas, mostrando como a estrutura de navegação definida anteriormente é apresentada ao usuário. Ele é dividido em uma parte estática e outra dinâmica. A parte estática é representada por uma forma particular de diagrama de classes que usa uma notação de composição de classes. Elementos de apresentação são representados por classes estereotipadas e suas partes internas são representadas graficamente dentro do elemento de modelo "classe", podendo ter vários níveis de "aninhamento de elementos gráficos". Ferramentas CASE comuns não conseguem representar esse tipo de notação, porém, segundo os autores, ela é a mais apropriada para a modelagem de interfaces gráficas, por permitir a representação espacial do elemento de interface com o usuário.

As classes que representam tais elementos podem ser ligadas num diagrama de seqüência ou de comunicação da UML formando roteiros (*storyboards*) que compõem a parte dinâmica do projeto de apresentação. Para refinar os casos de uso, diagramas de atividade

podem ser utilizados, modelando de forma mais detalhada a interação com o usuário. Por fim, diagramas de implantação podem ser usados para documentar a distribuição dos componentes da aplicação *Web*.

2.1.7. Método de Projeto Hipermídia Orientado a Objetos

Segundo Díaz et. al (2004), o paradigma hipermídia se baseia na idéia de organizar as informações como uma rede de nós inter-relacionados que podem ser navegados livremente pelos usuários via *links* ou outras estruturas avançadas de navegação, como índices e mapas.

OOHDM (*Object Oriented Hypermedia Design Method*) (SCHWABE e ROSSI, 1998) nasceu da necessidade de utilizar abstrações capazes de facilitar a especificação de aplicações hipermídia. Metodologias tradicionais de engenharia de software não proviam a noção de *links* e pouco era dito sobre como incorporar texto à interface. Também faltavam primitivas para especificar navegação, que é uma das chaves para o sucesso de uma aplicação Hipermídia .

O processo de construção de uma aplicação hipermídia com OOHDM se dá em cinco etapas: Levantamento de Requisitos, Projeto Conceitual, Projeto de Navegação, Projeto de Interfaces Abstratas e Implementação. Tais atividades são precedidas pela especificação de requisitos e são executadas numa mistura de ciclo de vida incremental e iterativo, construindo e evoluindo os modelos pouco a pouco a cada passo. Os fundamentos da abordagem OOHDM são: (i) a noção de que objetos de navegação são visões de objetos conceituais; (ii) o uso de abstrações apropriadas para organizar o espaço de navegação, com a introdução de contextos de navegação; (iii) a separação de questões de interface das questões de navegação; e (iv) uma identificação explícita de que algumas decisões de projeto só precisam ser feitas no momento da implementação (SCHWABE e ROSSI, 1998).

Durante o Projeto Conceitual, constrói-se um esquema conceitual representando objetos, suas relações e colaborações existentes no domínio em questão. Tal esquema conceitual é descrito em termos de classes, relacionamentos e sub-sistemas, podendo ser modelado em UML.

Segundo o *website* de OOHDM[7], no Projeto de Navegação a estrutura de navegação da aplicação hipermídia é descrita em termos dos seus contextos navegacionais e classes como: nós, *links*, índices, caminhos guiados (*guided tours*) etc. Os nós representam visões lógicas dos objetos conceituais definidos na fase anterior e diferentes modelos de navegação podem ser construídos para o mesmo esquema conceitual. Os *links* são derivados das relações

7 http://www.tecweb.inf.puc-rio.br/oohdm/

conceituais definidas na primeira fase de OOHDM. Ao definir a semântica navegacional em termos de nós e *links*, o projetista pode modelar o movimento no espaço de navegação, independentemente do modelo conceitual, permitindo que este modelo evolua independentemente e simplificando a manutenção do sistema.

Depois que a estrutura de navegação foi definida, seus aspectos de interface são especificados no Projeto de Interfaces Abstratas, definindo a forma na qual diferentes objetos de navegação serão apresentados, quais objetos de interface ativarão uma navegação e outras funcionalidades da aplicação e quando as transformações de interface serão feitas.

Por fim, na Implementação, os modelos construídos até então são mapeados para objetos de implementação a partir de modelos (*templates*). Nessa fase, o ambiente de execução específico será levado em conta e o projetista deve decidir como os itens de informação serão armazenados e como a interface, aparência e comportamento do sistema serão implementados em HTML (e suas possíveis extensões).

O método OOHDM foi estendido, dando origem ao método SHDM (*Semantic Hypermedia Design Method*). SHDM mantém os modelos definidos em OOHDM, estendendo-os com algumas primitivas, tais como subrelações e classes anônimas, inspiradas nas linguagens RDF e DAML+OIL (LIMA e SCHWABE, 2003). O objetivo é propor uma abordagem para a criação de aplicações hipermídia para a *Web* Semântica (ver Seção 3.1).

Outra extensão de OOHDM que interessa particulamente a este trabalho é a OOHDM-Java2 (JACYNTHO et al., 2002). A extensão consiste de uma arquitetura baseada e componentes e um *framework* de implementação para construção de aplicações *Web* complexas baseada em arquiteturas modulares (ex.: Java *Enterprise Edition*). Jacyntho et al. (2002) descrevem 7 passos para utilização de OOHDM-Java2:

1. Definição da estrutura e comportamento dos objetos de negócio (modelo conceitual);
2. Definição dos eventos de negócio da aplicação e especialização do componente "Tradutor de Requisições HTTP", presente no *framework*;
3. Personalização do componente "Executor": para cada objeto de evento de negócio, implementa-se a lógica de execução do evento;
4. Especialização do componente "Seletor de Visão", indicando quais elementos de visão (páginas) devem ser apresentados em quais resultados dos eventos;
5. Especialização do componente de "Contexto de Navegação", definindo a estrutura do espaço de navegação e identificando os contextos (conjuntos de nós);
6. Refino do componente de "Contexto de Navegação", especificando os atributos

que estarão visíveis para cada cliente;

7. Criação das páginas JSP e conseqüente definição do leiaute da aplicação *Web*.

2.1.8. Outras metodologias

Resumimos a seguir outras metodologias encontradas durante a revisão bibliográfica e que consideramos menos relevantes à nossa pesquisa.

A Metodologia Baseada em Processos de Negócio (*Business Process-Based Methodology* – BPBM) (ARCH-INT e BATANOV, 2003) é uma metodologia para desenvolvimento de sistemas de informação na *Web* a partir de componentes. A BPBM combina as vantagens dos paradigmas estruturado e orientado a objetos para identificação e modelagem de componentes de negócio e atividades de negócio.

BPBM divide-se em modelagem de componentes de negócio e modelagem de implementação *Web*. A primeira parte decompõe o sistema em um conjunto de processos de negócio que, por sua vez, são também decompostos em atividades de negócio. Usando modelos de Entidades e Relacionamentos (ER) estendidos, representam tais objetos de negócio, que, posteriormente, dão origem a componentes de projeto. A segunda parte da metodologia, modelagem de implementação *Web*, é um guia para implementação do modelo criado durante a fase anterior em uma infra-estrutura específica baseada em tecnologia *Web*.

A Metodologia de Desenvolvimento de Comércio na Internet (*Internet Commerce Development Methodology* – ICDM) (STANDING, 2002) é uma metodologia para o desenvolvimento de aplicações de comércio eletrônico negócio-consumidor (*business-to-consumer* – B2C) para a *Web*. ICDM provê um conjunto de diretrizes para guiar o desenvolvedor nessa tarefa.

ICDM enfatiza tanto os aspectos técnicos quanto questões estratégicas, de negócio, gerenciais e da cultura organizacional. ICDM envolve toda a organização, provendo diretrizes que podem ser adaptadas da forma que for mais adequada. Atividades de gerência são realizadas em três níveis (gerencial, componentes e implementação) continuamente durante todo o processo.

JAFAR2 (*J2EE Architectural Framework 2.0*) (RIES e CHABERT, 2003, apud GUELFI et al., 2003) é um *framework* para construção de aplicações *Web* de comércio eletrônico utilizando *Java 2 Enterprise Edition* (J2EE), versão 1.4 (SHANNON, 2003). O *framework* provê soluções técnicas para problemas comuns no desenvolvimento desse tipo de aplicação e provê padrões que simplificam o trabalho do desenvolvedor.

O *framework* funciona como um protótipo para exploração do domínio de

transformação de modelos dentro da pesquisa em torno de MEDAL (*UML Generic Model Transformer Tool*) (GUELFI et al., 2003), uma ferramenta de transformação de modelos em desenvolvimento dentro do contexto do projeto FIDJI (PERROUIN et al., 2002, apud GUELFI et al., 2003). Para apoiar o uso de JAFAR2, uma metodologia é proposta, sendo esta bastante voltada para a automatização em uma ferramenta CASE para desenvolvimento de projetos com JAFAR2, usando MEDAL como ferramenta de transformação de modelos.

2.2. Linguagens de modelagem para a *Web*

Descrevemos nesta seção algumas linguagens de modelagem propostas para a Engenharia *Web*. Linguagens de modelagem definem notações a serem usadas na criação de modelos abstratos da solução de problemas a serem resolvidos. A Linguagem de Modelagem Unificada (*Unified Modeling Language* – UML) (BOOCH et al., 2005), por exemplo, é uma linguagem de modelagem que define em seu meta-modelo notações padronizadas para diversos tipos de diagramas, dentre eles diagramas de classes, diagramas de casos de uso, diagramas de estado etc., sem, no entanto, definir quando e com que propósito tais diagramas devem ser utilizados.

De fato, por ser amplamente utilizada, muitas vezes a UML é usada como base para as linguagens apresentadas nesta seção. Ela possui mecanismos de extensão que nos permitem definir uma nova linguagem com base em sua notação, a saber:

- estereótipo (*stereotype*) – definição de um novo tipo de elemento de modelo baseado em um já existente;
- valor etiquetado (*tagged values*) – uso de pares <etiqueta, valor> para anexar informações textuais arbitrárias a elementos de modelo; e
- restrição (*constraint*) – condição ou restrição que permite a especificação de nova semântica a um elemento de modelo em uma linguagem formal ou informal.

São apresentadas a seguir, as seguintes linguagens de modelagem: a Linguagem de Modelagem *Web*, as Extensões de Aplicações *Web* e a linguagem de modelagem da UWE.

2.2.1. Linguagem de Modelagem *Web*

A Linguagem de Modelagem *Web* (*Web Modeling Language* – WebML) (CERI et al., 2000) (CERI et al., 2002) (CERI et al., 2002b) é uma linguagem de modelagem para aplicações *Web* que permite que projetistas modelem as funcionalidades de um *site* em um alto nível de abstração, sem se comprometerem com detalhes de alguma arquitetura específica.

WebML é uma linguagem baseada em XML, mas remete a representações gráficas intuitivas e pode ser facilmente suportada por uma ferramenta CASE, além de poder ser usada para comunicação com pessoal não-técnico (clientes etc.). Sua sintaxe XML a torna apta a servir de entrada para geradores automáticos, produzindo a implementação do *website* automaticamente.

A especificação de um *site* em WebML consiste de quatro perspectivas ortogonais:

- Modelo Estrutural (*Structural Model*): expressa os dados do *site*, ou seja, suas entidades e relacionamentos, compatível com notações clássicas como diagramas de Entidades e Relacionamentos ou diagrama de classes da UML;
- Modelo de Hipertexto (*Hypertext Model*): descreve os documentos hipertexto que podem ser publicados no *site*. Cada hipertexto define uma visão do *site*, que é descrita em dois submodelos: de composição (especifica as páginas) e de navegação (especifica o relacionamento entre as páginas);
- Modelo de Apresentação (*Presentation Model*): descreve o leiaute e a aparência gráfica das páginas, independentemente da linguagem final que representará as páginas (HTML, WML etc.);
- Modelo de Personalização (*Personalization Model*): modela explicitamente os usuários e grupos de usuários para armazenamento de informações específicas dos mesmos.

A Figura 2.6 mostra um modelo de hipertexto representado pelos sub-modelos de composição (elementos internos às caixas pontilhadas) e de navegação de páginas (setas de uma caixa pontilhada a outra). Os elementos gráficos nos diagramas de composição representam unidades de dados (cilindro), unidades direcionais (seta) e unidade de índices (linhas). A WebML define diversos outros tipos de unidades, tais como unidades multidados, unidades de rolagem, unidades de filtragem etc.

Informações mais atuais sobre a WebML e uma lista de artigos e outros tipos de documentação podem ser encontradas em http://www.webml.org.

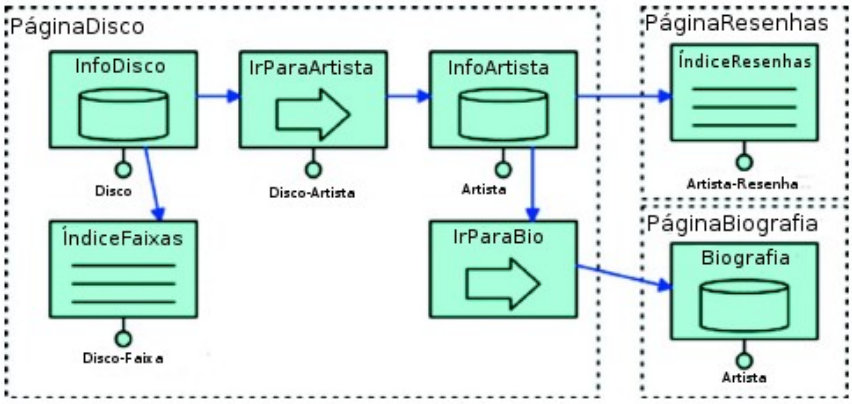

Figura 2.6: Exemplo de diagrama de hipertexto (composição e navegação) em WebML (CERI et al., 2000)

2.2.2. As Extensões de Aplicações *Web*

As Extensões de Aplicações *Web* (*Web Application Extensions* – WAE) (CONALLEN, 2002) são extensões ao meta-modelo da UML para a modelagem de características específicas de aplicações *Web*. A WAE define extensões apropriadas para modelar diversos componentes típicos do desenvolvimento de aplicações *Web*, usando elementos do meta-modelo da UML adicionados de estereótipos pré-definidos para simbolizar os conceitos necessários. Além dos estereótipos, a WAE prevê a definição de valores etiquetados (*tag values*), representados entre colchetes, e restrições (*constraints*), representadas entre chaves, para alguns elementos.

Em relação às extensões propostas para apoiar a elaboração de modelos de análise, merece destaque a notação para o modelo de experiência do usuário (*User Experience* – UX), discutido na Seção 2.1.4. O modelo UX se propõe a definir a aparência de uma aplicação *Web*, seus caminhos de navegação e a estrutura das páginas, sendo composto de dois artefatos principais: mapas de navegação e roteiros.

Mapas de navegação mostram as telas que compõem o sistema. Uma tela é algo que é apresentado ao usuário, contendo uma infra-estrutura padrão de interface *Web* (texto, *links*, formulários etc.) e possuindo nome, descrição, conteúdo (estático, de lógica de negócio ou gerenciado), campos de entrada e possíveis ações a serem executadas. Telas são modeladas como classes estereotipadas. Uma tela comum recebe o estereótipo **`<<screen>>`**, um compartimento de tela (ex.: um frame HTML) é modelado como **`<<screen compartment>>`** e um formulário recebe a denominação **`<<input form>>`**, como ilustra o exemplo da Figura 2.7.

No exemplo dessa figura, uma **`Tela de Login`** possui um texto de abertura como

conteúdo gerenciado (estereótipo **<<managed>>**) por algum sistema de gerência de conteúdo (*Content Management System* – CMS). Nessa tela é possível que o usuário efetue seu *login* no sistema por meio da operação homônima e do **Formulário de Login**, que possui dois campos de texto: **login** e **senha**. A navegação é modelada por associações. Por exemplo, se o cliente for corretamente identificado, segue para a tela **Home**, que possui dois compartimentos: **Menu** e **Tela Inicial**. Esta última possui um texto de boas-vindas, estático (a inclusão de conteúdo estático no modelo é opcional), e exibe a foto do cliente, que é um conteúdo do tipo lógica de negócio, ou seja, proveniente da camada de negócio. Telas podem ter a elas associadas classes normais do domínio do problema, como é o caso da associação entre **Tela Inicial** e **CarrinhoCompras**, simbolizando que os itens do carrinho podem ser exibidos nessa tela.

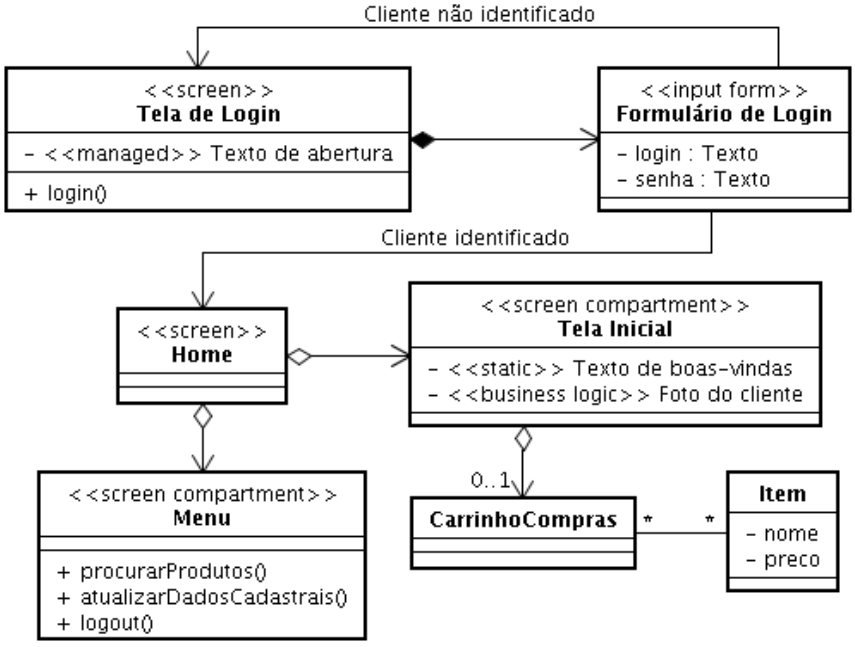

Figura 2.7: Um exemplo de mapa de navegação do modelo de UX

Para cada classe estereotipada, um ícone especial é atribuído, sendo que algumas ferramentas CASE dão apoio a esses ícones especiais, como é o caso da ferramenta Rational Rose[8] da IBM.

Um roteiro, por sua vez, é uma maneira de contar uma história através do uso de imagens estáticas discretas. Um roteiro pode ser capturado por um diagrama de colaboração da UML, por se parecer mais com um roteiro tradicional (teatro, cinema etc.), mas pode também ser modelado por meio de diagramas de seqüência ou de atividade.

8 http://www.ibm.com/software/rational/

A WAE, contudo, é mais voltada para apoiar a elaboração de modelos de projeto, tendo em vista que essa fase é mais suscetível à tecnologia. Conforme discutido na Seção 2.1.4, a fase de projeto é dividida em duas visões: a visão lógica, que está em um nível mais alto de abstração, definindo classes e associações, e a visão de componentes, que trata dos arquivos que efetivamente comporão o sistema implementado (páginas e diretórios).

Para a visão lógica, são definidos três estereótipos principais aplicados sobre o elemento classe do meta-modelo da UML e diversos estereótipos de associação, como mostram as Tabelas 2.2 e 2.3, respectivamente. Tais estereótipos podem ser utilizados para a criação de diagramas de classes que representem os elementos *Web* que compõem o sistema, chegando a um nível de detalhes maior do que os diagramas de classe da fase de análise (pois já incluem informações da plataforma de desenvolvimento), mas ainda num nível de abstração lógico. A Figura 2.8 mostra o diagrama de classes do formulário de *login* usado no exemplo anterior.

Assim como no modelo UX, Conallen sugere ícones especiais para cada estereótipo de classe proposto.

Tabela 2.2: Estereótipos de classe utilizados na visão lógica do projeto

Estereótipo	O que a classe estereotipada representa
<<server page>>	Uma página *Web* dinâmica que efetua processamento no servidor e gera um resultado possivelmente diferente a cada requisição. Suas operações representam funções do *script* e seus atributos representam variáveis visíveis no escopo da página.
<<client page>>	Uma página *Web* estática que é simplesmente retornada ao cliente quando requisitada, sem gerar processamento. Pode conter *scripts* do lado do cliente (*client-side*), portanto seus atributos e operações referem-se a tais *scripts*.
<<HTML form>>	Formulários HTML para entrada de dados. Seus atributos representam os campos do formulário. Tais formulários não possuem operações. Qualquer operação que interaja com o formulário deve ser propriedade da página que o contém.

Além desses elementos básicos, para o que Conallen chama de "Projeto Avançado", existem estereótipos para representação de outros elementos da visão lógica da camada *Web*, a saber: conjunto de quadros (classe com estereótipo **<<frameset>>**), alvo de *link* (classe com estereótipo **<<target>>**), *link* de destino (associação múltipla com estereótipo **<<target link>>**), bibliotecas de *script* (classe com estereótipo **<<script library>>**) e objeto de

script (classe com estereótipo **<<script object>>**).

Para a visão de componentes, são definidos três estereótipos para o elemento de modelo componente da UML: **<<static page>>**, componente que representa páginas estáticas, ou seja, sem processamento no lado do servidor; **<<dinamic page>>**, componente que representa páginas dinâmicas; e **<<physical root>>**, pacote de componentes representando uma abstração de uma hierarquia de arquivos que contém recursos disponíveis à solicitação no servidor *Web*.

Tabela 2.3: Estereótipos de associação utilizados na visão lógica do projeto

Estereótipo	O que a associação estereotipada representa
<<link>>	Um *link* normal entre páginas, representado em HTML pela *tag* **<a>**. Parâmetros codificados como parte da URL segundo o protocolo HTTP podem ser representados como atributos da associação.
<<build>>	Relacionamento direcional entre uma *server page* e uma *client page* que indica que a saída HTML do processamento da página do servidor é a página do cliente.
<<submit>>	Relacionamento direcional entre um *html form* e uma *server page*, representando o envio dos dados dos campos do formulário para a página do servidor para processamento.
<<redirect>>	Ação de redirecionamento de uma página para outra.
<<forward>>	Ação de delegação de uma página para outra. Difere do redirecionamento por manter o contexto da requisição feita à primeira página.
<<object>>	Relacionamento de confinamento entre uma página e um objeto, como uma Applet ou controle ActiveX.
<<include>>	Uma associação direcional entre uma *server page* e uma outra página que indica que o conteúdo desta última será processado (somente no caso de uma página do servidor) e incluído na primeira.

Por meio de diagramas de componentes, a visão de componentes busca modelar o mapeamento entre os arquivos físicos (representados pelos componentes com os três estereótipos citados) e as representações lógicas apresentadas na visão lógica (representadas por classes estereotipadas, conforme discutido anteriormente). A Figura 2.9 mostra um diagrama de componentes com os arquivos que fisicamente implementam as páginas lógicas

de *login* da Figura 2.8. A tela de login e o formulário são ambos implementados pela página estática **`index.htm`**, enquanto toda a parte dinâmica, desde o processamento do *login* até a geração das páginas do cliente **`Home`** ou **`ErroLogin`** é feita pela página dinâmica **`processaLogin.php`**.

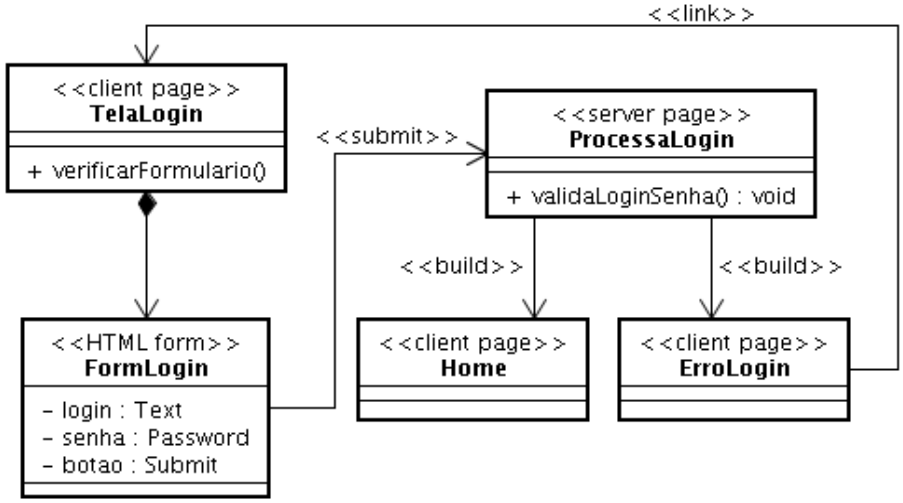

Figura 2.8: Diagrama de classes de visão lógica da camada *Web* do sistema

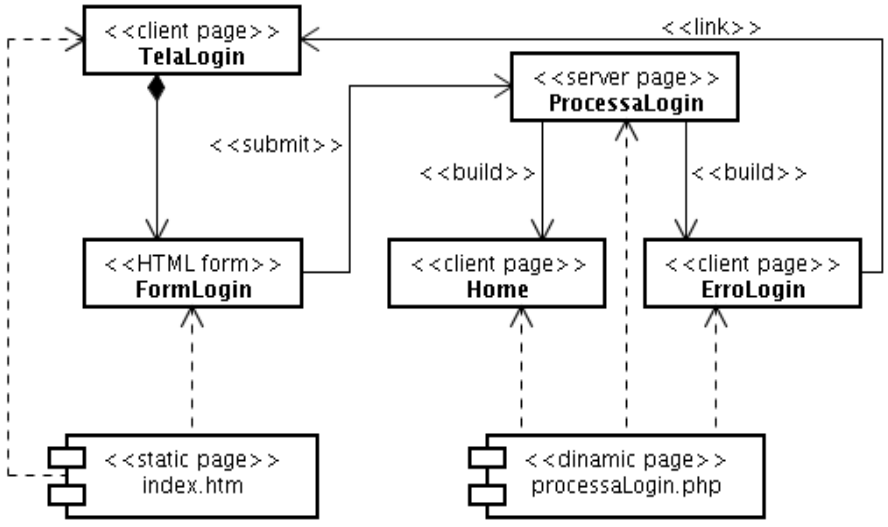

Figura 2.9: Diagrama de componentes ligando a visão lógica aos componentes físicos

Além dos estereótipos básicos, para o projeto avançado, Conallen define também: biblioteca de *script* (componente com estereótipo **`<<script library>>`**), raiz virtual (pacote com estereótipo **`<<virtual root>>`**), recurso HTTP (componente com estereótipo **`<<HTTP resource>>`**), biblioteca de *tags* JSP (pacote com estereótipo **`<<JSP tag library>>`**) e *tag* JSP (classe com estereótipo **`<<JSP tag>>`**), sendo esta última composta por um elemento que

representa o corpo da *tag* (dependência com estereótipo **<<tag>>**) e opcionalmente um elemento que reúne informações extras (dependência com estereótipo **<<tei>>** - *tag extra info*).

2.2.3. A linguagem de modelagem da UWE

A linguagem de modelagem associada à metodologia Engenharia Web Baseada em UML (*UML-based Web Engineering* – UWE) (KOCH et al., 2000) (KOCH e KRAUS, 2002) utiliza como base a UML e seus mecanismos de extensão (estereótipos, valores etiquetados e restrições), produzindo um perfil UML (*UML Profile*) específico para a metodologia. Segundo seus autores, a notação é leve, no sentido de ser facilmente apoiada por ferramentas e não causar impacto nos formatos de troca de dados.

Esse perfil UML é utilizado na construção de modelos de análise e projeto dentro da metodologia UWE para o desenvolvimento de *WebApps*, descrita na Seção 2.1.6. A Figura 2.10 mostra um modelo de estrutura de navegação de uma biblioteca virtual, exibindo os seguintes elementos do perfil UML da UWE:

- **Classe de navegação** (estereótipo **<<navigational class>>**): representa um conceito do domínio do problema, cujas instâncias são visíveis aos usuários durante a navegação;
- **Navegação direta** (associação com indicação de navegabilidade): indica a possibilidade de navegação de um elemento de navegação a outro;
- **Índice** (estereótipo **<<index>>**): é um objeto composto que contém um número qualquer de itens de índice, ou seja, objetos que possuem um nome e que fazem ligação com alguma classe de navegação;
- **Passeio guiado** (estereótipo **<<guidedTour>>**): consiste em um objeto que provê acesso seqüencial às instâncias de uma classe de navegação;
- **Consulta** (estereótipo **<<query>>**): possui uma frase de consulta como atributo e é utilizado para selecionar instâncias de classes de navegação mediante um determinado filtro embutido na consulta;
- **Menu** (estereótipo **<<menu>>**): representa um número fixo de itens de menu que fazem ligação com uma classe de navegação, índice, passeio guiado ou consulta.

Estes são somente alguns dos elementos definidos pelo perfil UML da UWE, que inclui vários outros, como contexto, classe de apresentação, quadro, conjunto de quadros, janela, texto, âncora, botão, imagem, áudio, vídeo, formulário etc. Muitos destes representam

conceitos bastante conhecidos para desenvolvedores de páginas *Web*. A definição de todas essas extensões vem acompanhada de ícones especiais que ferramentas CASE podem escolher dar apoio (KOCH et al., 2001).

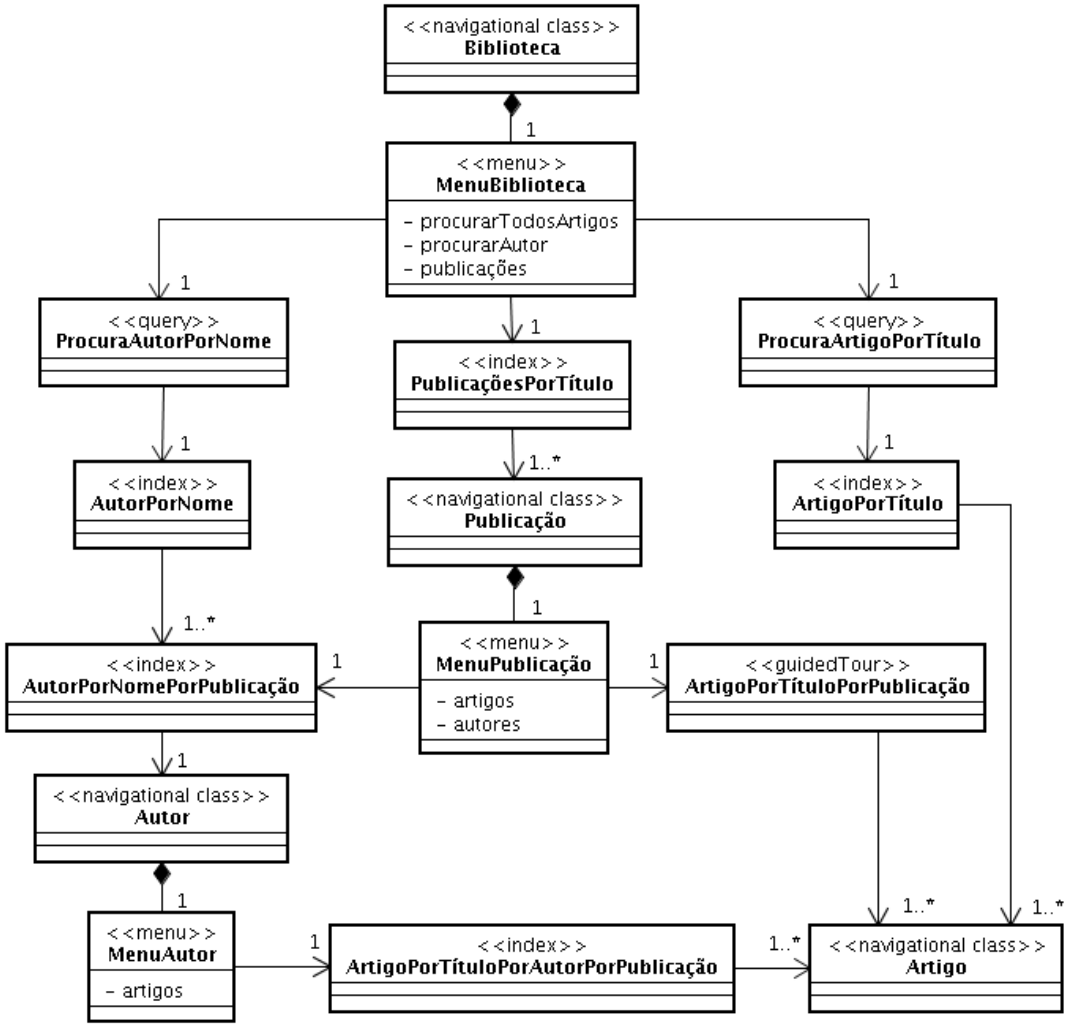

Figura 2.10: Exemplo de modelo de estrutura de navegação em UWE (KOCH et al, 2000)

2.3. *Frameworks* para desenvolvimento *Web*

Na área de pesquisa de Engenharia *Web*, existem diversas propostas de metodologias, linguagens e ferramentas que alcançam várias atividades do processo de desenvolvimento de um *WIS*. No entanto, se separássemos as propostas por atividade, provavelmente veríamos que a atividade de implementação possui o maior número de propostas, reunindo um amplo conjunto de ferramentas que visam facilitar a atividade de codificação de um sistema de informação *Web*.

Os WISs 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.

A união de diversas soluções de infra-estrutura (*frameworks*) dá origem ao que denominamos "arquiteturas baseadas em *containers*". Um *container* é um sistema que gerencia objetos que possuem um ciclo de vida bem definido. Um *container* para aplicações distribuídas, como os *containers* da plataforma Java *Enterprise Edition* (SHANNON, 2003), gerenciam objetos e lhes prestam serviços, como persistência, gerência de transações, distribuição, serviços de diretórios, dentre outros.

A partir de nossa experiência no desenvolvimento de WISs utilizando a plataforma Java, nos deparamos com diversos desses *frameworks* e pudemos organizá-los em seis categorias diferentes, a saber:

- *Frameworks* MVC (Controladores Frontais);
- *Frameworks* Decoradores;
- *Frameworks* de Mapeamento Objeto/Relacional;
- *Frameworks* de Injeção de Dependência (Inversão de Controle);
- *Frameworks* para Programação Orientada a Aspectos (AOP);
- *Frameworks* para Autenticação e Autorização.

Nas subseções seguintes, explicamos cada uma dessas categorias e sua influência na arquitetura de um sistema de informação *Web*, citando exemplos de *frameworks* de código-aberto ou gratuitos que podem ser utilizados, caso a linguagem Java seja a plataforma escolhida para implementação. Concluímos com uma breve análise das arquiteturas baseadas em *containers* e sua relação com os *frameworks* já citados.

2.3.1. Frameworks MVC (Controladores Frontais)

MVC 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 2.11 mostra como funciona esse padrão arquitetural.

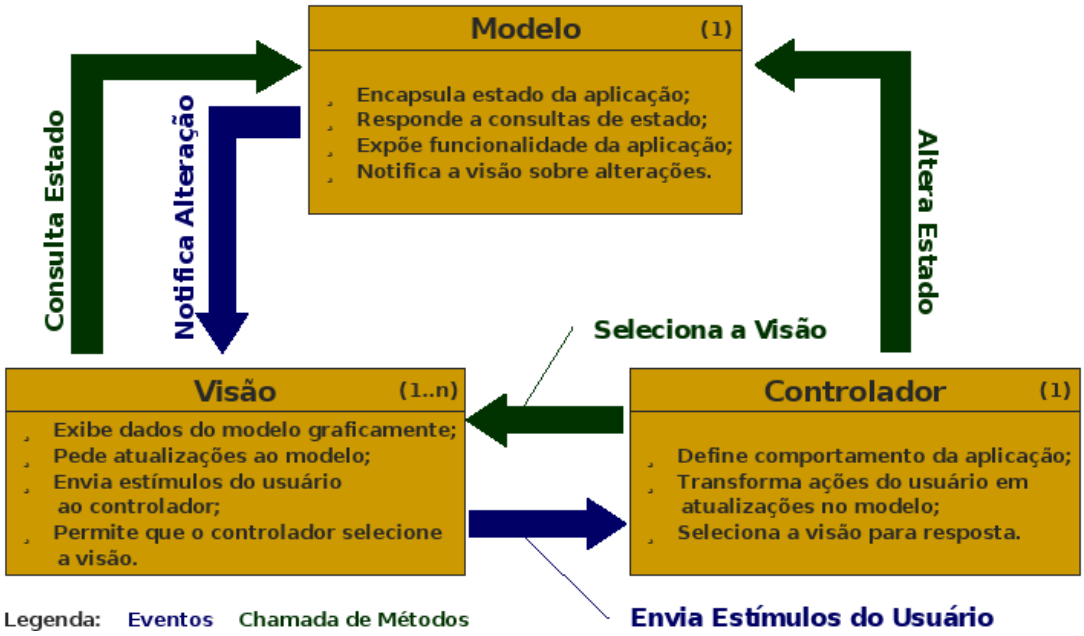

Figura 2.11: Funcionamento do padrão Modelo-Visão-Controlador

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.

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 2.12.

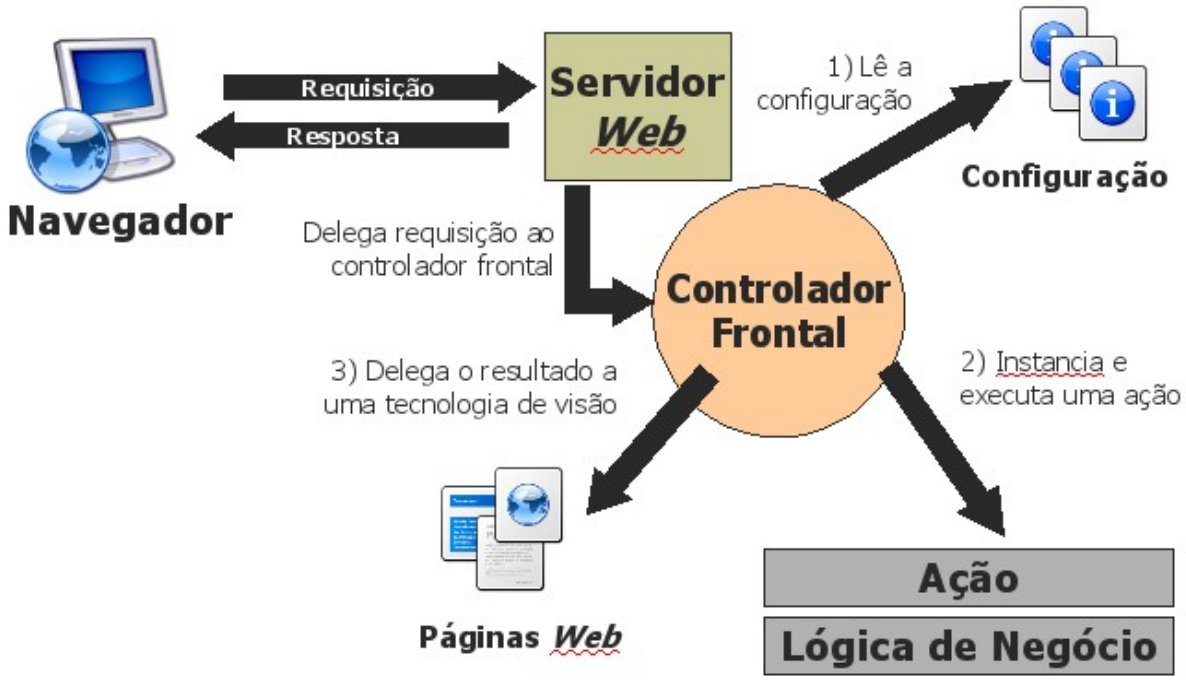

Figura 2.12: Funcionamento do padrão arquitetônico "Controlador Frontal"

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 construção 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.

Note que o padrão "Controlador Frontal" é aplicado apenas à camada *Web* da aplicação, que possivelmente pode ter sido dividida em outras camadas arquitetônicas. Se tomarmos a

arquitetura definida por Coad & Yourdon (COAD e YOURDON, 1993), dentre as quatro camadas por eles definidas, Domínio do Problema (CDP), Gerência de Tarefas (CGT), Gerência de Dados (CGD) e Interação Humana (CIH), a camada *Web* (e, portanto, toda a arquitetura MVC apresentada) encontra-se inserida na CIH. Desta maneira, é responsabilidade das classes de ação, e não mais das páginas *Web*, comunicarem-se com as classes da CGT para execução das tarefas (casos de uso), manipulando os objetos da CDP para exibição nas páginas *Web*. Tal organização de código evita que a lógica de negócio seja espalhada em *scripts* embutidos em páginas *Web*, aumentando a modularidade, coesão e manutenibilidade do código.

Somente para a plataforma Java, podemos encontrar mais de 50 *frameworks* MVC de código aberto implementados. Os mais notáveis são:

- WebWork (http://www.opensymphony.com/webwork);
- Struts (http://struts.apache.org/1.x/index.html);
- Struts2, união de Struts e WebWork (http://struts.apache.org/2.x/index.html);
- Spring MVC (http://www.springframework.org);
- VRaptor2 (projeto brasileiro – http://vraptor2.sourceforge.net);
- Wicket (http://wicket.sourceforge.net).

2.3.2. *Frameworks* Decoradores

Frameworks Decoradores surgiram para automatizar a trabalhosa tarefa de manter toda uma aplicação *Web* com o mesmo visual, ou seja, cabeçalho, rodapé, barra de navegação, esquema de cores e demais elementos gráficos de leiaute integrados num mesmo projeto de apresentação, elaborado pela equipe de *Web Design*.

Esse tipo de *framework* funciona como o padrão de projeto “Decorador” (*Decorator*) (GAMMA et al., 1994), se posicionando como um filtro entre uma requisição do cliente e um servidor *Web*, como mostra a Figura 2.13.

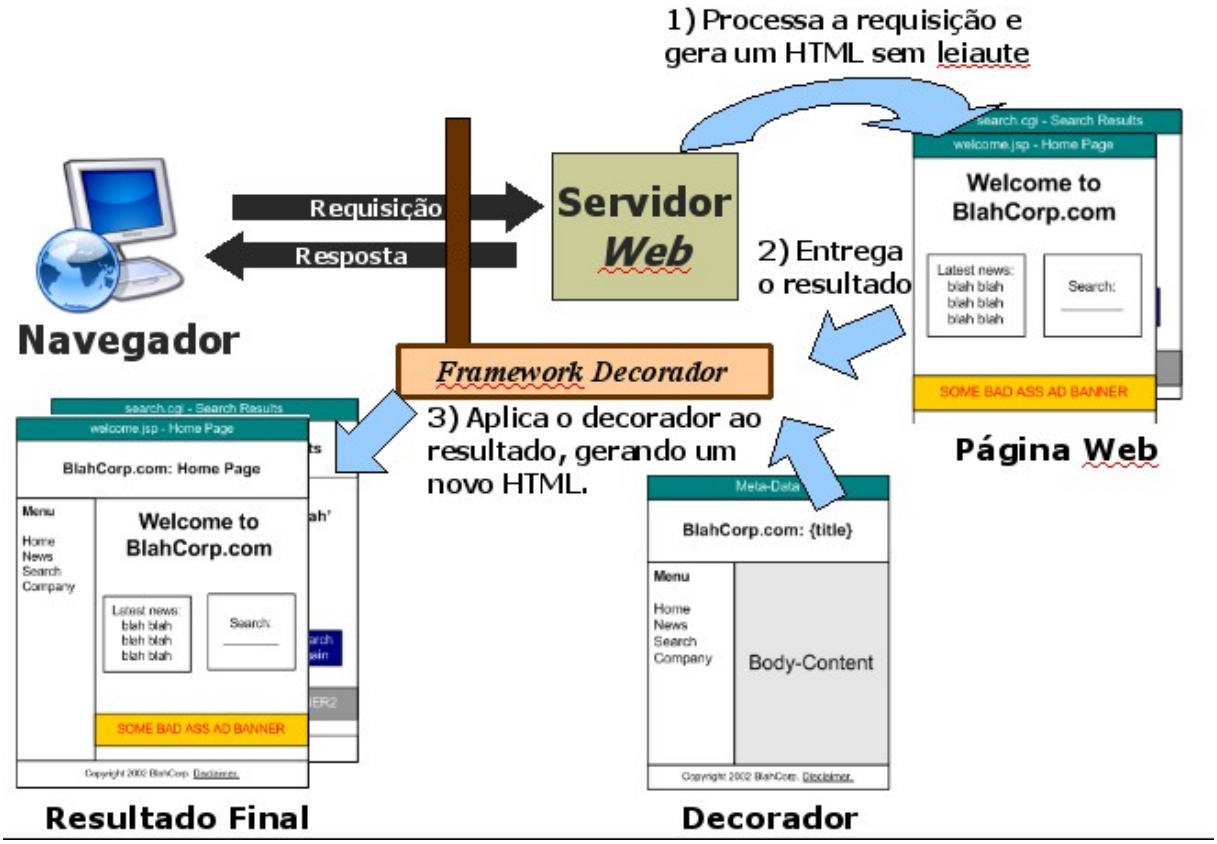

Figura 2.13: Funcionamento de um *Framework* Decorador

Tal solução é superior a outras comumente encontradas, como o uso de inclusão de páginas ou replicação do código HTML responsável pelo desenho do *site* em todas as páginas, pois além de diminuir custos de manutenção no caso da troca do leiaute, permite a seleção dinâmica do decorador, de forma a facilitar a implementação de versões alternativas das páginas, como, por exemplo, uma versão para impressão.

Dentre os *Frameworks* Decoradores livres para a plataforma Java, destacamos:

- SiteMesh (http://www.opensymphony.com/sitemesh);
- Tiles (http://struts.apache.org/struts-tiles).

2.3.3. *Frameworks* de Mapeamento Objeto/Relacional

Sistemas de Bancos de Dados Relacionais (SGBDR) têm sido há décadas o padrão de fato para armazenamento de dados. Por contarem com uma base teórica bastante formal (álgebra relacional) e uma indústria forte por trás, não há indicações de que esse cenário vá mudar tão cedo. Por conseguinte, mesmo aplicações desenvolvidas no paradigma orientado a objetos (OO) têm utilizado bancos de dados relacionais para persistência de seus objetos.

Tal combinação produz o que Christian Bauer e Gavin King chamam de

"incompatibilidade de paradigmas" (*paradigm mismatch*), visto que "operações de SQL [linguagem estruturada de consultas para SGBDRs] como projeção sempre unem resultados em uma representação em tabelas de dados resultantes. Isso é bem diferente do grafo de objetos interconectados usados para executar uma lógica de negócio em um aplicativo Java!" (BAUER e KING, 2005). Tal incompatibilidade se manifesta no uso de conceitos OO, como herança, identidade, associação e navegação pelo grafo de objetos.

Dentre as diversas opções para tratar esse problema, surgiu na década de 1980 a idéia do Mapeamento Objeto/Relacional (*Object/Relational Mapping* – ORM), que vem ganhando muita aceitação desde o final da década de 1990. ORM é a persistência automática (e transparente) de objetos de um aplicativo OO para tabelas de um banco de dados relacional, utilizando meta-dados que descrevem o mapeamento entre o mundo dos objetos e o mundo relacional (BAUER e KING, 2005).

Em outras palavras, ao invés de obter os dados dos objetos e mesclá-los a uma *string* de consulta SQL que será enviada ao SGBDR, o desenvolvedor deve informar ao *framework* de Mapeamento Objeto/Relacional como transformar objetos e seus atributos em tabelas e colunas e chamar métodos simples, como `salvar()`, `excluir()` e `recuperarPorId()`, disponíveis no *framework*. Tal idéia está representada na Figura 2.14. Em geral, esses *frameworks* disponibilizam também uma linguagem de consulta similar à SQL, porém orientada a objetos, para que consultas possam ser realizadas com igual facilidade.

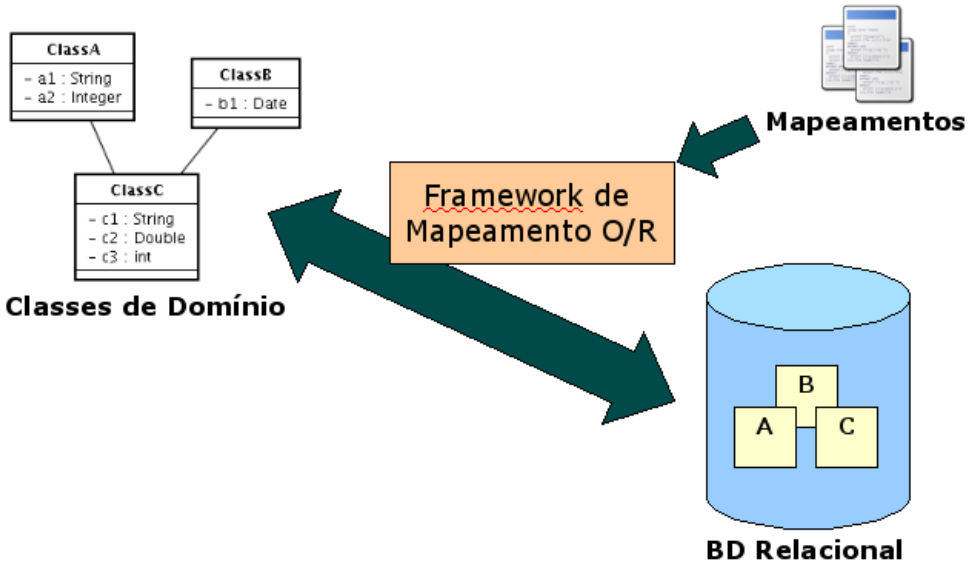

Figura 2.14: Funcionamento de um *framework* de mapeamento O/R.

A idéia já vem sendo adotada na plataforma Java há alguns anos e hoje em dia existem diversos *frameworks* maduros para ORM, cada vez mais se firmando como boas opções para

todos os tipos e tamanhos de aplicativos. Dentre eles, destacamos:

- Hibernate (http://www.hibernate.org);
- Java Data Objects – JDO (http://java.sun.com/products/jdo);
- Apache ObjectRelationalBridge – OJB (http://db.apache.org/ojb/);
- Oracle Toplink (http://www.oracle.com/technology/products/ias/toplink).

Naturalmente, *Frameworks* ORM não são exclusividade de sistemas de informação *Web*, podendo ser utilizados em diversas plataformas diferentes.

2.3.4. *Frameworks* de Injeção de Dependência (Inversão de Controle)

O paradigma orientado a objetos trouxe conceitos como modularidade e coesão como guias para o desenvolvimento de aplicações bem estruturadas. Existem inúmeras formas de dividir uma arquitetura em camadas, mas que, de forma geral, se assemelham à arquitetura de Coad e Yourdon (COAD e YOURDON, 1993), já citada anteriormente.

Classes relacionadas ao domínio do problema (representando conceitos do mundo real – CPD), à gerência de tarefas (implementando casos de uso – CGT), à gerência de dados (responsáveis pela persistência dos objetos – CGD) e à interação humana (interface com o usuário – CIH) ficam em camadas diferentes. No entanto, elas precisam estar integradas, pois ações do usuário na CIH devem acionar casos de uso implementados na CGT, que manipulam objetos da CDP que, por sua vez, possivelmente, são armazenados de forma persistente pela CGD.

Segundo Fowler (2006), quando criamos classes cujas instâncias dependem de objetos de outras classes para a realização de um determinado serviço, é interessante que estejam relacionadas apenas às interfaces dos objetos das quais dependem, e não a uma implementação específica daquele serviço. Esse aspecto também é tratado por GAMMA et al. (1994) com seus padrões de projeto de criação, como "Método Fábrica" (*Factory Method*), "Fábrica Abstrata" (*Abstract Factory*) e Construtor (*Builder*).

Por exemplo, se uma classe de gerência de tarefas, que implementa um determinado caso de uso, necessita de uma classe de gerência de dados para que esta realize a persistência de um determinado objeto, a classe da CGT não precisa saber como a classe da CGD realizará a persistência, que pode ser comunicando-se diretamente com um SGBDR, utilizando um *framework* ORM, convertendo dados em arquivos XML etc. Ela precisa saber somente que, ao chamar um método cuja assinatura é `salvar(objeto)`, o objeto passado como parâmetro será gravado em mídia persistente. Em outras palavras, precisa conhecer a interface do objeto da CGD.

Seguindo essa abordagem, surgiram vários *frameworks* de Injeção de Dependência (*Dependency Injection* – DI), também conhecidos como *frameworks* de Inversão de Controle (*Inversion of Control* – IoC). O objetivo deles é permitir que o desenvolvedor especifique por meio de meta-dados todas as dependências entre classes que existem em seu sistema e, ao obter instâncias dessas classes por meio do *framework*, todas as suas dependências já tenham sido satisfeitas ou, em outras palavras, "injetadas". O termo IoC é também utilizado pelo fato do controle de criação de objetos ter sido invertido, saindo da classe que depende diretamente do objeto a ser criado para um elemento externo, no caso, o *framework*.

A Figura 2.15 ilustra o funcionamento de um *framework* de Injeção de Dependências.

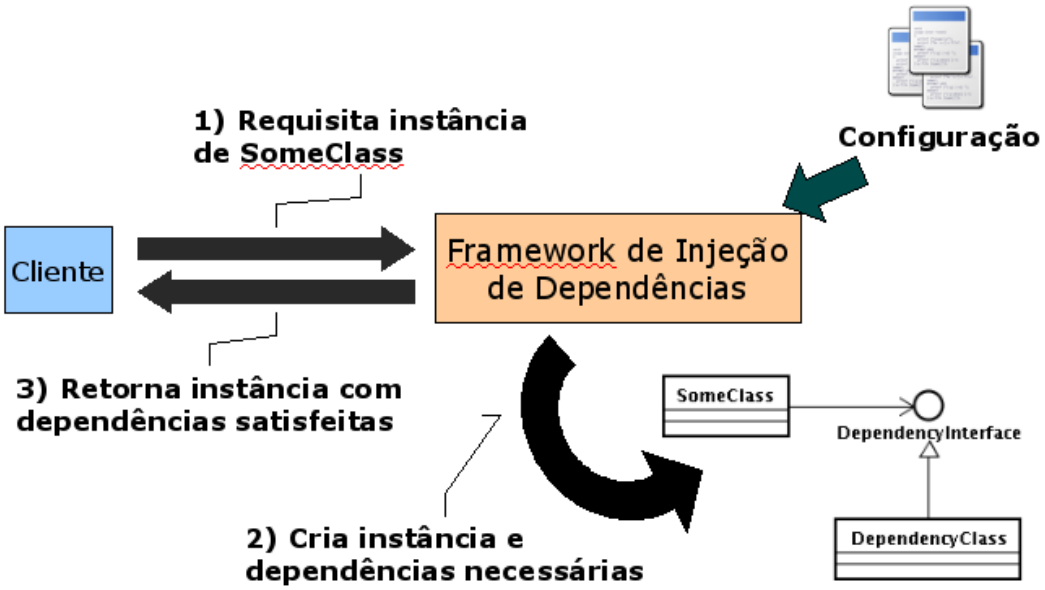

Figura 2.15: Funcionamento de um *framework* de Injeção de Dependências.

Destacam-se no universo de *frameworks* de Injeção de Dependência livres em Java os seguintes:

- Spring Framework (http://www.springframework.org);
- PicoContainer (http://www.picocontainer.org);
- Apache Excalibur (http://excalibur.apache.org);
- Apache HiveMind (http://jakarta.apache.org/hivemind).

Assim como os *Frameworks* ORM, *Frameworks* IoC também podem ser utilizados em diversos tipos de plataforma, apesar de se integrarem mais suavemente a aplicativos que executam dentro de *containers*, como é o caso dos WISs.

2.3.5. *Frameworks* para Programação Orientada a Aspectos (AOP)

A Orientação a Aspectos é considerada por muitos uma evolução da Orientação a

Objetos que complementa esta última de modo a simplificar o desenvolvimento de sistemas, permitindo que softwares cada vez mais complexos possam ser criados e mantidos com menor esforço. Assim como seu predecessor, esse novo paradigma abrange todo o processo de software, porém sua maior influência tem sido na fase de codificação, em torno da qual surge a Programação Orientada a Aspectos (*Aspect Oriented Programming* – AOP).

O paradigma da Orientação a Aspectos está centrado na idéia da separação de preocupações (*separation of concerns*). Segundo Resende e Silva (2005), separação de preocupações pode ser definida como "a teoria que investiga como separar as diversas preocupações pertinentes a um sistema, propiciando que cada uma das preocupações seja tratada separadamente, a fim de reduzir a complexidade do desenvolvimento, evolução e integração do software".

Em termos práticos, podemos citar como exemplos os sistemas que têm como preocupações serem distribuídos, acessarem bancos de dados ou registrarem suas ações em relatórios (*logs*). Com a programação orientada a objetos, tais tarefas demandariam que trechos de código relacionados a essas tarefas de infra-estrutura fossem repetidos (ou em terminologia AOP: espalhados) pelos vários módulos da aplicação, algo que pode ser feito automaticamente por uma ferramenta orientada a aspectos. A Figura 2.16 ilustra essa situação.

No quadro superior, vemos um código orientado a objetos no qual as classes de aplicação necessitam realizar tarefas relacionadas ao acesso ao banco de dados: iniciar a transação antes de realizar suas operações e confirmá-la (*commit*) ou cancelá-la (*rollback*) ao final. Com o uso da AOP, essas tarefas idênticas, que se encontram espalhadas pelo código, denominadas aspectos, são retiradas dos vários módulos do sistema e implementadas uma só vez, num só lugar, como demonstra o quadro intermediário. Por fim, o quadro inferior explica o papel do *framework* AOP: interceptar a requisição aos métodos que necessitam do aspecto relacionado ao banco de dados e garantir que o código seja executado antes e depois do método, resultando na mesma execução que havíamos anteriormente, porém com o código mais bem estruturado, sem repetições.

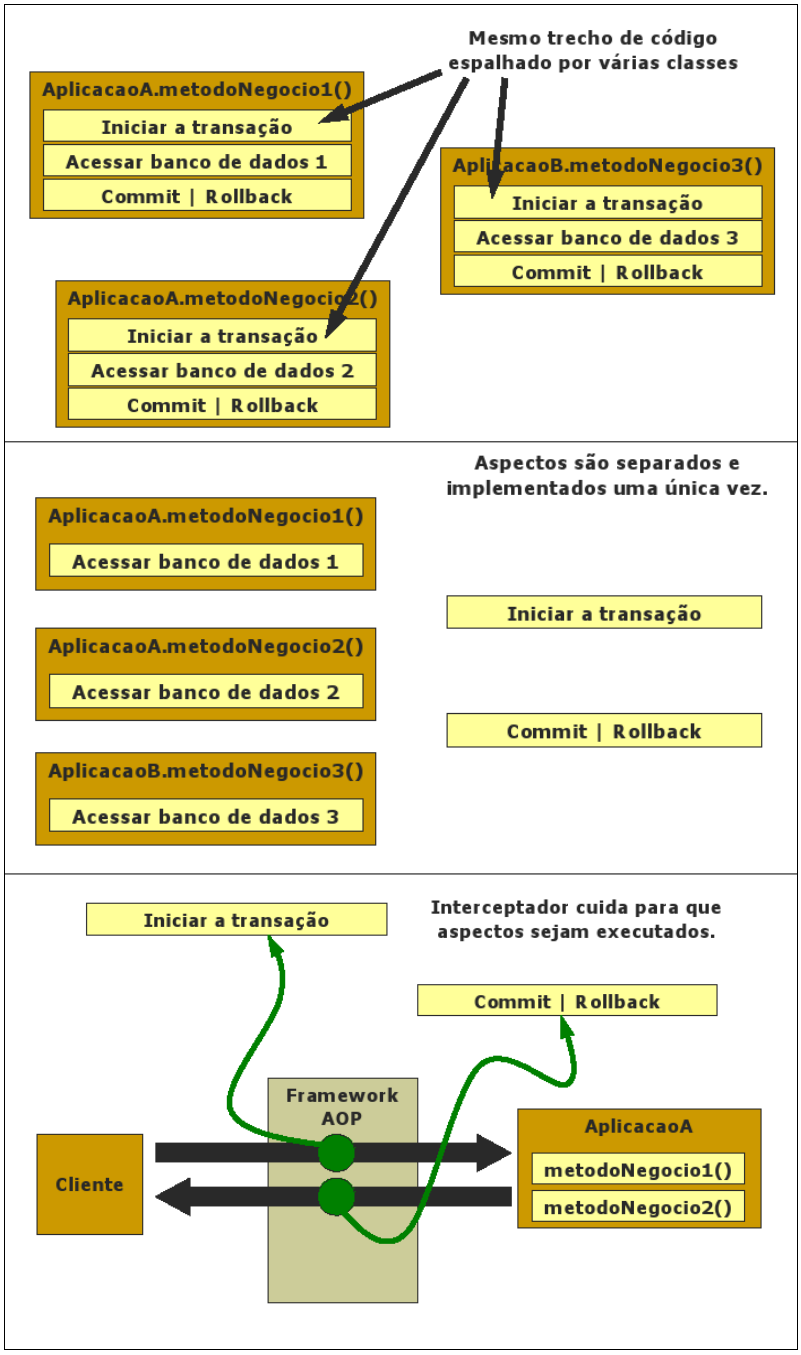

Figura 2.16: Funcionamento de um *Framework* AOP

Esse espalhamento automático dos aspectos é chamado de "entrelaçamento" ou "costura" (*weaving*) e pode ser feito em tempo de execução por um *framework*, como descrevemos, ou em tempo de compilação, por um compilador AOP. Neste segundo caso, o código compilado é o resultado da união do código-fonte original e o código dos aspectos. Em ambos os casos, o desenvolvedor precisa informar à ferramenta em quais pontos do código – chamados de "pontos de corte" (*pointcuts*) – determinados aspectos devem ser entrelaçados.

A popularidade de ferramentas relacionadas à AOP cresce a cada dia, e dentre as

ferramentas de código aberto que podem ser utilizadas na plataforma Java, destacamos:

- Spring Framework (http://www.springframework.org);
- AspectJ (http://www.eclipse.org/aspectj);
- AspectWerkz (http://aspectwerkz.codehaus.org);
- JBoss AOP (http://labs.jboss.com/portal/jbossaop).

2.3.6. Frameworks para Autenticação e Autorização

Uma outra característica que é bastante comum em sistemas de informação *Web* é a presença de mecanismos de segurança para autenticação e autorização. Autenticar consiste em verificar se uma chave de acesso (geralmente um par login / senha) é válida para o acesso a uma aplicação. Autorizar, por sua vez, consiste em verificar qual é o nível de acesso do usuário autenticado, permitindo que ele use somente as funcionalidades autorizadas para o seu nível.

Como é de se esperar, existem *frameworks* que realizam essas duas tarefas mediante configuração em meta-dados. Tais ferramentas apóiam diversos tipos de autenticação (bancos de dados, arquivos, servidores de diretório etc.) e autorização (geralmente via meta-dados e mapeamento de URLs).

Os *frameworks* de Autenticação e Autorização de código aberto e para a plataforma Java mais conhecidos são:

- Acegi Security for Spring (http://www.acegisecurity.org);
- Apache Cocoon Authentication (http://cocoon.apache.org);
- Java Authentication and Authorization Services – JAAS, utilizado pelos servidores de aplicação Java EE (http://java.sun.com/products/jaas).

2.3.7. Arquiteturas baseadas em *containers*

Antes da popularização de *frameworks* como Hibernate, Spring e Struts, o desenvolvimento de aplicações com necessidades de distribuição, escalabilidade e robustez era guiado por padrões como a plataforma Java *Enterprise Edition* (Java EE) (SHANNON, 2003). Tais padrões definem serviços que devem ser prestados por *containers*, que funcionam como servidores gerenciando objetos projetados pelo desenvolvedor e provendo serviços diversos, como gerência de persistência, gerência de transações, serviço de diretório, acesso a sistemas legados, dentre outros.

O surgimento de uma grande quantidade de *frameworks* que realizam um serviço já provido por *containers*, porém de forma independente, se deu em grande parte por

deficiências nas especificações que, dentre outros problemas, tornava burocrática a tarefa de construção de um componente, impunha uma série de restrições para os mesmos e os tornava muito dependente dos *containers*, impedindo que fossem reutilizados em outros contextos (JOHNSON e HOELLER, 2004).

Com a adoção dos *frameworks* por grande parte dos desenvolvedores, novos padrões para arquiteturas baseadas em *containers* foram adequando-se às preferências dos usuários. A versão 5.0 da plataforma Java EE (SHANNON, 2006), lançada recentemente, tem como arquitetura *Web* um padrão semelhante à arquitetura MVC (JavaServer Faces), define seu modelo de persistência de componentes (Enterprise JavaBeans) com base no *framework* ORM Hibernate e faz uso da técnica de Injeção de Dependências para satisfazer dependências entre componentes gerenciados pelo *container*.

Grande parte das aplicações *Web* de médio a grande porte utilizam *frameworks* ou *containers* cujos serviços são baseados em idéias similares a dos *frameworks*.

2.4. Outros trabalhos relacionados

Durante a pesquisa realizada, foram encontrados alguns trabalhos relacionados com o desenvolvimento de aplicações *Web* que, contudo, não se enquadravam propriamente como metodologias, *frameworks* ou linguagens de modelagem. Tais trabalhos são citados brevemente na seqüência.

Em (SCHARL, 2001), o autor examina o papel da modelagem conceitual de sistemas *Web* como o meio principal e padronizado de comunicação visual entre organizações e departamentos de uma mesma organização. Ele apresenta a Técnica de Projeto Estendida da *Web* (extended *World Wide Web Design Technique – eW3DT*) como uma linguagem consistente e semanticamente rica para troca de conhecimento sobre projetos em andamento ou já implantados.

Em (TAM et al., 2000), um *framework* de linguagem de script para o desenvolvimento rápido e sistemático de aplicações *Web* é proposto. O trabalho incluiu o desenvolvimento de um protótipo de um analisador de scripts e um ambiente integrado de desenvolvimento (*Integrated Development Environment – IDE*) para desenvolvimento rápido de aplicações *Web*.

Uden (2002) propõe um modelo de projeto para desenvolvimento de *WebApps* baseado em projeto de interface com o usuário orientado a objetos e ciências cognitivas. A técnica de Análise de Tarefas Cognitiva Aplicada (*Applied Cognitive Task Analysis – ACTA*) é apĺicada

no levantamento de requisitos e produz resultados que são usados no projeto de interface gráfica.

Richardson (2000) discute o uso de métricas simples para pequenos sistemas desenvolvidos para a *Web*, focando em como uma análise dessas informações pode auxiliar na reconstrução do projeto da aplicação de forma mais adequada. As métricas sugeridas pelo artigo incluem contadores de acesso, livros de visitas e analisadores de clientes.

Ginige e Murugesan (2001b) exploram práticas de desenvolvimento de sistemas *Web* e apresentam perspectivas multidisciplinares que podem ajudar a melhorar essa área. Os autores resumem dez fatores-chave para o desenvolvimento bem sucedido de *WebApps*.

Novos desafios para a colaboração no desenvolvimento de aplicações *Web* é o tema de (VOGELSAN e CARTENSEN, 2001), que apresenta resultados preliminares de um projeto de desenvolvimento de um sistema *Web* em andamento, discutindo o que os autores consideram como um dos maiores desafios da área: a cooperação entre grupos heterogêneos no desenvolvimento de aplicações.

Scott (2003) explora diversas práticas de gerência e planejamento estratégico, mostrando sua aplicabilidade para a área de desenvolvimento de sistemas *Web*.

Por fim, Chang et al. (2004) apresentam um *framework* para o desenvolvimento de aplicações *Web* adaptáveis e independentes de plataforma. Os autores justificam que, devido à grande quantidade de sistemas operacionais e navegadores *Web* (*browsers*) com características diferentes, se faz necessária uma maneira de construir um sistema independente de ambiente de execução, que pode ser convertido automaticamente por um tradutor para diversos ambientes diferentes.

Além desses trabalhos, também não nos aprofundamos em métodos hipermídia por considerá-los uma categoria de propostas fora do escopo deste trabalho. Métodos hipermídia focam mais nas páginas *Web* e sua navegação ao invés de funcionalidades, sendo mais adequados para aplicações *Web* com foco em conteúdo. O método OOHDM, descrito na Seção 2.1.7, por ser o mais citado representante dessa categoria de métodos, foi o único incluído nesta revisão bibliográfica.

2.5. Considerações finais do capítulo

Como podemos ver, a quantidade de propostas para a área de Engenharia *Web* é bastante vasta, o que demonstra que acadêmicos e profissionais da área de Engenharia de Software ainda não elegeram uma metodologia ou linguagem de modelagem padrão para o

desenvolvimento de aplicações *Web*. Além disso, não há nenhuma indicação de que isso ocorrerá nos próximos anos, dado que existem várias propostas para contextos específicos como desenvolvimento baseado em componentes (a proposta de Lee & Shirani – Seção 2.1.2), desenvolvimento baseado em prototipação (MODFM – Seção 2.1.1) ou desenvolvimento de aplicações B2C (ICDM – Seção 2.1.8).

Apesar de notarmos um crescimento na utilização de *frameworks* ou arquiteturas baseadas em *containers* para o desenvolvimento de WISs, não encontramos nenhuma abordagem voltada para essa categoria de *WebApps*. Esse contexto motivou-nos a propor um novo método, apresentado no Capítulo 4. Ao final deste, na Seção 4.4, comparamos nossa proposta com outras que não se enquadrem em contextos específicos, como as que acabamos de citar, para justificar a necessidade de um novo método para o projeto de sistemas de informação *Web*.

Nossa proposta, no entanto, se estende também à área da *Web* Semântica, sugerindo uma abordagem para construção de WISs preparados para o que por muitos é considerado o futuro da Internet. Para fundamentarmos nossa discussão sobre este assunto, apresentamos uma revisão dos conceitos e trabalhos desta área no capítulo seguinte.

Capítulo 3: A *Web* Semântica

A *Web* Semântica (*Semantic Web*) é considerada o futuro da Internet, mais especificamente uma evolução da *World Wide Web* atual, referida pelo termo "*Web* Sintática". Nesta última, "os computadores fazem apenas a apresentação da informação, porém o processo de interpretação fica a cabo dos seres humanos mesmo" (BREITMAN, 2006). O objetivo da *Web* Semântica é permitir que softwares (agentes) instalados nos computadores sejam capazes de interpretar o conteúdo de páginas da *Web* para auxiliar humanos a realizarem suas tarefas diárias na rede.

Este capítulo introduz conceitos da *Web* Semântica (Seção 3.1) e discute áreas estreitamente relacionadas com a sua materialização, como ontologias (Seção 3.2), agentes (Seção 3.3), diferentes abordagens para a adição de semântica às aplicações *Web* (Seção 3.4) e linguagens usadas na *Web* Semântica (Seção 3.5).

3.1. A *Web* Semântica

Berners-Lee, Hendler & Lassila publicaram na revista *Scientific American* em 2001 um artigo no qual apresentavam cenários de utilização da *Web* no futuro (BERNERS-LEE et al., 2001). Em um desses cenários, uma mulher, chamada Lucy, precisava marcar uma consulta com um ortopedista para sua mãe, além de uma série de sessões de fisioterapia. Diversos critérios deveriam ser atendidos para a marcação desses compromissos: (a) ser em um horário que Lucy e seu irmão estivessem livres para levar sua mãe; (b) ser, de preferência, em um local próximo à casa dela; e (c) contar com profissionais qualificados e que atendessem pelo plano de saúde de sua mãe. Na *Web* Semântica, agentes no computador portátil de Lucy iriam auxiliá-la nessa tarefa, realizando as seguintes ações:

1. Requisitar ao agente do médico que prescreveu a consulta e as sessões as características do tratamento prescrito;
2. Procurar em catálogos de serviços médicos uma lista de ortopedistas e fisioterapeutas, selecionando aqueles que atendessem pelo plano de saúde de sua mãe e, de preferência, que possuíssem consultório num raio de 20 milhas de sua casa;
3. Solicitar a um serviço de classificação de profissionais de saúde quais dos profissionais de sua lista estão classificados como excelentes ou muito bons;

4. Tentar casar os horários da agenda de Lucy e de seu irmão com os horários vagos dos profissionais que atenderam a todos os critérios, consultando os agentes ou *sites* dos médicos.

Hoje em dia é possível a Lucy realizar todas essas tarefas manualmente, porém seria difícil conceber um agente que pudesse auxiliá-la a realizar automaticamente essas e outras tarefas similares. O que impede que tais softwares de auxílio pessoal sejam construídos é o fato de que as páginas *Web* são escritas para interpretação por seres humanos, não descrevendo seu conteúdo de forma a permitir que agentes de software analisem-nas e raciocinem sobre as informações ali descritas.

Analogamente, considere que um estudante de Informática, sem uma sólida formação em química e biologia, depare-se com a seguinte página na Internet:

> *Proteína é uma macromolécula cujos monómeros são aminoácidos. Parte constituínte dos tecidos biológicos, muitas funcionam como enzimas. São substâncias sólidas, incolores, coloidais. Possuem estrutura complexa e massa molecular elevada (entre 15000 e 20000u). São sintetizadas através da condensação de muitos alfa-aminoácidos em ligações peptídicas.*
>
> *Texto adaptado da enciclopédia Wikipedia (www.wikipedia.org), capturado em agosto/2006.*

Provavelmente o estudante não conseguiria extrair o conhecimento contido nessa página, visto que não possui um modelo conceitual suficientemente rico sobre esse determinado domínio.

Da mesma forma, agentes de software não possuem um modelo conceitual de "senso comum" ou de "linguagem natural" para conseguir interpretar as páginas da Internet da mesma forma que seres humanos conseguem. Eles precisam entender a semântica do conteúdo das páginas.

A *Web* Semântica é vista como uma extensão da *Web* atual na qual, além de ser passível de compreensão por humanos por meio de navegadores *Web*, documentos são anotados com meta-dados. Tais meta-dados descrevem o conteúdo dos documentos em uma forma que possa ser processável por software. A representação explícita de meta-dados, acompanhada por teorias de domínio, abre espaço para um novo nível de serviços na *Web* (DAVIES et al., 2003).

Diferentes tecnologias vêm apoiar a construção deste novo paradigma da Internet e discutimo-las nas seções que se seguem. Ontologias (Seção 3.2) estão por trás das teorias de domínio e dos meta-dados que anotam os documentos da *Web* semântica. Agentes (Seção 3.3)

devem ser construídos para tirar proveito deste novo nível de serviços com semântica embutida. Porém, são os autores da Internet que possuem a grande responsabilidade de tornar real esta evolução, anotando suas páginas e serviços *Web* (Seção 3.4) para que estejam disponíveis para os agentes de software. Por fim, para que tais anotações sejam feitas, é preciso utilizar linguagens de representação desses meta-dados (Seção 3.5).

3.2. Ontologias

Para que agentes possam interpretar o conteúdo de páginas *Web*, é preciso explicitá-lo em estruturas formais de representação de conhecimento. Na área de Inteligência Artificial, diversas estruturas já foram propostas, como: facetas (*features*), dicionários, índices, taxonomias, *thesaurus*, redes semânticas etc. (BREITMAN, 2006). Mais recentemente, as ontologias despontaram como uma forma mais rica de representação do conhecimento.

O termo "ontologia" possui diferentes significados (GUIZZARDI, 2007):

- Ontologia é o ramo mais importante da metafísica, uma das disciplinas da filosofia. Uma de suas subdivisões, chamada Ontologia Formal, visa criar teorias gerais sobre aspectos da realidade que não são específicos a nenhum campo da ciência;
- Uma ontologia é uma representação de teorias gerais (chamada de ontologia fundamental) ou de teorias específicas de determinadas áreas de conhecimento (chamada de ontologia de domínio);
- Um artefato concreto que contém tal representação codificada em uma linugagem de representação adequada também é chamado de ontologia.

Especificamente no ramo da Computação, Nicola Guarino (1998) define ontologia como um artefato de engenharia, constituído por um vocabulário específico usado para descrever uma certa realidade, somado a um conjunto de proposições explícitas com relação ao significado proposto para as palavras desse vocabulário. 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).

Guarino (1998) propõe uma classificação de ontologias em quatro categorias, como

mostra a Figura 3.1. As ontologias de nível superior descrevem conceitos genéricos (ex.: pessoa, local etc.), comuns a diversos domínios de aplicação. Ontologias de domínio e de tarefas se baseiam numa ontologia de nível superior para representar conceitos específicos de um domínio (ex.: engenharia de software) ou de uma tarefa genérica (ex.: planejamento). Por fim, ontologias de aplicação estendem ontologias de domínio e de tarefas para representar conceitos existentes em uma classe de aplicações específica (ex.: aplicações de apoio ao planejamento de projetos de software).

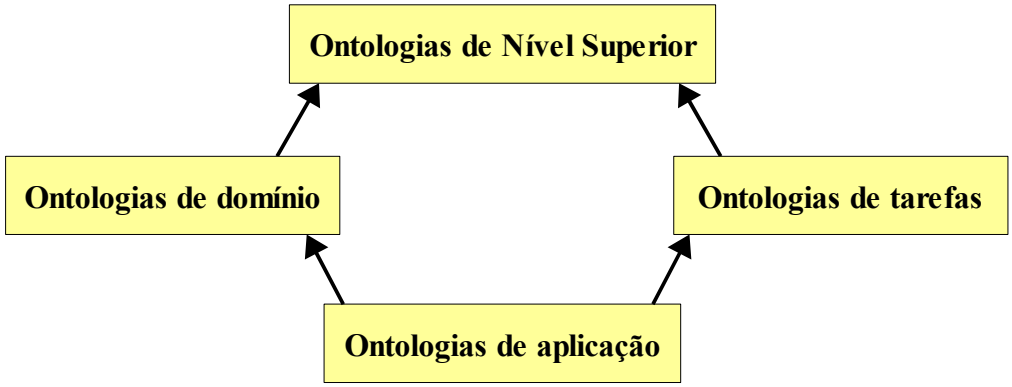

Figura 3.1: Classificação de ontologias segundo (GUARINO, 1998).

Para uma construção efetiva da *Web* Semântica, ontologias de nível superior como Dolce (GANGEMI et al., 2003), WordNet (FELLBAUM, 1998) e Cyc (MATUSZEK et al., 2006) devem guiar a construção de ontologias mais específicas de domínio e de tarefas. Estas poderiam ser compostas na criação de ontologias de aplicação e todas elas, em conjunto, seriam utilizadas pelas páginas e aplicações *Web* para descrever seus conteúdos e serviços.

Ontologias não são utilizadas somente no contexto da *Web* Semântica. Na Engenharia de Software, são utilizadas em atividades de Engenharia de Domínio, para identificar objetos e operações de uma classe de sistemas similares num domínio de problema em particular (GUIZZARDI, 2000). Quando construímos software, o objetivo é resolver um problema de um dado domínio de especialidade, como medicina, vendas ou siderurgia. Se o domínio é analisado antes da análise do problema, o conhecimento que é formalizado sobre o domínio pode ser reutilizado quando outro problema do mesmo domínio necessitar de uma solução via software (FALBO et al., 2002).

Apesar de também se tratar de um modelo conceitual, é preciso distinguir ontologias de modelos conceituais de aplicação. Atividades como análise de requisitos e projeto de sistema produzem modelos conceituais (usualmente representados por diagramas de classes da UML), que representam elementos de domínio relacionados ao problema que está sendo resolvido. O

escopo das ontologias é mais amplo, abrangendo todo o domínio no qual o software está inserido, inclusive conceitos, relações, atributos e restrições que não são relevantes ao problema tratado pelo sistema computacional. Como são modelos consensuais, não devem conter detalhes específicos de um problema particular, mas apenas aquilo que é válido para as aplicações no domínio como um todo.

Construir ontologias não é uma tarefa simples. Métodos e linguagens de modelagem têm sido propostos para apoiar esta tarefa. Na subseção a seguir, discutimos algumas metodologias que apóiam a construção de ontologias.

3.2.1. Construção de ontologias

Uma vez que construir ontologias não é uma tarefa trivial, é preciso adotar uma metodologia de forma que esse processo não seja conduzido de forma *ad-hoc*. Diversas metodologias já foram propostas e Breitman (2006) apresenta uma descrição resumida de algumas delas, a saber:

- Processo proposto pelo Método Cyc: resultado do projeto Cyc (MATUSZEK et al., 2006), foi o primeiro método para construção de ontologias publicado e divide-se em três etapas de extração: manual, apoiada por computadores e gerenciada por computadores;
- Metodologia proposta por Uschold: primeira metodologia completa, proposta por um grupo da Universidade de Edimburgo com base na prática da construção da ontologia *Enterprise*. O processo é guiado por cenários de motivação e é dividido em fases de identificação de propósito, construção, avaliação e documentação;
- Metodologia do Projeto TOVE: a metodologia do Projeto TOVE (Toronto Virtual Enterprise) foi derivada da experiência de Gruninger e Fox no desenvolvimento de ontologias para domínios de processos de negócios e corporativo. Utiliza os cenários de motivação propostos por Uschold e a geração de questões de competência como requisitos para a ontologia. Utiliza lógica de primeira ordem para formalização de tais questões, especificando formalmente os axiomas e verificando sua completude;
- Methontology: é, de fato, um *framework* desenvolvido no laboratório de Inteligência Artificial do Politécnico de Madri, que fornece apoio automatizado para a construção de ontologias em nove fases: planejamento, especificação, conceituação, formalização, integração, implementação, avaliação, documentação e manutenção;
- Método utilizado pelo Projeto KACTUS: proposto pelo projeto europeu Esprit-

KACTUS, é fortemente baseado na possibilidade de reúso de ontologias, seguindo passos de especificação da aplicação, desenho preliminar baseado em categorias relevantes de ontologias de topo, refinamento e estruturação;

- Método 101: proposto por Natalya Noy e Deborah McGuiness como um processo simplificado, envolve sete passos: determinar o domínio e escopo da ontologia, considerar o reúso de outras ontologias, enumerar os termos importantes da ontologia, definir classes e a hierarquia de classes, definir as propriedades das classes, definir os valores das propriedades e criar instâncias;

1

Falbo (2004) sugere uma abordagem sistemática chamada SABiO (Systematic Approach for Building Ontologies), esquematizada na Figura 3.2. Questões de competência são formalizadas durante a identificação do propósito e a modelagem se dá por iterações sucessivas de captura e formalização da ontologia, até o nível de refinamento desejado. Ontologias existentes podem ser importadas e atividades de avaliação e documentação são conduzidas durante todo o processo.

York Sure e Rudi Studer, em (DAVIES et al., 2003), apresentam a metodologia On-To-Knowledge (OTK), composta por cinco passos: um estudo de viabilidade avalia fatores de sucesso e fracasso na iniciativa de criação da ontologia e identifica os envolvidos no processo. No “pontapé inicial”, um documento de especificação de requisitos da ontologia é criado, elencando objetivos, diretrizes, fontes de conhecimento, cenários de uso potenciais, questões de competência e aplicações que usarão a ontologia. A ontologia é modelada e, na fases seguintes, refinada e avaliada de forma iterativa. A última fase da metodologia concerne a manutenção e a evolução da ontologia.

York Sure e Rudi Studer, em (DAVIES et al., 2003), apresentam a metodologia On-To-Knowledge (OTK), composta por cinco passos: um estudo de viabilidade avalia fatores de sucesso e fracasso na iniciativa de criação da ontologia e identifica os envolvidos no processo. No “pontapé inicial”, um documento de especificação de requisitos da ontologia é criado, elencando objetivos, diretrizes, fontes de conhecimento, cenários de uso potenciais, questões de competência e aplicações que usarão a ontologia. A ontologia é modelada e, na fases seguintes, refinada e avaliada de forma iterativa. A última fase da metodologia concerne a manutenção e a evolução da ontologia.

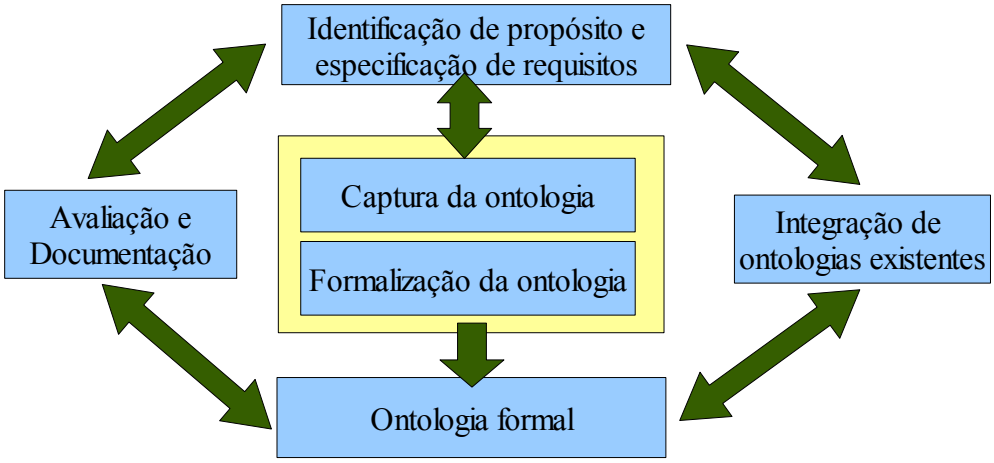

Figura 3.2: Processo de Desenvolvimento de SABiO.

3.2.1.1. Meta-modelo para Definição de Ontologias

Em 2003, a OMG[9] publicou um pedido de propostas (*request for proposals*) para um metamodelo com o propósito de definir ontologias, chamado *Ontology Definition Metamodel* (ODM) (BROCKMANS et al., 2004). O padrão, formado a partir das propostas da IBM[10] e Sandpiper Software[11], é mantido pela OMG e encontra-se disponível em (OMG, 2006).

ODM é uma família de meta-modelos MOF[12], mapeamentos entre esses modelos e UML, e um conjunto de perfis que permitem a modelagem de ontologias por meio de ferramentas baseadas em UML. Os meta-modelos que compõem ODM refletem a sintaxe abstrata de diversas linguagens de representação de conhecimento e modelagem conceitual que já se tornaram ou estão em vias de se tornar padrões internacionais (OMG, 2006).

Os perfis de ODM fornecem representações gráficas baseadas na UML para elementos definidos pelas linguagens RDF(S) e OWL, definidas pela W3C como padrões para representação de meta-dados na *Web* Semântica. Utilizando mecanismos de extensão e definindo a semântica para elementos UML como classes, propriedades e associações dentro do contexto de modelagem de ontologias, ODM apresenta-se como uma ferramenta bastante útil para construção de meta-dados no contexto da *Web* Semântica. A Figura 3.3 mostra exemplos simples de representação de componentes OWL em ODM.

9 *Object Management Group*, http://www.omg.org/.

10 http://www.omg.org/docs/ontology/04-01-01.pdf

11 http://www.omg.org/docs/ad/03-08-06.pdf

12 *Meta-Object Facility*, especificação fundamental para linguagens de modelagem da OMG.

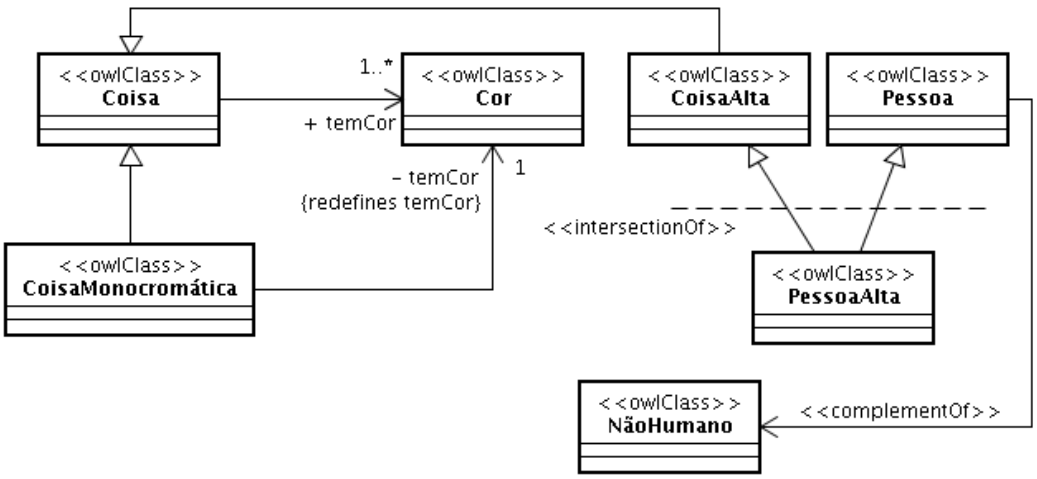

Figura 3.3: Exemplos de elementos OWL representados em ODM (OMG, 2006)

Elementos como classes (**`owl:Class`**), associações entre classes (**`owl:ObjectProperty`**), interseção (**`owl:intersectionOf`**) e complemento entre classes (**`owl:complementOf`**) estão demonstrados na Figura 3.3. A definição dos demais componentes OWL e outros exemplos de uso de ODM podem ser encontradas em (ODM, 2006).

3.2.1.2. Perfil UML para Ontologias

Utilizando os meta-modelos definidos em ODM, Đurić (2004) criou um perfil UML mais simplificado, no qual constam apenas os elementos que representam conceitos utilizados no desenvolvimento de ontologias na prática, como classes, indivíduos e propriedades. O perfil, chamado *Ontology UML Profile* (OUP), tem como proposta ser mais próximo da prática da Engenharia de Software e, ainda assim, ser compatível com MOF, sendo capaz de armazenar os modelos em repositórios baseados em MOF ou compartilhá-los via XMI[13] (ĐURIĆ, 2004).

A Figura 3.4 mostra um exemplo de representação de ontologias utilizando OUP. Classes ontológicas são representadas com estereótipo **`<<OntClass>>`** e suas associações com outras classes ontológicas são representadas por uma classe UML com o estereótipo **`<<ObjectProperty>>`**. O mesmo ocorre com associações com tipos de dados (**`<<DatatypeProperty>>`**). Associações são representadas desta maneira, pois podem participar de hierarquias ou serem compostas, o que inviabiliza sua representação com a notação de propriedade de classe da UML.

13 XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm

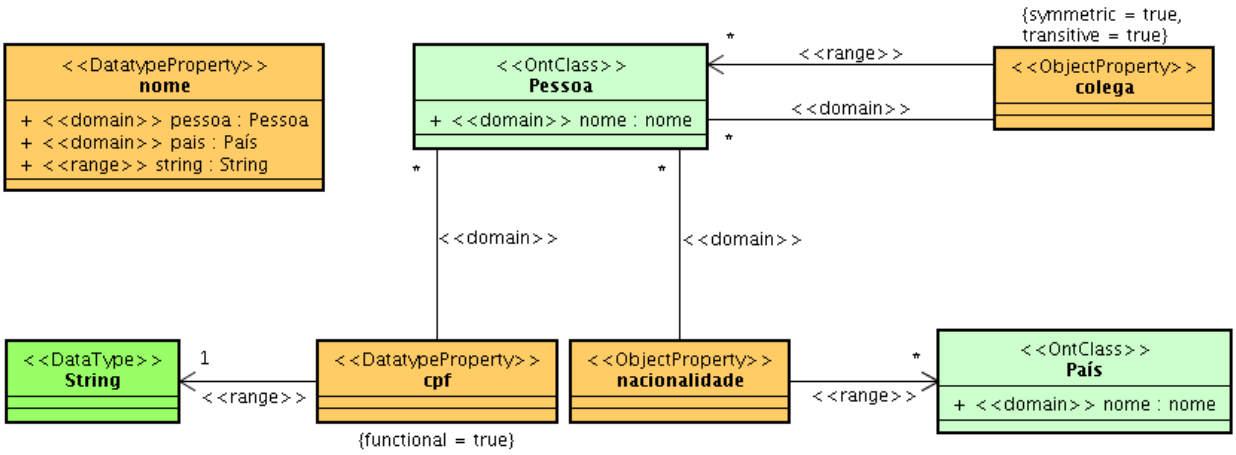

Figura 3.4: Exemplos de elementos OWL representados em OUP (ĐURIĆ, 2004)

É possível agrupar associações com tipos de dados com mesmo nome e tipo, como mostra a propriedade **`nome`**, na figura. Todas as associações possuem indicação de cardinalidade e diferenciação entre domínio (**`<<domain>>`**) e imagem (**`<<range>>`**) para orientar a construção do documento OWL. Propriedades das associações como simetria e transitividade podem ser representadas por restrições.

Além de construir ontologias para descrever o conteúdo das páginas *Web*, é necessário que haja agentes que consigam processar este conteúdo e realizar tarefas automaticamente. A próxima seção discute aspectos relacionados a esta tecnologia.

3.3. Agentes da *Web* Semântica

Em seu artigo seminal, Wooldridge (1999) define agentes como sistemas computacionais capazes de ações autônomas em algum ambiente, a fim de alcançar seus objetivos de projeto. “Um agente, tipicamente, sente seu ambiente e disponibiliza um repertório de ações que podem ser executadas para modificar o ambiente, o qual pode responder não determinadamente à execução dessas ações” (SCHWAMBACH, 2004).

A visão da *Web* Semântica é que cada internauta possua um agente pessoal que irá receber algumas tarefas e preferências da pessoa, buscar informações de fontes na *Web*, comunicar com outros agentes, comparar informações sobre requisitos e preferências do usuário, selecionar certas opções e dar a resposta ao usuário (ANTONIOU e VAN HARMELEN, 2004).

Portanto, um agente da *Web* Semântica possui autonomia para buscar, comparar,

selecionar e apresentar informações ao usuário. Porém é este quem deve tomar a decisão com relação ao que fazer. O exemplo citado no início do capítulo, em que Lucy quer marcar uma consulta com um ortopedista para sua mãe ilustra bem esse aspecto.

Diversas questões estão envolvidas na construção de agentes para a *Web* Semântica: (a) como fazer com que os agentes compreendam as mais várias ontologias utilizadas pelas páginas *Web*? (b) Que vocabulário será usado por um agente para se comunicar com outro? (c) Como o agente encontrará as fontes de informação que deseja espalhadas pela *Web*? Essas e outras questões precisam ser resolvidas para que a visão da *Web* Semântica se torne realidade no âmbito de toda a Internet e não só restrita a uma Intranet na qual é possível definir o vocabulário e conhecer todos os serviços disponíveis.

De maneira geral, os agentes da *Web* Semântica trabalham com os meta-dados disponíveis nas páginas e aplicações *Web*. A seção a seguir discute diferentes abordagens para adição desses meta-dados nos *websites*.

3.4. Abordagens para adição de semântica às aplicações *Web*

Existem diferentes formas de se adicionar semântica às informações dispostas na *Web*. Nesta seção, discutimos sucintamente duas delas: anotação de *websites* e *Web Services* semânticos.

3.4.1. Anotação de *websites*

As metodologias discutidas na Seção 3.2.1 podem ser utilizadas para a construção de uma ontologia que represente parcialmente o conteúdo a ser exibido em um determinado *website*. Cada página, então, pode ser manualmente anotada utilizando linguagens apropriadas (vide Seção 3.5). Esse procedimento, no entanto, pode ser bastante trabalhoso, tedioso e propenso a erros. Ferramentas de construção de ontologias como OILEd[14] e Protégé[15] podem auxiliar nessa tarefa.

Contudo, a abordagem de anotação manual não se aplica a *websites* centrados em dados (*data-intensive websites*). Nesse tipo de *site*, as páginas são geradas dinamicamente por meio de consultas a repositórios de dados, tipicamente bancos de dados relacionais. Tais páginas possuem a vantagem de separar dados e leiaute, porém trazem limitações como serem invisíveis aos mecanismos de buscas (que não conseguem indexar URLs geradas dinamicamente) e ter conteúdo em HTML e, portanto, não passível de compreensão por um

14 http://oiled.man.ac.uk/

15 http://protege.stanford.edu/

agente de software (STOJANOVIC et al., 2002).

Lima e Schwabe (2003) propõem uma abordagem semântica para o desenvolvimento de aplicações hipermídia baseada em OOHDM (vide Seção 2.1.7). Segundo a abordagem, chamada SHDM (*Semantic Hypermedia Design Method*), requisitos são capturados em forma de cenários e diagramas de interação com o usuário e posteriormente usados como base para modelagem conceitual, produzindo diagramas baseados em UML que representam os conceitos do domínio em que a aplicação hipermídia se encaixa. Num passo seguinte, os modelos são complementados com informações de navegação e, finalmente, transformados em componentes hipermídia para implantação e uso.

SHDM integra o modelo conceitual com o repositório de dados e modelos definidos pelo usuário no nível de implementação, provendo uma abordagem baseada em modelos. Dado que o modelo conceitual é facilmente convertido para uma linguagem de representação de ontologias (OWL, por exemplo), os componentes hipermídia possuem, assim, as anotações necessárias para processamento por agentes de software. No entanto, consideramos que essa abordagem é mais adequada para aplicações *Web* com foco em conteúdo, pois não atribui um foco muito grande à modelagem da lógica de negócio.

Faz-se necessária, portanto, uma abordagem automatizada para anotação de páginas dinâmicas para atender os sistemas de informação *Web*. Fuchs et al. (2003) tratam a questão de como gerar anotações semânticas automaticamente para as inúmeras páginas dinâmicas geradas em aplicações *Web* a partir de informações em bancos de dados. Utilizando uma ferramenta chamada VizCo, autores poderiam definir amarrações entre o modelo de domínio e objetos das páginas *Web* (ex.: componentes de interface com o usuário). Tal amarração, definida em RDF (vide Seção 3.5), seria utilizada em tempo de execução para disponibilizar anotações sobre o conteúdo das páginas geradas.

Este trabalho propõe uma abordagem similar, com o intuito de integrá-la ao *framework Web* a ser utilizado. Ao solicitar uma página ao servidor, o *framework* deve detectar se a solicitação provêm de um humano ou de um agente de software. No caso deste último, ao invés da página HTML que normalmente é exibida ao primeiro, um documento escrito em alguma linguagem de representação de ontologias (vide Seção 3.5) é apresentado, trazendo como conteúdo instâncias das classes definidas na ontologia daquele *website*, geradas automaticamente, representando entidades recuperadas do banco de dados pela solicitação específica daquela página.

Apesar da solução parecer ideal, muitos aspectos ainda precisam ser endereçados, como, por exemplo, como os agentes encontrarão as páginas dinâmicas e como saberão de que forma

devem interagir com elas (que dados devem ser enviados). Hepp (2006) advoga que a anotação semântica de conteúdo estático ou derivado de bancos de dados não é suficiente e que a visão original da *Web* Semântica só poderá tornar-se realidade com a utilização de *Web Services* Semânticos.

3.4.2. *Web Services* Semânticos

Atualmente, a *Web* não é composta apenas de páginas (estáticas ou dinâmicas) interconectadas. Nos últimos anos, aplicações *Web* evoluíram de documentos hipertextuais estáticos para complexos sistemas de informação. Grandes sistemas B2C (*Business-to-Consumer*, como lojas virtuais) e B2B (*Business-to-Business*, como controle de fornecimento) já fazem parte da grande rede de computadores, permitindo uma interação rápida e complexa entre pessoas e organizações.

Um componente-chave na construção de sistemas B2B na *Web* são os Serviços Web (*Web Services*). Segundo o glossário da W3C, um *Web Service* é um sistema de software projetado para apoiar a interação interoperável de máquina a máquina em uma rede. Possui uma interface descrita em um formato processável por computador (especificamente WSDL[16]). Outros sistemas interagem com o *Web Service* na maneira prescrita por sua descrição, usando mensagens SOAP[17], tipicamente enviadas pelo protocolo HTTP por meio de serialização XML[18] em conjunto com outros padrões da *Web* (W3C, 2006).

Páginas *Web* anotadas com meta-dados podem apenas ser lidas por agentes de software, sendo assim, sua informação é estática, não muda com muita freqüência e não há execução de nenhum serviço por parte do agente. Para oferecer uma forma pela qual os agentes da *Web* Semântica poderiam solicitar serviços ou consultar informações atualizadas dinamicamente, os *Web Services* apresentam-se como uma boa opção, visto que foram projetados para comunicação entre dois sistemas, sem interação humana.

As pesquisas, portanto, voltam-se para o desenvolvimento de *Web Services* Semânticos, que são formados a partir da adição de anotações semânticas de *Web Services* para que se tornem interpretáveis por software (MCILRAITH et al., 2001).

Meta-dados sobre os serviços são descritos por uma linguagem de marcação, descrevendo suas propriedades e capacidades, a interface para sua execução, seus pré-

16 *Web Services Description Language*, padrão da W3C para descrição de serviços *Web* usando XML.

17 *Simple Object Access Protocol*, padrão da W3C para intercâmbio de informações estruturadas em um ambiente descentralizado e distribuído.

18 Transformação de dados em formatos binários proprietários (ex.: um objeto em memória) para um formato textual e estruturado em XML.

requisitos e conseqüências de uso (MCILRAITH et al., 2001). Com isso, espera-se automatizar diversas tarefas, incluindo a descoberta, evocação, interoperação, seleção, composição e monitoramento de serviços (NARAYANAN e MCILRAITH, 2002).

Diversos modelos conceituais já foram propostos para descrever o domínio dos *Web Services* de forma a servirem como base para criação destas anotações semânticas. Dentre eles, destacam-se OWL-S (MARTIN et al., 2004), WSMF (FENSEL e BUSSLER, 2002) e WSMO (ROMAN et al., 2005).

3.5. Linguagens da *Web* Semântica

A representação formal de uma ontologia para ser processada por agentes de software é feita por meio de uma linguagem de representação de ontologias. BREITMAN (2006) cita diversos requisitos estabelecidos pelo *World Wide Web Consortium* – W3C para que tais linguagens sejam utilizadas na *Web* Semântica, a saber:

- Ontologias devem ser artefatos distintos e poder ser explicitamente estendidas;
- Termos devem poder ser referenciados de forma não-ambígua através de URIs (*Universal Resource Identifiers*);
- Recursos devem ser alocados explicitamente a ontologias;
- A linguagem deve permitir a inclusão de meta-dados e informações de versão nas ontologias;
- A linguagem deve ser capaz de expressar definições complexas de classes da ontologia (subclasses, combinações de expressão etc.) e de propriedades dessas classes (subpropriedades, restrições de domínio e escopo etc.);
- A linguagem deve fornecer um conjunto-padrão para tipos de dados e suporte à definição e utilização de tipos de dados compostos;
- A linguagem deve incluir mecanismos capazes de estabelecer que duas classes/propriedades são equivalentes, além de decidir se dois identificadores representam o mesmo indivíduo;
- A linguagem deve fornecer uma maneira de anotar as sentenças com informações adicionais;
- A linguagem deve ter suporte a tratamento de classes como instâncias;
- A linguagem deve oferecer suporte a restrições de cardinalidade nas associações;
- A linguagem deve permitir que se definam múltiplas opções de etiquetas (*labels*) e dar suporte a caracteres utilizados em vários idiomas (Unicode).

Ao invés de propor uma arquitetura nova, que implicaria numa difícil reestruturação de toda Internet, Tim Berners-Lee propõe um modelo em camadas, aproveitando tecnologias já existentes e utilizadas. A Figura 3.5 mostra essa proposta de construção gradativa das camadas para formação da *Web* Semântica de forma incremental.

Na base, o conjunto de caracteres Unicode permite expressar termos usando caracteres de qualquer linguagem do mundo, enquanto a URI, já utilizada amplamente em documentos HTML (*Hypertext Markup Language*), permite identificar os diferentes recursos espalhados pela *Web*. Logo acima, o padrão XML (*Extensible Markup Language*) permite a construção de documentos com foco na estruturação de dados. XML Namespace (NS) permite a definição de espaços de nomes e XML Schema define formatos de documentos e tipos de dados simples.

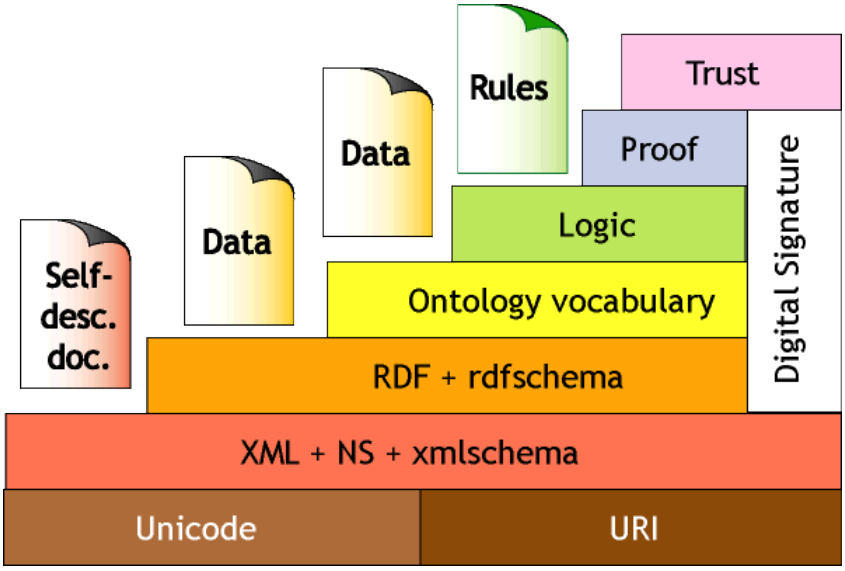

Figura 3.5: Arquitetura para a Web Semântica proposta por (BERNERS-LEE, 2000).

Na camada seguinte, a tecnologia RDF (*Resource Description Framework*) e sua extensão RDF Schema formam a base para construção de ontologias, permitindo a definição de taxonomias de classes e suas propriedades. Em cima dessa fundação, linguagens de descrição de ontologias buscam atender as necessidades das aplicações para a *Web* Semântica. Diversas linguagens já foram propostas, sendo a linguagem OWL (*Web Ontology Language*) (W3C, 2004) a recomendação do W3C para linguagem de representação de ontologias e, portanto, a mais utilizada para tal.

No entanto, a maioria das linguagens da *Web* Semântica, incluindo OWL, foram

definidas com preocupações epistemológicas e computacionais e não ontológicas (GUIZZARDI, 2006) (GUIZZARDI, 2007). Em outras palavras, ontologias representadas nessas linguagens devem ser interpretáveis por software e logicamente decidíveis. Por este motivo, nem todos os elementos que compõem uma ontologia podem ser representados nessas linguagens. Isso não deve sugerir, porém, que autores de *websites* não devam gastar tempo anotando suas aplicações e páginas *Web*, pois o aumento da demanda por linguagens mais completas motivará as pesquisas, que caminham em direção à definição de linguagens ricas e ao mesmo tempo processáveis na prática.

3.6. Conclusões do Capítulo

Este capítulo apresentou uma visão geral da *Web* Semântica e revisou alguns conceitos relacionados, como ontologias, agentes, anotação semântica e linguagens de representação. Os benefícios dessa nova forma de disponibilizar e encontrar informação na *Web* são claros e as pesquisas avançam para que essa visão possa, um dia, tornar-se realidade.

Desta maneira, apresentamos nossa proposta para construção de WISs baseados em *frameworks* no próximo capítulo e, no Capítulo 5, complementamos tal proposta com uma abordagem voltada para a *Web* Semântica, de forma que as aplicações *Web* construídas utilizando este método disponibilizem informações semânticas aos agentes. Esperamos, desta forma, contribuir a longo prazo com o fomento da *Web* Semântica.

Capítulo 4: O método FrameWeb

FrameWeb é um método de projeto para construção de sistemas de informação *Web* (*Web Information Systems* – WISs) baseado em *frameworks*. No Capítulo 2, apresentamos as motivações principais para tal proposta:

(a) O uso de *frameworks* ou arquiteturas baseadas em *containers* similares a eles se tornou o padrão *de facto* para o desenvolvimento de aplicações distribuídas, em especial os baseados na plataforma *Web*;

(b) Existem diversas propostas para Engenharia *Web*, incluindo metodologias, métodos, linguagens, *frameworks* etc., porém não encontramos nenhuma que tratasse diretamente aspectos característicos dos *frameworks* utilizados comumente na construção de WISs;

(c) O uso de métodos que se adequam diretamente à arquitetura de software adotada promove uma agilidade maior ao processo, característica que é bastante desejada na maioria dos projetos *Web* (PRESSMAN, 2005).

Em linhas gerais, FrameWeb assume que determinados tipos de *frameworks* serão utilizados durante a construção da aplicação, define uma arquitetura básica para o WIS e propõe modelos de projeto que se aproximam da implementação do sistema usando esses *frameworks*.

Sendo um método para a fase de Projeto, não prescreve um processo de software completo. No entanto, sugere o uso de um processo de desenvolvimento que contemple as fases apresentadas na Figura 4.1. Para uma melhor utilização de FrameWeb, espera-se que sejam construídos diagramas de casos de uso e de classes de domínio (ou similares) durante as fases de Requisitos e Análise. Além disso, como já mencionado anteriormente, agilidade é uma característica desejada num processo de software para a *Web* e, portanto, sugere-se que princípios de agilidade sejam seguidos, em especial os seguintes propostos pela Modelagem Ágil (AMBLER e JEFFRIES, 2002):

- Modele com um propósito: crie apenas os modelos que adicionam informação útil;
- Viaje com pouca bagagem: ao passar de uma fase para outra no processo de desenvolvimento, alguns modelos precisarão ser adequados à nova fase. Leve apenas

aqueles que serão úteis na fase seguinte;

- Conteúdo é mais importante que apresentação: utilize a notação e a linguagem mais adequadas ao objetivo principal de um modelo, que é a transmissão da informação;
- Conheça os modelos e as ferramentas: equipes de modelagem e desenvolvimento devem ser altamente proficientes nas linguagens e ferramentas utilizadas na modelagem;
- Adapte-se localmente: as equipes devem ser capazes de se adaptar às necessidades específicas de um projeto.

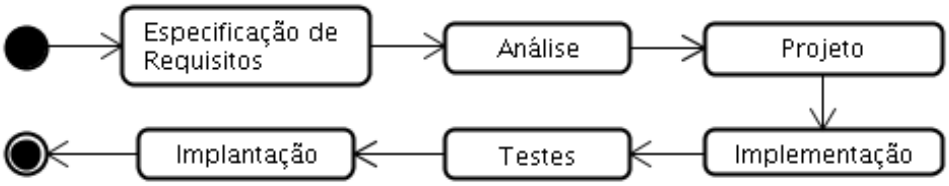

Figura 4.1: Um processo de desenvolvimento de software simples sugerido por FrameWeb.

A fase de Projeto concentra as propostas principais do método: (i) definição de uma arquitetura padrão que divide o sistema em camadas, de modo a se integrar bem com os *frameworks* utilizados; (ii) proposta de um conjunto de modelos de projeto que trazem conceitos utilizados pelos *frameworks* para esta fase do processo por meio da criação de um perfil UML que faz com que os diagramas fiquem mais próximos da implementação.

A fase de codificação é facilitada pelo uso dos *frameworks*, especialmente porque os modelos de projeto exibem componentes relacionados a eles. O uso de *frameworks* pode também ter impacto nas fases de teste e implantação. No entanto não foram traçadas considerações sobre essas atividades do processo neste trabalho.

FrameWeb é apresentado em detalhes neste capítulo. Para ilustrar seu uso, apresentamos brevemente na Seção 4.1 a análise de requisitos do sistema *Web* "Portal do LabES", desenvolvido em (PERUCH, 2007) e utilizado como exemplo nas seções que se seguem. Nas Seções 4.2 e 4.3 são apresentadas as principais propostas do método, respectivamente, a arquitetura padrão e a linguagem de modelagem. Por fim, a Seção 4.4 faz uma comparação de FrameWeb com as demais propostas apresentadas no Capítulo 2.

4.1. O Portal do LabES

Para ilustrar o uso do método, apresentamos aqui brevemente a análise de requisitos do Portal do LabES, um sistema de informação *Web* desenvolvido para o Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES / UFES).

O objetivo do portal é prover uma melhor interação entre a comunidade de Engenharia de Software e o LabES. Para tal, provê um conjunto de serviços e informações sobre o LabES, incluindo: projetos, áreas de interesse, publicações, eventos e materiais diversos. O portal é dividido em dois subsistemas, *ControleUsuario* e *ControleItens*, mostrados nas Figuras 4.2 e 4.3.

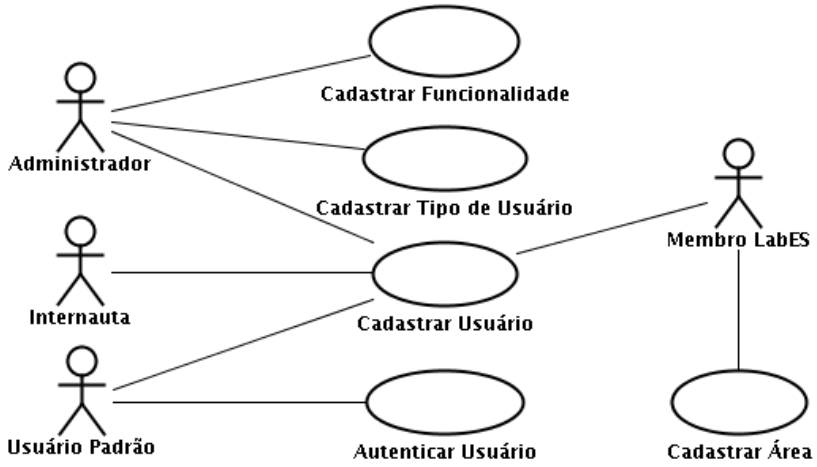

Figura 4.2: Diagrama de casos de uso do subsistema *ControleUsuario* do Portal do LabES.

O subsistema *ControleUsuario* reúne funcionalidades básicas de cadastro, enquanto as funcionalidades principais encontram-se no subsistema *ControleItens*. Neste último, professores podem cadastrar projetos e membros do laboratório podem cadastrar publicações, eventos da área e materiais diversos para *download*. Tais informações ficam disponíveis para o público em geral por meio de uma ferramenta de busca.

O modelo estrutural do sistema também foi dividido nos pacotes *ControleUsuario* e *ControleItens*, como mostram as Figuras 4.4 e 4.5. O primeiro contém elementos de infra-estrutura, tais como registro de usuários, áreas de interesse, tipos de usuário e funcionalidades. O segundo agrega os diversos materiais e informações que são cadastrados e disponibilizados no portal, como projetos, eventos, publicações etc.

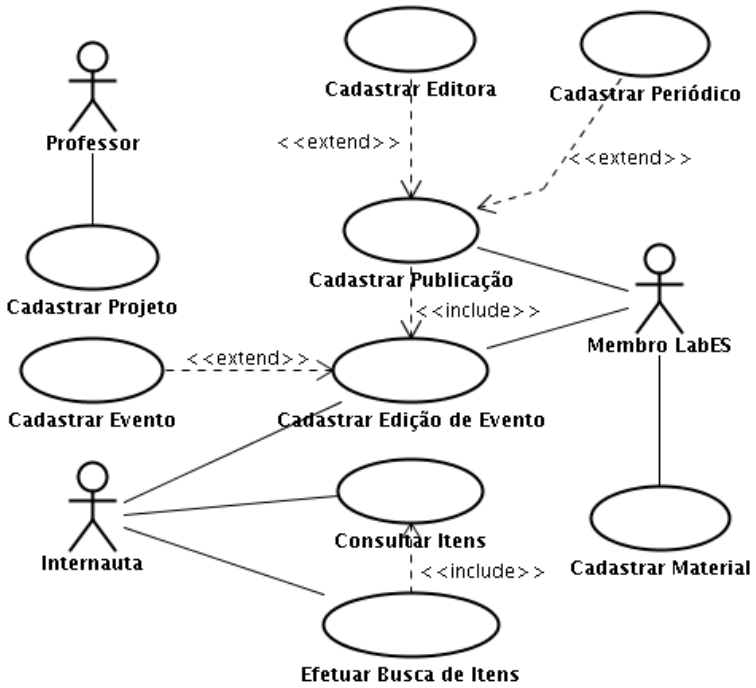

Figura 4.3: Diagrama de casos de uso do subsistema *ControleItens* do Portal do LabES.

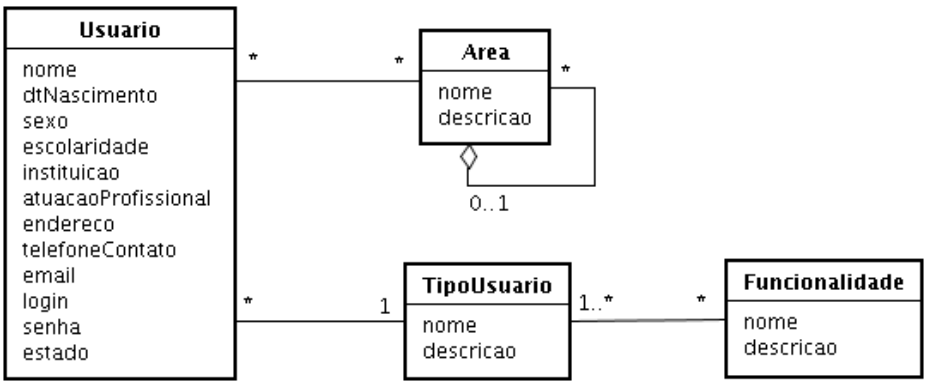

Figura 4.4: Diagrama de classes do pacote *ControleUsuario* do Portal do LabES.

A fase de projeto foi conduzida utilizando o método FrameWeb. Foi escolhido um conjunto de *frameworks* (Struts2, Spring Framework, Hibernate e Sitemesh) e utilizada a arquitetura padrão sugerida pelo próprio método, a qual apresentamos na seção seguinte.

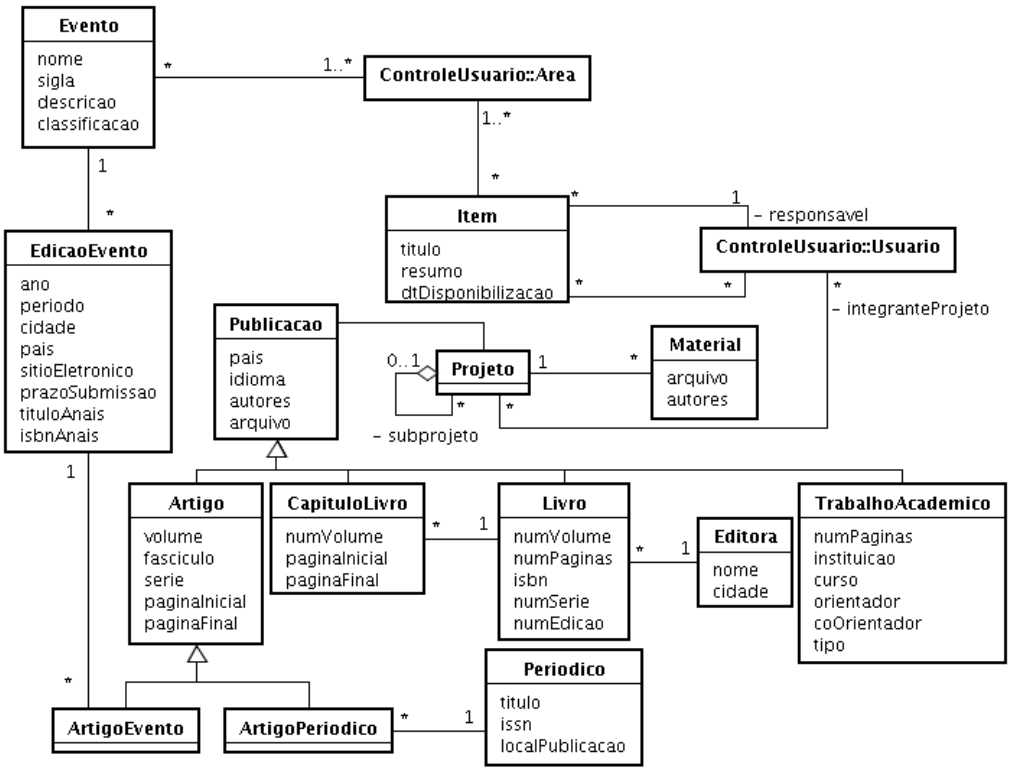

Figura 4.5: Diagrama de classes do pacote *ControleItens* do Portal do LabES.

4.2. Arquitetura *Web* baseada em *frameworks*

A fase de Projeto de Sistema, tradicionalmente executada após o levantamento e a análise dos requisitos, tem por objetivo descrever as arquiteturas lógica e física do sistema, bem como, partindo da análise do sistema, apresentar modelos estruturais e comportamentais da aplicação adicionados de características específicas da plataforma de implementação escolhida.

FrameWeb define uma arquitetura lógica padrão para WISs baseada no padrão arquitetônico *Service Layer* (Camada de Serviço), proposto por Randy Stafford em (FOWLER, 2002, p. 133). Como mostra a Figura 4.6, o sistema é dividido em três grandes camadas: lógica de apresentação, lógica de negócio e lógica de acesso a dados.

A primeira camada tem por objetivo prover interfaces gráficas com o usuário. O pacote `Visão` contém páginas *Web*, folhas de estilo, imagens, *scripts* que executam do lado do cliente, modelos de documentos e outros arquivos relacionados exclusivamente com a exibição de informações ao usuário. O pacote `Controle` envolve classes de ação e outros arquivos relacionados ao *framework* Controlador Frontal (Seção 2.3.1). Esses dois pacotes

possuem uma dependência mútua, visto que elementos da **Visão** enviam estímulos do usuário para classes do **Controle** que, por sua vez, processam as respostas utilizando páginas, modelos e outros componentes da **Visão**.

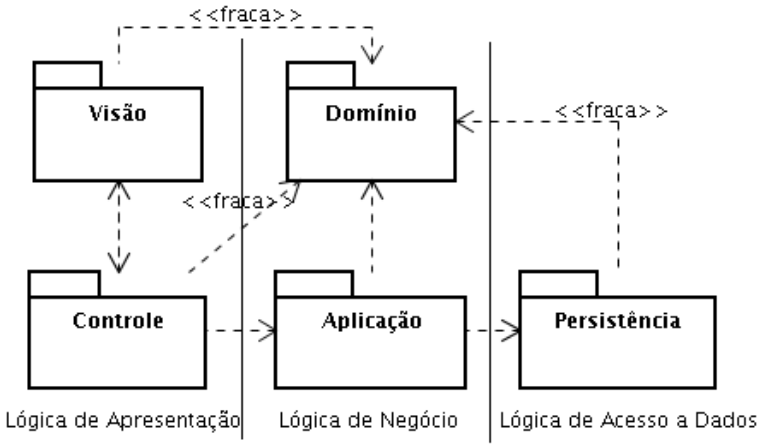

Figura 4.6: Arquitetura padrão para WIS baseada no padrão arquitetônico *Service Layer* (FOWLER, 2003).

A lógica de negócio encontra-se implementada na segunda camada, dividida em dois pacotes: **Domínio** e **Aplicação**. O primeiro contém classes que representam conceitos do domínio do problema identificados e modelados pelos diagramas de classes na fase de Análise e refinados durante o Projeto. O último tem por responsabilidade implementar os casos de uso definidos na especificação de requisitos, provendo uma camada de serviços independente da interface com o usuário. Como para isso a **Aplicação** manipula objetos de **Domínio**, possui um relacionamento de dependência com este pacote.

O pacote **Controle**, na camada de apresentação, depende do pacote de **Aplicação**, que provê ao usuário acesso às funcionalidades do sistema. Estímulos dos usuários provenientes da camada de **Visão** são transformados em chamadas de métodos no pacote **Aplicação** pelas classes de **Controle** para executar, desta maneira, os casos de uso. **Controle** e **Visão** possuem, também, relacionamento de dependência com o pacote **Domínio**. No entanto, este relacionamento é estereotipado como **<<fraco>>**, representando um baixo acoplamento. O estereótipo indica que não são feitas alterações em entidades de domínio persistentes: objetos de **Domínio** são utilizados na camada de apresentação apenas para exibição de seus dados ou como parâmetros nas evocações de métodos entre um pacote e outro.

A terceira e última camada, acesso a dados, contém apenas o pacote **Persistência**, que é responsável pelo armazenamento dos objetos persistentes em mídia de longa duração, como

bancos de dados, arquivos, serviços de nome etc. No caso de FrameWeb, espera-se a utilização de um *framework* de mapeamento objeto/relacional (ORM, Seção 2.3.3) por meio do padrão de projeto *Data Access Object* (DAO) (ALUR et al., 2003, p. 462). O padrão DAO adiciona uma camada de abstração a mais, separando a lógica de acesso a dados da tecnologia de persistência de maneira que a camada de aplicação não conheça qual *framework* ORM está sendo utilizado, permitindo que o mesmo seja trocado, se necessário. O uso deste padrão também facilita a execução de testes unitários na camada de aplicação.

Como ilustra o diagrama, o pacote **`Aplicação`** depende do pacote **`Persistência`** para recuperar, gravar e excluir objetos de domínio como resultado da execução dos casos de uso. O pacote **`Persistência`**, por sua vez, manipula objetos de **`Domínio`** apenas para mapeá-los ao banco de dados relacional e, portanto, uma relação de dependência **`<<fraca>>`** é representada.

Esta arquitetura provê uma sólida base para construção de WISs baseados nos *frameworks* apresentados na Seção 2.3. Cada pacote contém classes ou outros elementos que se integram com os *frameworks* citados e, para modelar todas esses elementos, FrameWeb propõe uma linguagem de modelagem baseada em UML, apresentada a seguir.

4.3. Linguagem de modelagem de FrameWeb

Durante a fase de projeto, além de especificar as arquiteturas lógica e física do sistema, também modelamos os artefatos que serão implementados pelos programadores na fase seguinte. Devido à grande integração com os *frameworks* apresentados na Seção 2.3, sentimos necessidade de uma linguagem de modelagem específica, que representasse diretamente os conceitos existentes em tais *frameworks*.

Seguindo a mesma abordagem utilizada por outras linguagens de modelagem para *Web*, como WAE (Seção 2.2.2) e UWE (Seção 2.2.3), FrameWeb define extensões leves (*lightweight extensions*) ao meta-modelo da UML para representar componentes típicos da plataforma *Web* e dos *frameworks* utilizados, criando um perfil UML que é utilizado para a construção de diagramas de quatro tipos:

- Modelo de Domínio;
- Modelo de Persistência;
- Modelo de Navegação;
- Modelo de Aplicação.

Esses modelos são apresentados nas subseções seguintes.

4.3.1. Modelo de Domínio

O modelo de domínio é um diagrama de classes da UML que representa os objetos de domínio do problema e seu mapeamento para a persistência em banco de dados relacional. A partir dele são implementadas as classes da camada de `Domínio` na atividade de implementação.

Os passos para sua construção são:

1. A partir do modelo de classes construído na fase de análise de requisitos, adequar o modelo à plataforma de implementação escolhida, indicando os tipos de dados de cada atributo, promovendo a classes atributos que devam ser promovidos, definindo a navegabilidade das associações etc.;
2. Adicionar os mapeamentos de persistência.

Os mapeamentos de persistência são meta-dados das classes de domínio que permitem que os *frameworks* ORM (Seção 2.3.3) transformem objetos que estão na memória em linhas de tabelas no banco de dados relacional. Por meio de mecanismos leves de extensão da UML, como estereótipos e restrições, adicionamos tais mapeamentos ao diagrama de classes de domínio, guiando os desenvolvedores na configuração do *framework* ORM. Apesar de tais configurações serem relacionadas mais à persistência do que ao domínio, elas são representadas no Modelo de Domínio porque as classes que são mapeadas e seus atributos são exibidas neste diagrama.

A Tabela 4.1 descreve os mapeamentos objeto/relacionais possíveis para o Modelo de Domínio. Para cada mapeamento, a tabela apresenta qual mecanismo de extensão é utilizado e quais são os possíveis valores ou sintaxe para seu uso. Nenhum dos mapeamentos é obrigatório e a maioria deles possui valores *default* baseados no bom senso, ou seja, os valores mais comumente utilizados são colocados como o padrão, caso não sejam especificados. Isso reduz a quantidade de informações que devem ser colocadas no modelo. Esses valores *default* são mostrados na coluna “Valores possíveis” em negrito ou entre parênteses.

Tabela 4.1: Possíveis mapeamentos objeto/relacionais para o Modelo de Domínio

Mapeamento	*Extensão*	*Valores possíveis*
Se uma classe é persistente, transiente ou mapeada (a classe não é persistente, mas suas propriedades serão se alguém herdá-las)	Estereótipo de classe	**<<persistent>>** <<transient>> <<mapped>>
Nome da tabela na qual serão salvos objetos da classe	Restrição de classe	table=*name* (o *default* é o nome da classe)
Se um atributo é persistente ou transiente	Estereótipo de atributo	**<<persistent>>** <<transient>>
Se um atributo pode ser nulo ou não	Restrição de atributo	**null** not null
Precisão do tipo "data e hora": armazenar somente a data, somente a hora ou ambos	Restrição de atributo	precision = (date | time | **timestamp**)
Se um atributo é chave-primária da tabela	Estereótipo de atributo	<<id>>
Como é gerado o valor da chave-primária: automaticamente, obtido em uma tabela, coluna identidade, em sequência ou pelo usuário (não é gerado)	Restrição de atributo	generation = (**auto** | table | identity | sequence | none)
Se um atributo é coluna de versionamento da tabela	Estereótipo de atributo	<<version>>
Se um atributo deve ser armazenado em um campo grande (CLOB, BLOB e afins)	Estereótipo de atributo	<<lob>>
Nome da coluna na qual será armazenado um atributo	Restrição de atributo	column=*name* (o *default* é o nome do atributo)
Tamanho da coluna na qual o atributo será armazenado (para os tipos que isso se aplica)	Restrição de atributo	size=*value*
Precisão decimal da coluna na qual o atributo será armazenado (para os tipos que isso se aplica)	Restrição de atributo	precision=*value*
Escala decimal da coluna na qual o atributo será armazenado (para os tipos que isso se aplica)	Restrição de atributo	scale=*value*
Se uma associação deve ser embutida (ao invés de um mapeamento de associação, as propriedades do objeto embutido são gravadas na tabela da classe que o "embute")	Estereótipo de atributo	<<embedded>>

Tabela 4.1: Possíveis mapeamentos objeto/relacionais para o Modelo de Domínio

Mapeamento	*Extensão*	*Valores possíveis*
Estratégia de mapeamento de herança: uma tabela para cada classe usando UNION, uma tabela para cada classe usando JOIN ou uma única tabela para a hierarquia	Estereótipo de herança	`<<union>>` `<<join>>` **`<<single-table>>`**
Coleção que implementará uma associação: *bag* (lista não indexada), lista, conjunto ou mapa	Restrição de associação	`collection = ( bag | list |` **`set`** `| map )`
Ordenação de uma associação: natural (da classe) ou por colunas (**`c1 asc, c2 desc`** etc.)	Restrição de associação	`order = ( natural |` *`column names`* `[asc | desc] )`
Cascateamento de uma associação: nada, criação, alteração, exclusão, atualização ou tudo	Restrição de associação	`cascade = (` **`none`** `| persist | merge | remove | refresh | all )`
Estratégia de recuperação de uma associação: normal (*eager*) ou preguiçosa (*lazy*)	Restrição de associação	`fetch = ( lazy |` **`eager`** `)`

O Modelo de Domínio do pacote **`controleusuario`** é mostrado na Figura 4.7. De acordo com os valores *default*, todas as classes do modelo são persistentes e os nomes das tabelas e colunas que armazenarão seus valores serão os mesmos nomes das classes e atributos, respectivamente. Nenhuma das classes possui um atributo identificador (chave-primária), pois o mesmo é herdado de uma classe do pacote **`utilitario`**, mostrada na Figura 4.8.

A classe **`ObjetoPersistente`** define os atributos de identidade e de versão, além de herdar um id universal que permite que cada objeto possua uma identidade única em memória[19]. Como foi declarada como mapeada, ela não é uma entidade persistente em si, mas suas sub-classes herdarão seus atributos juntamente com seus mapeamentos de persistência. Os tipos **`I`** e **`V`** são genéricos, permitindo que as sub-classes escolham diferentes tipos para seus atributos **`id`** e **`versao`**.

19 A relação entre a identidade de um objeto em memória e sua chave-primária no banco de dados traz diversas questões abordadas no artigo "Hibernate, null unsaved value and hashcode: A story of pain and suffering" de Jason Carreira (http://www.jroller.com/page/jcarreira?entry=hibernate_null_unsaved_value_and), do qual foi tirada a idéia o identificador universal.

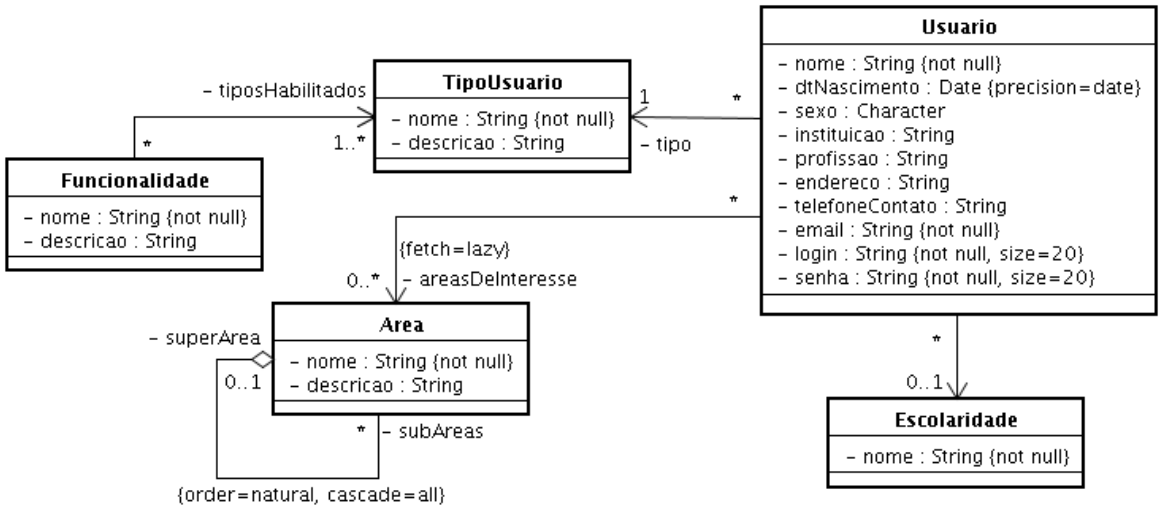

Figura 4.7: Modelo de Domínio do pacote `controleusuario` do Portal do LabES.

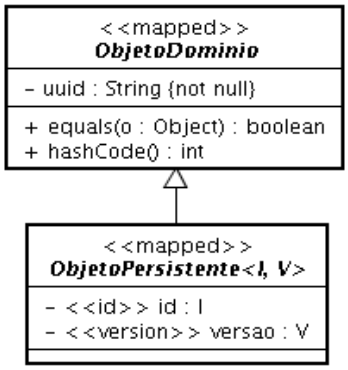

Figura 4.8: Classes do pacote `utilitario`, superclasses das classes de domínio.

Os demais mapeamentos indicam, por exemplo, que quando um objeto da classe **`Usuario`** é salvo no banco de dados, seu nome, e-mail, login e senha não podem ser nulos e somente o dia, mês e ano de sua data de nascimento será registrado (eventuais informações de hora, minuto e segundo serão perdidas). A associação recursiva de **`Area`** será implementada por um conjunto (**`Set`**, que é o *default*), com ordenação natural (a própria classe **`Set`** se incumbe de ordenar os objetos de acordo com critérios implementados na própria linguagem de programação) e cascateamento completo das operações, ou seja, se uma área é salva, suas sub-áreas também o serão e se uma área é excluída, suas sub-áreas também o serão.

4.3.2. Modelo de Persistência

Como já dissemos, FrameWeb indica a utilização do padrão de projeto DAO (ALUR et al., 2003, p. 462) para a construção da camada de acesso a dados. O Modelo de Persistência é

um diagrama de classes da UML que representa as classes DAO existentes, responsáveis pela persistência das instâncias das classes de domínio. Esse diagrama guia a construção das classes DAO, que pertencem ao pacote **`Persistência`**.

Os passos para a construção desse modelo são:

1. Criar as interfaces e implementações concretas dos DAOs base;
2. Definir quais classes de domínio precisam de lógica de acesso a dados e, portanto, precisam de um DAO;
3. Para cada classe que precisa de um DAO, avaliar a necessidade de consultas específicas ao banco de dados, adicionando-as como operações nos respectivos DAOs.

O Modelo de Persistência apresenta, para cada classe de domínio que necessita de lógica de acesso a dados, uma interface e uma classe concreta DAO que implementa a interface. A interface, que é única, define os métodos de persistência existentes para aquela classe, a serem implementados por uma ou mais classes concretas, uma para cada tecnologia de persistência diferente (ex.: um DAO para o *framework* Hibernate, outro para o *framework* OJB etc.).

Para que não seja necessário repetir em cada interface DAO operações que são comuns a todas elas (ex.: **`salvar()`**, **`excluir()`**, **`recuperarPorId()`** etc.), podemos apresentar DAOs base que declaram esses métodos – novamente, uma interface e várias implementações. Automaticamente, todas as interfaces DAO de todos os diagramas herdam as definições da interface base, ocorrendo o mesmo com as implementações concretas de cada tecnologia de persistência, sem que isso precise estar explícito no diagrama. A Figura 4.9 mostra a interface e a implementação para o *framework* Hibernate do DAO base utilizado no Portal do LabES.

Tanto a interface quanto a classe **`DAOBaseHibernate`** são declaradas usando tipos genéricos, deixando a cargo de suas sub-interfaces e sub-classes a especificação da classe gerenciada por cada DAO. O DAO base define métodos para recuperar todos os objetos de uma determinada classe, recuperar um objeto dado seu identificador, salvar e excluir um objeto.

A Figura 4.9 mostra mais uma característica do Modelo de Persistência que visa reduzir a quantidade de elementos modelados: não é necessário exibir os métodos do DAO na implementação e na interface, basta modelá-los em apenas um dos dois. No caso do DAO Base, subentende-se que todos os métodos públicos de **`DAOBaseHibernate`** são definidos na interface **`DAOBase`**.

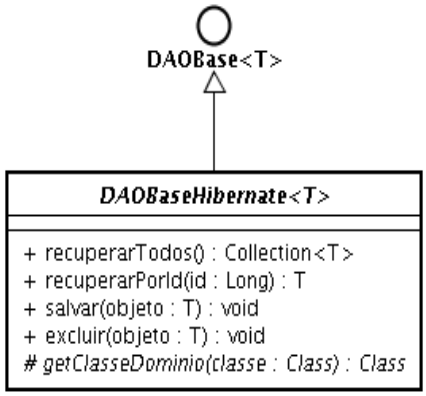

Figura 4.9: Interface e implementação usando Hibernate do DAO base utilizado no Portal do LabES.

Por fim, sugere-se seguir um padrão de nomes para as classes DAO. As interfaces devem seguir o padrão ***<nome da classe de domínio>*DAO**, enquanto as classes concretas seguem o padrão ***<nome da interface><tecnologia de persistência>***. A Figura 4.10 mostra o Modelo de Persistência do pacote **`controleusuario`** do Portal do LabES, que inclui interfaces DAO para as classes de domínio **`Usuario`**, **`Area`**, **`TipoUsuario`** e **`Funcionalidade`**, além de implementações das interfaces para o *framework* ORM Hibernate.

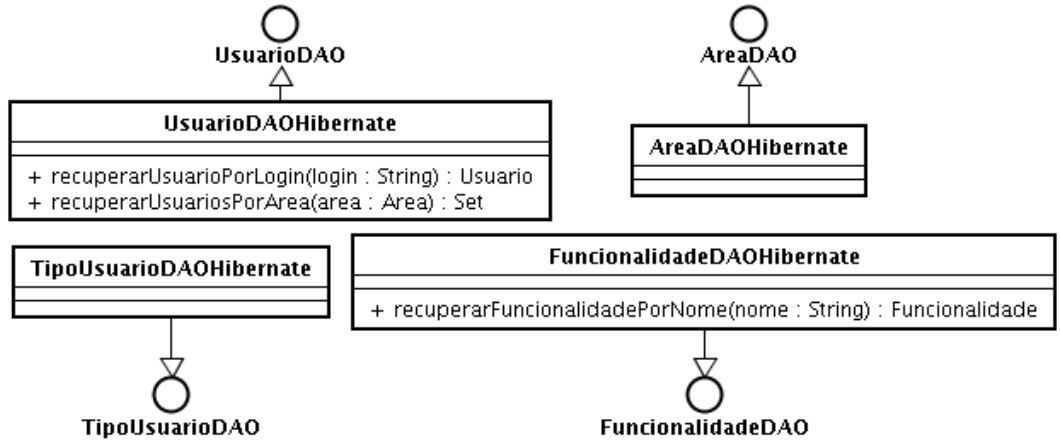

Figura 4.10: Modelo de Persistência do pacote `controleusuario` do Portal do LabES.

Segundo os padrões estabelecidos por FrameWeb, todas as interfaces DAO são sub-interfaces de **`DAOBase`**, enquanto todas as implementações Hibernate são sub-classes de **`DAOBaseHibernate`**, herdando os métodos básicos **`recuperarTodos()`**, **`recuperarPorId()`**, **`salvar()`** e **`excluir()`**. Os demais métodos que foram declarados no diagrama se referem a consultas específicas que devem ser disponibilizadas para o funcionamento de determinados

casos de uso (último passo do processo de construção do Modelo de Persistência).

Por exemplo, para o caso de uso "Autenticar Usuário", é necessário recuperar o usuário que possui um determinado *login*, para conferir se sua senha está correta. Já no cenário "Excluir Área" do caso de uso "Cadastrar Área", é preciso cancelar a exclusão de uma área caso ela tenha sido escolhida por um usuário. Como não há navegabilidade no sentido **`Area`** – **`Usuario`**, se faz necessária uma consulta que recupere todos os usuários que escolheram uma determinada área como de interesse.

Como é possível perceber, o Modelo de Persistência não define nenhuma extensão da UML para representar os conceitos necessários da camada de acesso a dados, mas apenas regras que tornam essa modelagem mais simples e rápida, por meio da definição de padrões.

4.3.3. Modelo de Navegação

O Modelo de Navegação é um diagrama de classe da UML que representa os diferentes componentes que formam a camada de Lógica de Apresentação, como páginas *Web*, formulários HTML e classes de ação do *framework* Front Controller (Seção 2.3.1). A Tabela 4.2 mostra os estereótipos UML utilizados pelos diferentes elementos que podem ser representados no Modelo de Navegação. Esse modelo é utilizado pelos desenvolvedores para guiar a codificação das classes e componentes dos pacotes **`Visão`** e **`Controle`**.

Tabela 4.2: Possíveis mapeamentos objeto/relacionais para o Modelo de Domínio

Estereótipo	***O que representa***
(nenhum)	Uma classe de ação, para a qual o *framework* Front Controller delega a execução da ação.
`<<page>>`	Uma página *Web* estática ou dinâmica.
`<<template>>`	Um modelo (*template*) de uma página *Web*, processado por um *template engine* que retorna um documento HTML como resultado.
`<<form>>`	Um formulário HTML.
`<<binary>>`	Qualquer arquivo binário que pode ser recuperado e exibido pelo navegador Internet (imagens, relatórios, documentos etc.).

Para páginas *Web* e modelos, atributos representam informações que são exibidas ao usuário na página. Os tipos possíveis são os mesmos tipos definidos pela plataforma de implementação. Relacionamentos de dependência entre páginas e modelos indicam um *link* HTML entre as mesmas, enquanto associações de composição entre páginas ou modelos e

formulários denotam a presença daquele formulário dentro da página ou modelo.

Em formulários HTML, atributos representam campos do formulário, que devem ter seus tipos definidos com o nome do campo segundo o padrão HTML (ex.: **`input`**, **`checkbox`**, **`button`** etc.) ou da *tag* JSP utilizada pelo *framework* Front Controller para renderizar o campo HTML (ex. no Struts2: **`textfield`**, **`checkbox`**, **`checkboxlist`** etc.).

A classe de ação é o principal componente do modelo. Suas associações de dependência ditam o controle de fluxo quando uma ação é executada. A Tabela 4.3 indica o significado desse tipo de associação, que varia dependendo dos componentes envolvidos.

Tabela 4.3: Associações de dependência entre a classe de ação e outros componentes

De	***Para***	***O que representa***
Página / modelo	Classe de ação	Um *link* entre a página/modelo e a classe de ação. Quando o *link* é seguido, a ação é executada.
Formulário	Classe de ação	Os dados do formulário são enviados à classe de ação para processamento.
Classe de ação	Página / modelo	A página ou modelo são exibidos como resultado da execução de uma determinada ação.
Classe de ação	Arquivo binário	Um arquivo binário é exibido como resultado da execução de uma determinada ação.
Classe de ação	Classe de ação	Uma outra classe de ação é executada como resultado da execução de uma primeira. Chamamos este processo de “encadeamento de ações”.

Os atributos da classe de ação representam parâmetros de entrada e saída relevantes àquela ação. Se um atributo da classe de ação possui o mesmo nome de um atributo de um formulário HTML que está sendo submetido a ela, significa que os dados do formulário são injetados pelo *framework* na ação e ficam disponíveis para o seu processamento. Analogamente, quando a classe de ação e a página ou modelo que apresenta seu resultado possuem atributos homônimos, indica que a página ou modelo exibem essa informação, obtendo-a da classe de ação.

Sempre que uma ação é executada, ela o faz pela evocação de um método definido como padrão pelo *framework* Front Controller. Caso seja permitido definir múltiplos métodos de ação na mesma classe, o projetista pode indicar qual método quer que seja executado por meio de uma restrição **`{method=nome-do-método}`** na associação de dependência. O mesmo

vale para as associações que representam resultados (as que partem da classe de ação), indicando que aquele resultado se refere à execução de um método específico da classe de ação. Todos esses métodos devem estar, naturalmente, modelados no diagrama e, preferencialmente, seguir o padrão de nomenclatura do *framework*.

No caso de encadeamentos de ações, será necessário, em determinados momentos, especificar qual foi o método da primeira ação que executou e qual o método que será executado na segunda. Nesses casos, utilizam-se as restrições **`outMethod`** e **`inMethod`**, respectivamente, que funcionam da mesma forma que **`method`**, apresentada anteriormente.

Quando representamos dependências que partem da classe de ação para páginas, modelos, arquivos binários ou outras classes de ação (resultados), além de poder indicar o método ao qual o resultado se refere, podemos também modelar múltiplos resultados (com a restrição **`{result=nome-do-resultado}`**) e tipos de resultados (restrição **`{resultType=nome-do-tipo-de-resultado}`**). Um resultado pode ser qualquer palavra-chave (sendo o valor *default* determinado pelo padrão do *framework*), enquanto os tipos de resultado são determinados pelo *framework*. Geralmente os seguintes tipos estão presentes:

- **`binary`**: um arquivo binário é retornado ao cliente como resultado do processamento da ação. Exemplos são relatórios em PDF, imagens geradas dinamicamente etc. É o resultado *default* quando a dependência é com um arquivo binário;
- **`chain`**: uma outra ação é executada como resultado da primeira (ações encadeadas). É o *default* quando a dependência é com uma classe de ação;
- **`dispatch`**: despacha a requisição para uma página dinâmica para processamento do resultado. É o *default* quando a dependência é com uma página;
- **`redirect`**: redireciona a requisição para uma página (estática ou dinâmica). A diferença entre o despacho e a redireção é que no primeiro os dados submetidos para a ação e obtidos no seu processamento estão disponíveis, enquanto no segundo não;
- **`template`**: um modelo é processado pelo *template engine* e seu resultado é retornado ao cliente. É o *default* quando a dependência é com um modelo.

O projetista é livre para escolher a granularidade das classes de ação, construindo uma para cada cenário de caso de uso, uma para cada caso de uso, uma para um conjunto de casos de uso, e assim por diante. Além disso, deve ponderar com relação à representação de várias ações no mesmo diagrama ou a utilização de um diagrama separado para cada ação.

Como exemplos temos as Figura 4.11 e 4.12. A primeira mostra em um único Modelo de Navegação ações para os três cenários do caso de uso “Autenticar Usuário” do subsistema *ControleUsuario* do Portal do LabES, enquanto a segunda serve exclusivamente ao cenário “Consultar Publicação” do caso de uso “Consultar Item” do subsistema *ControleItens*, ainda que a classe de ação sirva para os três cenários (note a presença dos métodos referentes aos demais cenários e repare que os atributos utilizados pela classe de ação na execução dos outros cenários não foram exibidos para não poluir o diagrama).

O Modelo de Navegação do caso de uso “Autenticar Usuário” mostra-nos que na página inicial do sistema (representada pela página **`web::index`**, convenção estabelecida para o projeto do Portal do LabES) possui um formulário com os campos **`login`** e **`senha`**. Esses dados são submetidos para a classe de ação para evocação do método **`executeEfetuarLogin()`** que deverá acessar a camada de Lógica de Negócio para execução do caso de uso. No caso das informações estarem corretas (**`result = success`**), o usuário é enviado para a página **`web::home`**, que representa a área de acesso restrito dos usuários autenticados. Caso contrário, o resultado retornará o resultado **`"input"`** (que indica problemas nos dados fornecidos) e o cliente será direcionado a uma página que exibirá novamente o formulário de *login*.

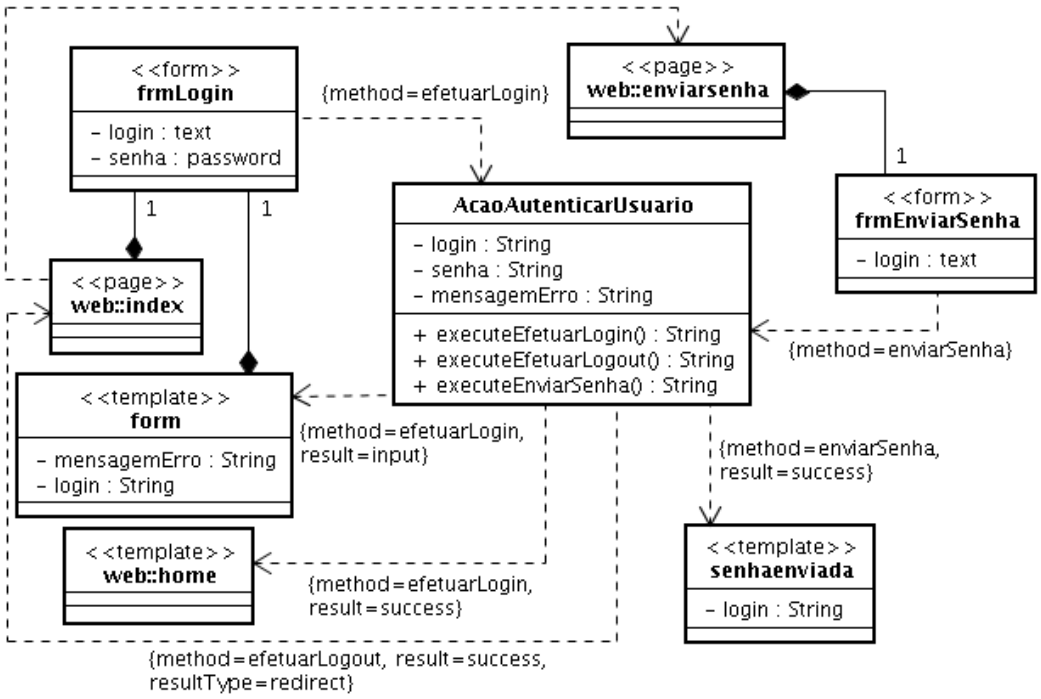

Figura 4.11: Modelo de Navegação para o caso de uso "Autenticar Usuário" do subsistema Controle Usuário do Portal do LabES.

No caso de esquecimento da senha, o usuário pode acionar um *link* na página principal para uma página que contém um formulário que pede seu login e o envia à ação **`enviarSenha`**. A classe de ação solicita à camada de Lógica de Negócio o envio da senha para o usuário por e-mail, exibindo como resultado uma página informando que a senha foi submetida. Para efetuar *logout*, basta que o usuário acione **`efetuarLogout`** que o sistema redirecionar-lhe-á novamente à página inicial, como se fosse um visitante comum.

O Modelo de Navegação de Consultar Publicação mostra que, a partir da página inicial, um internauta pode solicitar a listagem de publicações cadastradas por meio do método **`consultarPublicacao!input`** (esta é uma notação específica do Struts2, que permite a separação de um método em duas partes: exibição do formulário – com **`!input`** – e execução – sem **`!input`**). A classe de ação recupera o conjunto de publicações e os exibe no modelo **`listaPublicacoes`**. Este modelo possui uma série de formulários, um para cada publicação exibida, que permite que o internauta selecione uma das publicações para se obter mais detalhes.

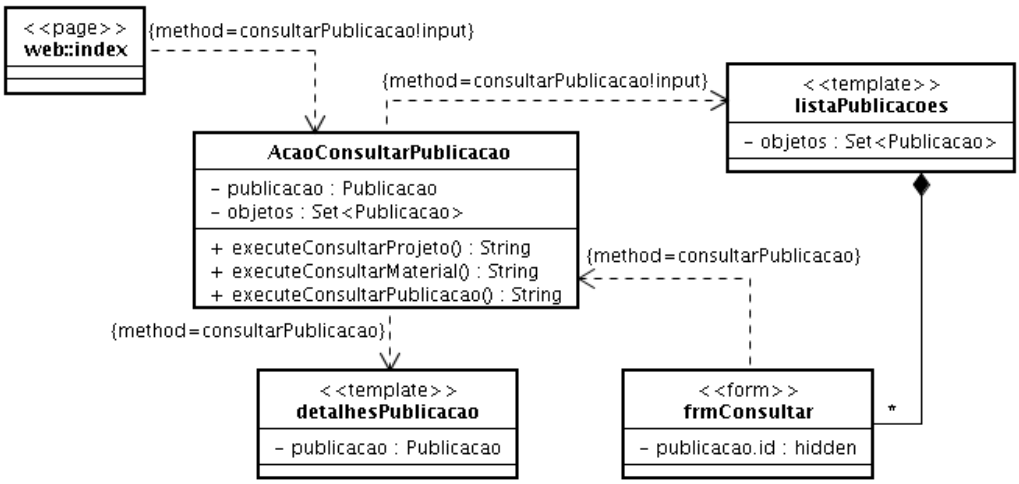

Figura 4.12: Modelo de Navegação do cenário "Consultar Publicação", do caso de uso "Consultar Item" do subsistema Controle Itens do Portal do LabES.

Durante a concepção de FrameWeb, no primeiro projeto experimental em que o método foi utilizado, discutiu-se junto à equipe de desenvolvedores qual diagrama UML melhor serviria como base para o Modelo de Navegação: o diagrama de classes ou o diagrama de seqüência. Duas razões foram as principais na escolha do diagrama de classes: (a) ele provê uma melhor visualização da estrutura interna das classes de ação, formulários, páginas e modelos; e (b) ele modela a composição entre páginas e formulários (ou modelos e formulários) com uma notação mais adequada. Apesar disso, se achar necessário, o projetista

é livre para usar diagramas de seqüência para representar navegação ou qualquer outra informação que desejar. Neste momento, consideramos interessante aplicar os princípios da Modelagem Ágil já citados, em especial “modele com um propósito”, “viaje com pouca bagagem” e “conteúdo é mais importante que apresentação”.

Os passos para a construção dos Modelos de Navegação são:

1. Analisar os casos de uso modelados durante a Especificação de Requisitos, definir a granularidade das classes de ação e criá-las, definindo seus métodos. Utilizar, preferencialmente, nomes que as relacionem com os casos de uso ou cenários que representam;
2. Identificar como os dados serão submetidos pelos clientes para criar as páginas, modelos e formulários, adicionando atributos à classe de ação;
3. Identificar quais resultados são possíveis a partir dos dados de entrada, para criar as páginas e modelos de resultado, também adicionando atributos à classe de ação.
4. Analisar se o modelo ficou muito complexo e considerar dividi-lo em dois ou mais diagramas.

4.3.4. Modelo de Aplicação

O Modelo de Aplicação é um diagrama de classes da UML que representa as classes de serviço, que são responsáveis pela codificação dos casos de uso, e suas dependências. Esse diagrama é utilizado para guiar a implementação das classes do pacote **`Aplicação`** e a configuração das dependências entre os pacotes **`Controle`**, **`Aplicação`** e **`Persistência`**, ou seja, quais classes de ação dependem de quais classes de serviço e quais DAOs são necessários para que as classes de serviço alcancem seus objetivos.

Assim como para as classes de ação do Modelo de Navegação, o projetista deve escolher o nível de granularidade das classes de serviço. Há também uma semelhança com o modelo de Persistência, pois no Modelo de Aplicação também não há definição de extensões UML e também valem as regras de programação por interfaces: cada classe de serviço deve ter uma interface e uma implementação.

Todas as classes de ação que dependem de uma classe de serviço exibida devem também aparecer no modelo, representando por meio de espaço de nomes que pertencem a outro pacote e indicando a dependência por meio de uma associação direcionada à interface da classe de serviço. Da mesma maneira, quando a classe de serviço depende de algum DAO, a interface deste deve aparecer no diagrama e estar associada à classe de serviço em questão.

A Figura 4.13 mostra um trecho do Modelo de Aplicação para o pacote

controleusuario do Portal do LabES. O modelo mostra que a ação de autenticação de usuários depende de uma classe de serviço que implementa a autenticação, isto é, implementa o caso de uso "Autenticar Usuário". Esta, por sua vez, para conseguir executar seus cenários de uso, precisa do DAO da classe **Usuario**, visto que efetuará consultas ao banco de dados para obter os dados de usuários a partir de seus *logins*. A ação e o serviço de cadastro de funcionalidades também são exibidos na figura.

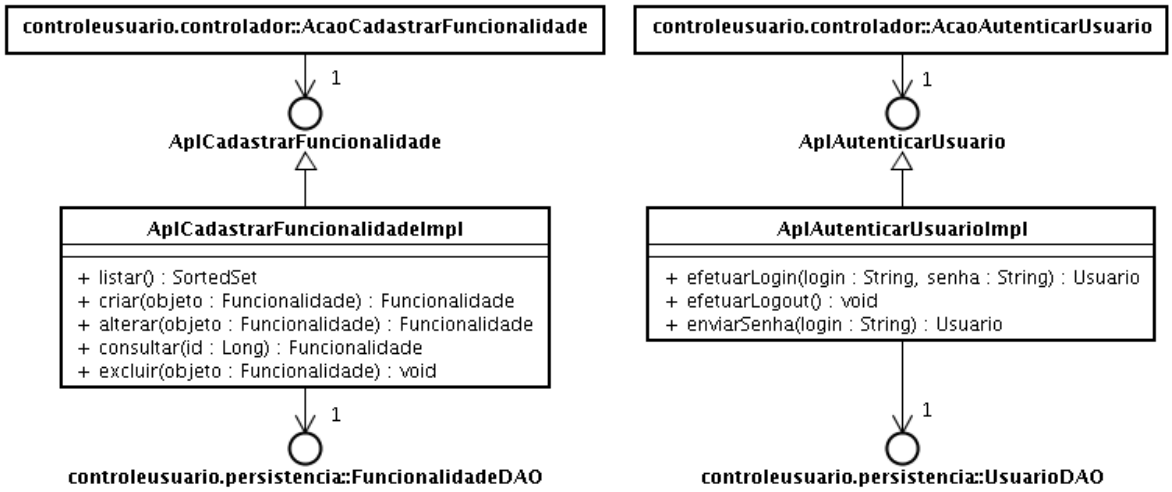

Figura 4.13: Parte do Modelo de Aplicação do pacote controleusuario do Portal do LabES.

Apesar de não ser expresso no diagrama, as classes de aplicação dependem das classes de domínio, visto que seu principal objetivo é manipular objetos de domínio para atender aos requisitos do sistema. Mostrar todas essas dependências no modelo de aplicação aumentaria a complexidade do diagrama desnecessariamente, visto que tais dependências podem ser obtidas analisando as especificações dos casos de uso atendidos pelas classes de aplicação.

Os passos para a construção do Modelo de Aplicação são:

1. Analisar os casos de uso modelados durante a Especificação de Requisitos, definir a granularidade das classes de serviço e criá-las. Utilizar, preferencialmente, nomes que as relacionem com os casos de uso ou cenários que representam;
2. Adicionar às classes/interfaces os métodos que implementam a lógica de negócio, com atenção ao nome escolhido (preferencialmente relacionar o método ao cenário que implementa), aos parâmetros de entrada e ao retorno (observar a descrição do caso de uso);
3. Por meio de uma leitura da descrição dos casos de uso, identificar quais DAOs

são necessários para cada classe de aplicação e modelar as associações;

4. Voltar ao modelo de navegação (se já foi construído), identificar quais classes de ação dependem de quais classes de serviço e modelar as associações.

4.4. Comparação com trabalhos relacionados

No Capítulo 2, apresentamos várias metodologias, métodos e linguagens de modelagem para Engenharia *Web*. Consideramos necessário justificar a apresentação de uma nova proposta, comparando-a com os trabalhos de propósito geral, a saber: a metodologia de Conallen / WAE (Seções 2.1.4 e 2.2.2), OOWS (Seção 2.1.5), UWE (Seções 2.1.6 e 2.2.3), WebML (Seção 2.2.1) e OOHDM (Seção 2.1.7).

FrameWeb baseou-se na mesma idéia que a linguagem WAE: a criação de um perfil UML para construção de diagramas que, no caso de FrameWeb, é voltado à arquitetura dos *frameworks* e, portanto, define estereótipos e restrições diferentes das propostas por Conallen. Outra diferença é a utilização da relação de dependência (linha pontilhada) ao invés da associação (linha sólida) para os relacionamentos entre os componentes da camada *Web*, pois consideramos que ela representa melhor a semântica dessas relações. Portanto, justifica-se a criação de um novo perfil ao invés de estender o perfil definido por Conallen.

Há ainda outras diferenças entre nossa proposta e a metodologia de Conallen: (a) FrameWeb introduz menos conceitos novos, facilitando a adoção por parte de desenvolvedores já proficientes com UML; (b) a metodologia de Conallen abrange todo o processo de software enquanto FrameWeb propõe um método para projeto de WISs, permitindo que organizações utilizem qualquer processo de software que lhes convém; e (c) no processo proposto por Conallen não há preocupação com a agilidade, enquanto FrameWeb baseia-se no uso de *frameworks* e em princípios da modelagem ágil para uma maior rapidez na construção de aplicações *Web*.

Ao contrário de WAE, OOWS propõe extensões não-padronizadas à UML, dificultando seu uso por organizações que não possuam ferramentas CASE específicas para o método. Neste sentido, FrameWeb se destaca por ser de aceitação mais fácil por parte de desenvolvedores que já possuam uma ferramenta CASE compatível com UML. Por outro lado, OOWS possui algumas características que faltam à nossa proposta: uma estratégia para geração de código a partir dos modelos e elementos de modelo que especificam mecanismos de indexação e filtragem, que são bastante comuns em aplicações *Web*.

UWE é bastante similar à FrameWeb no sentido de ser baseada em casos de uso e

diagramas de classe para construção de modelos de projeto. Assim como OOWS, apresenta algumas extensões não padronizadas e dá suporte à geração automática de código. Há uma ferramenta CASE construída especificamente para a linguagem, chamada ArgoUWE, que é baseada na ferramenta de código-aberto ArgoUML[20].

WebML, ao contrário das demais, é somente uma linguagem de modelagem, não estando associada com nenhuma metodologia específica. Isso traz uma grande vantagem para organizações que desejam manter seu processo de software inalterado. Sua grande desvantagem, porém, é não ser baseada em UML, o que pode dificultar sua aceitação por não haver muitas ferramentas disponíveis.

Por fim, OOHDM é um método para construção de aplicações hipermídia, provavelmente o mais conhecido. Como já mencionado na Seção 2.4, consideramos que o paradigma hipermídia não é suficiente para o desenvolvimento de WISs pelo fato do primeiro ter um foco maior na estrutura e navegação, enquanto o segundo atribui uma importância maior à funcionalidade. No entanto, algumas características como os modelos de navegação de OOHDM podem ser comparadas com FrameWeb, incidindo novamente na vantagem de nossa proposta ser baseada em UML e suas extensões padronizadas, enquanto muitos modelos em OOHDM são feitos com notações próprias.

Dadas todas as opções disponíveis, FrameWeb se apresenta como uma alternativa que foca uma arquitetura específica para sistemas de informação *Web* que utilizam *frameworks* ou que são baseados em *containers*. Neste contexto, destaca-se pela agilidade, pois seus modelos de projeto representam conceitos da arquitetura dos *frameworks*, permitindo rápido entendimento e implementação por parte de profissionais treinados. Também consideramos que FrameWeb introduz poucos conceitos novos, o que, aliado ao uso de padrões estabelecidos, facilita o aprendizado por parte dos projetistas e, por conseguinte, sua adoção nas organizações.

4.5. Conclusões do Capítulo

Frameworks são comumente utilizados atualmente para construção de WISs, ao ponto de influenciarem diretamente na definição de arquiteturas padronizadas para ambientes distribuídos, como a *Web*. A partir desse contexto, identificamos a necessidade de trazer o projeto de sistemas *Web* para mais próximo das tecnologias de implementação utilizadas e, para isso, propusemos FrameWeb. O método define uma arquitetura básica para WISs e quatro modelos de projeto que se utilizam de um perfil UML para uma representação direta de

20 http://argouml.tigris.org/

componentes dos *frameworks*.

O Portal do LabES, utilizado durante todo o capítulo para ilustrar o método, foi o projeto piloto de FrameWeb, desenvolvido por uma equipe de estudantes de computação que passaram por um treinamento sobre Engenharia Web, FrameWeb e um conjunto de *frameworks* (Struts2, Spring Framework, Hibernate e Sitemesh). Apesar de alguns desenvolvedores apresentarem dificuldade em compreender, em um curto intervalo de tempo, a arquitetura do *framework* Controlador Frontal, o desenvolvimento ocorreu de forma satisfatória, com a maioria dos módulos entregues dentro do prazo. Visto que todos eles conheciam a linguagem Java, mas poucos possuíam experiência com desenvolvimento para a *Web*, consideramos o resultado bastante satisfatório.

Ao final do projeto, foi solicitado aos desenvolvedores que comentassem sobre o trabalho realizado, dando um retorno em relação à utilização do método FrameWeb. Tal retorno pode ser resumido em três pontos:

- Permitir a modelagem direta de aspectos relacionados ao uso dos *frameworks* é o ponto forte de FrameWeb;
- Implementar em Java o que foi modelado na fase de projeto foi bastante facilitado pelo entendimento claro da semântica dos quatro modelos (domínio, persistência, navegação e aplicação);
- A simplicidade dos modelos facilitou a adoção de FrameWeb, com exceção do modelo de navegação, que adicionou uma certa complexidade ao método.

As Figuras 4.14 e 4.15 mostram telas do Portal do LabES, implementado por Peruch (2007).

É importante mencionar, porém, que a utilização de FrameWeb para construção dos quatro tipos de modelos de projeto apresentados neste capítulo não exime o projetista de utilizar outros métodos, modelos ou linguagens para descrever aspectos do projeto do sistema que não são cobertos pelos modelos de FrameWeb, como, por exemplo, o projeto visual da aplicação. FrameWeb define modelos que trazem informações relacionadas aos *frameworks* e à sua arquitetura padrão e deve ser combinado com outros tipos de modelos que se fizerem necessários.

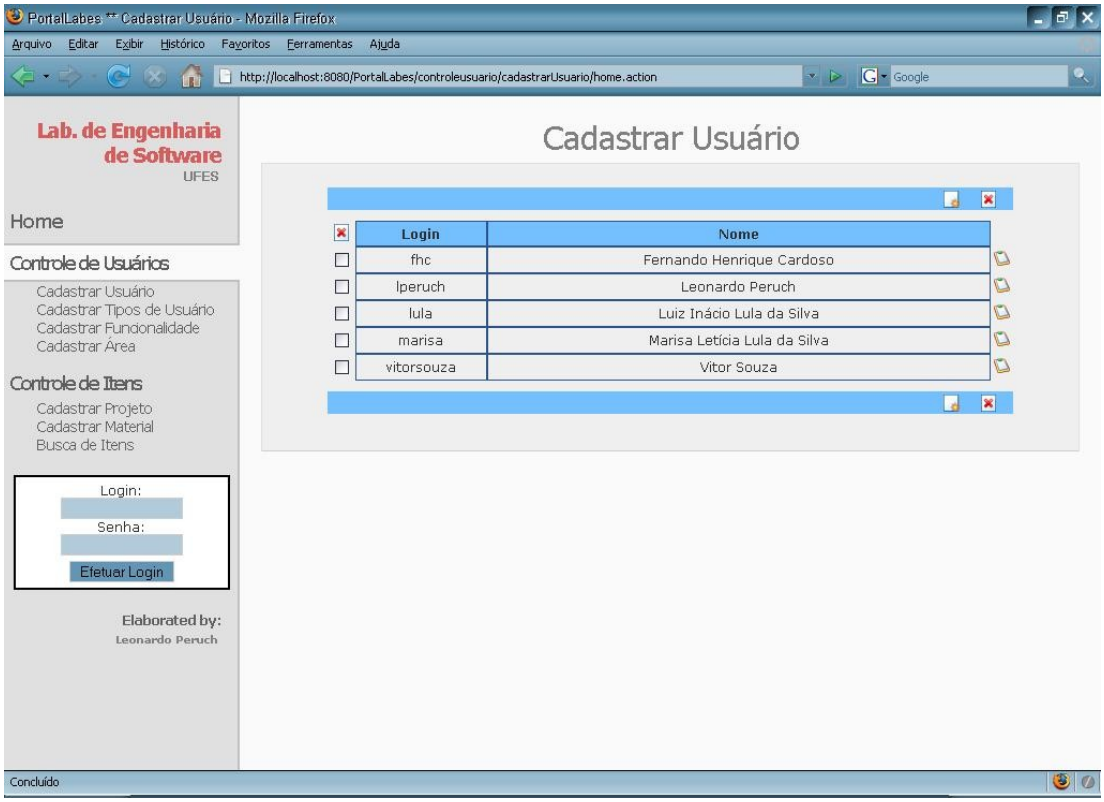

Figura 4.14: Tela de cadastro de usuários do Portal do LabES (PERUCH, 2007).

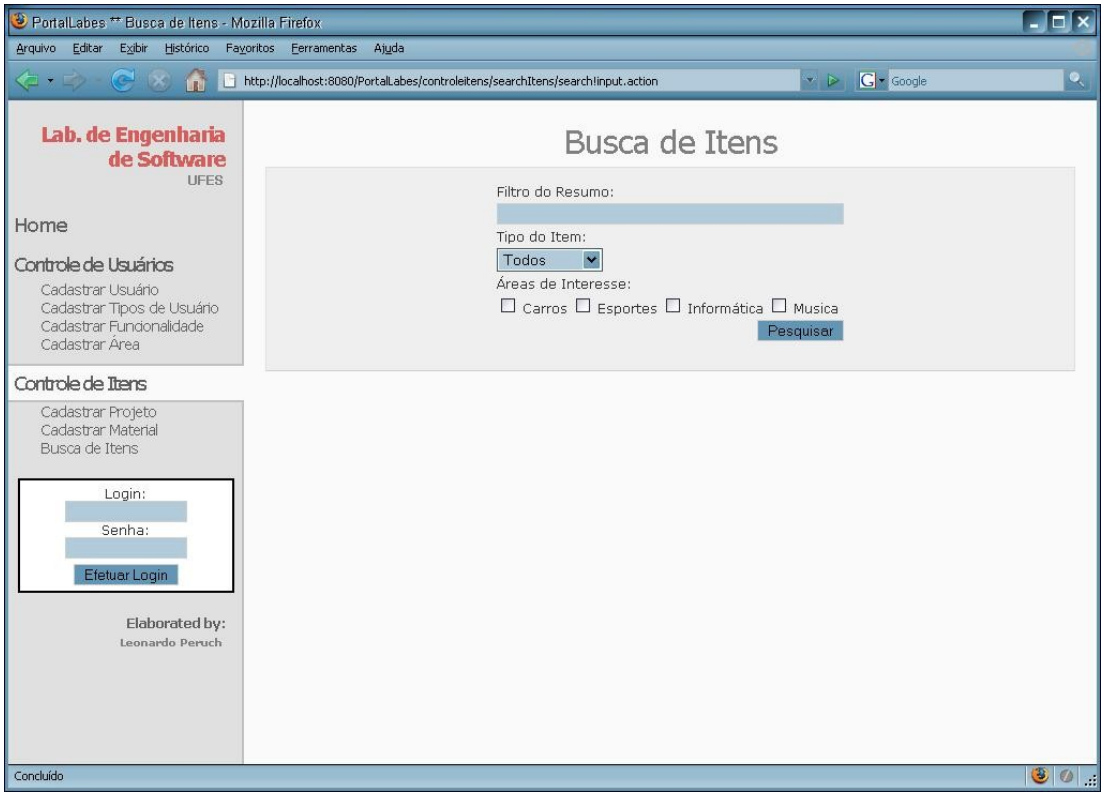

Figura 4.15: Tela de busca de itens do Portal do LabES (PERUCH, 2007).

Atualmente encontram-se em desenvolvimento outras iniciativas de uso de FrameWeb na prática. Tais iniciativas, assim como a experiência com o Portal do LabES, não caracterizam estudos formais de Engenharia de Software Experimental, mas apenas estudos de caso preliminares, que podem abrir espaço para trabalhos mais aprofundados. São elas:

- O Grupo de Usuários de Java do Estado do Espírito Santo[21] organiza grupos de desenvolvimento de software em Java e atualmente conduz o projeto JSchool[22], no qual uma equipe de 10 membros do JUG desenvolve um ambiente colaborativo de aprendizagem modelado com FrameWeb. A equipe recebeu treinamento sobre o método e os *frameworks* utilizados e o projeto encontra-se atualmente no estágio de implementação;
- Em (PERUCH, 2007), a partir do que já havia sido desenvolvido no projeto do Portal do LabES, Peruch integrou os componentes das diferentes equipes anteriores em um único portal e desenvolveu, em seguida, duas novas versões do sistema, utilizando *frameworks* Controladores Frontais diferentes, com o intuito de analisar a eficácia do método em diferentes contextos e sugerir alterações nos modelos para uma melhor adaptação a diferentes *frameworks*. O projeto encontra-se em fase final;
- Também como projeto final de graduação, Stênio Stein pretende desenvolver um sistema de gerência de projetos baseado nas necessidades da empresa em que atualmente conduz seu estágio. O projeto do sistema será feito com FrameWeb e permitirá avaliar o método no contexto de uma empresa de desenvolvimento de software. O trabalho encontra-se em fase inicial.

Em suma, este trabalho definiu a primeira versão de FrameWeb, um método para projeto de sistemas de informação *Web* com arquiteturas baseadas no uso de determinados tipos de *frameworks*. No entanto, há muito espaço para evolução do método. Um desses espaços, que resolvemos já começar a trilhar, é o desenvolvimento de aplicações *Web* preparadas para a *Web* Semântica usando FrameWeb. O próximo capítulo apresenta nossas propostas para adição de semântica a FrameWeb.

21 http://esjug.dev.java.net

22 http://jschool.dev.java.net

Capítulo 5: S-FrameWeb

Considerando que a *Web* Semântica será uma realidade somente quando os autores de páginas *Web* adicionarem semântica às suas páginas, pensamos ser importante incluir na proposta diretrizes para desenvolvimento de aplicações *Web* com semântica associada, de forma a auxiliar na construção desse novo paradigma da Internet.

Com o objetivo de auxiliar desenvolvedores a fazerem com que seus WISs gerem anotações dinâmicas para interpretação por agentes da Web Semântica, propomos neste trabalho algumas extensões para o método apresentado no Capítulo 4 que auxiliam o desenvolvedor a construir Sistemas de Informação *Web* (*Web Information Systems* – WISs) preparados para este novo paradigma da *World Wide Web*.

Este capítulo apresenta S-FrameWeb (SOUZA et al., 2007c), uma extensão de FrameWeb que visa apoiar o desenvolvimento de WISs Semânticos, e está organizado da seguinte forma: a Seção 5.1 apresenta uma visão geral de S-FrameWeb, apresentando o processo de desenvolvimento sugerido pela extensão; as seções seguintes abordam cada uma das fases desse processo: Análise de Domínio (Seção 5.2), Especificação e Análise de Requisitos (Seção 5.3), Projeto (Seção 5.4), Implementação, Testes e Implantação (Seção 5.5). Finalmente, a Seção 5.6 apresenta as considerações finais do capítulo.

5.1. Visão Geral de S-FrameWeb

As extensões propostas por S-FrameWeb a FrameWeb são resumidas nos seguintes pontos:

1. Uma atividade de Análise de Domínio deve ser conduzida no início do projeto para construção de uma ontologia para o domínio no qual o sistema *Web* será construído. Se uma ontologia daquele domínio já existir, deve ser reutilizada, refinando-a se necessário;
2. Atividades de Especificação de Requisitos e Análise seguem sem maiores alterações, exceto pelo fato de se sugerir que os modelos conceituais do domínio, na fase de Análise, sejam baseados na ontologia de domínio desenvolvida na Análise de Domínio;
3. Na fase de Projeto, o Modelo de Domínio de FrameWeb receberá anotações

semânticas baseadas na ontologia de domínio;

4. Na Implementação, o *framework* MVC utilizado deve ser estendido de forma a substituir as respostas dirigidas a humanos (ex.: páginas HTML) por documentos passíveis de serem interpretados por agentes de software (ex.: documentos OWL). Tal extensão deve ser capaz de decidir qual resposta utilizar por meio de um parâmetro na requisição HTTP.

Unindo as extensões propostas com o que já estabelece e sugere o método FrameWeb (apresentado no Capítulo 4), temos o processo de desenvolvimento sugerido por S-FrameWeb, apresentado na Figura 5.1, na qual estão destacados os artefatos mais significativos dentro do contexto de S-FrameWeb. A Tabela 5.1 resume o que cada um desses artefatos representa.

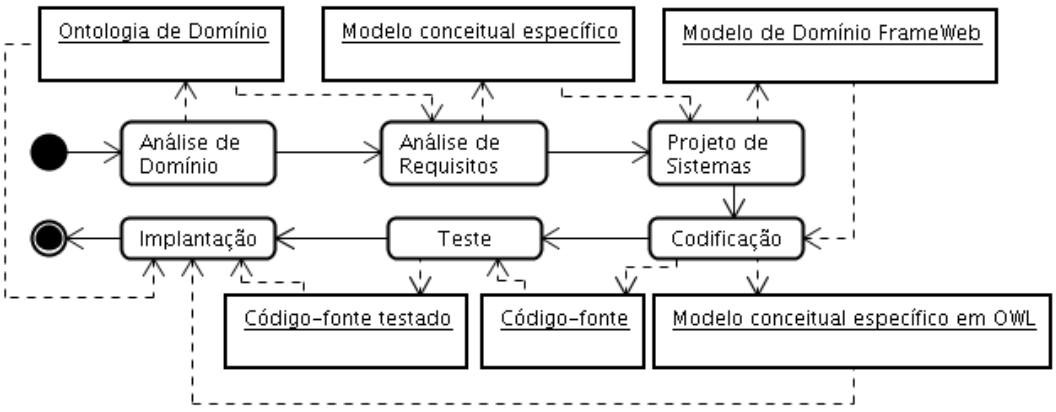

Figura 5.1: Processo de desenvolvimento de software sugerido por S-FrameWeb.

Tabela 5.1: Artefatos produzidos pelo processo sugerido por S-FrameWeb.

Atividade	***Artefato***	***O que o modelo representa***
Análise de Domínio	Ontologia de Domínio	Conceitos do domínio dentro do qual o software está sendo construído. É modelado em ODM e convertido para OWL para implantação.
Análise de Requisitos	Modelo conceitual específico	Conceitos específicos do problema que está sendo resolvido. Também modelado em ODM.
Projeto de Sistemas	Modelo de Domínio FrameWeb (MDF)	O mesmo que o modelo conceitual específico, adicionado de mapeamentos Objeto/Relacionais (O/R). Modelado usando o perfil UML de S-FrameWeb.
Codificação	Modelo conceitual específico em OWL	Representação em OWL do MDF, sem os mapeamentos O/R.

Nas seções a seguir, cada uma das atividades propostas por S-FrameWeb é abordada em mais detalhes. Ao longo dessas seções, utilizamos como exemplo o Portal do LabES, já apresentado na Seção 4.1. A implementação desse sistema utilizando FrameWeb se deu no contexto do trabalho de Peruch (2007) e sua adaptação para a *Web* Semântica utilizando S-FrameWeb ocorreu em (LOURENÇO, 2007).

5.2. Análise de Domínio

Para trazer um WIS para a *Web* Semântica, é imprescindível descrever formalmente o domínio no qual o sistema se encontra. Para isso, S-FrameWeb inclui no processo de desenvolvimento de WISs uma fase de Análise de Domínio, que é uma atividade voltada para a identificação de objetos e operações de uma classe de sistemas similares num domínio de problema particular (NEIGHBORS, 1981; FALBO et al., 2002).

Quando um software é construído, o propósito é resolver um problema que está inserido num determinado domínio de especialidade, como medicina, vendas ou siderurgia. Se o domínio for analisado antes de se analisar o problema específico, o conhecimento formalizado sobre o domínio pode ser reutilizado quando outro problema do mesmo domínio necessitar de uma solução via software (FALBO et al., 2002).

Conforme discutido no Capítulo 3, uma das formas mais utilizadas atualmente para representação de conhecimento de domínio são as ontologias. A atividade de Análise de Domínio produz um modelo de domínio que pode ser, portanto, uma Ontologia de Domínio. Neste capítulo, porém, evitamos usar o termo "modelo de domínio" para designar o produto da Análise de Domínio para não causar confusão com o Modelo de Domínio de FrameWeb. Para reforçar essa diferenciação, neste capítulo, utilizamos a sigla MDF para designar este último.

Na Seção 3.2.1 foram apresentadas algumas metodologias para a construção e linguagens para representação de ontologias. S-FrameWeb não preconiza o uso de nenhum método específico para a construção de ontologias.

Assim, o resultado da atividade de Análise de Domínio de S-FrameWeb é uma ontologia que representa os conceitos do domínio do problema. Para o Portal do LabES foi seguido o método SABiO (FALBO, 2004) e construída uma ontologia do domínio de portais educacionais com o objetivo de tratar as seguintes questões de competência (LOURENÇO, 2007):

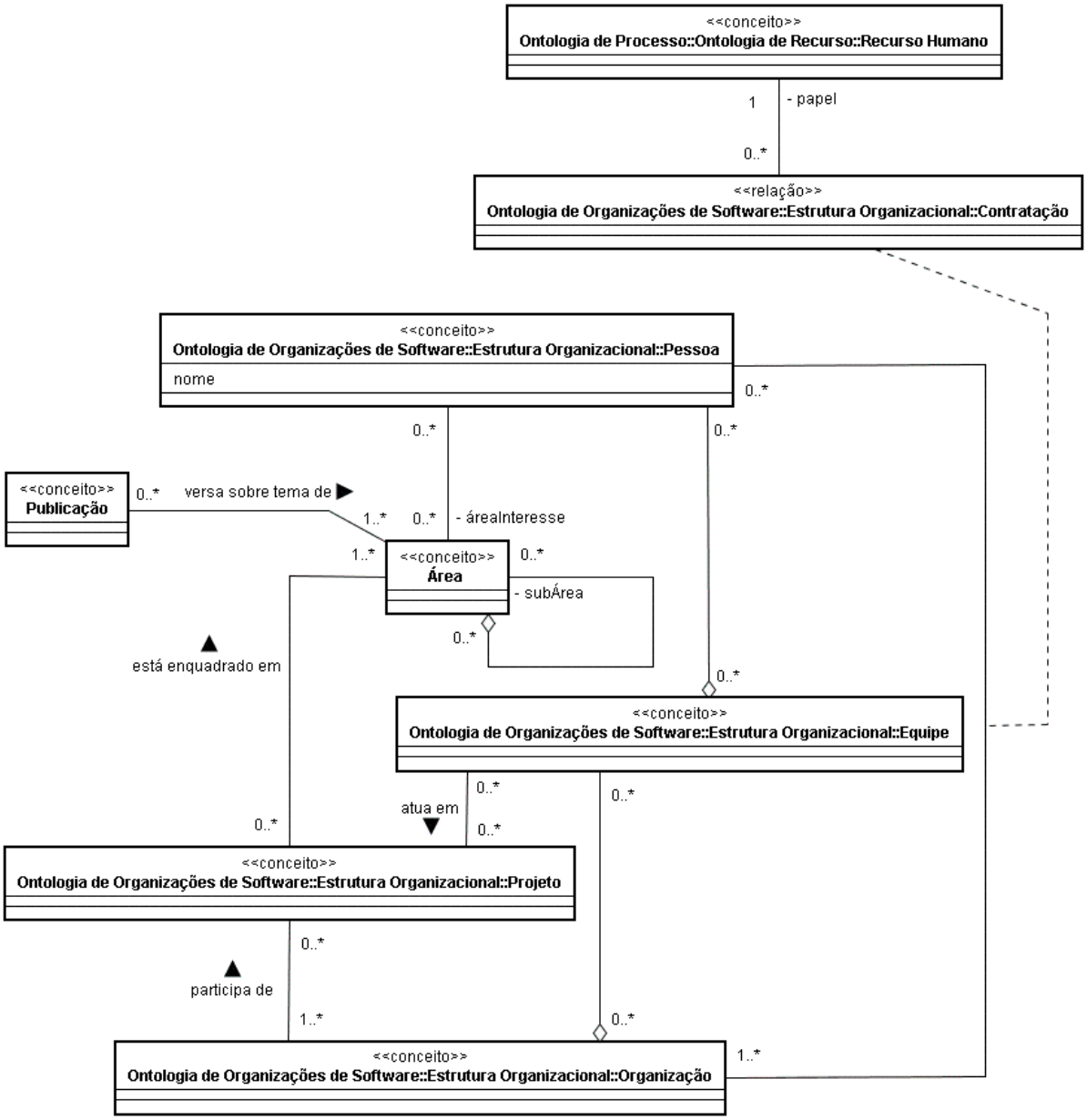

Figura 5.2: Modelo da parte estrutural da ontologia do domínio de portais educacionais (LOURENÇO, 2007)

QC 1 - Quais as pessoas relacionadas com a instituição em questão?

QC 2 - Que papéis essas pessoas desempenham na instituição?

QC 3 - Quais as áreas de pesquisa de interesse da organização?

QC 4 - Como essas áreas estão organizadas?

QC 5 - Quais as áreas de interesse das pessoas da organização?

QC 6 - Como as pessoas estão organizadas?

QC 7 - Quais os projetos em desenvolvimento na instituição?

QC 8 - Em que áreas esses projetos se enquadram?

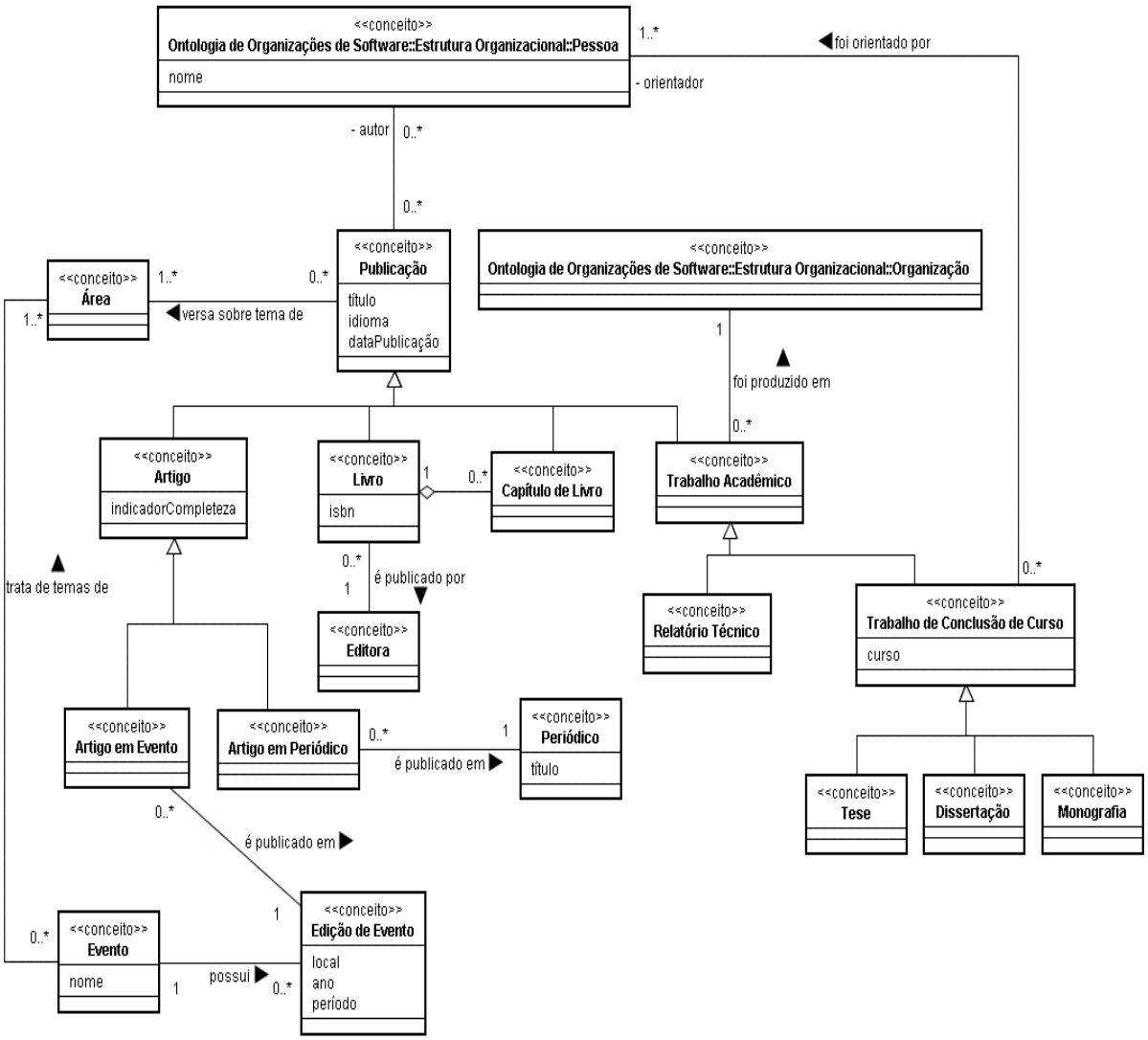

Figura 5.3: Modelo da parte de publicações da ontologia do domínio de portais educacionais (LOURENÇO, 2007)

QC 9 - Quais as instituições e grupos envolvidos nos projetos?

QC 10 - Quais as publicações produzidas?

QC 11 - Quais tipos de publicações produzidas?

Para simplificar o entendimento do problema, foram produzidos dois diagramas distintos, um para representar a estrutura geral dos portais educacionais e outro para tratar especificamente do conhecimento sobre publicações. As Figuras 5.2 e 5.3 mostram, respectivamente, tais diagramas.

A ontologia dos portais educacionais integra-se com a ontologia de organizações de software proposta por (RUY, 2006) e, nesse contexto, alguns conceitos da ontologia de organizações recebem sinônimos na ontologia dos portais educacionais, resumidos na Tabela

5.2, de forma a utilizar um vocabulário mais apropriado ao domínio em questão (LOURENÇO, 2007).

Tabela 5.2: Sinônimos entre as ontologias organizacional e de portais educacionais

Conceito na ontologia organizacional	*Sinônimo na ontologia de portais educacionais*
Instituição	Organização
Contratação	Vínculo
Equipe	Grupo de Interesse

A ontologia descreve um domínio no qual pessoas são vinculadas a organizações para exercer determinados papéis, tais como professor, aluno, pesquisador etc. (QCs 1 e 2). As pessoas possuem áreas de interesse e podem se organizar em grupos de interesse (QCs 5 e 6). Instituições e grupos de interesse podem envolver-se em projetos que, por sua vez, são enquadrados em áreas de pesquisa (QCs 7, 8 e 9). Por fim, publicações de diversos tipos são produzidas, versando sobre temas relacionados às áreas de pesquisa (QC 11) (LOURENÇO, 2007).

Esse diagrama é a base para a construção do modelo conceitual específico da aplicação (durante a fase de Análise de Requisitos), que deve derivar algumas classes e associações a partir dos elementos da ontologia de domínio, adicionando e modificando o que for adequado para o problema específico.

5.3. Especificação e Análise de Requisitos

As fases de especificação e análise de requisitos devem ser conduzidas pela equipe de desenvolvimento utilizando a metodologia de preferência da equipe: S-FrameWeb, assim como FrameWeb, não impõe nenhum método ou linguagem para esta fase. Apenas sugere, no entanto, que a linguagem ODM (OMG, 2006), descrita na Seção 3.2.1.1, seja utilizada para a representação gráfica dos modelos conceituais da Análise de Requisitos, de forma a facilitar a conversão destes para o MDF e, posteriormente, para código (OWL). Se desejado, ela poderia ser usada até mesmo na modelagem da ontologia de domínio para alcançar o mesmo benefício.

Nessa fase podem ser gerados diversos artefatos de Engenharia de Software como tabelas de requisitos, modelos de casos de uso, modelos conceituais, modelos dinâmicos etc. No contexto de S-FrameWeb, o artefato mais significativo é o modelo conceitual, que representa as classes do domínio do problema, seus atributos e relacionamentos, com foco

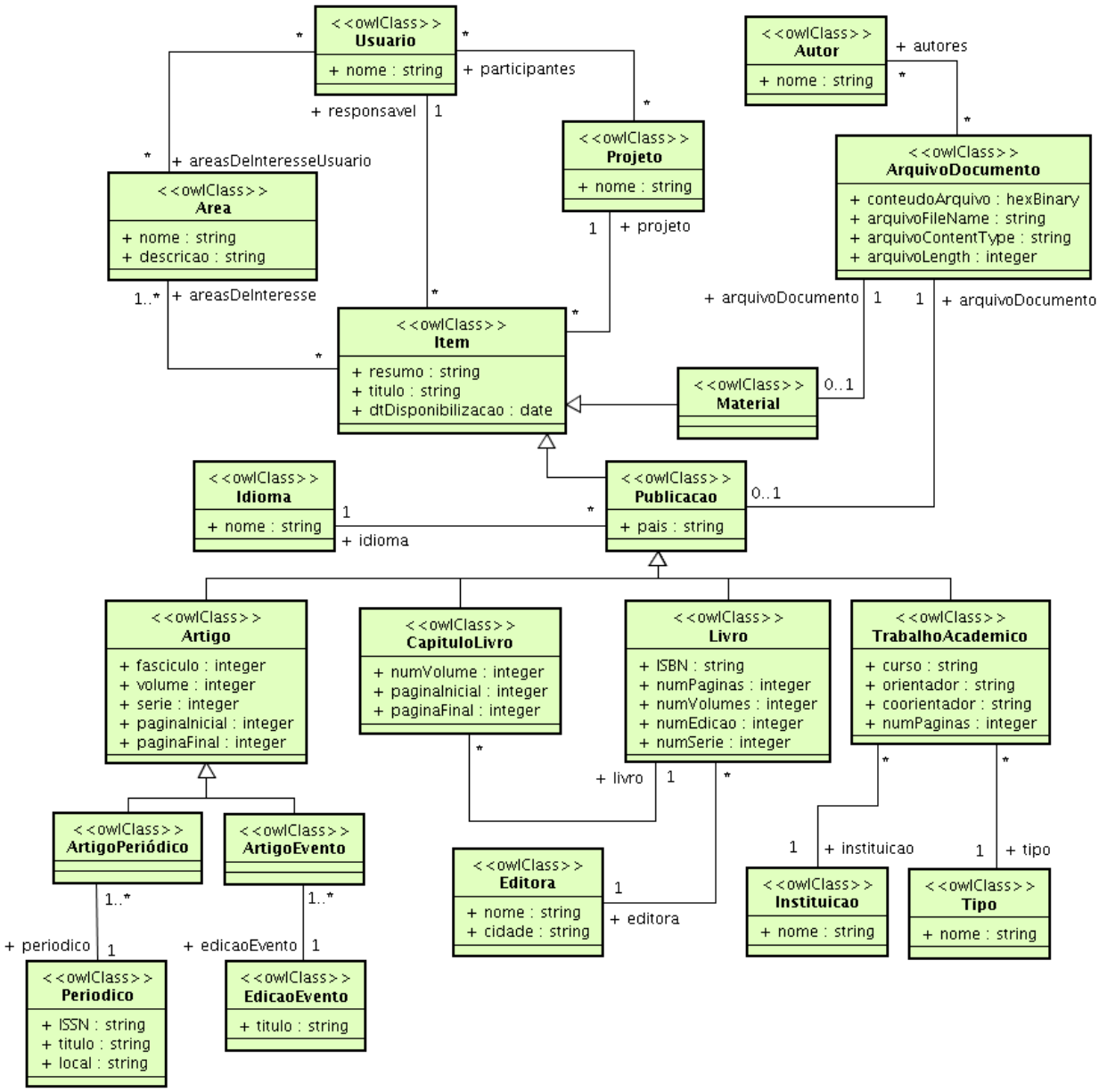

Figura 5.4: Modelo conceitual do Portal do LabES (módulo de controle de itens).

específico no problema sendo resolvido (a Seção 3.2 discute diferenças entre ontologias de domínio e modelos conceituais de aplicação).

Invariavelmente, diversos elementos presentes no modelo conceitual da aplicação se referem a conceitos do domínio do problema que já foram modelados num contexto mais amplo, a ontologia de domínio, durante a fase de Análise de Domínio. Portanto, a ontologia de domínio deve servir de base para a construção do modelo conceitual da aplicação, retirando-se o que não for relevante ao problema e adicionando conceitos que são muito específicos para estarem presentes na ontologia.

A Figura 5.4 mostra o modelo conceitual do Portal do LabES, construído com base nos conceitos da ontologia de portais educacionais apresentada anteriormente. O diagrama foi

construído utilizando ODM, com atributos simplificados, como sugerido por S-FrameWeb.

O modelo conceitual deve ser refinado na fase de projeto, ganhando novos elementos que representam preocupações arquitetônicas, específicas da plataforma de implementação.

5.4. Projeto

Na fase de projeto, FrameWeb propõe a criação de quatro tipos de diagrama, conforme discutido no Capítulo 4: Modelo de Domínio, Modelo de Persistência, Modelo de Navegação e Modelo de Aplicação. Tais modelos continuam sendo utilizados em S-FrameWeb, no entanto, o Modelo de Domínio de FrameWeb (MDF) deve ser alterado para uma representação mais adequada aos propósitos dessa extensão do método. S-FrameWeb, portanto, sugere um novo perfil UML para esse modelo, misturando o perfil definido por ODM com o perfil definido por FrameWeb.

Esse novo perfil consiste basicamente do perfil definido por ODM, com as seguintes modificações: (a) adição da navegabilidade das associações para implementação das classes; (b) adição dos mapeamentos Objeto/Relacionais para configuração do *framework* ORM, conforme discutido na Seção 4.3.1; e (c) utilização dos tipos de dados da plataforma de implementação, ao invés dos tipos definidos pelo padrão XSD (XML Schema Definition)[23]. A Figura 5.5 mostra o MDF do Portal do LabES.

Naturalmente, a construção do MDF deve ser baseada no modelo conceitual já construído em ODM na fase de Análise de Requisitos. Partindo desse modelo, basta proceder com as alterações descritas anteriormente: navegabilidade, mapeamentos O/R e mapeamentos de tipos XSD para os tipos específicos da plataforma de implementação. Uma comparação entre o MDF do Portal do LabES (Figura 5.5) e seu modelo conceitual (Figura 5.3) mostra o resultado desse processo.

A representação desse modelo num perfil UML que mistura ODM e FrameWeb visa facilitar, na fase de implementação, tanto a criação de um arquivo OWL que represente o modelo conceitual de projeto específico da aplicação, quanto a implementação do sistema usando o *framework* de mapeamento Objeto/Relacional escolhido.

23 O padrão XML Schema encontra-se no site da W3C em http://www.w3.org/XML/Schema. Os tipos de dados podem ser encontrados em página específica no endereço http://www.w3.org/TR/xmlschema-2.

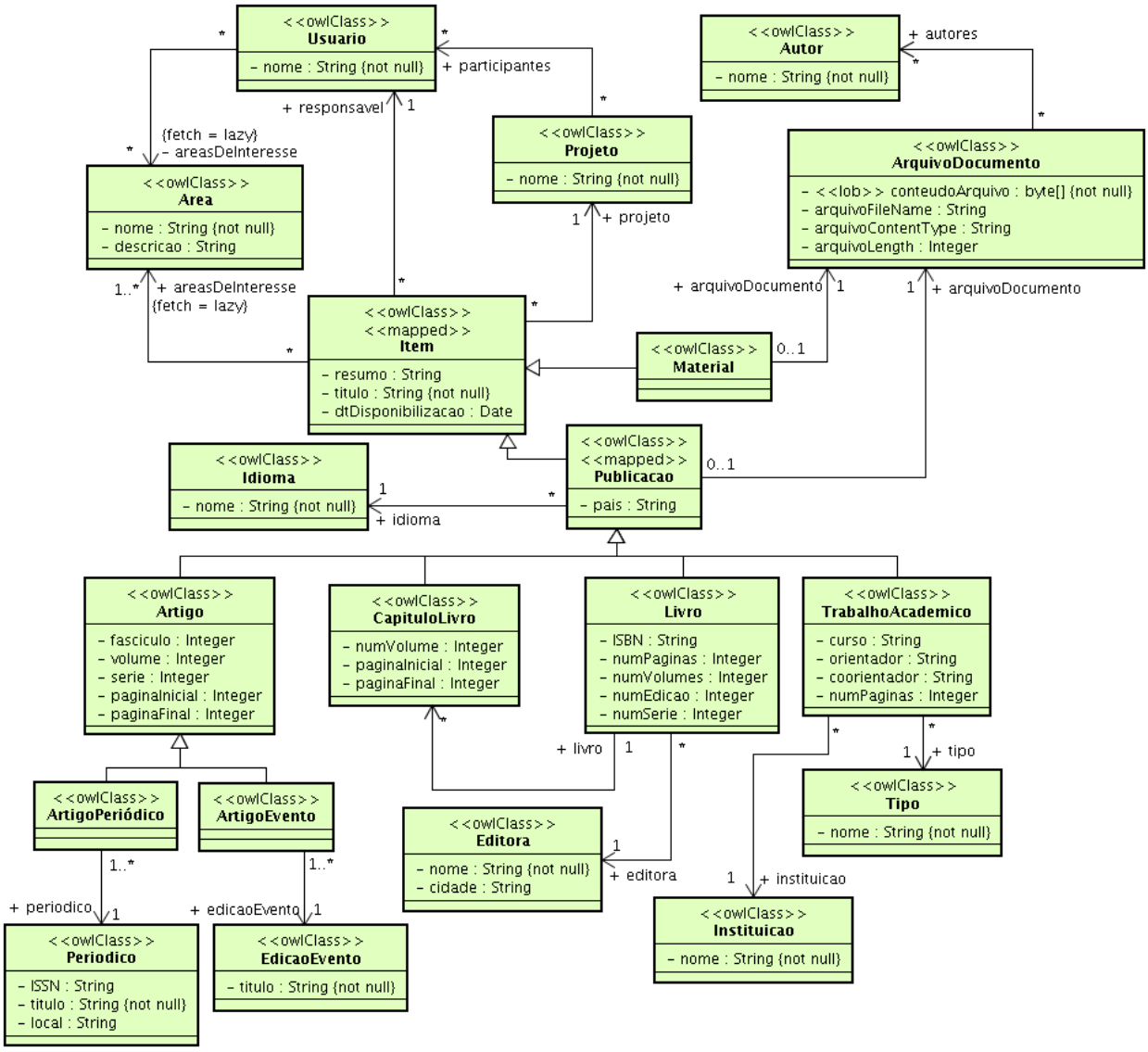

Figura 5.5: Modelo de Domínio de S-FrameWeb para o Portal do LabES (módulo de controle de itens).

5.5. Implementação, Testes e Implantação

Na fase de implementação são codificadas as classes que, juntamente com os *frameworks*, compõem a solução computacional para o problema em questão. S-FrameWeb adiciona uma nova tarefa para essa fase: a codificação da ontologia de domínio e do modelo conceitual específico da aplicação (a partir do MDF) em OWL.

A construção do documento OWL representando a ontologia de domínio e das classes que serão implementadas na aplicação é facilitada pela utilização de ODM nos modelos. No caso do MDF, como os tipos de dados representados referem-se a tipos existentes na

plataforma de implementação, o desenvolvedor deve ter a informação de que tipos do padrão XSD, usado em documentos OWL, representam os tipos específicos da plataforma de implementação escolhida. Se o modelo conceitual de análise foi construído em ODM, como no Portal do LabES, tal mapeamento pode ser derivado a partir de uma comparação entre o MDF e o modelo conceitual.

Tais documentos OWL são construídos para serem lidos e interpretados por agentes da *Web* Semântica. Eles veiculam informações estáticas sobre o domínio e a aplicação, porém as páginas do WIS são geradas dinamicamente e devem também produzir anotações para os agentes de software. Na Seção 3.4, foram discutidas duas abordagens para adição de semântica aos WISs. S-FrameWeb segue a abordagem de anotação dinâmica de *websites*, propondo uma extensão aos *frameworks* Controladores Frontais para a geração automatizada dessas anotações.

O funcionamento dos *frameworks* Controladores Frontais foi discutido na Seção 2.3.1 e representado na Figura 2.12. A extensão proposta por S-FrameWeb consiste em adicionar um parâmetro à requisição HTTP enviada a um WIS quando esta for proveniente de um agente de software. O *framework* deve, então, verificar a existência desse parâmetro e, caso identifique que a requisição vem de um agente, deve ler os arquivos OWL da ontologia de domínio e do modelo conceitual específico da aplicação e responder à requisição baseado nos resultados da ação que foi solicitada.

Utilizando alguma ferramenta de interpretação e raciocínio (*reasoning*) de ontologias codificadas em OWL, além do mecanismo de reflexão da linguagem de implementação escolhida, a extensão do *framework* Controlador Frontal deve executar a ação requisitada pelo agente de software da mesma forma que o faria se fosse solicitada por um ser humano. A diferença se dá apenas no resultado que, no primeiro caso, é mostrado em OWL e, no segundo, é exibido em HTML ou algum outro formato amigável ao ser humano.

Frameworks Controladores Frontais não possuem esse tipo de funcionalidade, porém geralmente trazem algum mecanismo de extensão que permite que o mesmo seja implementado, como é o caso do Struts2, *framework* utilizado na construção do Portal do LabES. Lourenço (2007) desenvolveu um protótipo dessa extensão para o *framework* Struts2, cujo esquema é exibido na Figura 5.6.

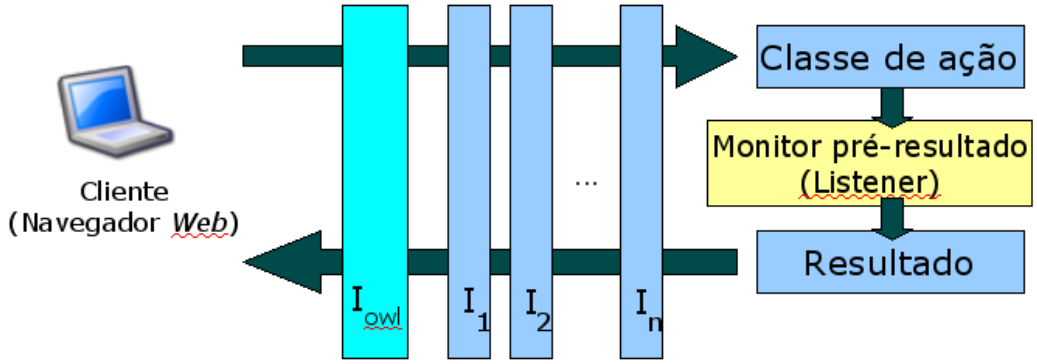

Figura 5.6: Extensão do *framework* Struts2 (SOUZA et al., 2007c)

O *framework* Struts2 pode ser estendido por meio de interceptadores. Esse mecanismo, que funciona como o padrão arquitetural dutos e filtros (*pipes and filters*) (SHAW e GARLAN, 1996), consiste do controlador frontal delegar a requisição a uma seqüência de interceptadores, que são classes construídas e registradas especificamente para este fim. Os interceptadores podem manipular a requisição e até mesmo influenciar em seu resultado final. Este comportamento é uma funcionalidade do Struts2 e muitas das funcionalidades do próprio *framework* são construídas utilizando esse mecanismo, como é o caso dos sistemas de conversão de tipos, validação de dados e envio (*upload*) de arquivos (SOUZA et al., 2007c).

Foi desenvolvido e configurado um "Interceptador OWL", cuja responsabilidade é verificar a existência de um parâmetro específico na requisição HTTP, indicando que a mesma foi feita por um software, e não um ser humano. Caso seja identificado que a requisição foi feita por um agente, o interceptador registra um monitor pré-resultado (*pre-result listener*) que interromperá o processamento normal do resultado e substitui-lo-á por um "Resultado OWL" (LOURENÇO, 2007).

Utilizando o mecanismo de reflexão da plataforma Java, o "Resultado OWL" identifica todos os atributos da classe de ação recém-executada que são instâncias ou coleções de instâncias de classes de domínio. Em requisições feitas por clientes humanos, esses atributos são utilizados para mostrar o resultado da requisição na página HTML e, portanto, representam a resposta à requisição feita pelo cliente. O "Resultado OWL", então, converte todas as instâncias para instâncias OWL, utilizando o mecanismo de inferência Jena (DICKINSON, 2007) e os apresenta juntamente com a ontologia de domínio e o modelo conceitual da aplicação (LOURENÇO, 2007).

Com as informações do domínio, da aplicação e das instâncias retornadas pela requisição, o agente tem condições de interpretar a resposta dada pelo sistema e produzir o

resultado esperado ao final. A Figura 5.7 mostra um trecho de um arquivo OWL retornado como resultado de uma busca por itens cujo resumo contém a palavra “FrameWeb”. Os resultados específicos da busca encontram-se entre as chaves **<result>** e **</result>**, enquanto os elementos entre **<instanceList>** e **</instanceList>** são instâncias de classes que são referenciadas pelos objetos retornados no resultado principal. As referências são representadas pelo identificador universal (UUID) de cada objeto. Por exemplo, na figura podemos ver que o artigo retornado como resultado faz referência às áreas de interesse “Web Semântica” e “WebApps” colocando o UUID destes dois objetos dentro das chaves **<areasDeInteresseItem></areasDeInteresseItem>**.

```
<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>
</result>
<instancesList>
	<instance>
		<uuid>130c2f70-5a37-4ad3-815b-841922584cd9</uuid>
		<name>Web Semântica</name>
	</instance>
	<instance>
		<uuid>93fcbf36-cdfe-4fd3-be23-a0d6ab3b45e8</uuid>
		<name>WebApps</name>
	</instance>
	<instance>
		<uuid>e5242491-b2be-4a34-b7b4-b0d9b7537517</uuid>
		<name>Vítor Souza</name>
		<dtNascimento>1979-06-05</dtNascimento>
		<sexo>M</sexo>
		<profissao>Professor</profissao>
		<instituicaoUsuario>UFES</instituicaoUsuario>
		<tipo>2e9c5b6e-0d0f-4da0-b99b-24178ca6873a</tipo>
	</instance>
</instancesList>
```

Figura 5.7: Trecho do arquivo OWL retornado como resultado de uma consulta ao Portal do LabES (LOURENÇO, 2007).

A fase de testes deve ser conduzida de forma a verificar a eficácia não só do código-fonte da aplicação, mas também das ontologias codificadas em OWL. Discussões acerca da

fase de testes encontram-se fora do escopo deste trabalho, porém consideramos de grande importância, no futuro, adequar os métodos de teste a S-FrameWeb.

A implantação de um WIS Semântico é feita normalmente, incluindo os arquivos OWL juntamente com o código-fonte compilado e testado, em local específico, de forma a estar disponível para que o *framework* Controlador Frontal possa encontrá-los.

5.6. Conclusões do capítulo

S-FrameWeb complementa as diretrizes trazidas por FrameWeb, acrescentando atividades que promovem a construção de WISs voltados para a *Web* Semântica. Por meio de uma análise do domínio do WIS, desenvolve-se uma ontologia do domínio e, durante a análise do sistema, o modelo conceitual da aplicação é modelado em ODM. Após o projeto do sistema e a criação do Modelo de Domínio de FrameWeb (MDF) utilizando um novo perfil UML, a ontologia e o MDF são codificados em OWL e utilizados pelo *framework* Controlador Frontal para responder a requisições feitas por agentes de software com documentos OWL, que poderão ser interpretados pelos mesmos.

Da mesma forma que FrameWeb promove agilidade no projeto e implementação de WISs baseados em *frameworks*, sua extensão para a *Web* Semântica traz benefícios para equipes de desenvolvimento que desejam levar à *Web* Semântica suas aplicações construídas com FrameWeb. Tais benefícios incluem:

- Promoção da atividade de Análise de Domínio, trazendo benefícios com a reutilização de conhecimento de domínio em novas aplicações;
- Transformação de diagramas ontológicos e conceituais para artefatos de código facilitada pelo uso de ODM e OWL;
- Implementação de anotação dinâmica das páginas geradas pelo WIS facilitada pela extensão do *framework* Controlador Frontal.

Este trabalho, no entanto, apenas abre caminho para várias outras possibilidades no campo da *Web* Semântica para S-FrameWeb. Para que amadureça, algumas questões, que surgem naturalmente das idéias apresentadas neste capítulo, pedem discussões mais aprofundadas:

- Anotação dinâmica de *websites* é realmente a melhor escolha para levar WISs para a *Web* Semântica? Como poderíamos utilizar a abordagem de *Web Services* Semânticos com S-FrameWeb?

- Como se dá a descoberta de serviços por parte dos agentes da *Web* Semântica? Em outras palavras, como os agentes saberão o endereço e os parâmetros de entrada das ações que disparam as funcionalidades existentes nos WISs?
- Como fazer para que os conceitos modelados na ontologia do domínio e no modelo conceitual da aplicação sejam compreensíveis para agentes de software construídos em qualquer lugar do planeta? Seria possível definir uma ou mais ontologias de topo como padrões e, no caso de mais de uma, criar dicionários que traduzam os conceitos de umas para as outras?
- Que outras sugestões ou indicações poderiam facilitar ou agilizar as fases de levantamento e análise de requisitos e, principalmente, de testes? Este trabalho não discute essas fases em profundidade.

Estas e outras questões poderiam ser abordadas em trabalhos futuros envolvendo FrameWeb e S-FrameWeb. É preciso conduzir uma extensa revisão bibliográfica em áreas como *Web Services* Semânticos, casamento de serviços (*match-making*) (PAOLUCCI et al., 2002), ontologias de topo etc.

Capítulo 6: Considerações Finais

O surgimento de necessidades de desenvolvimento de sistemas para a *Web* cada vez mais complexos promoveu o desenvolvimento da Engenharia *Web*, trazendo abordagens sistemáticas de engenharia para construção, implantação e manutenção de Sistemas de Informação *Web* (*Web Information Systems* - WISs) (MURUGESAN et al., 1999).

Em paralelo, a criação de diversos *frameworks* e *containers* para codificação de WISs foi bastante expressiva, demonstrando a necessidade de se organizar uma infra-estrutura básica para tais sistemas que auxilie na criação de softwares seguros, de fácil manutenção e de rápido desenvolvimento. Trazendo vários componentes prontos e extensivamente testados, os *frameworks* promovem o reúso e as boas práticas de implementação.

A grande utilização desses *frameworks* e *containers* na prática e a ausência de um método de projeto que fosse especificamente direcionado a eles levou-nos a propor FrameWeb, um método baseado em *frameworks* para o projeto de WISs. Os diversos esforços direcionados à pesquisa da *Web* Semântica nos motivaram a estender ainda este método, criando S-FrameWeb: um método baseado em *frameworks* para o projeto de WISs semânticos.

Foi proposto como objetivo principal deste trabalho que FrameWeb (a) fosse baseado em metodologias da Engenharia de Software Orientada a Objetos e linguagens de modelagem bem difundidas; (b) fosse direcionado à construção de aplicações baseadas em *frameworks*; (c) incorporasse idéias de desenvolvimento ágil; (d) incluísse diretrizes para a *Web* Semântica; e (e) não fosse restritivo em nenhuma de suas proposições. Ao longo desta dissertação mostramos como FrameWeb atende a todos os objetivos propostos.

Neste capítulo apresentamos uma avaliação geral do trabalho (Seção 6.1) e as perspectivas futuras (Seção 6.2).

6.1. Avaliação do Trabalho

Todo trabalho tem pontos positivos e negativos. Para uma análise dos pontos fortes e das fraquezas deste trabalho é necessário recapitular as propostas reunidas nesta dissertação. São contribuições deste trabalho:

- Diretrizes gerais para a condução de fases que precedem o projeto do WIS,

como os princípios da Modelagem Ágil ou a utilização de determinados diagramas que funcionam como insumos para atividades da fase de projeto;

- Uma arquitetura básica organizada em três camadas para divisão de responsabilidades dos componentes de um WISs em cinco pacotes: Visão, Controle, Aplicação, Domínio e Persistência;
- Um perfil UML para modelagem de quatro diagramas na fase de projeto: Modelo de Domínio, Modelo de Persistência, Modelo de Navegação e Modelo de Aplicação, representando componentes da solução integrados a componentes dos *frameworks* utilizados na implementação;
- Diretrizes gerais, passo a passo, para a construção dos quatro modelos de projeto citados e sugestões de simplificação de alguns desses modelos;
- Adição de atividades de análise de domínio, implementação e implantação de ontologias, além de alterações no perfil UML do Modelo de Domínio de FrameWeb para desenvolvimento de WISs Semânticos;
- Instruções para implementação de extensões aos *frameworks* Controladores Frontais para detecção automática do tipo de usuário e seleção da forma mais apropriada de apresentação dos dados gerados dinamicamente (LOURENÇO, 2007).

Consideramos como pontos fortes deste trabalho os seguintes:

- FrameWeb não impõe muitas restrições, deixando a equipe de desenvolvimento livre para utilizar o processo de software que melhor se encaixa em cada situação, além de propor algumas diretrizes com o objetivo de agilizar o processo;
- O uso da UML como linguagem padrão facilita a adoção de FrameWeb por desenvolvedores já proficientes nessa forma de representação;
- A indicação de "valores *default* baseados no bom senso" e sugestões simplificam bastante os modelos, facilitando o trabalho tanto de quem os escreve como de quem os lê;
- A divisão em camadas na arquitetura básica proposta integra-se bem com os *frameworks* utilizados, facilitando os processos de implementação e manutenção, visto que promove uma maior modularização, separação das preocupações de infra-estrutura e utilização de boas práticas de programação;
- A representação de componentes dos *frameworks* em modelos de projeto permite que o projetista defina com maior precisão como o sistema será implementado, evitando, assim, possível retrabalho causado por decisões inadequadas

dos implementadores;

- A inclusão de diretrizes para criação de WISs semânticos não só contribui para o desenvolvimento da *Web* Semântica como também promove o reúso de conhecimento ao sugerir que seja conduzida uma análise de domínio antes da análise do problema;
- A utilização de ferramentas de interpretação de ontologias combinadas com mecanismos de reflexão permitem a implementação de extensões aos *frameworks* Controladores Frontais de forma a reduzir consideravelmente o trabalho de anotação semântica dos dados gerados dinamicamente pelo WIS.

Algumas questões não foram abordadas com suficiente profundidade nesta dissertação, constituindo, assim, pontos fracos da proposta. São elas:

- A fase de testes e outras atividades de verificação e validação não foram incluídas no escopo do trabalho. No entanto, consideramos de grande importância que sejam analisadas no contexto do desenvolvimento de WISs baseados em *frameworks*;
- FrameWeb não traz nenhuma diretriz ou linguagem para criação de modelos visuais e de interação com o usuário. O projetista, no entanto, fica livre para utilizar a linguagem e o método que mais lhe agrade;
- Poucos experimentos foram conduzidos e nenhum deles utilizou de alto rigor de formalidade para alcançarmos conclusões mais precisas e fortes sobre, por exemplo, a eficácia do método, a facilidade de aprendizado, a agilidade proporcionada etc.
- Também foram realizados poucos testes com as diferentes instâncias dos *frameworks* para determinar a universalidade do método. (PERUCH, 2007) foi o primeiro trabalho nesse sentido, experimentando três *frameworks* Controladores Frontais diferentes utilizando FrameWeb;
- Apesar de utilizar uma linguagem simples, FrameWeb exige da equipe de projeto e desenvolvimento conhecimentos sedimentados sobre os *frameworks* utilizados, além de um certo grau de experiência para a compreensão rápida dos modelos, em especial do modelo de navegação;
- As questões apresentadas na conclusão do Capítulo 5, como anotação dinâmica vs. *Web Services* semânticos, descoberta de serviço, uso de ontologias de topo etc. precisam ser mais discutidas, de preferência com exemplos práticos que comprovem a viabilidade da utilização das idéias apresentadas num cenário real da *Web* Semântica.

6.2. Perspectivas Futuras

Esta dissertação lança a semente na definição de um método para o projeto de WISs baseado em *frameworks*, com extensão para a *Web* Semântica. Ao analisarmos os pontos fracos deste trabalho percebemos diversas perspectivas de projetos futuros que, abordando esses itens, complementam o presente trabalho e promovem a evolução de FrameWeb e S-FrameWeb:

- Discutir a atividade de testes no contexto de FrameWeb e S-FrameWeb;
- Propor modelos de leiaute e de interação com o usuário baseados em *frameworks* que tratam esse aspecto;
- Conduzir experimentos formais sobre o método para avaliar sua eficácia e universalidade;
- Aprofundar as discussões da *Web* Semântica e experimentar na prática o uso de S-FrameWeb para criação de WISs Semânticos.

A grande integração entre os modelos de projeto e o código-fonte implementado utilizando os *frameworks* sugere, também, a construção de ferramentas CASE que auxiliem na construção dos modelos de projeto de FrameWeb e que possibilitem a geração automática de código a partir desses modelos.

Referências

ALUR, D.; MALKS, D.; CRUPI, J. **Core J2EE Patterns: Best Practices and Design Strategies**. 2. ed. Prentice Hall, ISBN 0131422464, maio 2003.

AMBLER, S.; JEFFRIES, R. **Agile Modeling: Effective Practices for Extreme Programming and the Unified Process**. 1. ed. John Wiley & Sons, ISBN 0471202827, fevereiro 2002.

ANTONIOU, G.; VAN HARMELEN, F. **A Semantic Web Primer**. 1. ed. Londres: The MIT Press, ISBN 0262012103, abril 2004

ARCH-INT, S.; BATANOV, D. N. **Development of industrial information systems on the Web using business components**. Computers in Industry, v. 50, n. 2 (fevereiro), p. 231-250, Elsevier, 2003.

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

BERNERS-LEE, T. Semantic Web – XML 2000. Disponível em: <http://www.w3.org/2000/Talks/1206-xml2k-tbl/>. Acesso em: 11 jun. 2006. 2000.

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

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. 2. ed. Addison-Wesley Professional, ISBN 0321267974, 2005.

BREITMAN, K. K. **Web Semântica: A Internet do Futuro**, LTC, ISBN 8521614667, 2006.

BROCKMANS, S.; VOLZ, R.; EBERHART, A.; LÖFFLER, P. **Visual modeling of OWL DL ontologies using UML**. Proceedings of the 3rd International Semantic Web Conference, p. 198-213, Hiroshima, Japão, 2004.

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.

CERI, S.; FRATERNALI, P.; MATERA, M. **Conceptual modeling of data-intensive Web applications**. Internet Computing, v. 6, n. 4 (julho-agosto), p. 20-30, IEEE, 2002b.

CERI, S.; FRATERNALI, P.; BONGIO, A.; BRAMBILLA, M.; COMAI, S.; MATERA, M. **Designing Data-Intensive Web Applications**. Morgan Kaufmann, ISBN 1558608435, dezembro 2002.

CHANG, P.; KIM, W.; AGHA, G. **An adaptive programming framework for Web applications**. Proceedings of the 2004 Internation Symposium on Applications and the Internet, p. 152-159, IEEE, 2004.

COAD, P.; YOURDON, E. **Projeto Baseado em Objetos**, 1. ed. Editora Campus, 1993.

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

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.

DAVIES, J.; FENSEL, D.; VAN HARMELEN, F. **Towards the Semantic Web: Ontology-Driven Knowledge Management**, 1st ed. Wiley, ISBN 0470848677, janeiro 2003.

DÍAZ, P.; MONTERO, S.; AEDO, I. **Modelling hypermedia and web applications: the Ariadne Development Method**. Information Systems, v. 30, n. 8 (dezembro), p. 649-673,

Elsevier, 2004.

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

ĐURIĆ, D. **MDA-based Ontology Infrastructure**. Computer Science and Information Systems, v. 1, n. 1 (fevereiro), p. 92-116, ComSIS Consortium, 2004.

FALBO, R. A.; GUIZZARDI, G.; DUARTE, K. C. **An Ontological Approach to Domain Engineering**. Proceedings of the 14th International Conference on Software Enginering and Knowledge Engineering (SEKE), p. 351-358, Ischia, Itália, 2002.

FALBO, R. A. **Experiences in Using a Method for Building Domain Ontologies**. Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2004), pp. 474-477, International Workshop on Ontology In Action (OIA'2004), Banff, Alberta, Canadá, junho 2004.

FELLBAUM, C. **WordNet: An Electronic Lexical Database (Language, Speech, and Communication)**. The MIT Press, ISBN 026206197X, maio 1998.

FENSEL, D.; BUSSLER, C. **The Web Service Modeling Framework (WSMF)**. Electronic Commerce Research and Applications, v.1(2), Elsevier, 2002.

FONS, J.; PASTOR, O.; VALDERAS, P.; RUIZ, M. **OOWS: Un Método de Producción de Software em Ambientes Web**. In F. J. García (editor), Avances em Comercio Electrónico, Departamento de Informática y Automática de la Universidad de Salamanca, p. 121-136, 2002.

FONS, J.; VALDERAS, P.; RUIZ, M.; ROJAS, G.; PASTOR, O. **OOWS: A Method to 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. **Patterns of Enterprise Application Architecture**. Addison-Wesley, ISBN 0321127420, novembro 2002.

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.

FUCHS, M.; NIEDERÉE, C.; HEMMJE, M.; NEUHOLD, E. J. **Supporting model-based construction of semantic-enabled Web applications**. Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 2003), p. 232-241, 10-12 de dezembro, 2003.

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

GANGEMI, A.; NAVIGLI, R.; VELARDI, P. **The OntoWordNet Project: extension and automatization of conceptual relations in WordNet**. On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, p. 820-838, Springer, ISBN 3540204989, outubro 2003.

GINIGE, A.; MURUGESAN, S. **Web Engineering: An Introduction**. IEEE Multimedia, v. 8, n. 1 (Janeiro-Março), p. 14-18, IEEE, 2001.

GINIGE, A.; MURUGESAN, S. **The essence of web engineering - managing the diversity and complexity of web application development**. IEEE Multimedia, v. 8, n. 2 (Abril-Junho), p. 22-25, IEEE, 2001b.

GUARINO, N. **Formal Ontology and Information Systems**. Proceedings of the 1st International Conference on Formal Ontology in Information Systems (FOIS 98), Trento, Itália: IOS Press, 1998.

GUELFI, N.; RIES, B.; STERGES, P. **MEDAL: a case tool extension for model-driven software engineering**. Proceedings of the IEEE International Conference on Software: Science, Technology and Engineering (SwSTE '03), p. 33-42, 4-5 de novembro, 2003.

GUIZZARDI, G. **Uma abordagem metodológica de desenvolvimento para e com reuso baseada em Ontologias formais de Domínio**. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2000.

GUIZZARDI, G. **The Role of Foundational Ontology for Conceptual Modeling and Domain Ontology Representation**. Companion Paper for the Invited Keynote Speech, 7th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania, 2006.

GUIZZARDI, G. **On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models**. Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV, Olegas Vasilecas, Johan Edler, Albertas Caplinskas (Editors), ISBN 9781586036408, IOS Press, Amsterdam, 2007.

HAROLD, E.; MEANS, W. S. **XML in a Nutshell**, 3rd ed. O'Reilly and Associates, ISBN 0596007647, setembro 2004.

HEPP, M. **Semantic Web and semantic Web services - Father and Son or Indivisible Twins?** IEEE Internet Computing, v. 10, n. 2, p. 85-88, IEEE, 2006.

JACYNTHO, M. D.; SCHWABE, D.; ROSSI, G. **A Software Architecture for Structuring Complex Web Applications**. Proceedings of the WWW2002, Web Engineering Alternate Track, Honolulu, EUA. ISBN 1-880672-20-0, 2002.

JOHNSON, R.; HOELLER, J. **Expert One-on-One J2EE Development without EJB**. 1st ed. Wrox, ISBN 0764558315, junho 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.

KOCH, N.; KRAUS, A.; HENNICKER, R. **The Authoring Process of the UML-based Web Engineering Approach**. In Daniel Schwabe (editor), First Interanational Workshop on Web-oriented Software Technology (IWWOST01), publicação online, junho, 2001.

KOCH, N.; KRAUS, A. **The Expressive Power of UML-based Web Engineering**. In D. Schwabe, O. Pastor, G. Rossi e L. Olsina (editores), Proceedings of the Second International Workshop on Web-Oriented Software Technology (IWWOST02), CYTED, p. 105-119, junho, 2002.

KRUTCHEN, P. **The Rational Unified Process: An Introduction**, 2nd ed. Addison-Wesley, ISBN 0201707101, março 2000.

LEE, S. C.; SHIRANI, A. I. **A component based methodology for Web application development**. Journal of Systems and Software, v. 71, n. 1-2 (abril), p. 177-187, Elsevier, 2004.

LIMA, F.; SCHWABE, D. **Application Modeling for the Semanti Web**. First Latin American Web Conference (LA-WEB, Santiago, Chile), IEEE-CS Press, 2003.

LOURENÇO, T. W. **Uma Extensão para o *framework* Struts2 para a Implementação de Aplicações para *Web* Semântica com S-FrameWeb**. 2007. Monografia (Ciência da Computação) – Departamento de Informática, Universidade Federal do Espírito Santo, Vitória, 2007.

MARTIN, D.; PAULUCCI, M.; MCILRAITH, S.; BURSTEIN, M.; MCDERMOTT, D.; MCGUINNESS, D.; PARSIA, B.; PAYNE, T.; SABOU, M.; SOLANKI, M.; SRINIVASAN, N.; SYCARA, K. **Bringing Semantics to Web Services: The OWL-S Approach**. Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004 - California, USA), Springer, julho 2004.

MATUSZEK, C.; CABRAL, J.; WITBROCK, M.; DEOLIVEIRA, J. **An Introduction to the Syntax and Content of Cyc**. Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering (Stanford, CA), março 2006.

MCILRAITH, S. A.; SON, T. C.; ZENG, H. **Semantic Web Services**. Intelligent Systems, v. 16, n. 2 (março-abril), p. 46-53, IEEE, 2001.

MURUGESAN, S.; DESHPANDE, Y.; HANSEN, S.; GINIGE, A. **Web Engineering: A New Discipline for Development of Web-based Systems**. Proceedings of the First ICSE Workshop on Web Engineering (Australia), IEEE, 1999.

NARAYANAN, S.; MCILRAITH, S. A. **Simulation, Verification and Automated Composition of Web Services**. Proceedings of the 11th international conference on World Wide Web (Hawaii, USA), p. 77-88, ACM, 2002.

NEIGHBORS, J. M. **Software Construction Using Components**. Tese (Doutorado em Ciência da Computação) – Department of Information and Computer Sciences, University of California. Irvine, CA, EUA, 1981.

OMG. **Ontology Definition Metamodel Specification**, jun. 2006. Disponível em: <http://www.omg.org/docs/ad/06-05-01.pdf>. Acesso em: 07 jun. 2007.

PASTOR O.; GÓMEZ, J.; INSFRÁN, E.; PELECHANO, V. **The OO-Method Approach for Information Systems Modelling: From Objetct-Oriented Conceptual Modelling to Automated Programming**. Information Systems, v. 26, n. 7 (novembro), p. 507-534, Elsevier, 2001.

PERROUIN, G.; GUELFI, N.; HAMMOUCHE, D.; RIES, B.; STERGES, P. **FIDJI Project Overview**. Applied Computer Science Department Technical Report TR-DIA-02-07, Luxembourg University of Applied Sciences, Luxembourg-Kirchberg, Luxembourg, 2002.

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.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 6. ed. McGraw Hill, ISBN 007301933X, 2005.

REENSKAUG, T. **THING-MODEL-VIEW-EDITOR, an Example from a planning system**. Xerox PARC Technical Note, maio 1979.

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

RICHARDSON, O. **Gathering accurate client information from World Wide Web sites**. Interacting with Computers, v. 12, n. 6 (julho), p. 615-622, Elsevier, 2000.

PAOLUCCI, M., KAWAMURA, T., PAYNE, T. R., SYCARA, K. **Semantic Matching of Web Services Capabilities**. International Semantic Web Services Conference, 2002.

RIES, B.; CHABERT, S. **JAFAR2 Software Requirements Specifications**. Software

Engineering Competence Center Technical Report TR-SE2C-03-04, Luxembourg University of Applied Sciences, Luxembourg-Kirchberg, Luxembourg, 2003.

ROMAN, D.; KELLER, U.; LAUSEN, H.; DE BRUIJN, J.; LARA, R.; STOLLBERG, M.; POLLERES, A.; FEIER, C.; BUSSLER, C.; FENSEL, D. **Web Service Modeling Ontology**. Applied Ontology, v. 1(1): p. 77-106, IOS Press, 2005.

ROSENBERG, D.; SCOTT, K. **Use Case Driven Object Modeling with UML : A Practical Approach**. Addison-Wesley, ISBN 0201432897, março 1999.

RUY, F. **Semântica em um Ambiente de Desenvolvimento de Software. Dissertação de Mestrado**. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2006.

SCHARL, A. **Explanation and exploration. Visualizing the topology of web information systems**. International Journal of Human-Computer Studies, v. 55, n. 3 (setembro), p. 239-258, Elsevier, 2001.

SCHWABE, D.; ROSSI, G. **An Object Oriented Approach to Web-Based Application Design**. Theory and Practice of Object Systems 4 (4), Wiley and Sons, ISSN 1074-3224, 1998.

SCHWAMBACH, M. M. **OplA: Uma Metodologia para o Desenvolvimento de Sistemas Baseados em Agentes e Objetos**. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2004.

SCOTT, G. **Internet/web systems development: what can be learned from hi-tech new product strategic planning**. Proceedings of the 26th Annual Hawaii International Conference on System Sciences, p. 262-268, 6-9 de janeiro, IEEE, 2003.

SHANNON, B. **Java™ 2 Platform Enterprise Edition Specification, v1.4**. Sun Microsystems, 2003.

SHANNON, B. **Java™ Platform, Enterprise Edition (Java EE) Specification, v5**. Sun Microsystems, 2006.

SHAW, M.; GARLAN, D. **Software Architecture: Perspectives on an Emerging Discipline**. 1. ed. Prentice Hall, ISBN 0131829572, 1996.

SOUZA, V.; FALBO, R. **An Agile Approach for Web Systems Engineering**. Proceedings of the 11th Brazilian Symposium on Multimedia and the Web (WebMedia 2005 - Poços de Caldas, MG - Brasil), p. 1-3, dezembro 2005.

SOUZA, V.; FALBO, R. **FrameWeb - A Framework-based Design Method for Web Engineering**. Proceedings of the Euro American Conference on Telematics and Information Systems 2007 (EATIS 2007 – Faro, Portugal), maio 2007.

SOUZA, V.; FALBO, R.; GUIZZARDI, G. **A UML Profile for Modeling Framework-based Web Information Systems**. Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2007 – Trondheim, Norway), junho 2007b.

SOUZA, V.; LOURENÇO, T.; FALBO, R.; GUIZZARDI, G. **S-FrameWeb – a Framework-based Design Method for Web Engineering with Semantic Web Support**. Proceedings of the International Workshop on Web Information Systems Modeling (Trondheim, Norway), p. 55-66, 2007c.

STANDING, C. **Methodologies for developing Web applications**. Information and Software Technology, v. 44, n. 3 (março) p. 151-159, Elsevier, 2002.

STOJANOVIC, L.; STOJANOVIC, N.; VOLZ, R. **Migrating data-intensive Web Sites into the Semantic Web**. Proceedings of the 2002 ACM symposium on Applied computing (Madrid, Espanha), p. 1100-1107, ACM, 2002.

TAM, V.; FOO, W. K.; GUPTA, R. K. **A fast and flexible framework of scripting for Web application development: a preliminary experience report**. Proceedings of the First International Conference on Web Information Systems Engineering, v. 1, p. 450-455, 19-21 de junho, 2000.

UDEN, L. **Design process for Web applications**. Multimedia, v. 9, n. 4 (dezembro), p. 47-55, IEEE, 2002.

VOGELSAN, L.; CARTENSEN, P. **New challenges for the collaboration in Web-based information systems development**. Proceedings of the Tenth IEEE Internatinoal Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, p. 386-391, 20-22 de junho, IEEE, 2001.

W3C. **OWL Web Ontology Language Guide**, fev. 2004. Disponível em: <http://www.w3.org/TR/owl-guide/>. Acesso em: 13 nov. 2006.

W3C. **W3C Glossary and Dictionary**. Disponível em: <http://www.w3.org/2003/glossary/>. Acesso em: 13 nov. 2006.

WOOLDRIDGE, M. **Intelligent Agents**. Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence, p. 27-77, Londres: The MIT Press, 1999.

ZHANG, J.; CHANG, C.; CHUNG, J. **Mockup-driven Fast-prototyping Methodology for Web Requirements Engineering**. Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC '03), p. 263-268, 3-6 de novembro, 2003.

ZHANG, J.; CHUNG, J. **Mockup-driven Fast-prototyping Methodology for Web Applications**. Proceedings of the 2003 Symposium on Applications and the Internet (SAINT '03), p. 410-413, 27-31 de janeiro, 2003b.

Livros Grátis

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

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo