Uma ferramenta Web colaborativa para apoiar a engenharia de requisitos em software livre

Marco Aurélio Graciotto Silva

Uma ferramenta Web colaborativa para apoiar a engenharia de requisitos em software livre

Marco Aurélio Graciotto Silva

Orientador: ***Profa. Dra. Renata Pontin de Mattos Fortes***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para a obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

"VERSÃO REVISADA APÓS A DEFESA"

Data da Defesa: 01/11/2005

Visto do Orientador: _____

USP - São Carlos
Abril / 2006

Aos meus pais

Agradecimentos

Uma longa estrada percorrida. Mérito do autor, mas não só. Agradecimentos devem ser prestados. Omissões não existem, apenas esquecimentos. Dos nomes, não dos fatos. Perdoem esta pessoa cujo dom para guardar nomes é tão falho, mas lembrem-se de que o que importa são as pessoas, não o conjunto restrito e limitado de letras que tentam resumir tanto em tão pouco.

À Elisa, pela primeira orientação em trabalho científico. Ao Flávio, vulgo X, pelo apoio, até mesmo quando em tratamento médico, e pelas saudáveis polêmicas. A ambos pelo incentivo a seguir o caminho da pesquisa e pelas recomendações para o Mestrado. Agradecimentos também aos outros professores que apoiaram essa escolha: Dino (Edmundo), Maurício, Morandini (Marcelo). À Organização para Acontecimentos Estranhos (OAE), aos UMs Rorso (Robson), Marim (Flávio), Igorus (Igor), Schiavoni (Flávio), Neves (Alysson), Sidnelso (Marcel), BGA (Bruno), Cefernan (Carlos Fernando), Kiti (Cristiano†), Bibi (Sabrina), Vladi (Vladimir), Baldão (Alessandro), BH (Ben-Hur), Gelo (Ângelo), Déia (Andréia), Moe (Daniel), Beto (Luis Alberto), Camilla, Trooper (Douglas), Kengo, Sapo (Rodrigo), Zeddy (Marcelo). Com peso! Brutalidade! À minha turma de graduação, cc1998, a sobrevivente das greves da UEM, Motoca (Marcelo), Luz Clarita/Titanic (Márcio Zanardo), Poliquexosa (Gabriela), Froger (Roger), Renato, Magu (Gustavo), 20 (Augusto), Karina, Kenia, Patricia, Luciana, Burali (Marcelo), Isgarbi (Edson), Marcio Hirata, Hugo, Barrão (Fernando Nabarro), Pamonha (Fernando Panonte), Cleber, Magno, Sakae (Maurício), Chapareli (Fernando Caparelli), Laranja (Cleverson), Moralles (Fernando Moralles), Robertão, Vitor, Maurício (goleiro), Kiti (Cristiano†), Igor, Sidnelso (Marcel), Shimu (Eduardo), Patrick.

À Renata Fortes, orientadora deste trabalho, insistente, paciente, compreensiva. E disseminadora de software livre. Às aulas de seminários avançados em engenharia de software e as discussões do Maldonado com o André ("CMM é processo!"). Ao Kiko (Christian), *hacker* no sentido original da palavra. Ao Edilson, que me (eu) salvou (taquei) da (na)

fogueira na primeira aula do Mestrado e comemorou a defesa com tequila Jose Cuervo, sal e limão. À turma de pós-graduação de 2002, que tão pouco conheci e de poucos nomes me recordo: Danilo, Edilson, Rodrigo Plotz, Roberto Platz, Marisa, Aline, Bruno, André, Osnete. Ao LaBES, onde tanto vivi e vi. São eles: André Domingues, Adenilso, Erika, Ellen Francine, Tatiana, Thaise, Simone Domingues, Edilson, André, Osnete, Bira (Ubirajara), André (arquiteto), Willie, Tânia (Maria), KLB (Sandro), Antonielly, Auri, Maris, Alessandra, Lizandra, Paula, Valter, Rosana, Luciana, Andréia (Andrea), Débora, Otávio, Fabiano, Camila, Ré (Reginaldo), Rogério, Marcelo, Mateus, Dinho (Anderson), Marcela, Stanley. Claro, e aos fundadores: Masiero, Maldonado, Renata, Roseli, Fernão. Aos não LaBES, mas não menos amigos (alguns até quase LaBES): Cláudio, Juliana, Richard, Gawa (Ricardo), Menotti (Ricardo), Marinho (Mario), Sandra, Rosângela, Manu (Emanoela). À coordenação do curso, às eficientes e atenciosas secretárias da pós-graduação, Ana Paula, Laura e Beth. À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro.

À Nayd e família, que me hospedaram por dois anos, À Natalina, que me alugou a edícula neste último ano. Para a cidade de São Carlos, um verdadeiro celeiro tecnológico. Ao Centro de Divulgação da Astronomia (CDA), vulgo observatório da USP, por me abrir os olhos e direcioná-los para o céu.

Ao ano mundial da Física. Copérnico, Galileu, Gauss, Newton, Einstein, Hawking, Feymann e tantos outros ilustres cientistas.

A Deus. Aos meus pais, Apolo e Rosa Maria. À minha irmã Melissa e família. Ao meu irmão Victor Augustus e família.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de fragmentos de código

Lista de Acrônimos

ADV	*Abstract Data View*
AJAX	*Asynchronous JavaScript Technology and XML*
ANT	*Another Neat Tool*
API	*Application Program Interface*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CCM	*Change Control Management*
CDA	Centro de Divulgação da Astronomia
CERN	*Conseil Europeenne pour la Recherche Nucleaire*
CGI	*Common Gateway Interface*
CRUD	*Create*, *Read*, *Update*, *Delete*
ConOps	*Concept of Operations*
CVS	*Concurrent Versions System*
DAD	*Declarative Active Document*
DEC	*Digital Equipment Corporation*
DOM	*Document Object Model*
EIA	*Electronics Industry Association*
ESA	*European Space Agency*
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
FINEP	Financiadora de Estudos e Projetos
FSF	*Free Software Foundation*
GCC	*GNU Compiler Collection*
GLC	*Glossary Circularity*
GPL	*General Public License*
GNU	*GNU's Not Unix*
HDM	*Hypertext Design Model*
HERE	*Hypermedia Environment for Requirements Engineering*

HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
ICMC	Instituto de Ciências Matemáticas e de Computação
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
IRC	*Internet Relay Chat*
ISO	*International Organization for Standardization*
J2EE	*Java 2 Platform Enterprise Edition*
JAR	*Java Archive*
JAXB	*Java API for XML Binding*
JDBC	*Java DataBase Connectivity API*
JSP	*Java Server Pages*
JTA	*Java Transaction API*
JWSDP	*Java Web Services Developer Pack*
KDE	*K Destop Environment*
LAL	Léxico Ampliado de Linguagem
LGPL	*Lesser General Public License*
LISP	*LISt Processing language*
LOC	*Lines of Code*
Mac OS	*Macintosh Operating System*
MOV	*Minimality of Vocabulary*
MVC	*Model View Controller*
NCP	*Network Control Program*
OAE	Organização para Acontecimentos Estranhos
OHS	*Open Hypermedia System*
OOHDM	*Object-Oriented Hypermedia Design Model*
OSI	*Open Source Initiative*
PBR	*Perspective Based Reading*
PDF	*Portable Document Format*
POJO	*Plain Old Java Object*
RCS	*Revision Control System*
RDF	*Resource Description Framework*
REM	*REquirements Management tool*
RETH	*Requirement Engineering Through Hypertext*
RLE	*Run-Length Encoding*
RMM	*Relationship Management Methodology*
RQML	*Requirements Markup Language*
SAFE	*Software Engineering Available for Everyone*

SCR	*Software Cost Reduction*
SDL	*Specification and Design Language*
SGBD	Sistema Gerenciador de Banco de Dados
SI	Sistema Internacional de Medidas
SOHDM	*Scenario-Based Object-Oriented Hypermedia Design Methodology*
SMIL	*Synchronized Multimedia Integration Language*
SUN	*Stanford University Network*
SVG	*Scalable Vector Graphics*
TBD	*To Be Determined*
TCP	*Transmission Control Protocol*
UEM	Universidade Estadual de Maringá
UFMS	Universidade Federal do Mato Grosso do Sul
UM	*Underground Member*
UNESCO	*United Nations Educational Scientific and Cultural Organization*
UoD	*Universe of Discourse*
UP	*Unified Process*
URL	*Uniform Resource Locator*
USP	Universidade de São Paulo
USP/SC	Universidade de São Paulo/Campus de São Carlos
UTI	Unidade de Terapia Intensiva
VCM	*Version Control Management*
VDM	*Vienna Development Model*
VO	*Value Object*
VRML	*Virtual Reality Modeling Language*
XHTML	*eXtensible HyperText Markup Language*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*
XSLT	*eXtensible Stylesheet Language Transformations*
XUL	*XML-based User interface Language*
W3C	*World Wide Web Consortium*
WSDM	*Web Site Design Method*
WWW	*World Wide Web*
WYSIWYG	*What You See Is What You Get*

Resumo

A engenharia de requisitos em projetos de software livre é uma atividade de segunda classe, ao menos em face ao estado da arte da área. Um exemplo claro disso é a inexistência de especificações de requisitos nesses projetos. No entanto, softwares livres são reconhecidos como produtos de elevada qualidade e não é possível produzir softwares de sucesso sem que os requisitos de seus usuários sejam satisfeitos. Portanto, existe um processo de engenharia de requisitos, ainda que não formalmente definido. De fato, recentes estudos sobre o processo de desenvolvimento de software livre demonstraram que os requisitos são publicamente declarados *a posteriori* do desenvolvimento do código, dependendo das habilidades do desenvolvedor para a correta elicitação, análise e especificação dos requisitos. A natureza iterativa e aberta do desenvolvimento, com ciclos rápidos e resultados publicamente discutidos, permite que erros sejam detectados prematuramente, o que diminui o esforço necessário para as correções, viabilizando o processo de produção de software livre. Porém, existe a constante preocupação da documentação apenas do código-fonte e não dos requisitos. Uma das causas é que não existe uma ferramenta apropriada para armazenar esses requisitos e disponibilizá-los ao público, precisando os desenvolvedores recorrer a arquivos textos ou páginas Web cujo gerenciamento é trabalhoso ao ponto de sua constante atualização ser comprometida. Uma solução para o problema é a adoção de ferramentas ágeis de edição colaborativa para a Web, que permitam a rápida atualização dos documentos de requisitos por qualquer pessoa envolvida no desenvolvimento. Ademais, ela deve facilitar a associação dos requisitos com as discussões a seu respeito, geralmente armazenadas nos arquivos das listas de discussão e ferramentas de gerenciamento de alterações (como o *Bugzilla*). A Wiki/RE, proposta neste trabalho, visa disponibilizar um ambiente com tais características, voltado especificamente para a engenharia de requisitos. Ela é uma ferramenta *wiki* que permite a criação de hiperdocumentos de requisitos, provendo capacidades de gerenciamento do documento e permitindo a rápida avaliação da qualidade do mesmo.

Palavras-chaves: Engenharia de requisitos, software livre, wiki.

Abstract

Requirements engineering is a second-class citizen within free software projects development when facing the current state of art. A clear example is the absence of requirements specification for free software. However, free software is acknowledged as a high quality product, status that would be impossible to achieve if its user's requirements were not satisfied. There should be a requirement engineering process, even if not formally defined. Indeed, recent works about free software development processes state that requirements are publicly asserted after the code development, depending upon the developer's abilities for a correct elicitation, analysis and specification of the requirements. The iterative and open nature of the process, with short release cycles and results publicly available in mailing lists and *Internet Relay Chat* (IRC) networks, tolerates small errors, detecting them early in the process and easing their correction, making the free software process viable. However, there is a constant concern about documenting the source code but not the requirements. One reason is that there is no suitable tool to register those requirements and make them public, requiring the developers to use plain text files or web pages, whose management is difficult to the point the developers avoid updating it. A possible solution to the problem is the use of web enabled agile and collaborative edition tools, allowing fast documentation updates by any person, developer or user, involved with the software development. The tool should also register links between the document and the rationale underneath every requirement, usually stored in mailing lists archives and change control tools (such as Bugzilla). The Wiki/RE provides an environment that satisfies those needs, along with others desirable requirements for a requirement management tool. The Wiki/RE, tool developed within this work, is a wiki that supports the collaborative development of requirements hyperdocuments, supporting the management and quality assessment of the requirements document.

Keywords: Requirement engineering, free software, wiki.

Capítulo

1

Introdução

O desenvolvimento de software de qualidade nos prazos estipulados e a custos baixos motiva, há décadas, a pesquisa por novas técnicas, métodos, processos e sua transferência para a indústria. Apesar desse esforço, vários projetos ainda falham. Muitos desses fracassos são atribuídos à incorreta ou incompleta determinação das funcionalidades a serem oferecidas pelos softwares e às condições e restrições aplicáveis a sua operação (EBERLEIN, 2002).

A Engenharia de Requisitos, uma área chave da Engenharia de Software, é responsável pela criação e aperfeiçoamento de técnicas que permitam o correto desenvolvimento dos requisitos do software. Através da clara e precisa definição de todos os objetivos que o software deve cumprir para satisfazer aos seus usuários, evitam-se, ou, ao menos, reduzem-se os riscos de fracassos para os novos projetos.

Um diferencial da Engenharia de Requisitos, em relação às outras áreas da Engenharia de Software, é seu estreito relacionamento com os usuários finais. Em um mundo com cada vez mais usuários, reais e potenciais, e com a contínua inserção da computação nas atividades executadas no dia-a-dia, especificar, de maneira economicamente viável, os requisitos de novos softwares torna-se uma atividade ainda mais complexa. Concomitante a isso, a presença de concorrência competitiva e global exige o lançamento de produtos de melhor qualidade e no menor prazo possível.

Avanços nos processos de Engenharia de Software, como processos iterativos e centrados nos usuários, quando devidamente aplicados, permitem uma redução no tempo para lançamento no mercado e um aumento na qualidade dos produtos. Cada subárea da Engenharia de Software também desenvolve métodos otimizados para os problemas específicos que

enfrenta. Engenharia de Requisitos não é uma exceção: as últimas décadas observaram uma proliferação de técnicas como pontos de vista, casos de uso, metas, separação de interesses, etc[1].

Paralelamente às novas técnicas e processos, busca-se a redução e racionalização de custos através do desenvolvimento distribuído, com a divisão das responsabilidades entre vários parceiros. Entretanto, as mesmas empresas que concorrem em um segmento são parceiras em outro. Mecanismos para garantir a proteção da propriedade intelectual são imperativos nesse ambiente: patentes de software, direitos autorais e segredo industrial. Dado que um conhecimento encontra-se protegido, sua utilização depende de transferência de tecnologia entre as partes, pagamento de direitos de uso (*royalties*) ou licenciamento.

Tratam-se de mecanismos eficazes em um de seus propósitos, a proteção intelectual: todos são definidos por leis (com validade nacional) e até tratados internacionais (direitos autorais), com penas definidas de elevado custo. Porém, em seu outro propósito, o de permitir o avanço tecnológico, o abuso na utilização de tais mecanismos e nas condições impostas ao seu acesso é prejudicial. Atualmente, severas críticas são levantadas à indústria de software quanto ao uso excessivo de patentes (MCLAUGHLIN, 2004; MCORMOND, 2003), que impedem ou desencorajam o desenvolvimento de produtos e novas tecnologias, principalmente para empresas pequenas e médias que não podem arcar com os custos (tanto de criação quanto de defesa de uma patente violada).

Contrariando essa tendência, softwares sem restrições (patentes) quanto ao uso das tecnologias envolvidas e cujas licenças encorajam e protegem seu desenvolvimento atraíram a atenção da comunidade, empresas e usuários, no início desse milênio. São softwares livres.

O software livre (United Nations Educational, Scientific and Cultural Organization, 2001; RASCH, 2000) nasceu com os primeiros mini-computadores PD-10 produzidos pela *Digital Equipment Corporation* (DEC), no final dos anos sessenta. Naquela época, todo software era livre (a maioria era software básico, parte da própria máquina). Nos anos seguintes, com o barateamento e popularização dos computadores, as empresas começaram a oferecer seus produtos sob licenças mais restritivas. A comunidade de desenvolvedores, na maioria acadêmicos acostumados a alterar os *softwares* para atender às suas próprias necessidade, diante das restrições impostas pelas licenças então vigentes, passou a criar seus próprios softwares e disseminar o reúso do código desses. Destaca-se, nesse meio, Richard Stallman, responsável pela fundação da *Free Software Foundation* (FSF) em 1985.

O surgimento da Internet, em 1983[2], e sua difusão nas universidades americanas e européias permitiram o intercâmbio de softwares livres entre pessoas espalhadas pelo globo. O desenvolvimento distribuído tornou-se comum, mesmo sem sofisticadas ferramentas para

[1] O capítulo 2 apresentará essas técnicas com mais detalhes.

[2] A Internet foi criada em 1969, com a implementação da ARPANET. No entanto, essa rede utilizava o protocolo *Network Control Program* (NCP). Somente em 1983 foram adotados os protolocos *Internet Protocol* (IP) e *Transmission Control Protocol* (TCP), utilizados até hoje.

o gerenciamento da comunicação. Um exemplo é o Linux (TORVALDS et al., 1991), que teve seu desenvolvimento público iniciado com uma simples mensagem enviada a um grupo de discussão na *Usenet*.

O ano de 1997 foi marcado pelo lançamento do artigo "*The Cathedral and the Bazaar*", de Raymond (1997), um dos primeiros trabalhos a analisar o modelo de desenvolvimento do software livre. Em 1998, Eric Raymond e Bruce Perens criaram a *Open Source Initiative* (OSI), uma organização voltada para a promoção do código aberto. O software livre ganhou visibilidade com o lançamento de novos grandes projetos, como o Mozilla. A percepção da alta qualidade dos softwares livres motivou a realização de mais estudos sobre o processo utilizado.

Reis (2003) realizou um minucioso estudo dessa literatura e um amplo levantamento sobre o processo utilizado pelos projetos de software livre, buscando assim caracterizar esse processo. A investigação evidenciou diversas propriedades importantes:

- processo iterativo, com um enfoque em atividades de manutenção e evolução constante;
- desenvolvimento geograficamente distribuído;
- equipes pequenas (em média menos de cinco integrantes);
- existência de um ciclo de vida para os projetos, organizado em (1) criação, (2) lançamento público, (3) crescimento e organização, e (4) maturidade (o processo, a cada ciclo, torna-se mais complexo e controlado);
- importância dos aspectos humanos, com a valorização dos desenvolvedores e usuários no processo como parte essencial ao processo.

A pesquisa também levantou problemas a serem solucionados. Um deles era como incentivar a escrita de software livre, não apenas através da criação de padrões que seriam facilmente instanciáveis, mas também da capacitação de pessoas nesse novo tipo de desenvolvimento. Particularmente, enxergou-se a oportunidade para implantar esses processos aqui no Brasil, principalmente em pequenas empresas, muitas das quais não possuem um processo definido. Dado que tais empresas recrutam muitos de seus funcionários de universidades, seria possível, aliando o ensino de Engenharia de Software nas faculdades ao processo de software livre, capacitar o pessoal necessário em um curto prazo de tempo.

Em 2003, iniciou-se um projeto para atender tais metas. Em uma associação entre a Universidade de São Paulo (USP), a Universidade Federal do Mato Grosso do Sul (UFMS) e a empresa Async[3], com apoio financeiro da Financiadora de Estudos e Projetos (FINEP),

[3] http://www.async.com.br

elaborou-se o projeto *Software Engineering Available for Everyone* (SAFE) (FORTES; TURINE; REIS, 2004). O SAFE visa elaborar uma infra-estrutura para desenvolvimento de software, que seja simples o suficiente para atrair a colaboração de desenvolvedores nos diversos níveis de familiaridade com o processo de software livre. Tal infra-estrutura consiste em um suporte automatizado ao processo de software livre por meio da integração de ferramentas de software livre de apoio às atividades de engenharia de software.

Uma área da Engenharia de Software pouco contemplada no processo de software livre é a de Engenharia de Requisitos. Pouco se sabe sobre as atividades, relacionadas com a Engenharia de Requisitos, executadas em projetos de software livre. O levantamento de Reis (2003) considerou duas hipóteses quanto a esse tema:

- o desenvolvedor do software é também um usuário, o que contribuiria para a determinação dos requisitos do software (REIS, 2003, hipótese 2, p. 64);
- o trabalho de engenharia de requisitos é freqüentemente facilitado por levantamentos anteriores feitos por outras equipes, seja através de padrões estabelecidos ou documentados, seja através de código-fonte herdado de outro projeto (REIS, 2003, hipótese 8, p. 65).

Existe a percepção, na comunidade de software, de que uma parcela significativa dos softwares é criada por uma necessidade do autor, não atendida por softwares já existentes. Raymond (2001) defende, em seu livro "*The Cathedral and the Bazaar*", que "*every good work of software starts by scratching a developer's personal itch*" (Todo bom software começa por atender uma necessidade pessoal de seu desenvolvedor). De fato, 75% dos projetos analisados por Reis afirmaram que o desenvolvedor é um dos principais usuários do software (REIS, 2003, questão 1.2, p. 106).

O fato do desenvolvedor assumir, também, o papel de usuário demonstra-se vantajosa: existe motivação para que os requisitos sejam atendidos o mais rapidamente possível (favorecendo ciclos curtos de desenvolvimento), a avaliação de riscos é facilitada (os requisitos são propostos conhecendo, desde o princípio, as limitações tecnológicas aplicáveis), os conflitos são resolvidos eficientemente (através de discussões ou da implementação do requisito como um argumento favorável[4]).

Um segundo possível facilitador à engenharia de requisitos é a reutilização de requisitos de outros projetos. Em software livre, argumenta-se que vários projetos tentam replicar softwares já existentes ou implementam especificações abertas. Segundo Reis (2003, p. 137), não foi possível estabelecer uma conclusão para essa hipótese, mas os resultados do levantamento

[4] Um bordão típico em software livre é "*Show me the code*" (Mostre-me o código). Principalmente para requisitos polêmicos (com muito conflito entre os desenvolvedores), a implementação do requisito, na forma de uma prova de conceito, é um forte argumento para a tomada de decisões.

> [...] sugerem fortemente o uso de conhecimento pré-existente para estabelecer os requisitos do projeto. Aproximadamente um terço dos participantes afirmou fundamentar seu desenvolvimento em um padrão publicado anteriormente. Também é aproximadamente um terço o número de projetos que afirmou replicar de maneira significativa um outro produto de software.

Se, para uma parcela significativa dos projetos, os requisitos são obtidos de especificações definidas por organizações como a *Institute of Electrical and Electronics Engineers* (IEEE) e *International Organization for Standardization* (ISO), ou da replicação de funcionalidades de outros softwares, para os restantes desconhece-se a presença de documentos de requisitos ou do processo utilizado para obtê-los. Esse fato foi confirmado por estudos de Scacchi (2002) e Yamauchi et al. (2000). Dada a importância da correta engenharia de requisitos nos projetos, torna-se necessário disponibilizar, no *framework* do SAFE, um processo para sua execução e uma ferramenta que permita automatizá-lo.

Seria impossível, ou indesejável, concretizar algo (código) sem possuir uma idéia do que este deve atender (requisitos) e como alcançar tais metas (projeto). Em softwares livres, não se encontram documentos de requisitos e são raros os documentos de projeto, pregando-se documentos na linguagem de programação como único elemento obrigatório. Isso, em nenhum momento, implica que o processo de engenharia de software desses projetos não inclui atividades relativas a requisitos e projetos: apenas não se encontram documentos, bem formados, que resumem os resultados dos respectivos esforços de engenharia. De fato, evidências sobre atividades de engenharia de requisitos são encontradas nas listas de discussões, pedidos de alteração como os relatados em ferramentas como a Bugzilla (WEISSMAN et al., 1998), manuais de utilização das ferramentas e documentos detalhados sobre pontos importantes de cada software.

Scacchi (2002) verificou que, diferentemente do que se espera nos projetos tradicionais, os requisitos, quando descritos, o são após sua implementação. Sua disponibilização é feita, em linguagem natural, por arquivos textos ou páginas Web, editadas sem o apoio de ferramentas especializadas. Sempre que possível, são criadas ligações entre esses documentos e as discussões relativas à implementação (que se encontram arquivadas em gerenciadores de listas de email). Não existe um padrão para essa documentação, constituindo uma especificação de requisitos: ela é organizada o suficiente para que os desenvolvedores consigam encontrá-la futuramente.

Esse formato impõe alguns obstáculos na atualização e recuperação da informação. Sua natureza estática (arquivos textos simples disponíveis na árvore do código) dificulta a atualização das informações, além de restringir os tipos de dados que podem ser inseridos. A ausência de estruturas de acesso para esses arquivos e o alto volume típico das listas de discussões impedem que usuários encontrem, ao menos facilmente, as informações necessárias. Seria desejável que tais documentos de requisitos fossem organizados segundo recomendações

da indústria, tais como as propostas pela IEEE-830 (Institute of Electrical and Electronics Engineers, 1998b) e DID-IPSC-81443 (Departament of Defense, 1994), o que permitiria uma estrutura comum e conhecida pela comunidade.

O desafio reside em conciliar as exigências das normas aos termos da comunidade de software livre. A primeira impressão de documentos que seguem tais normas é de que eles exigem muitas informações e a dedicação dos desenvolvedores, que vêem a quantidade de tempo dedicada à codificação reduzida, o que, em um primeiro momento, parece significar morosidade do desenvolvimento (lançamentos menos freqüentes). Na verdade, não existe a percepção de que esse esforço resultará em um produto de melhor qualidade e com menos imprevistos.

O primeiro objetivo deste trabalho é propor um modelo de documento e um processo que permitam a criação de especificações de requisitos que, ao mesmo tempo que atendam às normas, reduzam o volume de informações, principalmente as duplicadas, inseridas pelos desenvolvedores e o custo de manutenção, especialmente quanto à rastreabilidade dos dados. A natureza híbrida e dispersa das informações, bem como o modelo de desenvolvimento de software livre, distribuído e aberto, torna os hiperdocumentos, disponibilizados via Web, uma solução adequada para o tipo de documento em questão.

Um problema em hiperdocumentos é a manutenção das ligações entre seus diferentes nós. Especificações de requisitos compartilham essa mesma dificuldade. A utilização de uma ferramenta que facilite a atualização e recuperação de requisitos e o gerenciamento de seus inter e intra-relacionamentos é essencial. Um tipo de aplicação, relativamente recente, que satisfaz esses requisitos são as *wikis*.

As *wikis* são um tipo de software Web de edição colaborativa criado por Cunnigham (1995). Seu conceito é bem simples. Cada *wiki* é um hiperdocumento, composto por nós e ligações. Os nós são identificados por um nome, cujo valor segue uma sintaxe diferenciada, com palavras capitalizadas e sem espaços as separando. Cada ocorrência de um nome no conteúdo de um nó transforma-se em uma âncora, ligando-a ao nó com o respectivo nome. Caso o nó não exista, ele é automaticamente criado. Observe que a citação do nome do nó sempre precede a criação do mesmo. Trata-se de uma maneira bem inocente de criar hiperdocumentos. Também se demonstra muito eficiente e bem recepcionada no mundo Web, conforme demonstrado pela Wikipedia[5], uma enciclopédia *on-line* mantida por seus próprios leitores. Atualmente, até mesmo alguns projetos de software livre já empregam *wikis* para gerir sua documentação: *K Destop Environment* (KDE)[6] (ETTRICH et al., 1996), *GNU Compiler Collection* (GCC)[7] (STALLMAN et al., 1987), para citar alguns importantes exemplos.

[5] `http://www.wikipedia.org`
[6] `http://wiki.kde.org/`
[7] `http://gcc.gnu.org/wiki`

Entretanto, as *wikis* possuem algumas limitações quanto ao tipo de dados que elas armazenam. Inicialmente, elas permitiam apenas o registro de textos em linguagem natural. Posteriormente, surgiram as que suportavam figuras e outros arquivos binários, mas sem seguir o conceito original, baseado na citação do nome precedendo a criação do conteúdo.

Considerando que muitas das técnicas de Engenharia de Requisitos podem ser armazenadas como texto em linguagem natural, as *wikis* atuais seriam suficientes. No entanto, quando se necessita gerenciar automaticamente as ligações, mantendo a rastreabilidade dos requisitos ao longo do desenvolvimento do software, e seguir as diretivas das recomendações e regras das normas, a diferenciação dos tipos dos nós faz-se necessária. Isso permite a rápida identificação do tipo de artefato representado em cada nó, sem depender de analisar, sintática ou semanticamente, o documento (o que é computacionalmente ineficaz, principalmente por se tratar de linguagem natural). Uma segunda vantagem é a possibilidade de oferecer mecanismos de edição diferenciados dependendo do tipo de dado, otimizando a criação do conteúdo. Esta possibilidade demonstra-se interessante, dadas as diversas técnicas de engenharia de requisitos existentes e os mecanismos de apoio as suas aplicações que poderiam assim ser disponibilizados.

O segundo objetivo deste trabalho é desenvolver a Wiki/RE, uma *wiki* que estende o modelo de *wikis* para o registro de diversos tipos de dados, permitindo a criação de hiperdocumentos multimídia de requisitos. Ela é uma ferramenta livre, disponibilizada sob a licença *General Public License* (GPL) e pertencente ao *framework* proposto no projeto SAFE.

Através da investigação sobre hiperdocumentos para apoiar o armazenamento e a recuperação de documentos, artefatos e experiências produzidos durante a engenharia de requisitos, espera-se o uso mais efetivo das informações obtidas durante o processo de engenharia de requisitos e de software e, pela utilização da ferramenta e um processo simples de engenharia de requisitos, fundamentado nos princípios das *wikis*, a produção de especificações de requisitos de qualidade.

O restante desta dissertação está organizado da seguinte forma. O capítulo 2 apresenta uma revisão da literatura de Engenharia de Requisitos. A seguir, no capítulo 3, estuda-se o estado da arte na utilização de hipermídia e hiperdocumentos na construção de documentos de requisitos. O capítulo 4 discute *wikis*. A Wiki/RE é descrita detalhadamente no capítulo 5. Finalmente, os resultados sobre o trabalho realizado e possíveis trabalhos futuros são apresentados na conclusão, o capítulo 6.

Capítulo

2

Engenharia de Requisitos

As primeiras tarefas a serem realizadas no desenvolvimento de um software são a correta determinação das funcionalidades a serem oferecidas e a identificação de condições e restrições aplicáveis. Embora aparentemente simples, essas atividades são responsáveis pelo fracasso de muitos projetos. A questão é bem simples: *"There is no sense in being precise about something when you do not even know what you are talking about"*[1] (John von Neumann). Não é possível, ou talvez seja muitíssimo improvável, construir um software que satisfaça os reais requisitos do usuário a partir do conhecimento incorreto (ou o desconhecimento) do problema em questão.

Obviamente que programas não são escritos a partir de dados aleatórios. Mesmo assim, não se espera que uma especificação dos requisitos seja ausente de erros. Medidas são necessárias para a identificação desses o mais cedo possível no processo, evitando o efeito cascata dos custos inerentes à correção dos mesmos (PRESSMAN, 2000) e à entrega de um produto com defeitos, que podem comprometer o sucesso do projeto.

Assim, a Engenharia de Requisitos justifica sua importância na Engenharia de Software. A Engenharia de Requisitos é responsável pelo estudo e desenvolvimento de técnicas que auxiliem e guiem o processo de definição de requisitos durante toda a vida do projeto. Através do desenvolvimento e aplicação de novas técnicas, torna-se possível a detecção prematura de erros, implicando em menores custos e prazos, e a satisfação das necessidades do usuário do software.

[1] "Não existe sentido em ser preciso sobre alguma coisa quando você nem sabe sobre o que está falando".

Nas próximas seções são apresentados os conceitos de Engenharia de Requisitos, sintetizados a partir de extensa revisão bibliográfica. As principais técnicas são apresentadas, algumas das quais utilizadas no desenvolvimento da Wiki/RE e no decorrer da dissertação (metas, casos de uso e casos de mau uso).

2.1 Conceitos

O homem, durante toda sua existência, desenvolveu sistemas que lhe permitiram alcançar metas vitais para sua sobrevivência. A capacidade de produzir sons e a associação desses com objetos e fatos do cotidiano possibilitou o desenvolvimento de um sistema de fala que foi um importante meio de comunicação com outros seres da mesma comunidade e espécie. Metas mais ousadas exigiram sistemas mais complexos e interoperáveis, os quais, para serem viáveis, requereram novas teorias, elementos e ferramentas.

> **Sistema**: *(1)* uma disposição das partes ou dos elementos de um todo, coordenados entre si, e que funcionam como estrutura organizada; *(2)* técnica ou método empregado para um fim precípuo. (FERREIRA et al., 1999)

A crescente sofisticação dos sistemas teve, como conseqüência natural, uma maior exigência quanto aos métodos necessários para compreensão e desenvolvimento dos mesmos. A Engenharia de Sistemas é responsável pela criação de sistemas artificiais. Ao seu encargo estão o desenvolvimento dos equipamentos e seus inter e intra-relacionamentos, o processo e a implantação (SOMMERVILLE, 2001). A preocupação reside no todo e não nas partes: estas são criadas, de acordo com o requerido pelo sistema, por grupos especializados.

O surgimento dos computadores, no século XX, possibilitou uma revolução na maneira com que sistemas são desenvolvidos. Em um primeiro instante, eles assumiram o papel de ferramenta, agilizando o desenvolvimento através da rápida e precisa realização de cálculos. Em um segundo momento, os computadores foram inseridos como um componente dos sistemas, permitindo que técnicas, cuja aplicação com os elementos "reais" eram impraticáveis, fossem possíveis com as ferramentas "virtuais", os softwares. De meros figurantes, hoje os softwares são atores principais da maioria dos sistemas, senão os próprios sistemas.

> **Software** é *(1)* uma especificação de instruções que, ao ser executada, gera ações e dados coerentes ao fim a que o software é destinado, fim este determinado pelo sistema, e *(2)* a documentação necessária à sua operação. (The Institute of Electrical and Eletronics Engineers, 1990; SOMMERVILLE, 2001)

A Engenharia de Sistemas, conseqüentemente, especifica as metas que o software deve atender. A realização deles é atribuída a um grupo especializado em Engenharia de Software, que iniciará um processo de engenharia de software, buscando garantir a alta qualidade do produto que lhe cabe a responsabilidade.

Engenharia de Software é a disciplina de engenharia responsável por todos os aspectos da produção de software. Envolve a pesquisa e desenvolvimento de teorias, métodos e ferramentas apropriados e a efetiva e eficiente aplicação desses no desenvolvimento de software (SOMMERVILLE, 2001).

A primeira etapa desse processo é levantar, a partir das metas atribuídas, as operações que o software deve realizar e as condições e restrições aplicáveis. Elas são expressas por sentenças em linguagem natural, diagramas ou especificações formais obtidos pela aplicação de técnicas de engenharia. Esses produtos constituem os requisitos do software.

Requisito de software é uma propriedade que o software possui para resolver um problema no mundo real (IEEE Computer Society, 2004).

Requisitos são usualmente classificados em funcionais e não-funcionais (SOMMERVILLE, 2001). Os funcionais são aqueles que executam uma função, causando uma alteração no estado do sistema. Em uma primeira instância, esses requisitos seriam os mais importantes para os usuários, dado que os resultados daqueles são necessários para atender às necessidades destes. No entanto, toda atividade é executada em um meio que impõe restrições. Exemplos clássicos seriam questões legais como impostos aplicáveis ao cálculo de preços (que influenciariam o requisito funcional de verificação de preço e compra de um produto) e respeito a normas de segurança em sistemas hospitalares (como um administrador intravenoso de medicamentos em uma UTI). Essas condições que os requisitos funcionais devem observar são requisitos não-funcionais.

Requisito funcional é uma ação ou reação que o software deve apresentar quando confrontado com dados de entrada, fornecidos pelos usuários, o meio ou o próprio sistema, ou situações particulares.

Requisito não-funcional é uma restrição ou condição imposta a um ou mais requisitos funcionais do software.

Ambos os tipos de requisitos são obtidos ou definidos por pessoas interessadas nos resultados obtíveis da execução do software. Essas pessoas, também denominadas *stakeholders*, incluem os usuários diretos e indiretos do programa, os desenvolvedores e as entidades que controlam o meio ao qual o software se aplica (governo, órgãos reguladores, etc).

Interessado, ou ***stakeholder***, é uma pessoa ou entidade que participa do processo de engenharia de requisitos ou possui influência na definição dos requisitos.

A obtenção dos requisitos a partir dos interessados e sua correta definição em um formato útil ao desenvolvimento do software, a especificação de requisitos, requer a execução ordenada de diversas atividades, constituindo assim um processo. O processo de engenharia de requisitos é caracterizado por quatro etapas: elicitação, análise, especificação e validação (IEEE Computer Society, 2004):

Elicitação: Obtenção de dados e subseqüente descoberta de requisitos. As fontes investigadas são os próprios interessados, documentos, softwares similares, etc.

Análise: Organização e refinamento dos requisitos coletados na elicitação, resolução de requisitos conflitantes, criação de modelos conceituais ou de negócio.

Especificação: Definição precisa dos requisitos e dos atributos de qualidade aplicáveis a cada um deles: correção, ausência de ambigüidade, completitude, consistência, verificabilidade, modificabilidade, rastreabilidade.

Validação: Certificar-se que os requisitos foram corretamente definidos (verificação) e realmente satisfazem as necessidades de seus interessados (validação).

A execução do processo é acompanhada por métodos e técnicas, buscando um equilíbrio no esforço investido e as metas a serem alcançadas no desenvolvimento, mitigando os problemas intrínsecos e comuns à Engenharia de Requisitos: comunicação, determinação do domínio, imprecisão. A seção 2.2 detalha esses problemas e as seções 2.3 e 2.4 demonstram as soluções disponíveis em um contexto macro e micro, respectivamente.

2.2 Problemas Enfrentados

Toda engenharia vive em um campo restrito de trabalho, tornando possível a construção do conhecimento necessário para a elaboração dos elementos do sistema a ela cabível. A

complexidade desse meio é um fator importante na lista de dificuldades enfrentadas no processo de engenharia. Em Engenharia de Requisitos, por exemplo, problemas recorrentes são: falhas na comunicação, domínios de aplicação mal definidos, inconsistências, falta de completitude, ambigüidade. A maioria desses problemas poderia ser facilmente resolvido pelo uso de linguagens precisas: a ambigüidade seria eliminada por definição e, juntamente com ela, muitas das falhas de comunicação; a incompletitude e inconsistências poderiam ser automaticamente identificadas. A utilização de métodos formais para a elaboração de requisitos, como *Z*, *Vienna Development Model* (VDM), *Software Cost Reduction* (SCR), apóia-se exatamente nessa idéia. Linguagens com bases matemáticas, se corretamente utilizadas, permitem a identificação e eliminação de quaisquer erros. Algumas linguagens, como *Specification and Design Language* (SDL), permitem a geração automática de código a partir das especificações.

O emprego de métodos formais, porém, não é uma garantia de um software que atenda os requisitos do usuário. Se a especificação, apesar de corretamente escrita, descreve um requisito que o usuário informou incorretamente, o programa gerado também conterá um erro.

O elo fraco da Engenharia de Requisitos é exatamente aquilo que a torna praticamente única na Engenharia de Software: o amplo universo de *stakeholders* inseridos no processo, provendo uma interface entre os (futuros) usuários do software e a respectiva equipe de desenvolvimento. Um fato de conhecimento comum em Engenharia de Software é que o usuário nunca sabe o que quer até ver o resultado. E o papel da Engenharia de Requisitos é exatamente documentar o melhor possível esses requisitos e transmiti-los às outras etapas da engenharia.

A primeira pergunta a ser respondida é "o que é documentar o melhor possível um requisito?" A questão deve ser analisada de, no mínimo, duas perspectivas: a dos usuários e a dos desenvolvedores. As características desejáveis por cada um deles são:

Usuários: Linguagem acessível (geralmente a língua do país e termos do domínio do usuário), alto nível de abstração, sem detalhes, representações gráficas, documentação breve.

Desenvolvedores: Linguagem acessível (geralmente uma mistura do idioma local e inglês, termos do domínio da computação), vários níveis de abstração, detalhamento, representações gráficas, documentação longa.

Os engenheiros sempre desejam a especificação a mais completa possível, o que facilita a execução das outras etapas de desenvolvimento. O volume de informações e o modo como elas são descritas e organizadas exigem técnicas elaboradas. O usuário final, por outro lado, não possui a formação para compreender tais técnicas e, na verdade, está mais interessado no produto final do que na documentação que acompanha a mercadoria.

Uma solução recorrente é manter a documentação no nível desejado pelo usuário. Todos os problemas típicos de engenharia de requisitos então surgem. O impacto no restante do desenvolvimento, com constantes alterações não planejadas no código, é um aumento de custos e prazos. Os usuários ficam insatisfeitos, tanto pelo preço e espera que eles enfrentam, quanto pelos freqüentes erros encontrados no produto.

Um problema que não pode passar desapercebido diz respeito aos "detalhes" citados na resposta quanto à documentação de requisitos. Em geral, eles são características essenciais do domínio para o qual o software destina-se. A caracterização falha desse domínio é um erro grave no desenvolvimento, omitindo requisitos óbvios do sistema do ponto de vista do usuário (que pertencem ao domínio), mas desconhecidos pelos desenvolvedores. Além disso, essa falha também dificulta a comunicação usuário/desenvolvedor e o estabelecimento de limites das responsabilidades do software.

A segunda pergunta é "como gerenciar os interessados e seus requisitos?" Existem dois aspectos a serem estudados. O mais vivenciado é a gerência dos requisitos. Conforme afirmado anteriormente, os interessados não conseguem precisar, em um primeiro instante, o que eles desejam do software. Uma conseqüência natural é que, durante o desenvolvimento, os usuários consigam definir mais precisamente os requisitos e, não apenas isso, descobrir que eles estavam errados ou "queriam algo um pouco diferente, não era bem isso que tinham em mente". Alterações de requisitos no início do processo são relativamente baratas e até esperadas, mas, se essa volatilidade se mantiver (e se manterá!) durante todo o desenvolvimento, o esforço necessário para realizar a alteração será elevado e, potencialmente, afetará negativamente a qualidade do produto.

Se alguns interessados já geram muitos requisitos voláteis, sistemas de alcance global, como as aplicações Web, potencializariam o problema e tornariam o desenvolvimento inviável. A adoção de critérios para a seleção dos requisitos a serem implementados torna-se essencial.

Uma bala de prata[2] para todos os problemas aqui descritos não existe. No entanto, algumas boas soluções encontram-se disponíveis para mitigá-los. Em um nível micro, técnicas e métodos contribuem para uma comunicação fluente entre os interessados, identificação de inconsistências, classificação e priorização, análise e validação. Em uma perspectiva mais ampla, gerência de requisitos e processos iterativos permitem um tratamento adequado a requisitos voláteis e quantidade elevada de *stakeholders*.

2.3 Processo de Engenharia de Requisitos

Processos são a maneira pela qual se realiza uma operação, segundo determinadas normas, métodos e técnicas. Em Engenharia de Requisitos, o processo é dividido em estágios, de acordo com a ênfase das atividades executadas, suas metas. Não existe um consenso quanto

[2] Em alusão à "bala de prata" discutida por Brooks (1986) e subseqüentes trabalhos.

a essa divisão em Engenharia de Requisitos. Sommerville (2001) divide o processo em quatro estágios: (1) estudo de viabilidade, (2) elicitação e análise de requisitos, (3) especificação de requisitos e (4) validação de requisitos. Kotonya e Sommerville (1998) organizam-no em (1) elicitação, (2) análise e negociação, (3) documentação e (4) validação. Já Pressman (2000) divide-o em (1) elicitação, (2) análise e negociação, (3) especificação, (4) modelagem do sistema, (5) validação e (6) gerenciamento. Apesar desse aparente desentendimento, no conjunto das atividades executadas, as diferentes definições são praticamente iguais, variando apenas em seu início (Sommerville cita estudo de viabilidade, algo geralmente delegado para a Engenharia de Sistemas). Para este trabalho, assume-se a definição do SWEBOK (IEEE Computer Society, 2004) (itens 1 – 4), acrescida de uma opcional (item 0): (0) preliminares, (1) elicitação, (2) análise, (3) especificação e (4) validação.

Além das atividades identificadas como características de engenharia de requisitos, existem as de apoio. Gerenciamento de requisitos é uma disciplina presente em qualquer processo. Sua maior preocupação é a identificação dos requisitos e a manutenção da rastreabilidade dos mesmos quanto aos demais artefatos utilizados no processo de engenharia de software. Salienta-se também a importância do gerenciamento de configuração, realizado em sincronia com o gerenciamento de requisitos, permitindo o controle das diferentes versões dos requisitos e a relação desses com as configurações do software.

A execução das atividades de engenharia de requisitos sugere um desenvolvimento em cascata (modelo seqüencial linear). Os processos modernos mantêm a execução em ordem, mas em iterações curtas. De fato, o ciclo de vida evolutivo, principalmente o modelo espiral, demonstra-se vantajoso para a engenharia de requisitos, permitindo a rápida avaliação dos progressos realizados, atualização dos riscos e correção das falhas detectadas. Tratando-se de um processo tão próximo do usuário e mais susceptível a erros, admite-se até a volta para a etapa anterior, invertendo temporariamente o ciclo (SOMMERVILLE, 2001).

2.3.1 Preliminares

Existem dois cenários básicos para o desenvolvimento de um software: o software como parte de um sistema e o software como o sistema. O primeiro caso permite que algumas atividades sejam economizadas: o domínio do sistema e, conseqüentemente, da aplicação já foi limitado e estudado; os principais interessados foram identificados; a viabilidade do sistema já foi avaliada. Certamente serão necessários ajustes, como detalhamento do domínio e gerenciamento de riscos (muitos dos quais levantados durante o estudo de viabilidade), mas o esforço inicial já foi feito.

No caso do software ser o sistema, é necessária a execução de atividades preliminares, com os mesmos objetivos daquelas executadas para sistemas. A análise do domínio da aplicação permite identificar o escopo do software, os principais interessados, as metas e os riscos existentes.

O domínio da aplicação é um subconjunto do sistema no qual o software será aplicado. Ele inclui pessoas, fontes de informação e escopo do software, delimitando o que será feito. Existem várias maneiras de se representar o conhecimento sobre o domínio. Dois modos comumente utilizados são a representação do domínio através de modelos conceituais e a definição de modelos de negócio (KRUCHTEN, 1999).

Para representar o domínio, geralmente se faz uma modelagem através de um diagrama de classes, elaborado por um pequeno grupo de especialistas do domínio e desenvolvedores. Esse diagrama identifica os objetos do domínio do sistema, enfatizando o problema a ser resolvido, sem pensar nas classes em nível de projeto ou implementação. Juntamente com um glossário, permite que uma linguagem e um entendimento comum sejam estabelecidos entre as pessoas envolvidas no desenvolvimento do sistema.

Os modelos de negócio representam não somente os objetos do sistema, mas também as operações a serem realizadas ou disponibilizadas. Uma prática usual é utilizar um modelo de caso de uso simplificado, com um nível maior de abstração e não especificando os detalhes dos cenários. O relacionamento com o modelo conceitual dá-se pela rastreabilidade dos atores e dos itens de informação, dos quais os casos de uso dependem ou produzem, e as classes identificadas no modelo conceitual. Um processo que utiliza esse tipo de modelo é o *Rational Unified Process* (KRUCHTEN, 1999), representado pelo modelo de objetos de negócio (*business object model*). Esse é composto por entidades de negócio (*business entities*), *workers* e unidades de trabalho (*work units*) (KRUCHTEN, 1999). Uma unidade de trabalho é um conjunto de entidades de negócio. Para identificar esses três elementos, parte-se de um modelo de caso de uso do negócio, no qual tem-se a identificação dos atores de negócio (*business actors*) e casos de uso de negócio (*business use cases*).

A vantagem do emprego de modelos de negócio é que a razão dos elementos do domínio existirem está claramente apresentada através do modelo de casos de uso de negócio. Essas informações extras permitem diagnosticar mais rapidamente falhas na determinação do domínio (extrapolando o escopo definido pela especificação de sistema) e aumentam a precisão e a confiabilidade do modelo, elemento essencial para o início do processo de engenharia de requisitos.

2.3.2 Elicitação

Esta etapa consiste na descoberta de dados sobre o sistema, a serem utilizados na definição de requisitos. Assim, diversas fontes de dados são consultadas nessa tarefa: pessoas, livros, leis, documentos em geral, sistemas anteriores. De acordo com as fontes, encontram-se disponíveis vários meios para obtenção ou extração de dados. A tabela 2.1 mostra algumas técnicas para elicitação de requisitos, suas respectivas qualidades e deficiências segundo Leite (2001).

Uma importante heurística, válida para várias das técnicas identificadas na tabela 2.1, é a 5W1H (LEITE, 2001): O que (*What*)? Quem (*Who*)? Quando (*When*)? Onde

Técnica	Qualidades	Deficiências
Leitura de documentos	facilidade de acesso às fontes de informação, volume de informação	dispersão das informações, volume de trabalho requerido para identificação de fatos
Observação	baixo custo, pouca complexidade da tarefa	dependência do ator (observador), superficialidade decorrente da pouca exposição ao universo de informações
Entrevistas	contato direto com os atores, possibilidade de validação imediata	conhecimento tático, diferenças culturais
Reuniões	múltiplas opiniões, criação coletiva	dispersão, custo
Questionários	padronização de perguntas, tratamento estatístico	limitação das respostas, pouca interação/participação
Etnografia	visão de dentro para fora, contextualização	tempo, pouca sistematização
Participação ativa dos atores	envolvimento de clientes e usuários, validação	treinamentos, falsa impressão de eficácia
Análise de protocolos	fatos não observáveis, melhor compreensão dos fatos	foco na performance, o que se diz não é o que se faz
Engenharia reversa	disponibilidade de informação (código), reutilização	descontinuidade de informações, informação muito detalhada
Reutilização	produtividade, qualidade	nível de abstração (requisitos), possibilidade de reutilização real

Tabela 2.1: Técnicas para elicitação de requisitos. Fonte: Leite (2001).

(*Where*)? Por quê (*Why*)? Como (*How*)? Ela auxilia não apenas na obtenção dos dados, mas também da justificativa de serem necessários e suas dependências com outros elementos do sistema.

2.3.3 Análise

A elicitação gera uma quantidade elevada de dados que servem como base para a descoberta de requisitos. À etapa de análise cabe identificar os requisitos, refinando-os e garantindo o entendimento comum dos mesmos por todos os interessados, resolvendo possíveis conflitos. Enquanto que, na elicitação, as técnicas são comuns às engenharias, com uma forte ênfase nos aspectos humanos e sociais e, conseqüentemente, contando com o auxílio de outras áreas (como a psicologia), a análise contém técnicas voltadas especificamente para a engenharia de requisitos.

A criação de modelos, com base nos requisitos coletados, é uma importante atividade na análise de requisitos. Os modelos propiciam uma representação concisa das informações e abstrações envolvidas, facilitando o entendimento dos problemas a serem abordados pelo

sistema e a definição dos requisitos. A possibilidade de exploração do modelo por meio da própria linguagem utilizada, sua execução ou simplesmente através de verificações feitas pelos engenheiros (de maneira mais efetiva devido às suas características intrínsecas), permite a identificação precoce de erros e inconsistências. Casos de uso, casos de mau uso e cenários, por exemplo, permitem demonstrar as funções desempenhadas pelo sistema e a iteração com os seus usuários: sua representação gráfica permite mostrar, claramente, as relações entre as diferentes funções e interessados; a descrição de cada elemento presente no modelo, por sua vez, fornece os detalhes necessários às etapas posteriores de engenharia de software.

Uma necessidade na etapa de análise é a correta identificação e tratamento dos riscos presentes. A utilização de um método orientado a metas permite identificar as metas de um software e os *soft-goals*, fatores que ameaçam ou contribuem para o alcance das metas. Os fatores negativos podem ser interpretados como riscos. Essas informações podem ser relacionadas a outros artefatos (como casos de mau uso), garantindo que os riscos serão debelados ou, ao menos, amenizados.

Durante a definição dos requisitos, é natural o conflito entre os conceitos utilizados e elementos definidos pelos diversos interessados. Essa dificuldade advém da próprio natureza do processo, executado por grupos heterogêneos de pessoas. Obviamente, um consenso deve ser alcançado antes de prosseguir às próximas etapas do desenvolvimento.

Um modelo conceitual é traçado na etapa preliminar, permitindo um entendimento comum mínimo entre os *stakeholders*. Não obstante, esse modelo torna-se ineficiente com o subseqüente refinamento dos artefatos e a participação de um contingente maior de pessoas. Um mecanismo simples que deve sempre ser empregado é um glossário, que trará as definições consideradas como corretas e que serão utilizadas ao longo do processo.

Uma solução mais refinada ao glossário é a técnica de Léxico Ampliado de Linguagem (LAL). Ela define como serão descritas as definições, sugerindo uma sintaxe e aplicando critérios para avaliar a qualidade do glossário. Em termos simples, ela verifica a percentagem de termos, da definição de um termo, que são definidos no glossário (princípio da circularidade) e a percentagem daqueles termos que não pertencem ao glossário (princípio da minimalidade). Em outras palavras, busca-se um artefato (o modelo conceitual gerado pelo LAL) cujo entendimento depende apenas dele mesmo.

Evitados os conflitos quanto ao significado dos termos, restam aqueles provenientes de discordância quanto à lógica, ao conteúdo do requisito. Não é incomum um usuário possuir uma necessidade antagônica a de outro, ou várias opiniões sobre um mesmo aspecto do software divergirem entre os participantes. Para essas questões, não existe uma técnica "mágica" que resolva os problemas: cabe ao engenheiro, juntamente com os usuários, descobrir a melhor solução possível. São inúmeros os fatores a serem considerados: obrigações legais impostas por organismos externos (governo), quantidade de interessados a ser atendida, relevância do interessado quanto a determinado tema, prazos, etc. Não se trata apenas de

um problema de engenharia de requisitos, envolvendo de maneira significativa a gerência de projeto. Nesse cenário complexo, uma preocupação constante deve ser a de justificar adequadamente as decisões tomadas, definindo critérios técnicos e garantindo que haja justiça.

O primeiro passo é identificar o conflito (problema), suas fontes (objeto que apresenta o problema) e os envolvidos (pessoas responsáveis pelo objeto). O conflito pode ser registrado de maneira análoga a uma alteração de software, descrevendo detalhadamente o problema, o objeto e, possivelmente, a solução (ou o que a pessoa acredita ser o correto). Esse relato deve ser encaminhado ao responsável pelo objeto, que contactará as pessoas que participaram no processo de sua criação (usuários e desenvolvedores) e consultará os dados da licitação que o originaram para discutir o conflito. A técnica de pontos de vista, quando utilizada na elicitação e durante a análise, permite identificar facilmente os interessados envolvidos e o grau de importância das opiniões que eles tecem sobre o assunto. Um exemplo simples seria uma questão relacionada à segurança: o ponto de vista de uma pessoa especialista em segurança é mais relevante que aquele de um gerente de banco quando se trata dos mecanismos de acesso a informações, enquanto que seria menos importante para as políticas de acesso.

2.3.4 Especificação

Após a análise, os requisitos devem ser disponibilizados em um documento de formato de fácil acesso. Usualmente utilizam-se dois tipos de documentos: o documento de conceito de operações e a especificação de requisitos.

O documento de conceito de operações, também conhecido pela sigla ConOps (*Concept of Operations*), é voltado principalmente ao usuário. Ele descreve o sistema como um todo, em um alto nível de abstração, explicando desde a situação que motivou o desenvolvimento do software, as melhorias desejadas para essa situação, como seria o software e sua operação, e quais os impactos, benéficos e maléficos, de seu uso.

A especificação de requisitos documenta todos os requisitos, levantados e analisados, que serão encaminhados para a equipe de projeto, implementação e teste de software. Trata-se de um documento eminentemente técnico, voltado aos desenvolvedores. Devido ao fato dele guiar o subseqüente desenvolvimento do software, ele é redigido mais rigorosamente que o *Concept of Operations* (ConOps). Espera-se o seguinte conjunto de atributos: corretitude, ausência de ambigüidades, completitude, consistência, classificação dos requisitos, verificabilidade, modificabilidade e rastreabilidade. Alguns desses atributos, como a rastreabilidade (a habilidade de descobrir a origem de cada requisitos e seus intra-relacionamentos), exige um grande esforço em sua construção e manutenção, dependente de um eficiente gerenciamento de requisitos.

Existem diversos padrões para documentos de requisitos. Os para ConOps, como o IEEE Std 1362 (Institute of Electrical and Electronics Engineers, 1998a), DI-IPSC-81430 (Departa-

ment of Defense, 1994), requisitos na subseção 5.1.1.1 da ISO/IEC 12207 (International Organization for Standardization (ISO); International Electrotechnical Commission (IEC), 1995), etc, dada a natureza das informações envolvidas, possuem uma estrutura semelhante (Institute of Electrical and Electronics Engineers, 1998a).

Os padrões para especificações de requisitos mais conhecidos são o DID-IPSC-81433 (Departament of Defense, 1994) e a IEEE-830 (Institute of Electrical and Electronics Engineers, 1998b). Eles definem o formato da especificação e as características esperadas dos requisitos nela definidos. O DI-IPSC-41433, apesar de oficialmente abandonado em 1998, encontra-se presente (quanto ao conteúdo) no padrão J-STD-016 F.2.4 (Institute of Electrical and Electronics Engineers; Electronics Industry Association (EIA), 1995), ainda utilizado para projetos governamentais da área militar. A IEEE-830, conforme definido em seu anexo "Guidelines for compliance with IEEE/EIA 12107.1-1997", permite a criação de documentos de requisitos de software que atendam as exigências do ISO/EIA 12207[3] (Institute of Electrical and Electronics Engineers; Electronics Industry Association, 1998). Portanto, ambos demonstram-se importantes no cenário atual de especificações de engenharia de requisitos.

2.3.5 Validação

Verificação e validação não são atividades exclusivas da engenharia de requisitos. Avaliar a corretitude no emprego do processo e do produto em si é vital para garantir o sucesso dos projetos. Revisões permitem a descoberta de erros nos artefatos e de falhas no processo utilizado. Um exemplo de técnica de revisão que pode ser empregada é a *Perspective Based Reading* (PBR) (SHULL; RUS; BASILI, 2000).

A técnica de pontos de vista também pode ser utilizada para validação de requisitos. Quando adotada uma linguagem de especificação para as visões, como a proposta por Leite e Freeman (1991), é possível identificar automaticamente conflitos entre as perspectivas e validá-las.

2.3.6 Atividades de Apoio

Além dos processos típicos de apoio ao ciclo de vida de software (documentação, gerenciamento de configuração, garantia de qualidade, verificação, validação, revisões conjuntas, auditorias e resolução de problemas) (International Organization for Standardization (ISO); International Electrotechnical Commission (IEC), 1995), o gerenciamento e classificação de requisitos são importantes atividades de apoio.

[3] O ISO/EIA 12207 é adaptação da IEEE para o padrão internacional ISO/IEC 12207 (International Organization for Standardization (ISO); International Electrotechnical Commission (IEC), 1995).

Gerenciamento de Requisitos

A gerência de requisitos assemelha-se ao gerenciamento de configuração em suas principais funções: identificação dos itens de configuração (requisitos); controle de versão; definição de configurações e *baselines*; consulta aos itens e configurações. Uma importante função adicional é manter a rastreabilidade entre os itens de configuração.

Todo requisito deve ser unicamente identificado. Além de permitir distinguir cada item, o identificador é o instrumento utilizado para associar o requisito aos artefatos produzidos em etapas posteriores do processo de engenharia de software. Os requisitos são usualmente identificados por números. Essa estratégia facilita a especificação de quantidades elevadas de requisitos, característica presente, por exemplo, quando a maioria dos requisitos são atômicos (indivisíveis e, portanto, muito pequenos): seria inviável nomear, distintamente, cada um desses elementos.

Definidos os itens de configuração, a criação de *baselines* segue idéia análoga àquela utilizada para arquivos: alcançado um marco nos principais artefatos (como a aprovação da especificação de requisitos pelos clientes), cria-se uma marcação ou ramificação dos itens. Alterações nos itens, a partir desse momento, exigem um processo mais apurado, com um maior controle de qualidade.

Gerenciados os itens, resta controlar os relacionamentos entre os mesmos. Em engenharia de software, busca-se justificar cada novo artefato. Em um processo bem definido e completo, tal justificativa sempre englobaria algum outro artefato pré-existente. O ponto inicial, "a ponta do *iceberg*", concentra-se nos proponentes do projeto e os motivos que os levaram a tal empreitada. A análise desses dados permite a elaboração dos primeiros artefatos, como um esboço do documento de ConOps. Iteração após iteração, novas fontes de dados são descobertas e requisitos criados e refinados. Essa produção irá alimentar as equipes de projeto, implementação e testes, que seguem processo análogo. O registro dessas ligações forma uma verdadeira teia de informação que, se devidamente organizada, permite o desenvolvimento controlado de software.

Esses relacionamentos podem ocorrer em duas direções: dos artefatos gerados antes da engenharia de requisitos para os requisitos (rastreabilidade para trás) e dos artefatos gerados após a engenharia de requisitos para os requisitos (rastreabilidade para frente). Também existe um sentido nessas associações: dos artefatos para os requisitos e dos requisitos para artefatos. Uma representação adequada para essa estrutura é um grafo direcionado, potencialmente convexo: expresso nesses termos, a complexidade do problema torna-se evidente.

Abordagens para criação das associações entre artefatos variam de geração automática de ligações (como em busca de definição de termos no glossário), sugestão de relacionamentos entre artefatos (com base em seus conteúdos) e tão simplesmente a disponibilização de mecanismos para criação manual das ligações. Quanto mais abordagens forem utilizadas,

identificando corretamente as relações entre os artefatos, mais eficiente será a posterior busca de informações na base de dados.

Considerando que os relacionamentos são criados naturalmente durante o processo (o que não é sempre verdade), o gerenciamento seria relativamente simples. No entanto, requisitos tendem a ser voláteis: usuários mudando ou corrigindo opiniões, conflitos esclarecidos, alterações nos requisitos do sistema, novos requisitos não-funcionais, etc. Nessas condições, torna-se inevitável a alteração dos requisitos. Quando o requisito é apenas modificado, a questão é relativamente simples (do ponto de vista de engenharia de requisitos): os artefatos que dependem dos requisitos são determinados e revisados, verificando-se os impactos das mudanças. Porém, se o requisito é excluído ou dividido em vários novos requisitos, a manutenção da consistência dos relacionamentos torna-se complexa: relacionamentos, cujos alvos eram o requisito original, devem ser redirecionados aos novos requisitos ou imediatamente atualizados. Esta última alternativa possui um custo: ela depende de intervenção manual. Se foram vários os requisitos afetados, torna-se ainda mais cara. Se as alterações são muito freqüentes, o esforço despendido na manutenção da rastreabilidade será mais alto que aquele despendido nas demais etapas do processo de engenharia, o que é indesejável. Uma solução interessante seria tolerar as inconsistências, redirecionando as consultas para o alvo dividido para os vários novos requisitos. Apesar do *recall* (objetos que são realmente a fonte da informação desejada) ser reduzido, a precisão mantem-se inalterada, ou seja, sempre existirá, dentre os alvos, ao menos um requisito que contém a informação.

A utilização de ferramentas para o gerenciamento de requisitos permite controlar os custos e utilizar eficientemente a estrutura de dados disponível. Trata-se de um campo fértil de pesquisa em Engenharia de Requisitos (REIFER, 2000; RAMESH; JARKE, 2001; WANG; LAI, 2001; CLELAND-HUANG; ZEMONT; LUKASIK, 2004; DAG et al., 2004; HAYES et al., 2004; HOFFMAN et al., 2004; LOCONSOLE, 2004; DAG et al., 2005) e com diversas soluções a disposição para aquisição: RequisitePro (IBM Rational, 2003), DOORS (Telelogic, 2005), etc. Infelizmente, não se encontram disponíveis ferramentas livres para gerência de requisitos até o presente momento.

Classificação de Requisitos

A quantidade de requisitos em projetos de software eleva-se continuamente durante o ciclo de vida de um software. A cada nova versão lançada, usuários detectam pontos a serem melhorados e problemas que devem ser corrigidos. Atender a todas essas requisições é uma tarefa inviável. A preservação do negócio depende da escolha daquelas mais importantes ao projeto, conciliando as metas dos interessados e do software, minimizando os riscos.

Por exemplo, uma estratégia para a classificação dos requisitos seria a seguinte. Iniciar-se-ia com a categorização dos requisitos quanto a tipo, contexto e severidade. O cruzamento desses dados com aqueles disponíveis sobre o proponente do requisito permitiria

a elaboração de uma lista de priorização. Essas atividades poderiam ser automatizadas sem grandes dificuldades. O desenvolvedor escolheria, então, os requisitos com maior prioridade para uma análise mais detalhada. As metas do projeto, o quanto os requisitos contribuiriam para o alcance delas e os riscos presentes seriam utilizados para determinar a aceitação do novo requisito.

Esse exemplo ilustra como poderia ser executada a classificação de requisitos. O ideal é utilizar o máximo de informações disponíveis com o menor custo possível. Para isso, a automatização na coleta e análise dos dados é essencial. O uso de diversas técnicas (como pontos de vistas e metas) e a rastreabilidade entre os artefatos são imprescindíveis para uma eficiente classificação.

2.4 Técnicas

Um processo de engenharia de requisitos deve escolher o conjunto mais apropriado de técnicas, combinando os artefatos gerados de maneira a otimizar o resultado final. São analisadas, nesta seção, apenas as técnicas julgadas apropriadas para o processo de engenharia de requisitos em software livre. Algumas dessas técnicas, como metas, casos de uso e casos de mau uso foram utilizados no desenvolvimento da ferramenta de engenharia de requisitos deste trabalho.

A seguir, são listadas as técnicas, encontradas na literatura, que oferecem suporte às respectivas atividades da Engenharia de Requisitos:

- **Elicitação**
  - Facilitadores de comunicação (LEITE; FRANCO, 1993; DAMIAN et al., 2000)
  - Pontos de vista (SOMMERVILLE; SAWYER, 1997)
- **Análise**
  - Metas (MYLOPOULOS; CHUNG; YU, 1999; LAMSWEERDE, 2001)
  - Casos de uso (JACOBSON et al., 1992)
  - Casos de mau uso (ALEXANDER, 2003)
  - Cenários (RIDAO; DOORN; LEITE, 2000; BREITMAN; LEITE, 2002)
  - Pontos de vista (LEITE, 1989; FINKELSTEIN; KRAMER; GOEDICKE, 1990; FICKAS; LAMSWEERDE; DARDENNE, 1991; NARAYANASWAMY; GOLDMAN, 1992; EASTERBROOK; NUSEIBEH, 1996; KOTONYA; SOMMERVILLE, 1995; SOMMERVILLE, 1996; SOMMERVILLE; SAWYER, 1997)
  - Padrões (RIDAO; DOORN; LEITE, 2000; TORO et al., 2000; HAGGE; LAPPE, 2005)

- **Especificação**
  - Modelos de referência (GUNTER et al., 2000)
  - Métodos formais (WIERINGA, 1995)
- **Validação**
  - Inspeção (SHULL; RUS; BASILI, 2000)
  - Pontos de vista (LEITE, 1989; LEITE; FREEMAN, 1991)
  - Revisão
  - Verificação automática (DURÁN; RUIZ; TORO, 2001)
- **Atividades de apoio**
  - Gerenciamento de Requisitos (RAMESH; JARKE, 2001; ZYLBERMANN; COHEN; GOLDIN, 2003; DAG et al., 2004; HAYES et al., 2004; HOFFMAN et al., 2004; CLELAND-HUANG; ZEMONT; LUKASIK, 2004; DAG et al., 2005; LOCONSOLE, 2004)
  - Classificação e priorização de requisitos (ZANLORENCI; BURNETT, 2000)

2.4.1 Léxico Ampliado de Linguagem

O Léxico Ampliado de Linguagem (LAL) (LEITE; FRANCO, 1993) é uma técnica facilitadora de comunicação que define uma heurística, independente de domínio, para derivar um modelo conceitual a partir de um léxico, implementando conceitos de semiótica em um hipergrafo. A técnica parte do princípio de que, no universo de informação, existe uma ou mais culturas e que cada cultura (grupo social) possui sua linguagem própria. Portanto, o principal objetivo a ser perseguido pelos engenheiros de requisitos é a identificação de palavras ou frases particulares ao meio social da aplicação sob estudo. Somente após a identificação dessas frases e palavras é que se procura seu significado. A estratégia de elicitação é ancorada na sintaxe da linguagem.

A construção do LAL é realizada em duas fases: a construção do léxico e a derivação do modelo conceitual. A primeira fase inicia-se com a coleta de fatos através de entrevistas, observação, leitura de documentos, etc. Sem se preocupar com a compreensão do problema, procuram-se palavras ou frases (símbolos) que pareçam ter um significado especial na aplicação: freqüentemente utilizadas, que geram dúvidas ou estão aparentemente fora do contexto. Cada item identificado é descrito como noção e impacto. Noção, ou denotação, é o significado do símbolo. Impacto, ou conotação, descreve efeitos do uso ou ocorrência do símbolo na aplicação ou do efeito de algo da aplicação sobre o símbolo. A figura 2.1 apresenta esses conceitos na forma de um diagrama de classes.

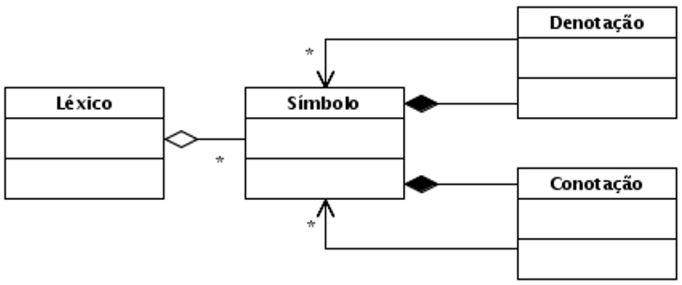

Figura 2.1: Modelo conceitual de Léxico Ampliado de Linguagem.

As descrições dos símbolos obedecem aos princípios da circularidade e do vocabulário mínimo. O primeiro dita que cada denotação e conotação deve fazer referência a outros símbolos. O princípio do vocabulário mínimo consiste em restringir as partes da denotação que não fazem parte do léxico a um conjunto reduzido de palavras.

2.4.2 Metas

As metas capturam, em diferentes níveis de abstração, os vários objetivos que um sistema deve atender (LAMSWEERDE, 2001). Em outras palavras, as metas são as ambições dos interessados quanto ao software: o que a utilização do software contribuirá, positivamente, para o alcance de seus objetivos pessoais. Se os requisitos definem o que o software deve fazer, metas são o porquê do software ser necessário. A justificativa de um software é elemento essencial da definição de requisitos, como afirmam Ross e Schoman (1977):

> *... requirements definition must say why a system is needed based on current or foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context.*[4]

A concretização de uma meta depende das contribuições dos componentes do ambiente em que o sistema será implementado: pessoas, software e dispositivos. Ao contrário do ambiente em si, passivo, os componentes são ativos, desempenhando papéis e assumindo responsabilidades. Devido a esse comportamento, eles são denominados de agentes.

A presença de agentes possibilita a divisão das responsabilidades quanto aos atendimentos das metas. Esse mecanismo permite a identificação dos requisitos do software, que são aqueles derivados das metas associadas ao agente "software". As demais metas cabem aos demais componentes, ativos e passivos. Esta é uma abordagem bem prática para a tarefa nada trivial de delimitar as responsabilidades do software.

[4] ... a definição de requisitos deve dizer a razão de um sistema ser necessário com base nas condições atuais ou futuras, que podem ser operações internas ou um mercado externo. Ela deve dizer quais as características do sistema atenderão e satisfarão este contexto.

As metas são um importante recurso para a avaliação das propriedades de uma especificação de requisitos: a completitude, pela existência de, ao menos, um requisito associado a cada meta; a corretitude, pela satisfação de, ao menos, uma meta para cada requisito; a rastreabilidade, por meio das diversas ligações entre os artefatos e as metas; a consistência, pela detecção de inconsistências; a modificabilidade, graças à separação das informações de acordo com sua volatilidade (quanto maior o nível de uma meta, maior sua estabilidade).

As metas são obtidas, inicialmente, dos interessados e de material previamente disponível. Deficiências identificadas em sistemas atuais e demais materiais, quando negadas, transformam-se em uma lista de metas do novo sistema. A partir de um conjunto inicial, uma parcela significativa das metas pode ser detectada pelo refinamento e abstração das metas e requisitos já conhecidos.

Toda meta possui um nome condizente ao que ela tem como objetivos. Além disso, define-se o tipo, atributos (geralmente dependentes do tipo) e especificação (dependente do método utilizado, como o KAOS (DARDENNE; LAMSWEERDE; FICKAS, 1993), o NFR (CHUNG et al., 1991) ou o i* (YU, 1997)). O modelo definido na figura 2.2 demonstra o modelo conceitual genérico de metas.

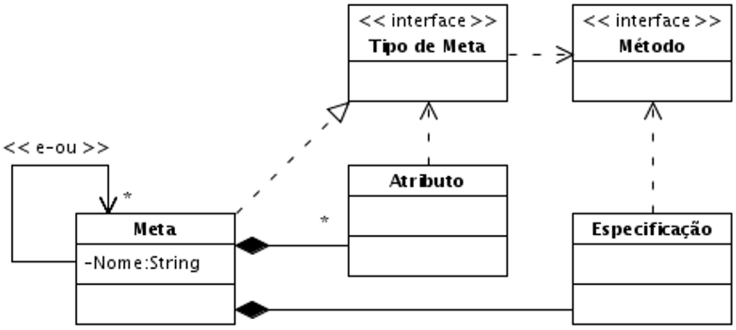

Figura 2.2: Modelo conceitual da técnica de metas.

Modelos apropriados foram criados para organização de metas. Árvore e/ou são estruturas comuns para armazená-los, dada sua propriedade de capturar os relacionamentos de refinamento das metas e a detecção de conflitos (quando a realização de uma meta impede que outra meta seja realizável).

Neste trabalho, utilizaram-se metas do tipo i*, classificadas em *hard-goals* (ou simplesmente *metas*) e *soft-goals*. A diferença entre ambas é que a última representa fatores que contribuem, positiva ou negativamente, para a realização da meta a qual ela está associada.

A figura 2.3 contém um exemplo de um diagrama segundo o método i*. Os desenhos ovais representam metas e as "nuvens" *softgoals*. Importante notar que, no modelo em questão, é impossível satisfazer completamente as metas: a *softgoal 2* contribui negativamente para a *meta 2*, ao mesmo tempo que a *softgoal 1* contribui positivamente. Caso todos os elementos do modelo fossem do tipo *metas*, existiria uma inconsistência no modelo (a *meta 2* não poderia ser satisfeita). A despeito disso, a *meta 1* sempre será satisfeita: ela é decom-

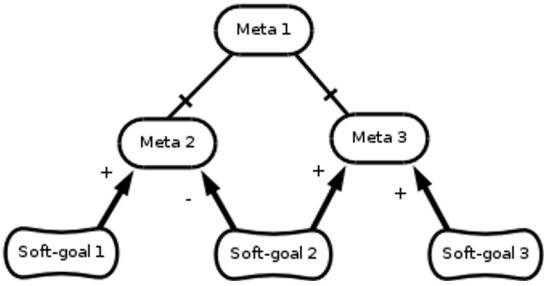

Figura 2.3: Exemplo de um modelo segundo o método i*.

posta em duas metas, *meta 2* e *meta 3*, e a satisfação de ao menos uma de suas sub-metas é suficiente para sua própria satisfação.

2.4.3 Cenários

Os cenários são descrições de situações do universo de discurso (e, conseqüentemente, do mundo real). Eles podem ser compostos por vários episódios e expressos como um episódio dentro de um contexto (e com as restrições desse último). Os cenários podem ter exceções, restrições e gatilhos que acionam outros episódios. Além disso, um cenário possui recursos, envolve atores e satisfaz metas. Os episódios são caracterizados por:

Título: nome que identifica o episódio.

Propósito: objetivo que a situação busca satisfazer.

Atores: atores (pessoas, aparatos ou organizações) envolvidos na situação.

Recursos: elementos necessários à execução da situação.

Tempo: momento específico sendo representado.

Lugar: contexto geográfico do acontecimento.

Restrições: pré-condições existentes.

2.4.4 Casos de Uso

Os casos de uso representam uma funcionalidade, com os atores participando dos casos de uso. Contêm os atores, os casos de uso e os relacionamentos entre os dois primeiros. Um relacionamento pode ser do tipo:

***Include*:** Somente o resultado do caso de uso é utilizado, não altera o comportamento do caso de uso que o inclui.

***Generalization*:** Altera o comportamento do caso de uso.

***Extend*:** Altera o comportamento de um caso de uso em determinado ponto se a pré-condição for atendida.

A identificação dos atores pode ser feita com base nos seguintes critérios:

- Quem está interessado nos requisitos.
- Quem se beneficiará do produto.
- Quem fornecerá informação ao produto.
- Quem usará informação do produto.
- Quem removerá informação do produto.

Em se tratando de casos de uso, os critérios para descrevê-los podem ser:

- Quais as tarefas de cada ator.
- Que informação cada ator cria, armazena, consulta, altera ou remove.
- Que informação cada caso de uso cria, armazena, consulta, altera ou remove.

Finalmente, nos relacionamentos:

- Quando e como o caso de uso se inicia.
- Como o caso de uso interage com os atores.
- Seqüência padrão dos passos do caso de uso.
- Seqüências de exceções e alternativas dos passos do caso de uso.

A utilização de diagramas facilita a rápida visualização dos casos de uso e atores do sistema, mas os casos de uso em si são descritos mais detalhadamente em separado, contendo as seguintes informações:

- Estado inicial (pré-condição).
- Como e quando inicia.
- Ordem em que as ações devem ser executadas.
- Como e quando termina.
- Definição de possíveis estados finais.

- Caminhos de execução permitidos.
- Caminhos de execução alternativos.
- Interação do sistema com o ator e o que é trocado entre eles.
- Uso de objetos, valores e recursos do sistema.
- Explicitação do que o ator e o sistema fazem.

2.4.5 Casos de Mau Uso

Os casos de mau uso são o complemento natural dos casos de uso. São sempre associados a requisitos não-funcionais, representando eficientemente cenários que ameaçam o correto funcionamento do software, tais como (in)segurança e (baixa) confiabilidade.

O relacionamento de casos de uso e de mau uso ocorre por dois tipos especiais de relacionamento: *threaten* e *mitigate*. Casos de mau uso podem ameaçar (*threaten*) os casos de uso. Detectadas as ameaças, novos casos de uso devem ser elaborados para mitigar (*mitigate*) os casos de mau uso.

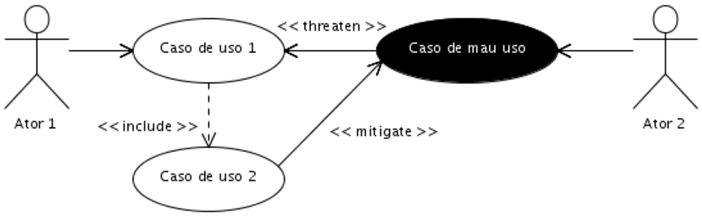

Figura 2.4: Exemplo de um modelo de caso de uso que utiliza casos de mau uso.

Um caso de mau uso é especificado de maneira idêntica a um caso de uso. De fato, é possível imaginar o caso de mau uso como um estereótipo de um caso de uso. O mesmo é válido para os relacionamentos. A representação dos elementos do modelo no diagrama é exemplificada na figura 2.4. A única diferença visível, além dos estereótipos dos relacionamentos, é a alteração da cor de preenchimento do caso de mau uso, simbolizando o caráter negativo do mesmo.

A principal vantagem dos casos de mau uso não é o tratamento de requisitos não-funcionais: ele já ocorreria, utilizando outras abordagens. No entanto, ao aproximar tais requisitos dos pontos que eles afetam, é facilitada a visualização das implicações, contribuindo positivamente para a completitude do modelo.

2.4.6 Modelos para Qualificação de Requisitos

A qualificação de requisitos, estabelecendo assim prioridades e avaliando riscos, possui papel importante no processo evolutivo de engenharia de requisitos. Em REQAV (ZANLORENCI;

BURNETT, 2000), por exemplo, são propostos critérios de valor e peso à informação dos *stakeholders*, calculados em cinco fases:

1. Descrição dos requisitos.
2. Qualificação dos requisitos.
3. Qualificação das fontes de informação.
4. Aplicação de parâmetros de qualificação.
5. Composição do quadro de avaliação de riscos de implementação do requisito.

A descrição dos requisitos consiste em:

- Planejamento.
- Pesquisa inicial do material existente.
- Identificação do *stakeholder*.
- Descrição inicial dos requisitos.
- Estruturação dos dados.
- Composição da versão inicial do documento de requisitos.

A qualificação de requisitos visa obter as propriedades de cada requisito e a relação de dependência entre eles. Três aspectos são analisados: qualificação funcional, área de origem e a relação de dependência entre os requisitos.

Durante a qualificação da fonte de informação, obtém-se a qualificação do *stakeholder* em função de seu ponto de vista, sua qualificação funcional na organização e a exigência da informação.

A aplicação de parâmetros de qualificação compreende a apropriação dos resultados das etapas de qualificação dos requisitos e das fontes de informação e o comparativo dos resultados (a ser usado na próxima fase).

Finalmente, na fase de composição do quadro de avaliação de riscos, a partir das informações obtidas na qualificação dos requisitos e das fontes de informação, juntamente com a aplicação de parâmetros de qualificação, um quadro de avaliação de riscos é elaborado.

As vantagens da utilização de modelos de qualificação de requisitos são:

- Permitir a fácil visualização de requisitos prioritários.
- Identificação dos requisitos que devem ser revisados (alto risco ou alto grau de dependência, por exemplo).

- Facilitação da manutenção do foco durante o desenvolvimento do sistema.

- Utilização das informações geradas para a negociação de requisitos.

2.4.7 Pontos de Vista

A engenharia de requisitos orientada a pontos de vista teve sua origem no reconhecimento de que os requisitos do sistema são gerados por várias fontes distintas e que tal realidade deve ser incluída explicitamente no processo de engenharia. Os primeiros trabalhos a respeito dessa técnica são do final da década de 1970, como o SADT (SCHOMAN; ROSS, 1977) e o CORE (MULLERY, 1979). Mais recentemente, na década de 1990, várias técnicas com essa abordagem foram desenvolvidas: o VORD (FINKELSTEIN; KRAMER; GOEDICKE, 1990), trabalhos de Leite e Franco (LEITE, 1989), Kotonya e Sommerville (KOTONYA; SOMMERVILLE, 1995), Sommerville e Sawyer (SOMMERVILLE; SAWYER, 1997) e Easterbrook e Nuseibeh (EASTERBROOK; NUSEIBEH, 1996).

Os pontos de vista podem ser informalmente definidos como um posicionamento do indivíduo quando examinando ou observando o domínio da aplicação. A visão, produto gerado a partir desse ponto de vista, consiste em um conjunto de fatos descritos segundo algum critério.

O conceito da técnica permite uma análise multifacetada do software e seus requisitos, o que é explorável em todas as etapas do processo de engenharia de requisitos. As diferentes técnicas de pontos de vista especificam quais dados devem ser obtidos das visões, como representá-los e processá-los, variando o grau de formalização e automatização da aplicação da técnica e o custo de sua execução. Por exemplo, a técnica de Leite (1989), direcionada para a validação de requisitos, define uma linguagem baseada em *LISt Processing language* (LISP) para descrever as visões e permite detectar, automaticamente, conflitos entre elas.

As vantagens dos pontos de vista são, principalmente, o melhor aproveitamento dos vários atores no domínio, obtendo mais requisitos (e mais completos), e a fácil integração em processos atuais de engenharia de software. Mas existe uma desvantagem, ou característica, a ser observada: a grande quantidade de pontos de vista manipulados e o custo inerente ao seu gerenciamento (FINKELSTEIN; SOMMERVILLE, 1996).

Os pontos de vista geram muitos dados, exigindo a utilização de ferramentas e o emprego de técnicas que filtrem os mais relevantes para o projeto. A realização dessa atividade de modo totalmente automático, no entanto, se mostra arriscada: pontos de vista geralmente são informais (característica esta inerente da interface *stakeholder* x engenheiro) e a avaliação da relevância dos pontos de vista tornar-se-ia imprecisa e potencialmente inseriria erros, o que se deseja evitar. Uma solução semi-automática evitaria tais equívocos. O sistema atribuiria um valor ao ponto de vista, calculado a partir do conteúdo, da pessoa que

o criou e os relacionamentos com outros pontos de vista. O desenvolvedor, de posse dessas informações, decidiria se um ponto de vista é relevante ou não.

Existe ainda uma questão de grande impacto na definição dos requisitos: a resolução de conflitos. A partir dos pontos de vista, podem-se identificar os conflitos e sugerir soluções adequadas, mas não o raciocínio para obtê-las. Na verdade, a identificação de conflitos é feita graças à formalização da maneira como os pontos de vista são descritos e à definição de regras que, uma vez desobedecidas, poderiam ser resolvidas por conjuntos bem conhecidos de ações. Infelizmente, isso pode parecer insatisfatório para o *stakeholder* devido à não explicitação dos motivos e idéias, ou seja, o caminho traçado para cada alternativa fornecida. Além disso, como o tempo de vida dos requisitos é grande e eles estão submetidos sempre a mudanças, mesmo que pequenas, seria normal a repetição de determinados problemas. Portanto, apresenta-se interessante a aplicação de um raciocínio utilizado anteriormente para evitar erros nesses casos. A captura do *design rationale* oferece uma possível solução (ou pelo menos um grande auxílio) na resolução desse problema (PAIVA; FORTES, 2005).

2.5 Qualidade

Os requisitos são reunidos em um documento denominado Especificação de Requisitos de Software. As seguintes características são desejáveis nesse documento (Institute of Electrical and Electronics Engineers, 1998b): corretitude, ausência de ambigüidade, completitude, consistência, classificação, verificabilidade, modificabilidade e rastreabilidade.

Corretitude

Uma especificação de requisitos é dita correta quando está de acordo com outros documentos previamente estabelecidos como, por exemplo, a especificação de sistema. Mas, muitas vezes, tais documentos não existem e uma definição mais independente deve ser utilizada. Assim, a corretitude implica em definir apenas requisitos que o software deva cumprir.

Ausência de ambigüidade

Qualidade de um requisito admitir mais de uma interpretação. A ambigüidade é um problema constante nas definições em requisitos, principalmente nas escritas em linguagem natural. Assegurar sua inexistência é possível por meio de métodos formais. No entanto, eles não são soluções adequadas em qualquer caso. Em especial durante a especificação, uma descrição em linguagem natural é muito mais facilmente desenvolvida do que em linguagem formal, apesar da perda de precisão. A adoção de um vocabulário comum e glossários inibe o problema desde a elicitação. Durante a análise, a criação de modelos permite uma visualização mais rápida do sistema e sua organização, facilitando a compreensão dos requisitos.

Completitude

A especificação de requisitos deve estar completa em dois níveis: quanto aos requisitos necessários e à definição dos mesmos. Todo requisito do sistema deve estar contido na especificação ou devidamente referenciado, de maneira que esse documento seja suficiente para descrever o software. Cada requisito definido, por sua vez, deve ter suas entradas (válidas e inválidas) e saídas especificadas.

Outro aspecto de completitude é em relação ao documento em si: todas as figuras, tabelas e diagramas devem estar rotulados; unidades de medidas e termos utilizados no texto devem estar claramente definidos.

Uma prática comum em especificações é anotar trechos incompletos ou ausentes, geralmente identificando-os com a sigla TBD (*To Be Determined*). É natural que nem todos os dados estejam disponíveis de antemão, mas, entre ignorar a ausência e identificá-la, assinalando que ela deve ser futuramente sanada, melhor essa última alternativa. No entanto, uma especificação não está completa até que todos os TBDs sejam resolvidos.

Consistência

Inconsistência é uma situação em que duas partes da especificação não obedecem a algum relacionamento mantido entre elas (EASTERBROOK; NUSEIBEH, 1996). Esses relacionamentos podem se referir aos aspectos sintáticos e semânticos, além de relacionamentos inerentes ao próprio processo. Um exemplo bem simples de inconsistência seria um requisito definir que a distância será em metros e um outro que será em jardas. Os trabalhos de Easterbrook e Nuseibeh (1996) tratam em detalhes essa questão, definindo abordagens e estratégias para identificação e resolução de inconsistências.

Classificação

Dentre os requisitos definidos na especificação, é natural uns terem importância maior do que outros. Essa classificação pode ser feita de acordo com os mais diversos fatores: volatilidade/estabilidade, importância e exigência legal. Além disso, podem-se considerar fatores organizacionais, ordenando os requisitos também de acordo com as necessidades da empresa (tempo de mercado, custo/benefício).

Verificabilidade

Para todo requisito da especificação, deve existir um mecanismo que permita avaliar se ele está sendo atendido. A criação de casos de teste nas primeiras iterações do processo de engenharia permite identificar rapidamente requisitos não verificáveis (que serão aqueles para os quais os casos de testes não puderam ser definidos). A metodologia ágil *eXtreme*

Programming (XP) (BECK, 1999), por exemplo, apóia-se fortemente em testes, especificando-os desde as primeiras iterações de desenvolvimento.

Modificabilidade

Os requisitos são inerentemente instáveis e voláteis, assim como o é a natureza humana. No início, existe a incerteza sobre o que é necessário e muitas idéias sobre o sistema são definidas. Posteriormente, após melhor compreensão, novas idéias surgem, possibilitadas pelos avanços conseguidos com o entendimento do sistema. E esse processo se repete indefinidamente. A estrutura da especificação de requisitos deve ser tal que mudanças possam ser acomodadas com o menor esforço possível. Isso implica uma boa modularização e organização, evitando redundâncias e buscando definir requisitos o mais atomicamente possível.

Rastreabilidade

A capacidade de saber as origens e os destinos dos requisitos é fundamental. A rastreabilidade permite guiar o processo de engenharia como um todo, identificando as causas de um problema e possíveis conseqüências de alterações. Muitas técnicas dependem da rastreabilidade para serem efetivas.

2.6 Considerações Finais

O início do capítulo é um importante registro de quão simples a Engenharia de Requisitos pode ser, lembrando de sua ligação com a Engenharia de Sistemas e o fato de que esta é uma antiga atividade humana, desempenhada há centenas de anos. Apesar de toda essa experiência acumulada, a realidade é que muitos projetos falham exatamente nesse ponto.

Um argumento seria que, para sistemas simples, não é necessário muito esforço de engenharia. A questão é que os softwares, hoje, são complexos em diversas dimensões, tanto técnicas (segurança, paralelismo, flexibilidade, etc) quanto econômicas (globalização, concorrência, legislação). A negligência dessas questões não é uma opção.

A engenharia de requisitos é inerentemente complexa e vários esforços vêm sendo realizados para torná-la uma realidade no processo de engenharia de software, garantindo produtos de qualidade. As diversas técnicas, aqui brevemente explicadas, testemunham os avanços feitos na área. Devidamente empregadas, com processos de engenharia de requisitos apropriados, um importante obstáculo no desenvolvimento de software pode ser transposto.

Não obstante, não se observa uma integração entre essas diferentes técnicas. Cada uma possui um enfoque distinto, dedicando-se a alguns dos problemas descritos na seção 2.2. A melhor solução seria combinar seus produtos, gerando assim uma Especificação de Requi-

sitos de Software de alta qualidade. Infelizmente, unir os artefatos gerados nesse documento não é trivial, principalmente frente aos seus atributos desejáveis.

Dois fatores são necessários para unir as técnicas: processo e documentação. O primeiro permite executá-las, o segundo registrá-las. Para o processo, este trabalho assume o processo de software livre e a liberdade que ele disponibiliza aos desenvolvedores. Quanto ao registro, o capítulo seguinte discute como fazê-lo, utilizando hiperdocumentos.

Capítulo

3

Engenharia de Requisitos e Hiperdocumentos

Uma das grandes dificuldades no emprego das técnicas de apoio a engenharia de requisitos, citadas no capítulo 2, se refere à integração das mesmas, relacionando, de maneira fácil e clara, seus produtos e experiências para melhorar o resultado final do processo de engenharia de requisitos.

Os documentos de especificação de requisitos são o recurso comumente utilizado para expor os requisitos, buscando a consolidação dos diversos artefatos produzidos pelas técnicas. Eles possuem uma organização linear das informações e apresentam os inter-relacionamentos, a rastreabilidade, geralmente por meio de tabelas ou referências cruzadas.

Um conhecido problema nas especificações de requisitos trata-se justamente dos custos de criação e manutenção da rastreabilidade. A identificação dos relacionamentos é uma tarefa manual ou semi-automática, realizada, geralmente, após a criação dos artefatos. A atualização desses, necessidade constante frente à volatilidade dos requisitos, exige os maiores esforços.

Uma abordagem para a racionalização do gerenciamento de requisitos é a utilização de sistemas hipermídia. A representação dos artefatos em nós e os relacionamentos através de ligações e âncoras mostram-se apropriados para a engenharia de requisitos e para a geração e manutenção dos documentos de requisitos. Os hiperdocumentos produzidos também facilitam a leitura não seqüencial das especificações, contribuindo positivamente para

o processo. A seguir, são apresentados os conceitos de hipermídia e as aplicações desses na engenharia de requisitos.

3.1 Conceitos de Hipermídia

Inspirado pelo trabalho de Bush (1945), idealizador do *memex*, e convencido de que transcrever, linearmente, o conhecimento não era uma solução adequada (NELSON, 1990), Nelson (1965, p. 96) cunhou o termo *hipertexto*:

> *[...] a body of written or pictorial material interconnected in such a complex way that it could not conveniently be presented or represented on paper. It may contain summaries, or maps of its contents and their interrelations; it may contain annotations, additions and footnotes from scholars who have examined it. [...] Such a system could grow indefinitely, gradually including more and more of the world's written knowledge. However, its internal file structure would have to be built to accept growth, change and complex informational arrangements.*[1]

Hipertextos (ou hiperdocumentos) armazenam as informações em estruturas denominadas nós e os relacionamentos em ligações (*links*), ancorados nos nós. Segundo a definição de Nelson, os nós de um hipertexto podem conter texto e figuras, de modo semelhante ao presente em livros (os "nós" seriam as "páginas" neste caso). Quando se associam outros tipos de mídia, como vídeos, sons, modelos tridimensionais (como aqueles descritos em VRML), denominam-se os documentos como hipermídia.

Nó: item de informação presente em um documento hipermídia.
Ligação: inter-relacionamento entre nós em um documento hipermídia.
Âncora: ponto do nó a que se prende a ligação e que, ao ser seguido, aciona a ligação e ativa os nós presos à outra ponta da ligação.

Os sistemas que permitem a criação de documentos hipermídia, ou simplesmente hiperdocumentos, são caracterizados, segundo o modelo Dexter (HALASZ; SCHWARTZ, 1994), por três camadas:

[1] [...] um corpo de material escrito ou pictográfico interconectado de maneira tão complexa que ele não poderia ser convenientemente representado em papel. Ele pode conter sumários ou mapas de seus conteúdos e inter-relacionamentos; ele pode conter anotações, adendos e notas de rodapé de estudiosos que o examinaram. [...] Tal sistema poderia crescer indefinidamente, incluindo gradualmente mais e mais do conhecimento escrito do mundo. No entanto, sua estrutura interna de organização teria que permitir o crescimento, troca e arranjos complexos de informações.

***runtime layer*:** instancia o hiperdocumento para apresentação e oferece mecanismos de interação ao usuário.

***storage layer*:** armazena a rede de nós e ligações que são a essência do hiperdocumento.

***within components layer*:** armazena o conteúdo interno aos nós e ligações.

Resumidamente, as características de um sistema hipermídia, observando o modelo Dexter (HALASZ; SCHWARTZ, 1994) e as demais definições da literatura (DEROSE; DURAND, 1994; NIELSEN, 1995; LENNON, 1997), são:

- hiperdocumentos compostos por nós, ligações e âncoras;
- nós simples e compostos;
- distinção entre estrutura e conteúdo;
- armazenamento de conteúdo dos nós em diferentes formatos;
- diversos tipos de ligações: específicas (unidirecionais), bidirecionais, locais, com vários alvos (cardinalidade superior a 2) e computadas (ou dinâmicas).

  ligação específica: possui uma âncora fonte e uma âncora destino e é usada, geralmente, para relacionar informações detalhadas sobre um assunto em particular.

  ligação bidirecional: é semelhante a uma ligação unidirecional, mas sem a distinção entre a âncora fonte e destino.

  ligação com vários alvos: reflete automaticamente em toda a ocorrência da âncora destino no documento corrente.

  ligação computada ou dinâmica: é resolvida em tempo de execução.
- aspectos de apresentação;
- aspectos de acesso.

As atuais implementações de sistemas hipermídia atendem às características de sistemas hipermídia em diferentes graus[2]. Aplicações Web típicas (baseadas em documentos HTML), por exemplo, permitem conteúdos em diferentes formatos, mas não exigem a distinção entre estrutura e conteúdo. Elas também não incorporam ligações bidirecionais e de cardinalidade um para muitos.

Uma categoria de sistemas hipermídia, que recebeu destaque na década de 90, foi a de sistemas hipermídia abertos (OHS - *Open Hypermedia System*). Seu diferencial é que

[2] Os autores do Dexter reconhecem que o modelo oferece mais recursos que os próprios sistemas hipermídia existentes (HALASZ; SCHWARTZ, 1990). Eles afirmam que o modelo foi especificado de modo a acomodar características de futuros sistemas hipermídia.

os conteúdos não estão restritos àqueles armazenados localmente pelo sistema, permitindo a integração com outros sistemas (DAVIS; LEWIS; RIZK, 1996). Para isso, existe não apenas a distinção entre estrutura e conteúdo, e sim uma total separação dos mesmos (MILLARD; DAVIS; MOREAN, 2000a; MILLARD; DAVIS; MOREAN, 2000b). Os recursos para criação e gerenciamento de ligações são um elemento a parte do sistema, geralmente na forma de um componente intermediário no ambiente computacional (*middleware*), o qual oferece funcionalidade hipermídia para aplicativos (o armazenamento e apresentação são de responsabilidade dos aplicativos e não mais do sistema hipermídia).

3.2 Sistemas Hipermídia de Apoio à Engenharia de Requisitos

Os sistemas hipermídia ainda são pouco explorados em Engenharia de Requisitos. Na literatura científica, são descritas as ferramentas C&L (FELICÍSSIMO; LEITE; BREITMAN, 2004; Grupo de Engenharia de Requisitos (PUC-RJ), 2004), Eudibamus (Frauhofer Institute, 2003) (criado pelo projeto KOJITO (Fraunhofer FIRST et al., 2003; MARSCHALL; SCHOENMAKERS, 2003)), RETH (KAINDL; KRAMER, 1995; KAINDL, 2004), REM (TORO, 2004) e HERE (PAPAIOANNOU; THEODOULIDIS, 1996).

Constata-se uma concentração da pesquisa no tópico rastreabilidade, com a criação automática (BOMPANI; CIANCARINI; VITALI, 2000) ou semi-automática de ligações (KAINDL; KRAMER, 1995; DAG et al., 2004; HAYES et al., 2004; DAG et al., 2005; CLELAND-HUANG; ZEMONT; LUKASIK, 2004), e modelos e visualização de documentos (BOMPANI; CIANCARINI; VITALI, 2000; GUDGEIRSSON, 2000).

A seguir, são revisados alguns dos trabalhos, da literatura, de cada um desses tópicos, iniciando com Aiken e sua visão sobre o uso de hipermídia para a engenharia de requisitos, abordagens para definir a rastreabilidade entre os requisitos, a descrição de ferramentas que permitem a criação de hiperdocumentos e de modelos de hiperdocumentos de requisitos.

3.2.1 Engenharia de Requisitos baseada em Hipermídia

Aiken (1990) salienta o uso de sistemas hipermídia para representar dados multimídia, permitindo assim a criação de definições ricas (sons e vídeos) para questões, cujas representações baseada em texto e diagramas seriam deficientes, além da exploração dos relacionamentos entre os diversos tipos de dados. O seu raciocínio fundamenta-se em um ponto: "a maneira com que a informação é representada é freqüentemente um fator crítico na determinação do grau de dificuldade da solução para o problema; ou na averiguação de uma aplicação funcionar ou não" (WEBSTER, 1988 apud AIKEN, 1990, p. 12). Sistemas hipermídia provêem

meios para a aplicação de métodos adequados para representar os requisitos e o fornecimento de uma representação eficiente das mídias.

A fim de descrever a relação entre engenharia de requisitos e soluções hipermídia, Aiken divide a engenharia de requisitos em oito tarefas, agrupadas em quatro grupos:

1. **Captura:** (1) elicitação.
2. **Organização:** (2) perfil das tarefas, (3) dos usuários e (4) do ambiente.
3. **Estruturação:** (5) análise, (6) modelagem e (7) projeto.
4. **Apresentação:** (8) prototipação.

A captura consiste na aquisição de informações sobre os usuários, as tarefas que o sistema deve desempenhar e as características relevantes da organização. A utilização de diversos métodos de captura (som, vídeo) e, conseqüentemente, a redução da necessidade de transformar os dados capturados em requisitos disponibilizam uma quantidade de informações maior e melhor para o engenheiro de requisitos. O esforço poupado na transformação dos dados elicitados em uma outra mídia (como textos) pode ser utilizado para a captura de mais requisitos, o que leva ao maior volume de informações. A melhora na qualidade dos dados advém não apenas da maior quantidade de dados obtidos, que retratam mais fielmente o domínio do produto a ser produzido, mas também do menor índice de erros desses dados, devido à não execução de transformações dos mesmos.

Os dados capturados devem ser organizados de maneira a facilitar seu posterior acesso. A fase de organização é considerada crítica por causa do maior volume de dados obtidos na tarefa de elicitação. As informações devem ser organizadas, aproveitando-se os diversos tipos de mídia disponíveis e transformando objetos não-estruturados em estruturados (utilizando-se pesquisas ou mineração de dados e mecanismos ágeis para manipulação dos objetos), permitindo assim a composição de representações apropriadas dos requisitos.

A estruturação visa aumentar a conectividade das informações previamente organizadas, ao mesmo tempo em que um modelo da aplicação, no ambiente em que ela será implantada, é criado. A criação desses modelos exige a análise apurada dos dados e deve contar com o suporte visual e os processos cognitivos associados à tarefa. Uma navegação rápida, ágil e fácil é imperativa, assim como facilidades para manipular e manter os requisitos individualmente.

A apresentação, por fim, deve fornecer uma visão uníssona entre os requisitos definidos e aqueles imaginados pelo usuário. A utilização de protótipos ajuda a garantir essa concordância.

O trabalho de Aiken (1990) explora os usos potenciais de hipermídia para a engenharia de requisitos. Existe um foco importante no usuário e na retratação fiel dos requisitos da

aplicação. À época de sua publicação, a captura de dados "ricos" (vídeos e sons) era um processo caro e a distribuição e o acesso aos mesmos eram uma grande dificuldade. No início dos anos 90, a Internet ainda não possuía a infra-estrutura e, principalmente, os serviços (Web) que encontramos hoje. Atualmente, seria possível concretizar os ideais de Aiken, necessitando, para isso, "apenas" desenvolver as aplicações e os métodos adequados.

3.2.2 Rastreabilidade

A rastreabilidade entre requisitos pode ser representada como ligações entre nós em hiperdocumentos. Essas ligações podem ser determinadas de modo manual, semi-automático e automático. As ligações manuais são aquelas criadas pelo engenheiro de requisitos de modo *ad hoc*. Considerando os engenheiros infalíveis, elas são sempre corretas. Porém, elas também são as mais custosas. A identificação automática de relacionamentos, por outro lado, possui um custo zero de utilização[3]. No entanto, apesar de ela utilizar-se de regras que reproduzem aquelas empregues pelos engenheiros, não é possível garantir que o conjunto de regras definido esteja completo: a criação de relacionamentos geralmente requer o uso de conhecimento tácito. Além disso, a maneira como o requisito é expresso (texto em linguagem natural, por exemplo) pode dificultar a identificação do conteúdo tratado no nó, impedindo a aplicação automática das regras. Para esses casos, existe a opção de ligações semi-automáticas: determinam-se, automaticamente, os possíveis relacionamentos entre os nós e oferece-se, ao engenheiro, a decisão quanto à validade das ligações.

O trabalho de Kaindl e Kramer (1995), "Geração Semi-Automática de Ligações de Dicionários em um Hipertexto", explora a geração manual e semi-automática de ligações em um ambiente industrial, evidenciando a importância de ferramentas que apóiem a aplicação da técnica. Em "Funcionalidades Hipertexto Sofisticadas para Engenharia de Software", Bompani, Ciancarini e Vitali (2000) apresentam uma abordagem automática para criação de ligações por meio de transformações de modelos.

Geração Semi-Automática de Ligações de Dicionários em um Hipertexto

Kaindl e Kramer (KAINDL; KRAMER, 1995; KAINDL, 2004) desenvolveram um método para criação semi-automática de ligações globais entre palavras-chave e ligações locais nos dicionários de dados. Semi-automática porque associar às palavras, em suas mais diversas variações, um significado é um processo imperfeito se realizado automaticamente. A intervenção humana permite a escolha do significado mais apropriado à palavra.

Uma ferramenta, denominada *Requirement Engineering Through Hypertext* (RETH), foi desenvolvida por Kaindl (2004) para automatizar a aplicação do método proposto. Ela

[3] Elas possuem um custo computacional, mas não requerem intervenção humana, que é o principal fator em consideração nesse caso.

identifica automaticamente, a partir dos termos presentes no dicionário de dados, potenciais associações no texto. Essas são oferecidas ao usuário, que as classifica como corretas ou não. Outra possibilidade é a criação de uma associação *ad hoc* pelo usuário. Nesse caso, o sistema tenta aprender esse novo relacionamento, utilizando-o em subseqüentes execuções.

Foram realizados três experimentos reais: no *Conseil Europeenne pour la Recherche Nucleaire* (CERN) (projeto Cortex), na Siemens e na *European Space Agency* (ESA) (*ESOC PASTEL Mission Planning System*). No primeiro, a ferramenta RETH ainda não estava completa, sendo necessária a criação das ligações manualmente. Isto aumentou muito a sobrecarga cognitiva, mesmo que a interface do sistema fosse simples e confortável. A geração semi-automática demonstrou-se necessária para explorar melhor a idéia. Nos dois últimos projetos, essa infra-estrutura estava disponível, facilitando consideravelmente a utilização da técnica: o único trabalho era verificar se a ligação era ou não apropriada (e geralmente ela estava correta).

Mesmo sendo uma técnica simples, os experimentos realizados na indústria e ambientes controlados demonstraram sua importante contribuição no correto desenvolvimento dos requisitos.

Funcionalidades Hipertexto Sofisticadas para Engenharia de Software

Bompani, Ciancarini e Vitali (2000) discutiram a utilização de hipertexto em um ambiente de engenharia de software utilizando-se de linguagens (XML, XPointer, XLink, etc) e protocolos (HTTP, WebDAV) de tecnologia Web. Foi proposto que o documento de especificação de requisitos fosse visualizado através de um documento XML, sucessivamente transformado até a obtenção de um documento HTML, com visualização avançada dos dados através de *displets* (*applets* Java). A esse hiperdocumento dá-se o nome de *Declarative Active Document* (DAD): documentos ativos que realizam computação, engajam metas e produzem resultados. As sucessivas transformações que geram o DAD são realizadas pela ferramenta XMLC, que associa a cada elemento XML um comportamento (*displet*).

O DAD, segundo Bompani, Ciancarini e Vitali (2000) acresce aos sistemas Web as seguintes características:

- nós e ligações tipadas;
- ligações com atributos e consultas baseadas em estrutura;
- transclusões, ligações “normais” e “quentes”;
- anotações;
- ligações públicas e privadas;
- ligações personalizadas computadas;

- bases de ligações externas e mecanismos de atualização de ligações;
- visões gerais e locais (*overviews*);
- rastros (*trails*) e visitas guiadas;
- *backtracking* e navegação baseada em histórico.

Essas características podem ser agrupadas em duas categorias: produção de informação e exploração das informações disponíveis. Atualmente, o XMLc permite a geração de DADs que possuem características voltadas à exploração.

A criação de um DAD ocorre em dois passos: determinação dos relacionamentos e processamento do documento, com a identificação dos objetos, seus relacionamentos (ligações) e pontos de conexão com outros objetos (âncoras) e a transformação XSLT. Esta gera páginas *Hypertext Markup Language* (HTML), compostas por *displets*, que permitem o acesso do documento por navegadores Web usuais.

O primeiro passo é permitir a criação automática de relacionamentos entre os artefatos gerados pelo processo de engenharia de software. Os relacionamentos propostos por Bompani, Ciancarini e Vitali (2000) são classificados, de acordo com o escopo dos artefatos, em:

Interfases: relacionamentos entre documentos pertencentes a diferentes fases do processo de engenharia de software.

Intrafases: entre documentos da mesma fase.

Interpartes: entre partes diferentes de um mesmo documento.

Intrapartes: dentro de uma mesma parte do documento.

Tal classificação é insuficiente para descrever a riqueza inerente ao desenvolvimento, pois não considera a semântica fornecida pelos processos em atuação e diferentes tipos de dados envolvidos. Bompani, Ciancarini e Vitali (2000) propuseram uma classificação complementar, baseada nos tipos de relacionamentos:

Esquema: conexão entre a estrutura das entidades do processo de software com seus elementos. Por exemplo, módulos de pacotes de software e pontos de função de um procedimento complexo.

Ontológico: conexão entre elementos de dados, programas, pessoas, processo e ambiente com seus respectivos parâmetros e descrições, o que permite fornecer informação genérica a respeito de uma entidade.

Ocorrência: conexão entre todos os usos de um elemento na aplicação.

Processo: conexão entre os elementos e a atividade à qual eles se envolvem.

Estrutural: conexão que possui restrições e referências a serem aplicadas entre os elementos relacionados.

Dinâmico: conexão que se torna aparente somente durante o tempo de vida da aplicação.

***Ad hoc*:** conexão cuja existência não pode ser determinada por uma regra.

Com base nessa classificação, é possível estabelecer uma relação entre os documentos gerados no processo de engenharia de software e as ligações ($Documento \rightarrow TipoXEscopo$). Esse conjunto de ligações, com exceção dos seus pares envolvendo relacionamentos dinâmicos e *ad hoc*, pode ser gerado automaticamente.

A ferramenta XMLC permite a geração de um hiperdocumento segundo o método proposto. Dado um documento XML como argumento, as seguintes operações são realizadas:

1. ler o documento *eXtensible Markup Language* (XML);
2. gerar a árvore *Document Object Model* (DOM) a partir do documento XML;
3. criar as âncoras da árvore através do XPointer (árvore DOM convertida em outra árvore DOM, resolvendo âncoras, utilizando o XPointer Manager);
4. criar as ligações entre os nós através do XLink (esta árvore DOM é convertida em outra árvore DOM, resolvendo as ligações por meio de XLink Manager);
5. aplicar transformações *eXtensible Stylesheet Language Transformations* (XSLT) a cada nó da árvore até que todo elemento XML tenha um *displet* para visualização de seu conteúdo. Os *displets* são associados da seguinte forma:
   - o nome do elemento do documento XML em questão determina a classe Java a ser usada;
   - os atributos do elemento determinam as propriedades da instância da classe;
   - o conteúdo do elemento é adicionado como filho do nó da árvore.
6. instanciar todos os displets.

Em linhas gerais, a transformação associa um *displet* para cada elemento identificado no DOM. O nome do elemento determina a classe Java (*displet*) a ser usada e os atributos do elemento determinam as propriedades da instância de classe.

3.2.3 Ferramentas

As ferramentas analisadas foram a REM, de Durán, e HERE, desenvolvida por Papaioannou e Theodoulidis. A ferramenta RETH foi descrita na seção 3.2.2, aplicando os conceitos para a criação semi-automática de ligações.

REM

A *REquirements Management tool* (REM) (TORO et al., 1999; DURÁN, 2000; TORO, 2000) é uma ferramenta experimental de apoio à engenharia de requisitos de acordo com a metodologia definida na tese de Toro (*A Methodological Framework for Requirements Engineering of Information Systems*). Ela define um projeto composto por quatro documentos:

- documento de requisitos orientado ao cliente;
- documento de requisitos orientado ao desenvolvedor;
- registro de conflitos e defeitos de requisitos;
- registro de solicitações de alteração de requisitos.

Os documentos são compostos por objetos REM, conforme pode ser visto no diagrama contido na figura 3.1. Esses objetos podem ser requisitos voltados ao cliente (`Requisitos C`), requisitos voltados ao desenvolvedor (`Requisitos D`) e interessados.

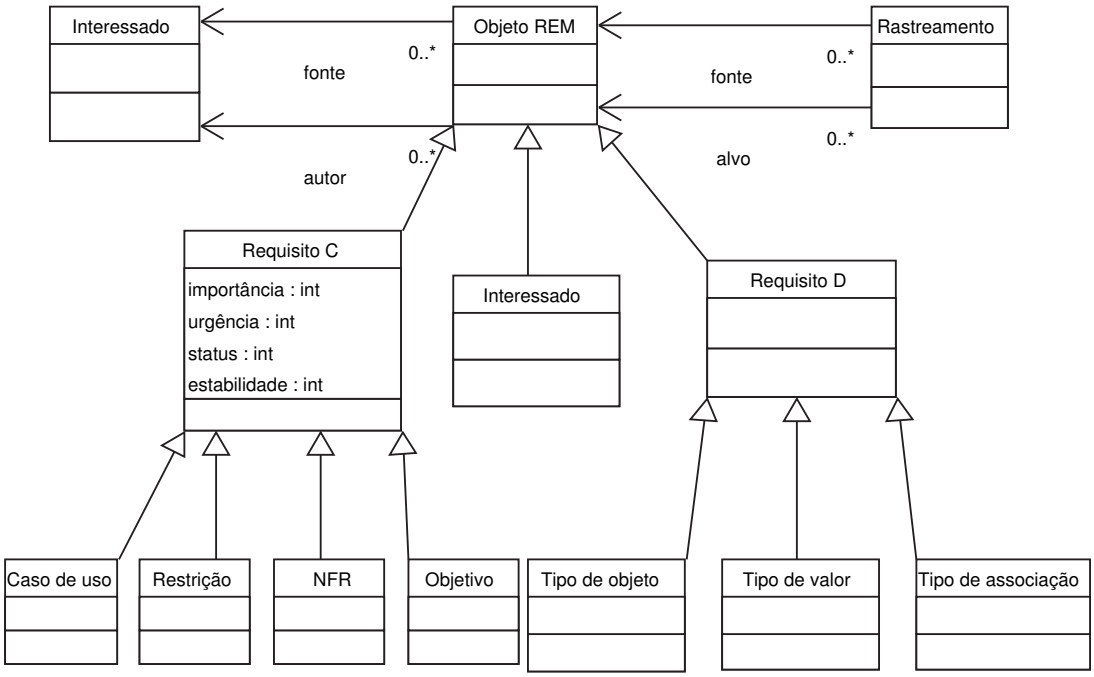

Figura 3.1: Modelo conceitual do documento utilizado pelo REM.

Os dados são armazenados em bancos de dados relacionais, mas a especificação de requisitos é gerada como um documento XML. Sobre esse documento são aplicadas transformações XSLT para:

- gerar os documentos que compõem o projeto;
- apresentar os dados (como páginas em HTML);
- mostrar a rastreabilidade entre os itens do documento.

Em trabalhos posteriores de Durán, Ruiz e Toro (2001), utilizam-se transformações XSLT para verificar automaticamente a especificação de requisitos, seguindo os seguintes atributos de qualidade:

Ausência de ambigüidade: verificada indiretamente através da medição da circularidade do glossário (GLC - *Glossary Circularity*) e minimidade do vocabulário (MOV - *Minimality of Vocabulary*):

GLC: razão entre itens de glossário e referências para itens de glossário a partir de outros itens;

MOV: razão entre o número de referências para itens de glossário e o número de requisitos.

Completitude: verificação da existência das várias seções do documento (caso utilizando a REM, isso é garantido pela própria ferramenta), ausência de informação incompleta (identificadas como TBD).

Rastreabilidade: verificação da existência de uma fonte para cada requisito descrito, detecção de requisitos não-funcionais não rastreáveis (e provavelmente não atendidos).

Essas não são as únicas verificações que podem ser feitas. Transformações XSLT são facilmente criadas e podem ser moldadas para projetos específicos, permitindo a identificação mais rápida e precisa de desvios de qualidade na especificação de requisitos.

Um Ambiente para Engenharia de Requisitos baseado em Web

Um grande problema da engenharia de requisitos é a complexidade do processo. O *Hypermedia Environment for Requirements Engineering* (HERE) (PAPAIOANNOU; THEODOULIDIS, 1996) classifica-a em complexidade de produto, organizacional e do processo. O HERE se propõe a atuar na complexidade do produto, ou seja, nos variados produtos da engenharia de requisitos em suas mais diversas formas, cobrindo níveis variados de conhecimento, visões e formalidade:

- ligar os modelos "*enterprise*", especificação de sistemas de informação e "*raw*" requisitos, com o intuito de capturar "*design rationale*" (requisitos são classificados em "*raw*" e de sistema);
- melhorar a comunicação entre os desenvolvedores do sistema e os clientes;
- melhorar a rastreabilidade de pontos de vista e representações dentro da empresa;
- facilitar a integração de pontos de vista e representações dentro da empresa;
- facilitar as ligações dos requisitos a recursos humanos e outras informações relativas aos requisitos.

O mecanismo principal do HERE é a habilidade de manter informação sobre modelos, conhecimentos e dados do domínio do problema em um conjunto gerenciável de conceitualização explícita compartilhada (uma ontologia). Essa ontologia é conhecimento (não confundir com o conhecimento que ela está representando) e é representada através de ligações entre estes três componentes e armazenada em um repositório de dados hipermídia.

O conhecimento no HERE é classificado em: requisitos de domínio, domínio da empresa e sistema de informação. Esse conhecimento é definido em modelos e metamodelos que descrevem conceitos e metaconceitos, respectivamente:

Conhecimento de modelos: contém conhecimento sobre descrição de modelos que podem ser usados em domínios (ou seja, contém informações de metamodelo). Esta informação envolve os três tipos de conhecimento: de sistema de informação, de requisito e de empresa.

Conhecimento de domínios: conhecimento modelado para um domínio específico. Na prática, é uma instância de um metamodelo do componente "conhecimento de modelos". Também abrange os três tipos de conhecimento: de sistema de informação, de requisito e de empresa.

Repositório: mantém toda forma de ontologia, além de toda informação relevante ao *Universe of Discourse* (UoD).

Os conceitos e metaconceitos, por sua vez, são descritos através de ontologias. A utilização do HERE consistiria, então, basicamente, de:

Criação de ontologias: uma ontologia pode incluir outras previamente existentes.

Edição de ontologias: definem-se, utilizando a linguagem O_2C[4], classes pertencentes à ontologia, podendo derivá-las de classes previamente existentes.

[4] Linguagem de banco de dados orientado a objeto de 4ª geração.

Instanciação de modelos: consiste em instanciar uma ontologia em um determinado domínio e preenchimento dos atributos dos objetos criados.

Todos esses dados estão automaticamente relacionados, podendo-se navegar desde o nível de ontologia até uma instância de um modelo.

3.2.4 Modelos de Documentos

Os modelos para os hiperdocumentos de requisitos especificam o modo como os conteúdos e seus inter-relacionamentos são definidos no hiperdocumento. Eles facilitam a criação de ferramentas que apóiem o desenvolvimento do hiperdocumento e avaliação da qualidade do mesmo. Apesar da importância de seu emprego, existe uma baixa incidência de pesquisas quanto a modelos de hiperdocumentos de requisitos, um provável reflexo da baixa incidência de uso de sistemas hipermídia para edição de documentos de requisitos. Estudou-se, nesse trabalho, a RQML, de Gudgeirsson.

RQML

A *Requirements Markup Language* (RQML) (GUDGEIRSSON, 2000) é uma definição de documento XML específica para documentação de requisitos. A utilização de um hiperdocumento visa superar as dificuldades quanto ao uso de documentos feitos em editores de texto comuns, ao mesmo tempo em que garante a interoperabilidade não encontrada em soluções baseadas em bancos de dados.

Procedeu, a determinação da linguagem, um estudo sobre características desejáveis para hiperdocumentos, observando cada etapa da engenharia de requisitos. As seguintes características foram consideradas relevantes e oferecidas, em grau variado, pela RQML:

- integração de outros artefatos (casos de uso, glossário);
- rastreamento (fonte das informações, identificação de *stakeholder*);
- controle de versão;
- reutilização;
- determinação de prioridades.

A partir disso, um modelo lógico da linguagem foi proposto, conforme ilustrado na figura 3.2.

Uma vantagem em adotar documentos XML é a facilidade de transformar seus dados através de XSLT. O trabalho de Gudgeirsson (2000) aborda duas transformações em especial: a representação do documento em uma página HTML, visível em qualquer navegador, e a criação de um texto descrevendo informações de rastreabilidade e classificação dos requisitos.

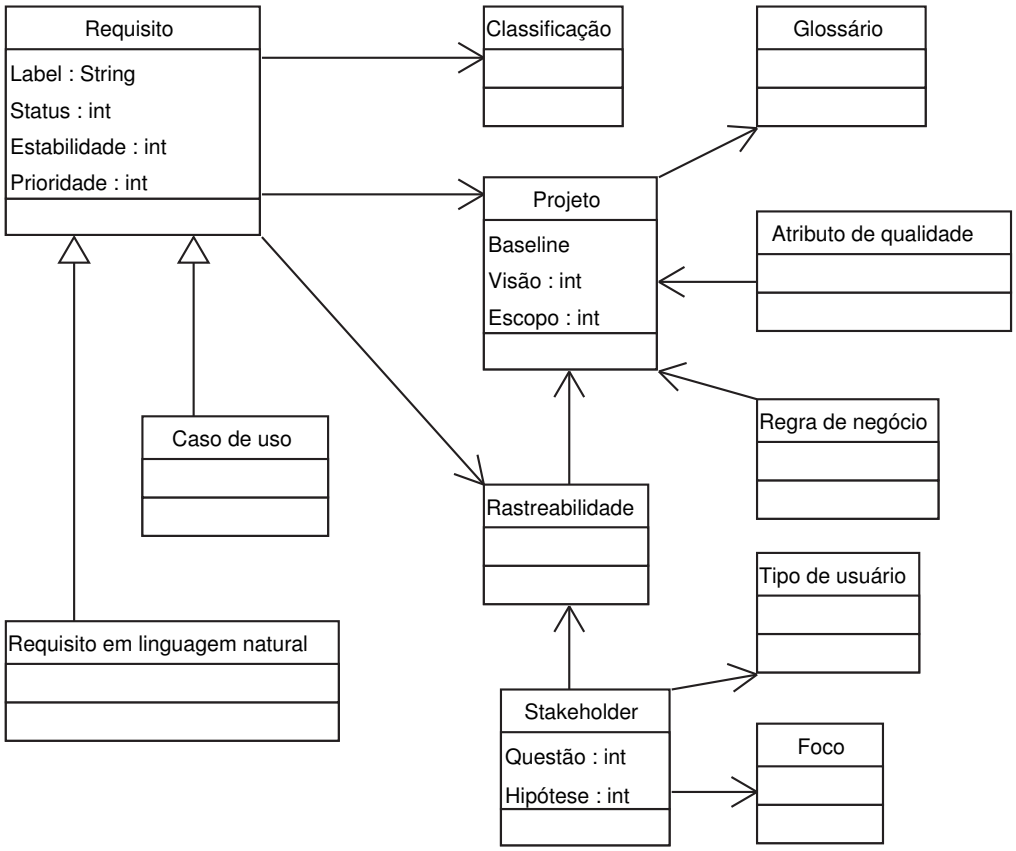

Figura 3.2: Modelo lógico da RQML (GUDGEIRSSON, 2000)

3.3 Considerações Finais

A partir do levantamento bibliográfico sobre a utilização de Hipermídia para suporte à Engenharia de Requisitos, observam-se três tipos distintos de trabalhos. Um grupo consiste na aplicação das idéias de hipermídia na engenharia de requisitos e seus possíveis benefícios. Essas pesquisas são oriundas do início da década de 90, nos primórdios da Internet, quando os sistemas hipermídia de grandes dimensões ainda estavam longe da realidade.

Posteriormente, surgem as primeiras aplicações: a XMLC, o RETH e o HERE, que se usufruem da Internet e das vantagens sugeridas nos primeiros trabalhos.

Finalmente, seguindo uma tendência de interoperabilidade entre aplicações, encontram-se trabalhos como o REM e RQML, que definem hiperdocumentos através de padrões da indústria para sistemas Web: XML, XPath, XPointer, XLink, dentre tantos outros padrões desenvolvidos pela W3C (W3C, 1994).

Comparou-se, ao final do estudo da literatura, as diferentes ferramentas hipermídia, de acordo com as funcionalidades esperadas de sistemas hipermídias, julgadas adequadas para a Engenharia de Requisitos, e a criação de hiperdocumentos que acomodem, de maneira flexível e extensível, os artefatos gerados pelas diferentes técnicas de Engenharia de Requisitos. As características analisadas foram:

Ligações globais: as ligações globais, criadas automaticamente preferencialmente, são um importante mecanismo para relacionar assuntos ortogonais aos diversos métodos de engenharia. No RETH, elas são empregues para relacionar termos encontrados no texto a um dicionário.

Metadados em ligações: metadados em ligações permitem adicionar informações aos relacionamentos: questões levantadas e tipo de ligação de que se trata. Na XMLC, apesar de existirem vários tipos de ligações, os metadados não são utilizados na implementação.

Integração de e para outros métodos: a integração deve ser observada sob dois aspectos: o da utilização de artefatos gerados por outros métodos e o da possibilidade de uso do próprio hiperdocumento em outras etapas da engenharia.

Gerenciamento de configuração: fator importante para gerenciar os requisitos e que não é abordado em nenhuma ferramenta. A REM e a RQML permitem identificar a versão dos componentes do hiperdocumento, mas estão muito além de fornecerem controle de versão, quanto mais gerenciamento de configuração.

Verificação de atributos de qualidade: algumas características desejáveis no documento de requisitos podem ser verificadas automaticamente. A utilização de documentos XML validados, no caso do REM e RQML, contribui para garantir a completitude do documento. Entretanto, muitos outros atributos podem ser verificados. Os trabalhos de Durán (DURÁN, 2000; DURÁN; RUIZ; TORO, 2001), com sua ferramenta REM, são os únicos nesse sentido, verificando a presença de ambigüidades e completitude, dentre outros.

A tabela 3.1 relaciona as técnicas revisadas neste capítulo e as características adicionais de sistemas hipermídia para apoio à engenharia de requisitos. As características presentes estão marcadas com um **X** e aquelas não informadas ou ausentes estão identificadas com um hífen.

Características	HERE	RETH	XMLC	RQML	REM
Ligações globais	-	X	-	-	-
Metadados nas ligações	-	-	-	-	-
Integração de outros métodos	-	-	X	X	X
Integração em outros métodos	-	-	X	X	X
Gerenciamento de configuração	-	-	-	-	-
Verificação de atributos de qualidade	-	-	-	-	X

Tabela 3.1: Características em hipermídia para apoio à engenharia de requisitos

A REM é a mais completa das ferramentas avaliadas. A capacidade de geração de documentos XML facilita não apenas a integração de e em outros métodos, mas também a

aferição da qualidade do documento, por intermédio de transformações XSLT. Esse mesmo arquivo poderia ter suas versões gerenciadas por um sistema qualquer de *Version Control Management* (VCM) baseado em arquivos. As ligações globais e com metadados seriam possíveis com outras técnicas compatíveis com o XML, como XLink e *Resource Description Framework* (RDF). No entanto, cabe ressaltar que a REM não é uma ferramenta Web e, para a criação de hiperdocumentos em software livre, o uso de ferramentas Web e a adoção de padrões abertos, como os definidos pela *World Wide Web Consortium* (W3C)[5], são essenciais.

[5] As tecnologias definidas pela W3C beneficiam-se do suporte dos navegadores Web, o que viabiliza o desenvolvimento de aplicações hipermídia sofisticadas e acessíveis.

Capítulo

4

Wikis

A *World Wide Web* (WWW) surgiu para facilitar o acesso às informações relativas aos experimentos conduzidos nos laboratórios do CERN (BERNERS-LEE, 1989). O sucesso da iniciativa possibilitou a Internet que a maioria das pessoas vêem hoje, com uma míriade de informações e serviços disponíveis e interligados, organizados em *sites* hospedados por todo o mundo. A simplicidade da HTML, linguagem utilizada para criação e interligação de hiperdocumentos para a WWW, contribuiu muito para essa popularização. No entanto, a crescente complexidade dos hiperdocumentos tornou onerosa a criação e manutenção dos mesmos.

A comunidade de padrões de software, em meados da década de 90, criou um *site* que funcionava como um repositório para descrição de padrões: o *Portland Pattern Repository*[1]. A natureza do conteúdo disponibilizado, constituída de colaborações enviadas por pesquisadores do mundo inteiro, e a experiência positiva da utilização da ferramenta *HyperCard*[2] (ATKINSON, 1987) para a documentação de padrões motivaram Cunnigham a desenvolver uma ferramenta Web para complementar o repositório: a WikiWikiWeb (CUNNIGHAM, 1995).

O objetivo da WikiWikiWeb era fornecer um ambiente simples e ágil para edição colaborativa de hiperdocumentos. O nome "WikiWiki" é proveniente do havaiano *wikiwiki*, cujo significado é rápido, depressa. As aplicações Web que se assemelham a WikiWikiWeb são classificadas como *wikis*, em homenagem ao nome do primeiro software dessa categoria já construído.

[1] O *site* do *Portland Pattern Repository* encontra-se hospedado em `http://c2.com/ppr/`.

[2] HyperCard foi um aplicativo hipermídia distribuído com o sistema operacional MacOS, em computadores Macintosh.

Inicialmente voltada para a manutenção de um repositório de páginas, que compõe um hiperdocumento, hoje as ferramentas *wiki* acumulam diversas funções. A MediaWiki (MANSKE, 2002) é utilizada na WikiPedia (`http://www.wikipedia.com`), uma enciclopédia gratuita composta por milhares de artigos escritos pelos seus próprios usuários e considerada a maior aplicação *wiki* pública do mundo. A *TWiki* (THOENY, 1998), além de suas características *wiki*, possui diversas extensões voltadas para aplicações de *groupware*, tais como calendário, acompanhamento de projetos, etc. Em Engenharia de Software, existem a *Trac* (BORGSTRÖM et al., 2004), um gerenciador de projetos de software integrado a um gerenciador de configuração, e a *Fitnesse* (Object Mentor, Inc, 2003), voltada para a automatização de testes de aceitação. A *Trac* permite a criação de relacionamentos de páginas *wiki* para *bugs* e itens de configuração. A *Fitnesse* possibilita a definição de casos de teste para aceitação de um software e a criação de páginas que executam automaticamente esses casos de teste e mostram os respectivos resultados.

Apesar da diversidade das implementações, as *wikis* ainda preservam um conjunto de princípios. A próxima seção descreve essas características, discutindo requisitos desejáveis para as aplicações que implementem esse modelo. A seguir, analisam-se as ferramentas atualmente em uso e define-se uma estratégia para a implementação de uma *wiki* voltada à Engenharia de Requisitos.

4.1 Princípios

Cunningham estabeleceu um conjunto de princípios para a *WikiWikiWeb* (CUNNINGHAM, 2005), estruturados ao redor de um único princípio, considerado fundamental: confiança. Crer nas pessoas, para a adição e alteração de páginas, e no processo, na fé de que uma base de conhecimento, em forma de hiperdocumento, surgirá. Neste alicerce, ele constrói o seguinte conjunto de princípios:

Aberta: Qualquer usuário poderá editar qualquer página.

Incremental: Uma página poderá referenciar outra página, mesmo que essa página não exista.

Orgânica: A estrutura e conteúdo do *site* estarão abertos para edição e evolução.

Mundana: Um número reduzido de convenções textuais proverá acesso para as marcações de página mais úteis.

Universal: O mecanismo de edição e organização serão os mesmos.

Evidente: O conteúdo apresentado deverá sugerir a formatação de entrada dos dados.

Unificada: Os nomes das páginas estarão em um espaço de nomes (*namespace* — conjunto de nomes que identificam as páginas) único.

Precisa: Os nomes das páginas serão definidos de maneira precisa, evitando o conflito com outras páginas, e compostos por substantivos.

Tolerante: O comportamento interpretável será preferível ao uso de mensagens de erro.

Observável: Atividades serão observáveis e revisáveis por qualquer pessoa.

Convergente: Duplicações de conteúdo serão desencorajadas e removidas através da busca e referência a materiais similares ou relacionados.

A partir desses princípios, é possível analisar os requisitos elementares de aplicações *wiki*.

4.2 Discussão Sobre os Princípios

Em aplicações *wiki*, toda página é identificada por seu nome. Não existe uma estrutura, tal como em sistemas de arquivos. O conceito de página inicial é apenas uma convenção adotada, tal quando realiza-se uma requisição à raiz de um *site* e assume-se que se deseja o arquivo `index.html`. A resposta a cada requisição conterá o conteúdo da página solicitada, caso a mesma seja encontrada, ou o redirecionamento para a criação da referida página. Na prática, nunca existirá a situação de uma página não encontrada, tão comum na Web. Satisfaz-se assim os seguintes princípios:

Incremental: O documento inteiro é criado com a filosofia incremental, iniciando com um hiperdocumento vazio e criando-se as demais páginas sob demanda.

Unificada: Todas as páginas estão em um mesmo espaço. Não é possível que duas requisições, solicitando nomes iguais, resultem em páginas diferentes.

A aplicação também não verifica a procedência da requisição. De fato, a intenção é permitir que qualquer pessoa possa acessar o hiperdocumento, alterando-o, se conveniente, para compartilhar um pouco de seu conhecimento. Não existe pré-condição alguma para a edição do conteúdo da página, tal como revisões, censura, pedidos para revisão e publicação:

Aberta: A aplicação permite que qualquer usuário acesse ou edite uma página.

A requisição a uma página não existente redireciona automaticamente o usuário para a função de edição de uma página, mas esse não é o único meio de ativar essa função. De fato, toda página do hiperdocumento é editável, a qualquer momento:

Orgânica: A edição de qualquer página está disponível a qualquer pessoa.

A linguagem utilizada na edição busca ser o mais natural possível: um parágrafo é um conjunto de linhas em seqüência, sendo cada parágrafo separado por uma linha em branco; citações são demarcadas por aspas; linhas horizontais por três hifens consecutivos; o título de uma seção por um linha procedida por uma linha contendo três hífens em seqüência. Basicamente, os mesmos recursos utilizados na época das máquinas datilográficas para destaque de texto e identificação de formato estão disponíveis para a edição de uma página *wiki*.

Mundana: As convenções textuais são poucas, limitando-se a usos de caracteres extras imediatamente antes ou depois do texto a ser dermarcado. Além disso, as convenções textuais para edição de páginas *wiki* são semelhantes àquelas adotadas em tecnologias antecessoras, às quais as pessoas já estão acostumadas.

Evidente: Os recursos de formatação disponíveis para edição são simples e comuns a vários meios de escrita. A associação entre o conteúdo apresentado e as instruções para realizá-lo torna-se assim clara.

Até o presente momento, o hiperdocumento não possui nenhuma estrutura: todas as páginas estão no mesmo espaço, mas sem as ligações necessárias para o estabelecimento de uma ordem. Regras para criar esses relacionamentos são necessárias. A primeira: toda página é nomeada com dois ou mais substantivos, eliminando qualquer outro tipo de termo entre eles. Por exemplo, "especificação de requisitos" seria escrito como "especificação requisitos". Entretanto, esse esquema não é muito eficiente, tanto do ponto de vista computacional quanto humano. Computacionalmente, identificar o tipo de um termo de uma frase é uma atividade cara: envolveria a utilização de analisadores de linguagem natural, que, atualmente, são propensos a erros. E, quanto à leitura do texto por uma pessoa, tal estrutura não seria facilmente identificável, não se destacando do restante da página ou confundindo-se com erros na escrita. A solução é adicionar mais uma regra: os substantivos deverão ser capitalizados, sem espaços entre as palavras. Esse modo de escrita é denominado CamelCase[3]. Por exemplo, "especificação requisitos" seria definido como "EspecificaçãoRequisitos". O algoritmo necessário para identificar essa construção é simples e, dado o contraste desse formato quando comparado ao restante do texto, torna-se evidente, ao usuário, de que se trata de um termo diferenciado.

Mundana: A única convenção pouco usual para edição é aquela que é efetivamente o diferencial de uma página *wiki*: a criação de um relacionamento para com outra página. Apesar disso, a sua definição é suficientemente simples para que qualquer pessoa entenda.

[3] Mais detalhes sobre o CamelCase, sua história e utilização na computação podem ser encontrados em `http://en.wikipedia.org/wiki/CamelCase`.

Evidente: As ligações são facilmente identificáveis e a maneira como são apresentadas mostram exatamente como elas são definidas na edição.

Universal: A organização do hiperdocumento é definida pelos relacionamentos entre suas páginas. Por sua vez, tais relacionamentos são definidos durante a edição das páginas, de modo transparente.

Precisão: A função gramatical de substantivos é denominar coisas, pessoas e lugares. Logo, os nomes são tão precisos quanto aqueles utilizados no mundo real. A eliminação de termos acessórios e o emprego de um gênero, número e grau padrões para todos os nomes evitam o conflito entre páginas.

Orgânica: A estrutura do hiperdocumento é definida pelos relacionamentos definidos pelas páginas (e a edição está aberta a todos).

Uma característica chave da *wiki* é a liberdade e transparência que ela oferece aos seus usuários. Todo o seu conteúdo é público e imediatamente disponível. Associando essa característica com as convenções para definição de nomes de páginas, a existência de conteúdo duplicado é desencorajado:

Observável: Todas as alterações são imediatamente publicadas.

Convergência: Conteúdos similares geralmente possuem partes de seu nome em comum ou diretamente relacionados, com uma probabilidade maior de encontrarem-se citados em uma mesma página. Além disso, dada a facilidade em relacionar diferentes páginas, torna-se mais conveniente aos autores dividir uma página extensa em diversas páginas menores, porém interligadas.

Finalmente, é preferível uma página com conteúdo incompleto ou incorreto (por exemplo, com referências definidas em uma sintaxe que não permite a geração automática de relacionamentos) do que a ausência da mesma. Posteriormente, o seu conteúdo poderá ser corrigido e aperfeiçoado:

Tolerante: Erros são tolerados sempre que possível, evitando-se a perda de trabalho do autor dada a possibilidade de posterior correção do conteúdo.

O seguimento de todos esses princípios, fundamentados no estabelecimento de uma relação de confiança entre os usuários e a *wiki*, propiciam a criação de complexos hiperdocumentos. Um exemplo bem sucedido é a *wiki* primordial, em uso até os dias atuais em `http://c2.com/cgi/wiki`. A próxima seção apresenta um estudo sobre outras implementações e suas diferenças quanto ao modelo ideal aqui descrito.

4.3 Implementações

A *wiki* idealizada apóia-se fortemente na confiança dos usuários entre si. Infelizmente, a Internet está exposta a qualquer tipo de usuário, inclusive os mal-intencionados. Os primeiros softwares que concretizaram os princípios de *wikis* enfrentaram sérios problemas com a alteração indevida de suas páginas, ocasionando a perda de dados e, principalmente, a quebra da corrente de confiança existente entre os usuários legítimos do hiperdocumento. Sem esse último elemento, a *wiki* perece. Tornou-se imperativo adicionar mecanismos que protegessem a idoneidade das ferramentas.

Uma profusão de implementações de *wikis*, adequadas às adversidades da Internet, surgiram na comunidade de software livre. Em um recente levantamento, foram identificadas 79 ferramentas (SILVA, 2005) e esse número continua crescendo.

Para a implementação da Wiki/RE, algumas *wikis* foram escolhidas para estudo, com o objetivo de facilitar a definição dos requisitos e implementação da Wiki/RE. Em meados de 2003, realizou-se um levantamento de *wikis* livres, posteriormente avaliadas de acordo com critérios apropriados.

O modelo de desenvolvimento (livre) difere daquele utilizado nos softwares tradicionais, com uma exposição maior dos aspectos de engenharia se comparado com o uso efetivo das aplicações pelos usuários (CROWSTON; ANNABI; HOWISON, 2003). A aplicação de técnicas usuais de avaliação de softwares aos softwares livres não representaria uma medida correta do sucesso dos projetos. Logo, a disponibilidade dos artefatos gerados durante o desenvolvimento (código, relatos de erro, emails com o *design rationale* de decisões de projeto e implementação) deve ser utilizada para corrigir essa distorção.

Crowston, Annabi e Howison (2003) propuseram um conjunto de indicadores para avaliar o sucesso de ferramentas de software livre sob sete pontos de vistas: qualidade do sistema, satisfação do usuário, uso, impacto no indivíduo ou na organização, produtos do projeto, processo de desenvolvimento, benefícios gerados aos participantes dos projetos. A tabela 4.1 relaciona os indicadores a esses pontos de vista.

Algumas medidas sugeridas por Crowston, Annabi e Howison (2003), conforme os autores afirmam, não podem ser obtidas diretamente dos dados livremente disponíveis pelos projetos ou com a precisão necessária (oportunidade de trabalho, salário, reputação dos desenvolvedores, geração de conhecimento a nível individual, implicações econômicas, quantidade de usuários, satisfação do usuário). As outras medidas também não são de obtenção trivial, dependendo da maneira com que os dados são disponibilizados pelos projetos sob análise. Em posterior trabalho, Crowston et al. (2004) analisaram quatro medidas para mensurar o sucesso (quantidade de membros, nível de atividade, tempo necessário para correção de erros e quantidade de *downloads*), obtidas sistemática e automaticamente de um repositório de software livre (`http://www.sourceforge.net`). Essas medidas também são

Ponto de vista	Indicadores
Qualidade do sistema	Qualidade do código (compreensibilidade, completeza, concisão, portabilidade, consistência, manutenibilidade, testabilidade, usabilidade, confiabilidade, estruturação, eficiência), qualidade da documentação
Satisfação do usuário	Avaliação realizada pelo usuário, opiniões em lista de discussão, levantamentos
Uso	Quão popular é o softwar, quantidade de usuários, quantidade de downloads, disseminação por meio de distribuições, popularidade (quantidade de acessos) da página principal do projeto, dependências para outros pacotes, reutilização de código.
Impactos nos indivíduos e organizações	Implicações econômicas
Produtos gerados pelo projeto	Mudança de estágios de desenvolvimento (alpha para beta, beta para estável), alcance das metas estabelecidas, satisfação do desenvolvedor
Processo	Quantidade de desenvolvedores, nível de atividade (contribuições dos membros do projeto: código, relato de erros, etc), tempo entre lançamento de novas versões, tempo para resolver relatos de erros ou implementar novas características no software.
Benefícios para os membros do projeto	Oportunidades de trabalho e salário para os indivíduos, reputação individual, geração de conhecimento.

Tabela 4.1: Medidas e indicadores sugeridos para a avaliação do sucesso de projetos de software livre. Fonte: Crowston, Annabi e Howison (2003).

facilmente calculáveis para os projetos de *wikis* livres. Para o levantamente de *wikis* livres realizado, determinou-se que a qualidade do código e da documentação também seriam avaliadas, bem como algumas funcionalidades consideradas essenciais para ferramentas *wikis*. A lista abaixo resume todas as medidas coletadas:

Software livre: A ferramenta deverá ser um software livre, disponibilizada preferencialmente por uma licença compatível com a GPL.

Qualidade do código: Quantidade de linhas de código, documentação da *Application Program Interface* (API), testabilidade (presença de casos de testes especificados e cuja execução seja automatizada por ferramentas como jUnit), portabilidade (linguagem de implementação que facilite a portabilidade entre sistemas, tais como a linguagem Java), clareza do código, manutenibilidade (padrão de codificação, arquitetura bem definida, reutilização de código de outros projetos).

Uso: Dependências para e de outros softwares, facilidade de instalação[4].

[4] Uma prática comum em software livre é o lançamento freqüente de novas versões (RAYMOND, 1997). A intenção é permitir que o software seja executado e testado o mais breve possível. Para que isso se concretize, é necessário que os procedimentos para instalação e configuração do software estejam devidamente documentados e, quando possível, automatizados.

Produtos do projeto: Alcance de metas planejadas, lançamento de versões alpha, beta e estáveis.

Processo de desenvolvimento: Quantidade de desenvolvedores, tempo entre lançamento de versões, verificação quanto a utilização de ferramentas de controle de versão e controle de alterações, seguindo o processo de software livre caracterizado por Reis (2003).

Wikis:

Seguimento dos princípios *wiki*: O ideal é que a ferramenta atenda aos princípios explicados na seção 4.1. A linguagem utilizada na definição das páginas e o mecanismo de ligações são os itens de maior interesse quanto a esse quesito.

Expansibilidade: Possibilidade da adição de novas funcionalidades para edição ou apresentação dos dados contidos na página.

Controle de versão: Várias ferramentas implementam controle de versão para as páginas, possibilitando a recuperação de versões anteriores para a eventualidade de um usuário mal-intencionado alterar indevidamente uma página.

Adotou-se, como critério de pré-seleção, a linguagem utilizada na implementação. Definiu-se Java com a linguagem preferencial devido a facilidades para desenvolvimento de aplicações Web e portabilidade por ela oferecida. Foram escolhidas, de acordo com a popularidade da ferramenta, aferida por meio do *site* `http://freshmeat.net` à epoca do levantamento, três *wikis* Java para estudo: a JSPWiki, a Snipsnap e a VeryQuickWiki. Analisou-se ainda a ferramenta CoTeia, desenvolvida em *PHP*, que foi utilizada para a documentação do desenvolvimento da Wiki/RE. O autor deste trabalho, no decorrer do mestrado, assumiu o desenvolvimento da ferramenta CoTeia, conforme descrito na subseção 4.3.4.

4.3.1 JSPWiki

A JSPWiki foi criada por Jalkanen (JALKANEN, 2001). Foram analisadas as versões 2.0.52, 2.1.91 (obtida do repositório CVS do projeto em 21/01/2004) e a 2.2.0.

A instalação é simples e bem documentada. O único requisito é um *container* Web, para o qual escolheu-se o Apache Tomcat. A configuração é feita na própria aplicação, sem precisar editar arquivos manualmente.

A linguagem utilizada nas páginas é exclusiva da JSPWiki e a criação de referências segue uma regra dessa linguagem (ao invés de ser uma convenção, como é comum em outras *wikis*). A figura 4.1 contém a página de entrada da ferramenta (com seus *links* sublinhados a fim de facilitar a identificação) e, no canto inferior direito, o formulário para edição daquela página.

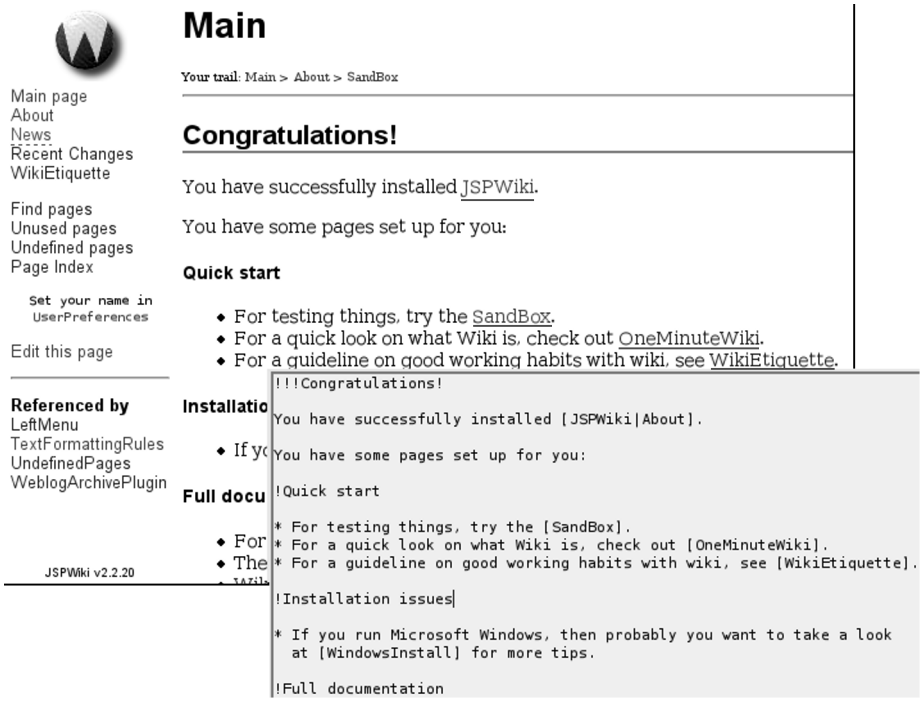

Figura 4.1: Tela inicial da JSPWiki e seu formulário para edição

As páginas são armazenadas em disco, utilizando mecanismos de controle de versão. São suportados o *Revision Control System* (RCS) e um mecanismo proprietário baseado em cópias integrais dos arquivos (ao invés do armazenamento apenas das diferenças entre as versões). A JSPWiki é expansível por meio de *plugins* especificados em Java e declarados nas páginas através de uma construção específica da linguagem de edição (por exemplo, `[{INSERT org.jspwiki.Image (filename=test.jpg)}]`). Além disso, é possível definir novos elementos de marcação e a respectiva regra de transformação a ser utilizada na apresentação do documento.

Ela foi desenvolvida em Java e disponibilizada sob a licença GPL, na versão 2.0.52, e *Lesser General Public License* (LGPL), na versão 2.1.91. O projeto utiliza um sistema de controle de versões para o gerenciamento do código e uma instância da ferramenta para o *site* do projeto. O código é claro e bem estruturado. Em sua versão 2.0.52, possui 15.019 *Lines of Code* (LOC); a 2.1.91 possui 21.462 LOC. Sua versão mais recente, a 2.2.0, contém 20.406 LOC.

4.3.2 Snipsnap

A Snipsnap pertence ao instituto *Fraunhofer Gesellschaft* e é desenvolvida por Jugel e Schmidt (2002). Foi analisada a versão 0.5a.

A instalação é simples e bem documentada. O único requisito é um *container* Web, para o qual escolheu-se o Apache Tomcat. A configuração é feita na própria aplicação, sem precisar editar arquivos manualmente.

A linguagem utilizada nas páginas segue a padrão das *wikis*. Referências a outras páginas podem ser criadas utilizando o convenção *CamelCase* ou construções específicas da linguagem. A figura 4.2 contém a página de entrada da ferramenta (com seus *links* sublinhados a fim de facilitar a identificação) e, no canto superior direito, o formulário para edição da página.

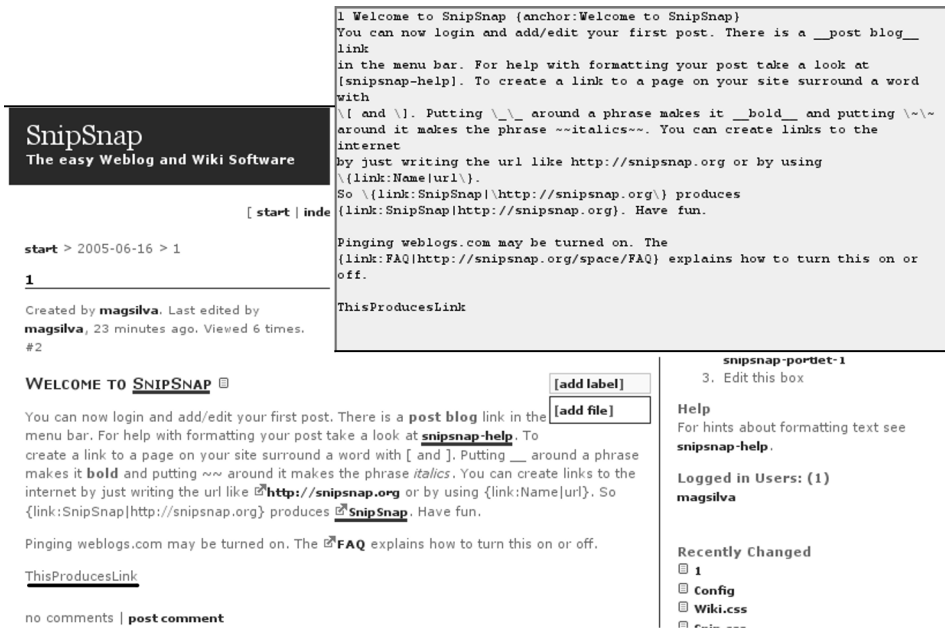

Figura 4.2: Tela inicial da Snipsnap e seu formulário para edição

As páginas são armazenadas em arquivos, sem o emprego de mecanismos de controle de versão. A Snipsnap é expansível por meio de macros e filtros. As macros seriam como *plugins*, disponibilizados globalmente pela aplicação. Os filtros são definidos nas próprias páginas. A criação e o uso de macros e filtros estão bem documentados, constituindo elemento essencial da arquitetura da aplicação.

Ela foi desenvolvida em Java e disponibilizada sob a licença GPL. O projeto utiliza um sistema de controle de versões para o gerenciamento do código (*Concurrent Versions System* (CVS) e, recentemente, *Subversion*) e uma instância da ferramenta para o *site* do projeto. A arquitetura e o projeto da ferramenta são bem documentados, porém o código não acompanha a qualidade da documentação. Foram contabilizadas 23.343 LOC.

4.3.3 VeryQuickWiki

A VeryQuickWiki foi criada por Cronin (CRONIN, 2000). As versões analisadas foram a 2.6.3, obtida a partir do repositório CVS em 21/01/2004, e a 2.7.1, sua última versão estável, lançada em junho de 2004.

A instalação da versão 2.6.3 não é simples. Existe uma dependência quanto a ferramenta *JBuilder*[5] (provavelmente o ambiente utilizado para o desenvolvimento). Nesse tocante, a versão 2.7.1 evoluiu muito, requerendo apenas a cópia da aplicação para um *container* Web (como o Apache Tomcat) e a alteração da configuração pela própria interface da ferramenta.

A linguagem utilizada nas páginas segue o padrão das ferramentas *wiki*. Referências a outras páginas são criadas exclusivamente através da convenção *CamelCase*. A figura 4.3 contém a página de entrada da ferramenta (com seus *links* sublinhados a fim de facilitar a identificação) e, no canto inferior direito, o formulário para edição daquela página.

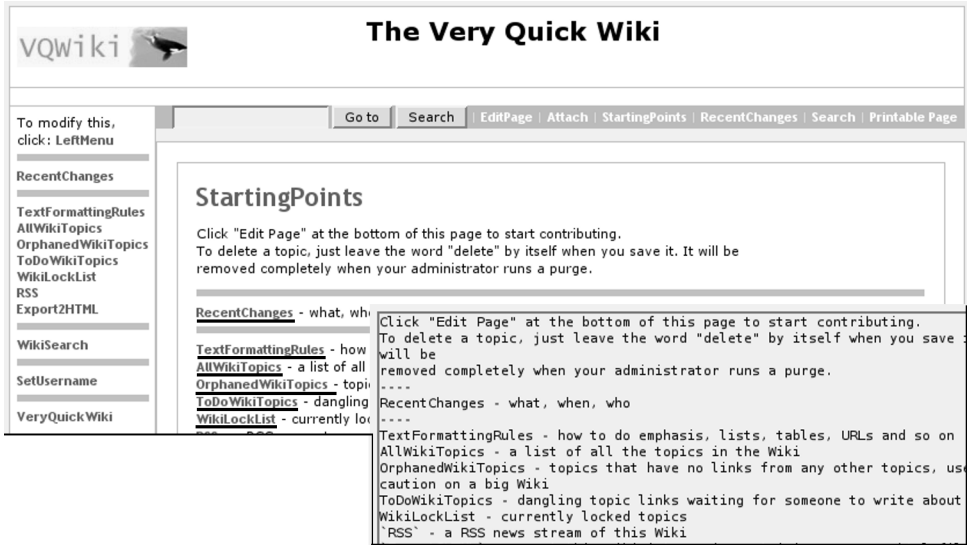

Figura 4.3: Tela inicial da VeryQuickWiki e seu formulário para edição

Quanto às funcionalidades, a evolução existente entre as versões 2.6.3 e 2.7.1 é notável. A versão antiga não suportava controle de versões e tampouco era extensível. Sua última versão já oferece ambas as funcionalidades. As páginas são armazenadas em disco ou em bases de dados, utilizando um mecanismo de controle de versões proprietário (baseado na manutenção da cópia integral das páginas). Ela é expansível por meio de plugins, desenvolvidos em Java e descritos por um documento XML, facilitando assim o acréscimo de novas funções à ferramenta.

Ela foi desenvolvida em Java e disponibilizada sob a licença LGPL. O projeto utiliza um sistema de controle de versões para o gerenciamento do código (CVS) e uma instância da ferramenta para o *site* do projeto. Não se encontram disponíveis documentos sobre a arquitetura e o projeto da ferramenta, porém o código é simples e bem documentado. Foram contabilizadas 22.260 LOC para a versão 2.7.1, o que impressiona se comparado com a contagem da versão 2.6.3, que foi de 3.259 LOC.

[5] O *JBuilder* é um ambiente de desenvolvimento para aplicações Java, desenvolvido pela empresa *Borland* (`www.borland.com/jbuilder/`)

4.3.4 CoTeia

A CoTeia é uma ferramenta colaborativa para edição desenvolvida por Arruda (JR; IZEKI; PIMENTEL, 2002). Ela é uma re-escrita da ferramenta Swiki (GUZDIAL, 1999), também conhecida como CoWeb.

Ela difere das outras ferramentas *wiki* por utilizar a linguagem *eXtensible HyperText Markup Language* (XHTML) para edição, com algumas construções adicionais, em XML, para a criação de relacionamentos entre páginas, arquivos e anotações. A escolha do HTML permite o emprego de ferramentas de edição de páginas *Web*, amplamente disponíveis no mercado. Além disso, existem programas que habilitam a edição de formulários de páginas HTML no modo *What You See Is What You Get* (WYSIWYG) — *TinyMCE* (Moxiecode Systems AB, 2005), *FCKeditor* (KNABBEN, 2005) —, facilitando ainda mais a edição das páginas. Em modo experimental, a CoTeia permite atualmente o uso do *TinyMCE*.

A criação de *links* na CoTeia é possível através de uma estrutura da linguagem, o marcador `<lnk>`. Originalmente ela não suportava a criação de relacionamentos através do *CamelCase*. Essa limitação advinha da utilização de XML e a maneira como as ligações eram computadas. Atualmente, a CoTeia suporta a criação automática de ligações através de *CamelCase*.

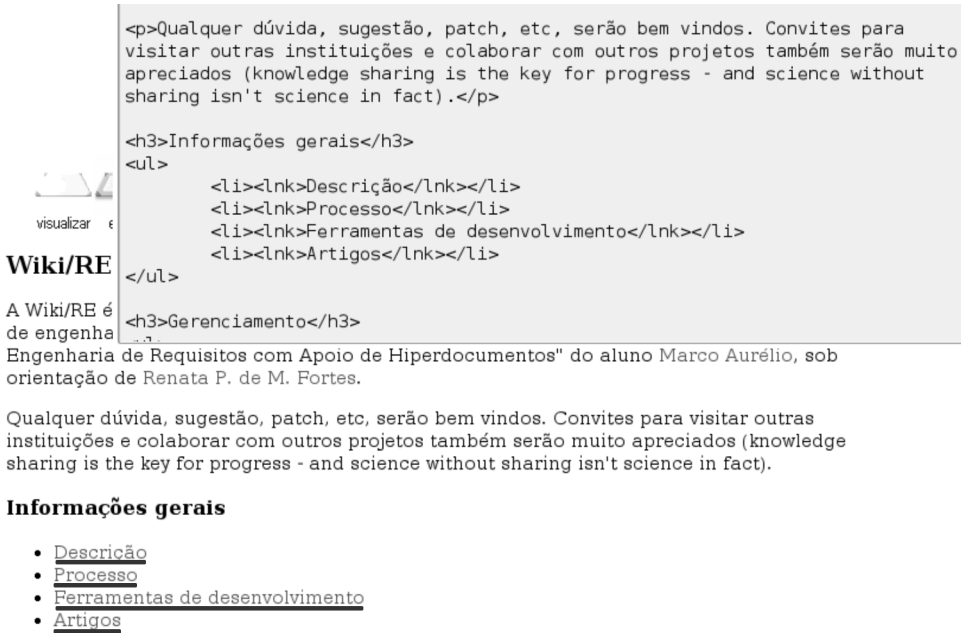

Figura 4.4: Tela inicial de uma *swiki* da CoTeia e seu formulário para edição

Cada página é armazenada em uma base de dados MySQL e um documento XML. Esse documento, quando requisitado pelo usuário para visualização, é apresentado, aplicando-se uma transformação XSLT. A transformação padrão produz um documento XHTML, mas é possível a escolha de outros formatos através da passagem de parâmetros na requisição *Hy-*

pertext Transfer Protocol (HTTP). As páginas XHTML geradas são colocadas sob controle de versão, utilizando-se o CVS.

A CoTeia teve origem acadêmica e está em uso no Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo/Campus de São Carlos (USP/SC) desde 2001, gerenciando o conteúdo utilizado no ensino de disciplinas para turmas de graduação e pós-graduação. Em 02 de fevereiro de 2004, ela foi lançada como software livre, sob a licença GPL, e administrada pelo autor desta dissertação (SILVA, 2004). Em junho do mesmo ano, foi registrada como um projeto da Incubadora FAPESP (`http://incubadora.fapesp.br/projects/coteia`), centralizando o desenvolvimento e aumentando a visibilidade da ferramenta.

Em sua primeira versão pública, várias falhas de segurança foram identificadas, além de erros no tratamento do conteúdo das páginas. A utilização da CoTeia para documentação do desenvolvimento da Wiki/RE requereu a correção desses problemas. Nesse processo, todo o código foi revisado. A correção de erros no mecanismo de geração de ligações e no tratamento dos dados exigiu a re-implementação do núcleo da aplicação. A desativação da diretiva "register_globals" do *PHP*, considerada uma boa prática de segurança (ACHOUR et al., 2005), levou à re-escrita de todas as páginas da ferramenta. Nesse processo, foram introduzidas melhorias na usabilidade da CoTeia, sugeridas durante a criação do hiperdocumento do processo de desenvolvimento da Wiki/RE e devidamente registradas e acompanhadas através da ferramenta *Bugzilla*. Apesar da abrangência das alterações, preservou-se a compatibilidade com as bases de dados de versões anteriores da CoTeia, conforme demonstrado na migração de uma instância da ferramenta utilizada no ICMC da USP/SC: as únicas alterações necessárias foram quanto a erros no conteúdo das páginas que a CoTeia anterior não detectava e que a nova versão prontamente identifica.

Atualmente, o código da CoTeia encontra-se bem documentado, assim como o modelo da ferramenta. Devido à amplitude das alterações realizadas, o autor deste trabalho possui pleno domínio sobre a ferramenta. O desenvolvimento agora concentra-se, por parte do mantenedor, na correção de falhas, delegando-se a adição de novas funcionalidades a projetos desenvolvidos por alunos da instituição.

4.3.5 Comparação das *Wikis* Analisadas

Estudadas individualmente as ferramentas *wikis*, procedeu-se uma análise comparativa das mesmas. A ferramenta CoTeia não foi considerada, dado o envolvimento do autor deste trabalho quanto ao desenvolvimento da mesma. A análise qualitativa de vários itens (portabilidade, manutenibilidade, instalação, seguimento dos princípios e expansibilidade) seria de pouca validade nesse cenário.

A tabela 4.2 resume os resultados da análise realizada sobre as ferramentas. Para efeito da comparação, utilizou-se a última versão de cada ferramenta analisada. As linhas

de código foram medidas com o auxílio da ferramenta "sloccount" (WHEELER, 2001). A portabilidade foi verificada quanto aos requisitos para uso, instalação e desenvolvimento da ferramenta. A manutenibilidade considerou a existência de documentação de projeto e a qualidade do código-fonte. As dependências, indicadas entre parênteses, foram calculadas pela contagem de pacotes Java (JAR) necessários a execução de cada ferramenta. A instalação verificava os requisitos básicos de instalação (no servidor) e o grau de facilidade para a configuração. A freqüência dos lançamentos de versões foi observada pelos arquivos disponíveis em seus repositórios de *downloads*. O uso de ferramentas de controle de versão e controle de alteração foi constatado através dos *sites* de cada ferramenta. O seguimento dos princípios *wikis*, expansibilidade e controle de versão foram analisados através da documentação e do código-fonte disponíveis, bem como pela execução das ferramentas.

Os sinais de positivo indicam o atendimento a uma determinada característica, podendo existir um ou dois sinais para cada item (satisfatório ou bom, respectivamente). O sinal de negativo indica que a ferramenta não possui a característica em questão.

Características	**JSPWiki**	**Snipsnap**	**VeryQuickWiki**
Licença do software			
Software de código aberto	++	++	++
Compatível com a GPL	++ (LGPL)	++ (GPL)	++ (LGPL)
Qualidade do código			
Linhas de código	21.462	23.342	22.260
Portabilidade	++	++	+
Manutenibilidade	++	+	+
Uso			
Dependências	+ (13)	- (24)	- (21)
Instalação	++	++	++
Processo			
Lançamento constante de novas versões	++	+	+
Usa ferramenta de VCM	++	++	+
Usa ferramenta de CCM	-	++	+
Características *Wiki*			
Segue os princípios *wiki*	+	++	++
Expansibilidade	++	++	+
Controle de versão	++	-	+

Tabela 4.2: Análise das *wikis* selecionadas.

A comparação mostra um resultado favorável à JSPWiki, com 19 pontos (20 positivos, 1 negativo). Em segundo lugar encontra-se a Snipsnap, com 16 (18 positivos, 2 negativos) e, por último, a VeryQuickWiki, com 14 pontos (15 positivos, 1 negativo). A ausência de controle de versões de páginas da Snipsnap teve uma importante influência no resultado, bem como o seu modelo de desenvolvimento, com lançamento inconstante de novas versões.

A JSPWiki, apesar de sua equipe de desenvolvimento reduzida e sem dedicação exclusiva a mesma (ao contrário da Snipsnap), possui todas as características desejáveis de uma *wiki*, falhando apenas quanto ao modo utilizado para criação de ligações e a ausência de controle de alterações. A VeryQuickWiki, por sua vez, não possui destaques importantes, mas também não possui deficiências notáveis.

4.4 Análise

A utilização básica de *wikis* pode ser modelada como especificado no diagrama de caso de uso apresentado na figura 4.5. Os usuários criam e editam páginas, especificando os relacionamentos entre as páginas. Ao visualizar as páginas, os relacionamentos são mostrados ao usuário, permitindo a navegação no hiperdocumento. A realização desses casos de uso leva o usuário a alcançar as metas das *wikis*, que são:

- o compartilhamento de conhecimento (para os usuários que colaboram em sua criação);
- aquisição de conhecimento (para aqueles que apenas visualizam as páginas).

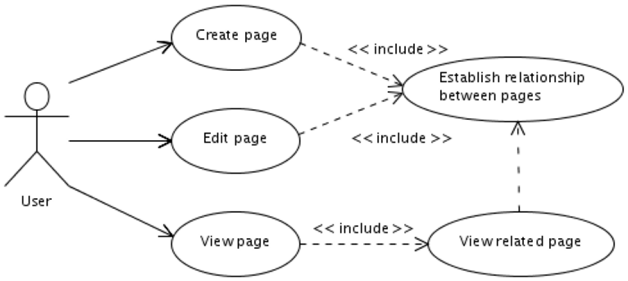

Figura 4.5: Casos de uso para uma *wiki*

Esse modelo atende satisfatoriamente aos princípios estabelecidos por Cunningham, mas não às adversidades inerentes ao ambiente em que ele é instanciado, a Internet. Trata-se de um modelo otimista, cujo emprego não é possível sem a adequada identificação de correção de suas deficiências. O diagrama de metas e *soft-goals* (figura 4.6) apresenta os princípios e adversidades, representados por *soft-goals*, com a identificação do impacto que elas causam entre si e nas metas.

As diferentes implementações de *wikis* buscam soluções que equilibrem as diferentes *soft-goals*, proporcionando o alcance das metas. Os principais problemas resumem-se em manter a rede de confiança entre os usuários (*trust*) e lhes fornecer os recursos de edição que precisam para transmitir seus conhecimentos.

Duas abordagens são utilizadas quanto à questão da confiança: o uso de controle de versão e o controle de acesso dos usuários. O armazenamento das diferentes versões de arquivos é uma atividade comum em engenharia de software: repositórios centralizados ou

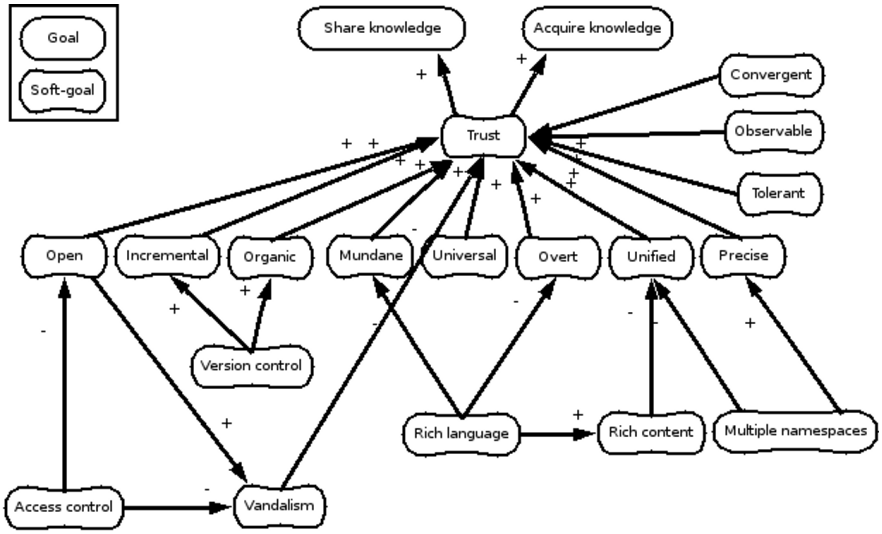

Figura 4.6: Metas e *soft-goals* das *wikis*

federados, acessados por diferentes usuários simultaneamente e, geralmente, associados a sistemas de gerenciamento de alterações. No entanto, nas *wikis*, os repositórios são sempre locais, mono-usuários e sem controle quanto às alterações. O único efeito prático desse modo de uso é a possibilidade de restaurar uma página quando detectada uma alteração indevida. Quando isolada a causa das modificações (por exemplo, bloqueando o acesso à *wiki* temporariamente, ou proibindo o acesso a partir de determinado endereço IP), a *wiki* pode ser restaurada ao seu estado normal.

Inicialmente, tal solução parece efetiva. No entanto, ela é apenas paliativa, não sanando efetivamente o problema. Dada a freqüência com que os atos de vandalismo podem ocorrer, trata-se apenas de uma questão de tempo e insistência para que os usuários do *site* percam a confiança no sistema e o abandonem. Conforme dito anteriormente, isso significa o fim de uma *wiki*.

A segunda solução, e mais efetiva, é a exigência da autenticação do usuário para que qualquer alteração seja realizada. Nada impediria pessoas mal-intencionadas de cadastrarem-se na ferramenta, mas é fato que tal requisito desencorajaria tais elementos. Para os demais usuários, essa condição é bem aceita, principalmente se claramente especificado que o motivo é a garantia da segurança.

O segundo desafio é oferecer uma linguagem rica, porém simples, para a edição de páginas. Os recursos básicos para edição de texto (negrito, itálico, cabeçalhos e título) são comuns entre as *wikis*. No entanto, a partir do momento em que se deseja utilizar estruturas mais complexas, como tabelas, e mídias diversas, como imagens, vídeos e fórmulas matemáticas, não se consegue definir um padrão entre as ferramentas. Não apenas isso, a própria definição desses elementos é pouco intuitiva, violando o princípio de evidência. Um exemplo disso é apresentado na figura 4.7: ela contém um trecho de uma tabela com a descrição das unidades do Sistema Internacional de Medidas (SI) (a tabela completa encontra-se em

`http://en.wikipedia.org/wiki/SI_base_unit`). Comparando-se com a figura 4.8, que especifica a figura 4.7 (com exceção da região borrada, na parte inferior direita), pode-se observar que o formato utilizado na especificação não lembra imediatamente uma tabela, a começar pelo cabeçalho, que configura parâmetros específicos de apresentação, até a definição de linhas e colunas, definidas separadamente em linhas.

SI Base units

edit

Name	Symbol	Quantity	Definition
kilogram	**kg**	Mass	The unit of mass is equal to the mass of the international prototype kilogram (a platinum-iridium cylinder) kept at the Bureau International des Poids et Mesures (BIPM), Sèvres, Paris (1st CGPM (1889), CR 34-38). Note that the kilogram is the only base unit with a prefix; the gram is defined as a derived unit, equal to

Figura 4.7: Parte de uma tabela com unidades do SI. Fonte: WikiPedia.

```
{| border="1" cellpadding="2" style="margin:0 auto; background:#f3f9ff;"
!  colspan="5" | <font size="+1">SI Base units</font><p align="right"><small>{{ed
SI_base_units|}}</small></p>
|-
|'''Name'''
|'''Symbol'''
|'''Quantity'''
|'''Definition'''
|-
|[[kilogram]]
|'''kg'''
|[[Mass]]
```

Figura 4.8: Código correspondente a parte de uma tabela com unidades do SI. Fonte: WikiPedia.

As primeiras *wikis* eram puramente textuais, não enfrentando tais problemas. Aliás, cabe lembrar que a primeira *wiki* criada permanece ativa até os dias atuais, utilizando somente texto como meio de comunicação com seus usuários. A razão desse sucesso é bem simples: apesar da aparência ser importante, o conteúdo é o essencial. No exemplo da tabela de medidas do SI, por exemplo, existia uma preocupação maior com a aparência do que com os dados em si (quando na visão de edição dos mesmos), esquecendo que uma sintaxe pouco intuitiva desencoraja o autor a transmitir seu conhecimento. No caso da *WikiPedia*, por exemplo, dado o volume de usuários e sua massa crítica, a probabilidade de um usuário dispor-se ao trabalho de utilizar soluções complexas para edição é razoável.

4.5 Modelo Proposto

O controle de acesso, apesar de contribuir negativamente com o princípio de abertura, contribui na elevação da confiança no uso de *wikis*. Observando a figura 4.6, é evidente o valor dessa contribuição, que afeta direta e positivamente as metas. O controle de versão também não encontra resistências: seu uso não causa impacto negativo.

No entanto, o uso de linguagens ricas para descrição das páginas, presente em todas as ferramentas *wiki* analisadas, representa uma questão delicada. Ela contribui para as metas através do conteúdo mais rico disponível aos autores, mas afeta o quão evidente e mundana é a edição das páginas. A sintaxe da linguagem em si não é um problema, dado que os usuários podem ignorar as estruturas mais complexas. O problema reside no fato de o único modo de inserir tal conteúdo é através da linguagem definida.

Uma outra questão é a maneira com que os conteúdos ricos (não-textuais) são referenciados nas páginas, utilizando uma sintaxe diferente daquela empregada para as páginas. Isso afeta diretamente o princípio de universalidade e do desenvolvimento incremental. Um ponto forte das *wikis* é a criação da referência antes da criação do conteúdo, estabelecendo, transparentemente, a rastreabilidade entre os diferentes elementos do hiperdocumento. Para os elementos em diferentes mídias, o fluxo é invertido, sendo necessário criar o elemento antes de estabelecer o relacionamento.

A proposta é disponibilizar diferentes modos para a criação de conteúdos para hiperdocumentos, preservando a universalidade. Para isso, propõe-se um modelo em que os elementos do hiperdocumento não são sempre páginas. A escolha do tipo do elemento é definida no momento de sua criação e os mecanismos para criação de referências aos elementos independem de seu tipo.

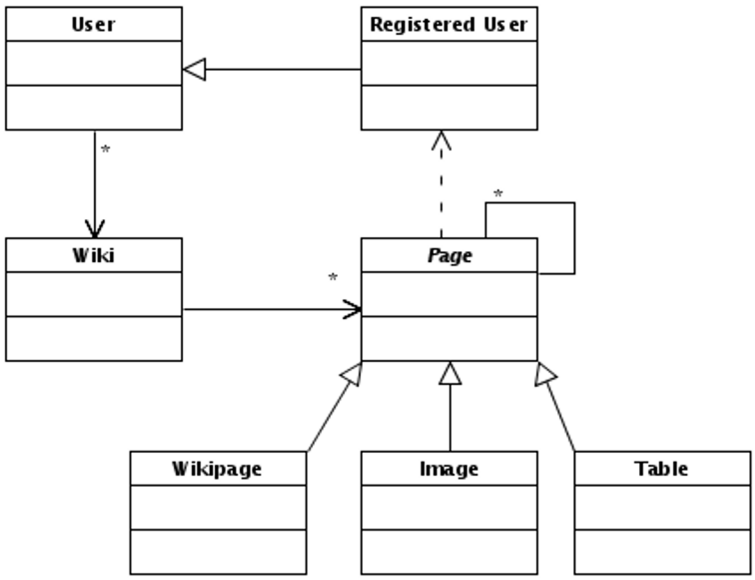

Figura 4.9: Modelo conceitual proposto para *wikis*.

Por exemplo, para a criação de uma tabela, o usuário poderia, ao invés de utilizar de uma intrincada sintaxe, como a mostrada na figura 4.8, criar uma referência para um elemento identificado como "Unidades de Medidas do SI". Ao acionar a ligação, seriam oferecidas ao usuário as opções quanto ao tipo de elemento desejado: uma página, uma figura, uma tabela, etc. Nesse caso, ele selecionaria "tabela" e lhe seria apresentado um ambiente apropriado à edição de tabelas.

Um argumento contrário a essa solução é que ela desfavoreceria o princípio mundano e evidente das *wikis*, utilizando uma linguagem para definição que não corresponderá a maneira como o elemento é apresentado. No entanto, os princípios não consideravam que as *wikis* seriam utilizadas para a criação de hiperdocumentos multimídia. Utilizar meios diferentes para descrever elementos diferentes é mais natural que utilizar um único meio para descrever todas as coisas.

4.6 Considerações Finais

Neste capítulo foram analisados os conceitos de *wikis* e como eles são atendidos por algumas implementações de *wikis* escritas em Java. Considerando os resultados obtidos, a melhor *wiki* para servir como base para a Wiki/RE seria a JSPWiki, seguida da VeryQuickWiki. No entanto, durante a análise detalhada das implementações atuais de *wikis* e o atendimento aos princípios (seção 4.4), verificou-se um problema não trivial no modo com que são tratados dados em diferentes mídias. Essa característica é essencial para a Wiki/RE, cujos diversos nós conterão artefatos gerados por técnicas diferentes de engenharia de requisitos. Decidiu-se, portanto, pela construção de uma nova *wiki*, sem tomar uma já existente como base.

A análise dessas ferramentas *wiki* e a experiência adquirida com a manutenção da CoTeia foram uma importante fonte de conhecimento para a construção da Wiki/RE. A comparação de suas características e princípios permitiram a detecção de pontos a serem melhorados, conforme demonstrados na seção 4.4, e a descrição de uma proposta para o problema (seção 4.5). Essa nova concepção, e sua aplicação à Engenharia de Requisitos, será aprofundada e testada na forma de uma *wiki* para Engenharia de Requisitos, a Wiki/RE, descrita no próximo capítulo.

Capítulo

5

A Ferramenta Wiki/RE

A Wiki/RE é uma ferramenta Web para engenharia de requisitos que disponibiliza um ambiente para construção de hiperdocumentos. Este capítulo discorre sobre seu desenvolvimento, descrevendo os problemas encontrados e discutindo as soluções adotadas. São abordados temas como o processo de desenvolvimento utilizado, as etapas da engenharia da ferramenta, com uma visão abrangente dos resultados. Informações mais detalhadas sobre o desenvolvimento encontram-se disponíveis no *site* oficial da Wiki/RE: `http://www.wikire.org/`.

A primeira seção descreve o processo de desenvolvimento utilizado para a Wiki/RE. Os resultados de cada etapa de desenvolvimento são então apresentados nesta ordem: requisitos, arquitetura, projeto, implementação e testes.

5.1 Processo de Desenvolvimento

As aplicações hipermídia criaram novos requisitos aos processos de engenharia de software. Além da necessidade da elaboração detalhada de modelos de hiperdocumentos e de navegação, existe uma maior exposição das aplicações aos mais diferentes tipos de usuários. Aspectos como internacionalização, localização e interface usuário-computador precisam de tratamento adequado. Exige-se também, do processo de desenvolvimento, agilidade para tratar os requisitos (voláteis) e evoluir as aplicações em curtos períodos.

Criaram-se diversos métodos para o desenvolvimento de aplicações Web nos últimos anos: HDM (GARZOTTO; PAOLINI; SCHWABE, 1993), RMM (ISAKOWITZ; STOHR; BALASUBRAMANIAN, 1995), FORM (STOTTS; FURUTA, 1989; FURUTA; STOTTS, 1990; NA; FURUTA,

2001), OOHDM (SCHWABE; ROSSI, 1995; SCHWABE; ROSSI; BARBOSA, 1996), SOHDM (LEE; LEE; YOO, 1998), WSDM (TROYER; LEUNE, 1998), dentre outros. Esses métodos tentam acompanhar não somente o amadurecimento das metodologias de engenharia de software, mas também a própria evolução da Web e das tecnologias que a sustentam. Nesse período, observa-se o surgimento de várias tecnologias, dentre as quais merecem destaque XML, XSLT, RDF, ECMAScript, DOM, XLink, XPointer, SVG, XUL, SMIL e *Web Services*. De fato, constata-se uma verdadeira revolução nos recursos tecnológicos disponíveis às aplicações hipermídia. Em função da rapidez com que essas tecnologias são criadas, ainda são poucas as aplicações que fazem amplo uso dessas tecnologias.

Nota-se, entretanto, que os métodos Web ainda não estão maduros o suficiente para o desenvolvimento de aplicações de grande porte. Sua adoção pela indústria é lenta (LANG; FITZGERALD, 2005). Atividades de gerenciamento de requisitos e de configuração, testes, bem como um processo disciplinado e iterativo, são realizados sem critério adequado. A ferramenta Wiki/RE, apesar de ser uma aplicação hipermídia, reúne características de aplicações corporativas (*enterprise*): longevidade, manutenibilidade, integração com outras aplicações. Para esse nicho, existem métodos adequados e amadurecidos, como o *Unified Process* (JACOBSON; BOOCH; RUMBAUGH, 1999) e o *Component Based Development* (CHEESMAN; DANIELS, 2001; WHITEHEAD, 2002). Para a Wiki/RE, o desejável seria um método que unisse os pontos fortes de ambas as metodologias (Web e *enterprise*).

O processo utilizado no desenvolvimento da Wiki/RE inspirou-se no *Unified Process* e no trabalho de Conallen (2002). Algumas características do projeto de interface abstrata *Object-Oriented Hypermedia Design Model* (OOHDM) foram utilizadas, em especial os conceitos dos *Abstract Data Views* (ADVs) e dos *ADV-Charts*. O embasamento no *Unified Process* garantiu um sólido suporte para a construção de aplicações. No que tange à engenharia de requisitos, empregaram-se técnicas para descrição de requisitos não-funcionais (interesses e casos de mau uso) que não constam nos demais processos. A próxima subseção oferece uma visão geral do processo e as demais subseções descrevem cada etapa do desenvolvimento.

5.1.1 Visão Geral

O processo de software utilizado neste trabalho foi guiado por requisitos, centrado na arquitetura e iterativo. O desenvolvimento foi organizado em ciclos (veja a figura 5.1), concluídos com o lançamento de cada nova versão do software. Cada ciclo foi composto por fases, para as quais se estabeleceram marcos a serem alcançados ao seu final. Cada fase constituiu-se de uma ou mais iterações, que por sua vez foram compostas por um conjunto de atividades estabelecidas em um fluxo de trabalho: requisitos, análise, projeto, implementação e implantação. Assim, o processo adotado fundamentou-se no *Unified Process* (UP) e pode ser visto como uma simplificação acadêmica do mesmo.

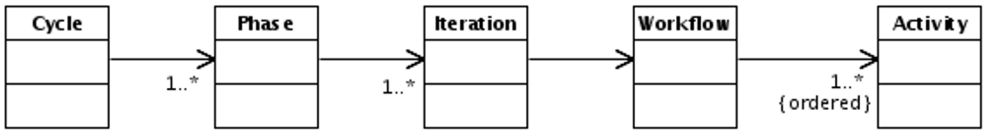

Figura 5.1: Visão de alto nível do processo de desenvolvimento.

A iteração foi o item de menor granularidade do processo. A cada iteração implementou-se um pequeno grupo (geralmente de cardinalidade um) de requisitos funcionais, controlando e mitigando-se os riscos. Ela é constituída de uma seqüência de atividades que englobam as práticas julgadas essenciais para o desenvolvimento de um software de qualidade. A reduzida duração (tipicamente da ordem de uma semana) e restrita dimensão das alterações realizadas permitiram um maior controle do desenvolvimento e do progresso da construção da aplicação, além de garantir uma maior agilidade na obtenção de *feedback* quanto aos resultados. Esses fatores colaboraram na redução dos riscos inerentes a cada iteração.

As iterações foram coordenadas de maneira que seus artefatos contribuíssem para o alcance dos marcos estabelecidos para cada fase. Em linhas gerais, o processo possuiu as seguintes fases e respectivos marcos:

1. **Concepção:** modelo de negócio, esboço da arquitetura e lista de riscos;
2. **Elaboração:** especificação de requisitos, projeto e arquitetura;
3. **Construção:** sistema executável (beta), documentação para o usuário;
4. **Transição:** sistema pronto para implantação.

Existe uma relação entre as atividades executadas nas iterações e a fase corrente, mantendo-se uma proporção de atividades, dados os artefatos por elas produzidos e o quanto esses artefatos contribuem para o marco da fase. A figura 5.2 demonstra como seria essa relação, mostrando a quantidade de determinada atividade executada por fase no desenvolvimento de um software, utilizando o *Unified Process*.

Em projetos executados por pequenos grupos (ou até mesmo individualmente), não é necessária muita sofisticação na gerência das iterações, dado que as mesmas serão seqüenciais. O presente trabalho se enquadra nessa categoria de projeto. As próximas seções detalham cada fase, definindo os fluxos de trabalho utilizados em suas iterações, assumindo a execução das iterações em seqüência.

5.1.2 Concepção

A concepção é a fase inicial do processo. Sua duração geralmente varia com a experiência existente na produção de sistemas no domínio de negócio da aplicação e pode exigir uma única iteração ou até mesmo meses de iterações para projetos totalmente novos. O objetivo

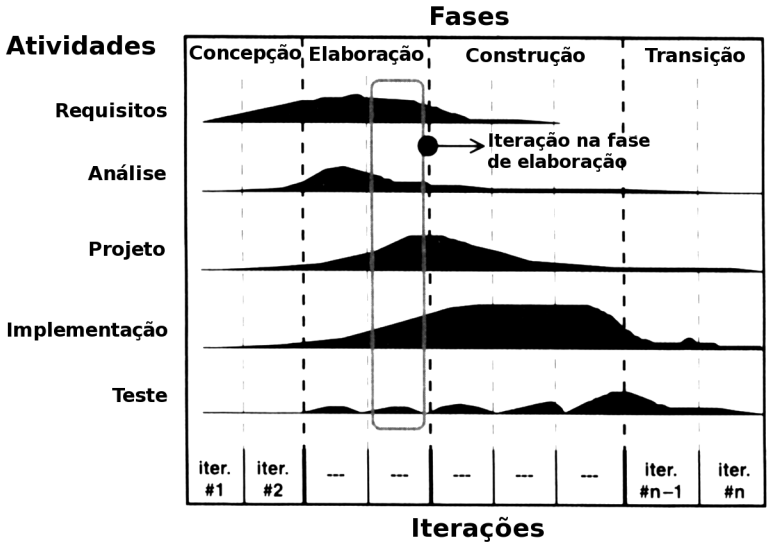

Figura 5.2: Exemplo da relação entre o tipo de atividades realizadas durante as fases do *Unified Process*. Fonte: Jacobson, Booch e Rumbaugh (1999, p. 11).

desta fase é demonstrar a viabilidade da execução do projeto. Inicia-se com a elaboração das idéias que fizeram o projeto surgir e termina com a criação de um modelo de domínio ou de negócio, um esboço da arquitetura da aplicação e uma lista com os principais riscos identificados.

Observa-se que um ponto de partida é necessário para o processo. Antes da concepção, o que existe são apenas idéias e as pessoas que as conceberam. A partir do momento em que uma das idéias demostra ser do interesse dos envolvidos e que decidem concretizá-la, novos grupos de interessados são então inseridos no processo, viabilizando a concretização dela.

Estabelecidos os interessados iniciais, é necessário explicitar o raciocínio da idéia escolhida e quais são os benefícios proporcionados no domínio ou negócio em que ela se insere, definindo-se, em linhas gerais, o produto almejado. Algumas outras informações também devem ser estabelecidas: a quem se destina, recursos disponíveis e prazo. Não se espera, nesse momento, uma precisão elevada nesses dados: é apenas uma visão geral do sistema e delimitação do escopo do trabalho.

Traduzindo a fase em atividades para a realização do mestrado, o ponto de partida foi o aluno, a instituição e o orientador. O período de concepção correspondeu aos primeiros meses do curso. Os marcos foram a entrega dos documentos Projeto de Mestrado e Proposta de Mestrado FAPESP. Durante esse período, várias idéias para possíveis projetos foram propostas pelo aluno. Após uma análise, em conjunto com a orientadora, escolheu-se a proposta mais promissora, respeitando-se os perfis de pesquisa de ambos e o potencial do trabalho [1]. Definiu-se pela construção de uma ferramenta hipermídia para engenharia de requisitos. Procedeu-se à elaboração do Projeto de Mestrado e da Proposta FAPESP. Para esse fim,

[1] As propostas encontram-se disponíveis em `http://coweb.icmc.usp.br/coweb/mostra.php?ident=59.2.9`.

realizou-se uma revisão da literatura, descrição dos objetivos do projeto e planejamento de atividades.

5.1.3 Elaboração

Os objetivos da elaboração são o detalhamento dos requisitos, a definição e validação da arquitetura e a mitigação os riscos. No presente trabalho, a elaboração teve como marcos a qualificação e a manutenção da ferramenta CoTeia.

O primeiro período do mestrado, correspondente à qualificação, compreendeu a revisão da literatura, a participação em disciplinas regulares do curso de pós-graduação e reuniões com a orientadora e com um representante da empresa Async. Durante a concepção, definiram-se os requisitos da ferramenta em reuniões com a orientadora e demais interessados e realizou-se uma cuidadosa revisão da literatura sobre engenharia de requisitos e hipermídia. A qualificação foi redigida e o aluno avaliado em prova oral (apresentação e argüição da monografia). Após a avaliação, decidiu-se pela implementação de uma ferramenta *wiki*, voltada para a engenharia de requisitos em software livre.

Os requisitos funcionais foram definidos pelos tradicionais casos de uso, aos quais estão associados casos de mau uso, que especificam requisitos não-funcionais. A cada um deles (casos de uso, tanto "bons" quanto maus), existe uma meta associada e, conseqüentemente, um ou mais interessados. São esses interessados e suas metas que se busca satisfazer com o produto em desenvolvimento. Os artefatos produzidos foram armazenados em uma *wiki*, mais precisamente uma instância da CoTeia. A manutenção da CoTeia permitiu a compreensão de importantes conceitos de *wikis* e auxiliou na definição da arquitetura da Wiki/RE.

Todas as atividades foram acompanhadas através de registros semanais[2], encaminhados à orientadora, e relatórios semestrais, encaminhados à instituição.

5.1.4 Construção

A fase de construção objetiva a implementação, em iterações, do software. As primeiras iterações de construção iniciaram-se ainda durante a elaboração. A cada iteração os requisitos foram refinados e implementados. Buscou-se a execução de atividades de garantia de qualidade, como controle de alteração, revisões e testes, durante todas as iterações. Todos os artefatos foram colocados sob controle de versão: o código-fonte e modelos do projeto armazenados no Subversion (CollabNet, 2000), os requisitos e demais documentos armazenados no CVS (GRUNE et al., 1986), com o intermédio da CoTeia (SILVA, 2004). As alterações foram controladas com a ferramenta Bugzilla (WEISSMAN et al., 1998), os testes implementados com a ferramenta JUnit (GAMMA; BECK, 2001). Adotou-se o uso de ferramentas automáti-

[2] O acompanhamento das atividades encontra-se em `http://coweb.icmc.usp.br/coweb/mostra.php?ident=59.2.1`.

cas para verificação da qualidade do código-fonte, como o Hammurapi (VLASOV et al., 2004), CheckStyle (BURN, 2001) e PMD (COPELAND et al., 2002).

5.1.5 Transição

A fase de transição destina-se ao testes *beta* da aplicação, treino de usuários, preparação de documentação, preparativos para implantação da ferramenta e revisão do projeto, identificando pontos a serem melhorados no processo.

Algumas atividades de transição já foram realizadas para a Wiki/RE. O processo de implantação da ferramenta foi definido, documentado e automatizado e pontos a serem melhorados no processo foram identificados. As demais atividades serão executadas após o lançamento público da ferramenta.

5.2 Requisitos

O processo de engenharia de requisitos iniciou-se com a identificação dos interessados. Cada um expôs suas metas, a partir das quais definiram-se os requisitos da ferramenta. Através de casos de uso e casos de mau uso, os principais requisitos foram analisados. Os resultados do processo encontram-se documentados nas próximas subseções.

5.2.1 Identificação dos Interessados

Os interessados que participaram ativamente do processo de engenharia de requisitos foram o presente mestrando, sua orientadora e o empresário Christian Reis, da empresa Async Open Source. O primeiro desenvolvia pesquisas em engenharia de requisitos desde a graduação, mas não possuía um conhecimento profundo da área de hipermídia e não possuía experiência profissional. A orientadora desempenhava pesquisas na área de Engenharia de Software e Hipermídia e contava com experiência profissional. Reis, cujo mestrado (REIS, 2003) foi realizado sob a orientação de Renata Fortes, aliava o conhecimento sobre Engenharia de Software ao Software Livre, do qual possuía ampla experiência profissional.

5.2.2 Definição das Metas

Os diferentes perfis dos interessados permitiu a identificação de diferentes metas. Os autores deste trabalho tinham enfoque em resultados acadêmicos, com viabilidade prática, enquanto Reis concentrava em aspectos práticos, que facilitariam a aceitação da ferramenta na indústria de software. As principais metas e interesses identificados foram:

1. Metas e interesses acadêmicos

1.1. documentar requisitos descritos por técnicas diversas;

1.2. utilizar autoria colaborativa para escrever os documentos de requisitos;

1.3. fornecer mecanismos que facilitem a descrição de requisitos;

1.4. fornecer mecanismos que facilitem a navegação no documento de requisitos;

1.5. extensibilidade da ferramenta.

2. Metas e interesses industriais

2.1. garantir a segurança do sistema e das informações nele armazenadas;

2.2. acessibilidade;

2.3. respeito aos direitos autoriais da ferramenta;

2.4. direitos autorais dos artefatos produzidos com a ferramenta;

2.5. internacionalização;

2.6. viabilizar o desenvolvimento sustentável da ferramenta;

2.7. importação e exportação de dados;

2.8. apoio para a execução do processo de engenharia de requisitos.

A partir das metas e interesses, iniciou-se a definição dos requisitos da Wiki/RE. Atender a todas as metas seria inviável, então selecionou-se um subconjunto delas: 1.1, 1.2, 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 2.5 e 2.6. Todas elas foram analisadas, em algum grau, durante a engenharia de requisitos. Algumas, como a 1.1 e a 1.2, receberam mais tempo para estudo e implementação, em detrimento a outras, como as 2.1, 2.2 e 2.4.

5.2.3 Casos de Uso

Analisando as metas 1.1 e 1.2, optou-se pelo desenvolvimento de um sistema hipermídia do tipo *wiki* para possibilitar a edição colaborativa do documento de requisitos. O diagrama, contido na figura 5.3, apresenta os casos de uso básicos da Wiki/RE.

As diferentes páginas do hiperdocumento podem conter artefatos produzidos por técnicas diversas de engenharia de requisitos. O fluxo de trabalho, em uma visão de alto nível (apresentada no diagrama da figura 5.4), do uso da Wiki/RE consiste na aplicação de uma técnica por um ou mais interessados, consultando, conforme necessário, dados registrados no hiperdocumento gerido pela Wiki/RE. Os resultadas obtidos seriam então documentados, criando-se novos nós, provavelmente a partir dos nós consultados durante a aplicação da técnica, e alterando os já existentes, seja para atualização ou correção de algum dado ou para criar as referências que permitirão a criação de novos nós.

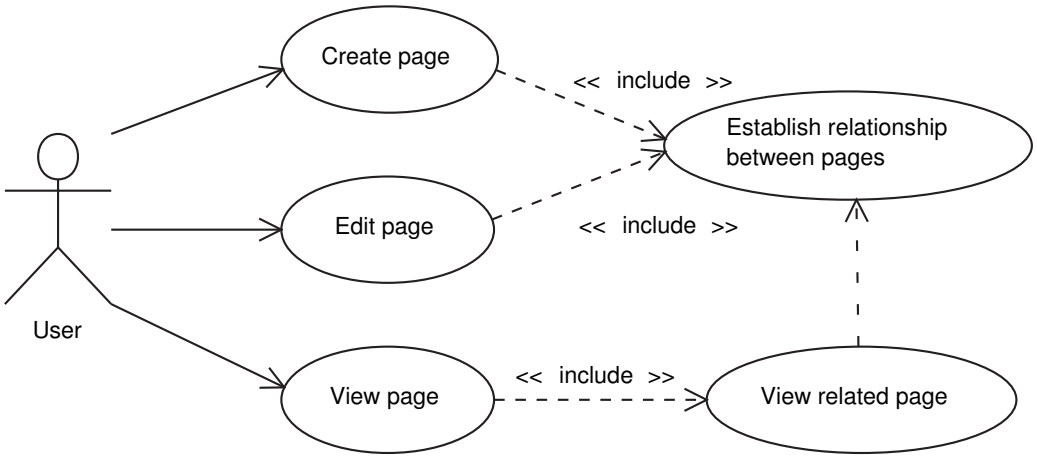

Figura 5.3: Diagrama de casos de uso sobre a essência do funcionamento da Wiki/RE.

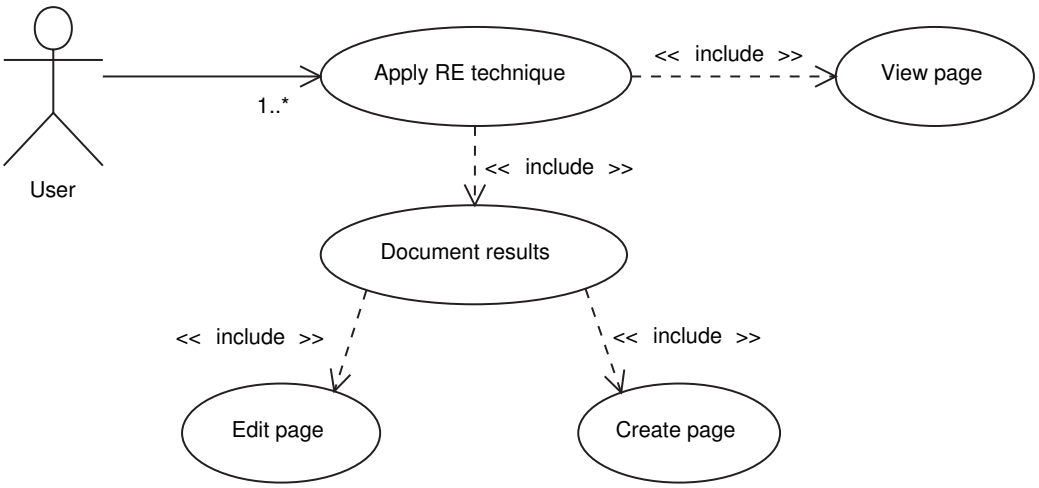

Figura 5.4: Diagrama de casos de uso da Wiki/RE para engenharia de requisitos.

Para prosseguir com o desenvolvimento dos casos de uso, é necessária a apresentação do diagrama da figura 5.5, correspondente aos casos de uso do padrão *Create*, *Read*, *Update*, *Delete* (CRUD). Em linhas gerais, seus casos de uso representam as operações típicas realizadas em objetos durante o ciclo de vida dos mesmos: criação, edição, consulta e remoção. Essas operações serão posteriormente estendidas para os casos de uso da Wiki/RE.

O diagrama da figura 5.6 relaciona todos os casos de uso da Wiki/RE que contribuem para as metas de documentação de requisitos provenientes de técnicas diversas (1.1) e de emprego de autoria colaborativa para a escrita dos documentos de requisitos (1.2).

Os primeiros casos de uso para qualquer hiperdocumento são os responsáveis pela visualização ou pela atualização de uma página *wiki* (ou simplesmente *wikipage*)[3]. Os demais casos de uso não podem ser ativados sem que exista uma página definida.

[3] Durante a análise dos casos de uso, utiliza-se o termo *wikipage* como sinônimo de nó do hiperdocumento.

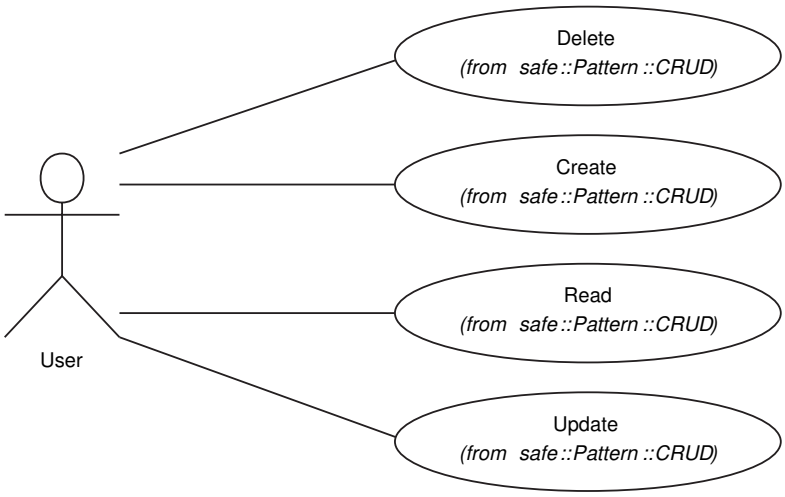

Figura 5.5: Diagrama de casos de uso do padrão CRUD.

Figura 5.6: Diagrama de casos de uso da Wiki/RE quanto às suas funções *wiki*.

O fato de qualquer desses dois casos de uso ser o inicial decorre do fato de que, em *wikis*, existe o conceito de que, no hiperdocumento, estão presentes todos os nós possíveis

(no caso da Wiki/RE, infinitos). A diferença é que alguns possuem um conteúdo definido, enquanto outros não. O acesso a um nó cujo conteúdo não esteja definido, ao invés de informar que tal nó não existe (como ocorre nas aplicações Web, apresentando-se uma resposta HTTP com código 404), redireciona, automaticamente, para o caso de uso de atualização do conteúdo do nó. Atende-se, com essa estratégia, o princípio incremental das *wikis*.

Nome: View wikipage

Pré-condições:

- O nome do nó deve estar especificado.

Passos:

1. (Sistema) Verificar se existe um nó com o nome especificado.
   - (Sistema) Se não existir um nó com o nome especificado, executar o caso de uso "Update wikipage".
   - (Sistema) Se existir apenas um tipo de nó cujo nome é idêntico ao especificado:
     1.1. (Sistema) Recuperar a última versão do nó solicitado.
     1.2. (Sistema) Apresentar o nó encontrado ao usuário.
   - (Sistema) Se existir mais de um tipo de nó cujo nome é idêntico ao especificado:
     1.3. (Sistema) Apresentar ao usuário os nós encontrados.
     1.4. (Usuário) Escolher o nó que deseja ver.
     1.5. (Sistema) Recuperar a última versão do nó escolhido .
     1.6. (Sistema) Apresentar o nó encontrado ao usuário.

Pontos de extensão:

- Escolha da versão do nó a ser recuperada (passos 1.1 e 1.5)

A Wiki/RE permite a criação de páginas com diferentes tipos de artefatos, possibilitando que os produtos de diversas técnicas de engenharia de requisitos sejam inseridos no hiperdocumento (meta 1.2). Antes da definição do conteúdo de um nó, define-se o tipo de dado que nele será armazenado (um caso de uso, uma meta, etc). A referência do hiperdocumento que acionou o caso de uso em questão, o "Update wikipage" não precisa especificar o tipo de nó desejado. Nesse caso, durante a visualização da página, conforme descrito no caso de uso "View wikipage", o acionamento da referência exibirá todos os nós do hiperdocumento com o nome especificado. Dessa forma, preserva-se o princípio de unificação e precisão.

Nome: Update wikipage
Pré-condições: • O nome do nó deve estar especificado.
Passos: 1. (Sistema) Verificar se existe um nó com o nome (e, se conhecido, também o tipo) especificado. • (Sistema) Se não existir um nó com o nome (e, se conhecido, também o tipo) especificado, executar o caso de uso "Create wikipage". • (Sistema) Se existir apenas um tipo de nó cujo nome é idêntico ao especificado: 1.1. (Sistema) Recuperar a última versão do nó solicitado. 1.2. (Sistema) Apresentar a visão adequada para a edição do nó (de acordo com o tipo do nó). 1.3. (Usuário) Alterar o conteúdo do nó (de acordo com o tipo escolhido). 1.4. (Sistema) Verificar se o conteúdo foi corretamente especificado. 1.5. (Sistema) Armazenar o conteúdo alterado do nó no sistema. • (Sistema) Se existir mais de um tipo de nó cujo nome é idêntico ao especificado: 1.6. (Sistema) Apresentar ao usuário os nós encontrados. 1.7. (Usuário) Escolher o nó que deseja ver. 1.8. (Sistema) Recuperar a última versão do nó escolhido 1.9. (Sistema) Apresentar a visão adequada para a edição do nó (de acordo com o tipo do nó). 1.10. (Usuário) Alterar o conteúdo do nó (de acordo com o tipo escolhido). 1.11. (Sistema) Verificar se o conteúdo foi corretamente especificado. 1.12. (Sistema) Armazenar o conteúdo alterado do nó no sistema.

A mesma concepção que justifica a relação entre os casos de uso de visualização e alteração, aplica-se ao caso de uso de criação de uma página *wiki*, o "Create wikipage". Segundo o modelo de *wikis*, não existiria necessidade de criar um *wikipage*. No entanto, a aplicação precisa distinguir entre alteração e criação de um novo nó, para efeito de controle de versão.

Nome: Create wikipage
Pré-condições: • O nome do nó deve estar especificado. • O tipo do nó deve estar especificado.
Passos: 1. (Sistema) Criar um nó com o nome e tipo especificados e conteúdo nulo. 2. (Sistema) Armazenar o novo nó no sistema.

A Wiki/RE preserva todas as versões de uma página *wiki*. Essas versões são identificadas única e globalmente: uma alteração de uma página não incrementa a versão apenas da página em questão, mas de todo o hiperdocumento. O objetivo é manter uma configuração consistente do hiperdocumento sem grandes custos computacionais. O caso de uso "Recover previous version", que estende o "View wikipage", permite a navegação no hiperdocumento, recuperando apenas as páginas com versões compatíveis (iguais ou inferiores) à solicitada pelo usuário.

Em Engenharia de Requisitos, esse recurso é importante para a rastreabilidade. Ele permite identificar, em um artefato que utilize o requisito (a página *wiki*), a identificação da exata versão consultada. Se o requisito for posteriormente alterado, é possível identificar os artefatos que devem ser revisados. Durante a validação do artefato (ou qualquer outra atividade de garantia de qualidade, como criação de casos de teste e revisões), a atividade poderá ser executada com os dados efetivamente utilizados para a criação do artefato.

Nome: Recover previous version

Pré-condições:

- A versão solicitada é válida.

Passos:

1. (Sistema) Verificar se, na versão solicitada do hiperdocumento, o nó especificado existe.
   - (Sistema) Se o nó especificado não existir na versão do hiperdocumento solicitada:
     1.1. (Sistema) Notificar o usuário de que o nó não existe na versão especificada do hiperdocumento.
   - (Sistema) Se o nó existir na versão especificada do hiperdocumento:
     1.2. (Sistema) Obter a versão do nó solicitada.
     1.3. (Sistema) Atualizar todas as referências para outros artefatos, definidas no nó recuperado, para que artefatos da mesma versão sejam recuperados.

Em software livre, as iterações rápidas e lançamentos freqüentes de novas versões compelem os desenvolvedores, mesmo aqueles que realizam o desenvolvimento em árvores ou ramos diferentes do principal, a utilizar a versão mais recente dos artefatos sempre que possível[4]. Comparar as alterações feitas a cada versão facilita a identificação de erros e o acompanhamento da evolução dos artefatos. O caso de uso "Compare wikipage" descreve essa funcionalidade, aplicada para páginas *wiki*.

Nome: Compare wikipage

Pré-condições:

- Especificação de dois nós a serem comparados.
- Especificação de versões válidas para a comparação.

Passos:

1. (Sistema) Recuperar as versões solicitadas dos nós.
2. (Sistema) Comparar os nós recuperados.
3. (Sistema) Apresentar ao usuário as diferenças detectadas entre os nós.

Uma conseqüência da evolução do hiperdocumento é a obsolescência de páginas antigas. A remoção de um página em *wikis* é um tema controverso: páginas que citam a página apagada precisam ser atualizadas e a recuperação do conteúdo da página removida,

[4] A utilização da versão mais recente facilita a integração do novo código à linha de desenvolvimento principal, o que é um importante marco no desenvolvimento de software livre.

caso necessário, não é trivial. Por outro lado, existem páginas que ficam "órfãs", ou seja, sem nenhuma ligação a partir de outras páginas do hiperdocumento. Fazendo uma analogia com algumas linguagens de programação (LISP, Java), a página seria uma candidata a ser removida pelo "coletor de lixo". Argumenta-se, no entanto, que alguma página poderia, futuramente, criar uma referência para uma página com o mesmo nome daquela apagada automaticamente e, conseqüentemente, perder-se-ia uma oportunidade de reutilização do conteúdo da mesma.

O interesse em remover páginas na Wiki/RE advém justamente da presença de páginas órfãs. Nenhuma informação do hiperdocumento deveria estar desprovida de rastreabilidade, uma razão para sua existência, em qualquer momento. Se a perda de rastreabilidade foi uma conseqüência natural da evolução do documento, deve-se revisar a página órfã, para ter certeza de que seu conteúdo é desnecessário, e removê-la. Se foi um equívoco, deve-se identificar em que momento a rastreabilidade foi quebrada e restaurá-la devidamente.

A Wiki/RE identifica as páginas removidas com uma ligação específica, que torna possível a recuperação da página removida, caso necessário. Desta forma, páginas que contêm ligações para páginas identificadas como apagadas ainda dispõem de acesso à sua última versão disponível.

Nome: Delete wikipage

Pré-condições:

- O nome do nó deve estar especificado.

Passos:

1. (Sistema) Verificar se existe um nó com o nome especificado.
   - (Sistema) Se existir apenas um tipo de nó cujo nome é idêntico ao especificado:
     1.1. (Sistema) Marcar o nó como apagado no hiperdocumento.
     1.2. (Sistema) Remover o nó do controle de versões.
   - (Sistema) Se existir mais de um tipo de nó cujo nome é idêntico ao especificado:
     1.3. (Sistema) Apresentar ao usuário os nós encontrados.
     1.4. (Usuário) Escolher o nó que deseja ver.
     1.5. (Sistema) Marcar o nó como apagado no hiperdocumento.
     1.6. (Sistema) Remover o nó do controle de versões.

Uma outra operação típica durante a evolução de um hiperdocumento de requisitos é a mudança de nome de um artefato. Além de alterar o nome e atualizar as ligações em outras páginas, é preciso garantir que, na ocorrência de uma requisição para uma página que fora renomeada, o usuário seja redirecionado para a página com o novo nome. Esse recurso é necessário para preservar a rastreabilidade das páginas perante referências contidas em documentos que não são controlados pela Wiki/RE (por exemplo, arquivos de projeto ou casos de teste que são criados a partir dos requisitos).

Nome: Rename wikipage
Pré-condições: • O nome do nó deve estar especificado. • O tipo do nó deve estar especificado. • O novo nome do nó deve estar especificado. • Não pode existir, no hiperdocumento, um nó, do mesmo tipo, com o novo nome.
Passos: 1. (Sistema) Recuperar todos os nós com o nome especificado. 2. (Sistema) Alterar o nome de todos os nós encontrados (independentemente do tipo). 3. (Sistema) Criar ligações dos nós (com os nomes antigos) para os nós (com os nomes novos). 4. (Sistema) Renomear o nome dos nós no sistema de controle de versão.

Durante a engenharia de requisitos, é esperado que ocorra o refinamento dos artefatos e a eventual divisão dos artefatos em outros. A Wiki/RE dispõe de um mecanismo de refatoração, que permite a divisão de uma página em duas páginas diferentes. Uma ligação é, então, especificamente criada, garantindo que, caso ocorra a requisição de uma página que sofreu refatoração, sejam oferecidas as páginas geradas após o processo de divisão, preservando-se assim a rastreabilidade.

Nome: Refactor wikipage
Pré-condições: • O nome do nó deve estar especificado.
Passos: 1. (Sistema) Verificar se existe um nó com o nome especificado. • (Sistema) Se existir apenas um tipo de nó cujo nome é idêntico ao especificado: 1.1. (Sistema) Recuperar a última versão do nó solicitado. 1.2. (Sistema) Apresentar a visão adequada para a refatoração do nó (de acordo com o tipo do nó). 1.3. (Usuário) Alterar o conteúdo do nó (de acordo com o tipo escolhido). 1.4. (Sistema) Verificar se o nó foi dividido em outros nós. Se for dividido: 1.4.1. (Sistema) Verificar se o nome dos novos nós são válidos. 1.4.2. (Sistema) Criar uma ligação do nó original aos novos nós criados. 1.5. (Sistema) Verificar se o conteúdo foi corretamente especificado. 1.6. (Sistema) Armazenar o conteúdo alterado dos nós no sistema. • (Sistema) Se existir mais de um tipo de nó cujo nome é idêntico ao especificado: 1.7. (Sistema) Apresentar ao usuário os nós encontrados. 1.8. (Usuário) Escolher o nó que deseja refatorar. 1.9. (Sistema) Recuperar a última versão do nó solicitado. 1.10. (Sistema) Apresentar a visão adequada para a refatoração do nó (de acordo com o tipo do nó). 1.11. (Usuário) Alterar o conteúdo do nó (de acordo com o tipo escolhido). 1.12. (Sistema) Verificar se o nó foi dividido em outros nós. – Se foi dividido: 1.12.1. (Sistema) Verificar se o nome dos novos nós são válidos. 1.13. (Sistema) Verificar se o conteúdo foi corretamente especificado. 1.14. (Sistema) Armazenar o conteúdo alterado dos nós no sistema.

5.2.4 Casos de Mau Uso

Analisaram-se possíveis ameaças ao sistema por meio de casos de mau uso. No capítulo 4, seção 4.4, uma das *soft-goals* que afetavam negativamente as *wikis* era o vandalismo. A partir dela, foi possível detectar os seguintes casos de mau uso:

Vandalismo *per se*: alteração grosseira do hiperdocumento e seu conteúdo.

Adulteração de informações: alteração discreta do conteúdo e que poderia passar desapercebida.

Negação de serviço: criação ou renomeação indevida e em massa de nós do hiperdocumento.

Esses casos de mau uso relacionam-se com os casos de uso da Wiki/RE de acordo com o diagrama de casos de mau uso da figura 5.7.

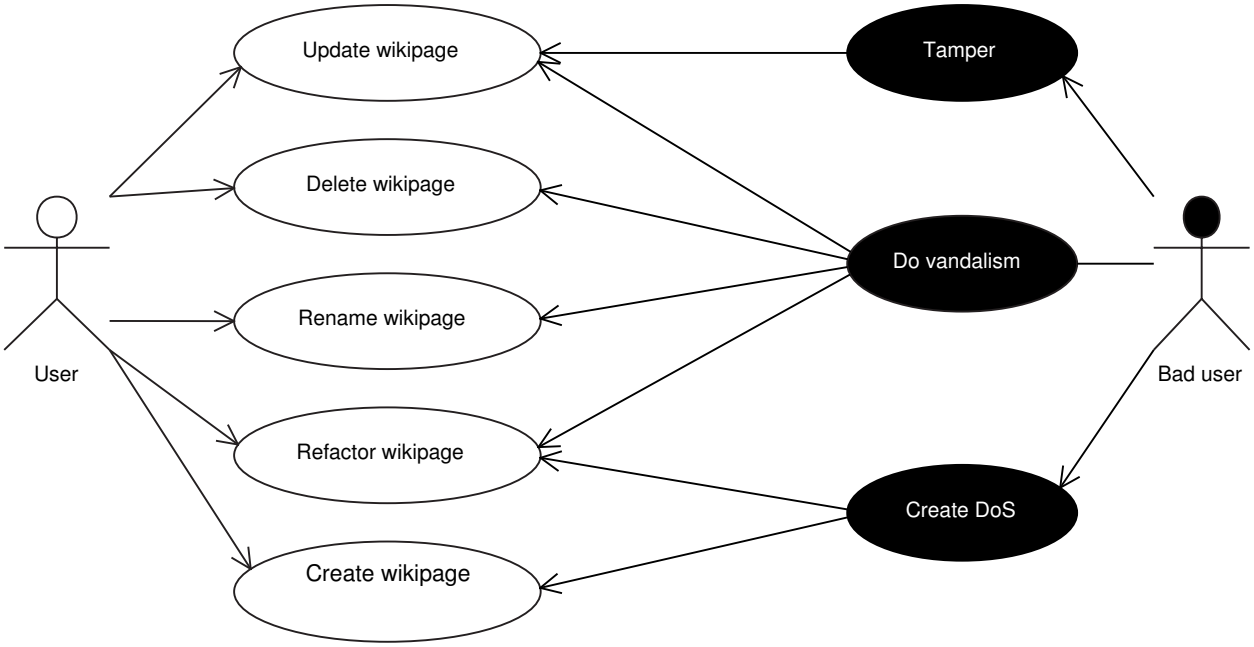

Figura 5.7: Diagrama de casos de mau uso para a Wiki/RE.

Observa-se que o simples acesso ao hiperdocumento (caso de uso "View wikipage") não causa transtornos à aplicação. Uma solução razoável para todos os casos de mau uso identificados seria a exigência de autenticação do usuário para qualquer alteração realizada no hiperdocumento. Esse mecanismo demonstra-se suficiente para impedir a maioria dos abusos em relação a aplicações *wiki*.

Considerando que a Wiki/RE será utilizada em um contexto mais amplo no projeto SAFE, integrada a outras ferramentas tanto técnicas quanto gerenciais, poder-se-ia restringir

as alterações apenas aos usuários autorizados pela gerência (clientes e desenvolvedores), ao invés de um modelo aberto típico das *wikis*.

Um nível adicional de segurança seria ainda possível com a utilização de criptografia assimétrica (chaves públicas e privadas) para a autenticação dos usuários e do conteúdo enviado para a ferramenta. Devido à dificuldade na utilização dessa tecnologia, ela seria restrita aos usuários desenvolvedores.

A Wiki/RE foi projetada considerando o uso de autenticação simples (sem o uso de criptografia assimétrica) integrada ao SAFE. A implementação desse mecanismo está prevista como um trabalho futuro, utilizando os serviços definidos pelo *framework* do SAFE.

5.3 Arquitetura

A arquitetura da Wiki/RE foi elaborada considerando-se a separação em camadas. Assim, a arquitetura consiste, inicialmente, em três camadas: persistência (***Storage***), negócio (***Business***) e apresentação (***Presentation***). Cada camada comunica-se apenas com a sua camada imediatamente vizinha. Essa separação permite uma clara divisão dos interesses da ferramenta, além de ser uma solução bem sucedida para aplicações Web. A camada de persistência é responsável por estender o tempo de vida dos objetos para além do tempo de execução da aplicação, armazenando seus dados em memória permanente. A camada de negócio contém a lógica da aplicação, representando e manipulando o modelo de dados dos requisitos. Enfim, a camada de apresentação permite diferentes visualizações desses dados do modelo de requisitos.

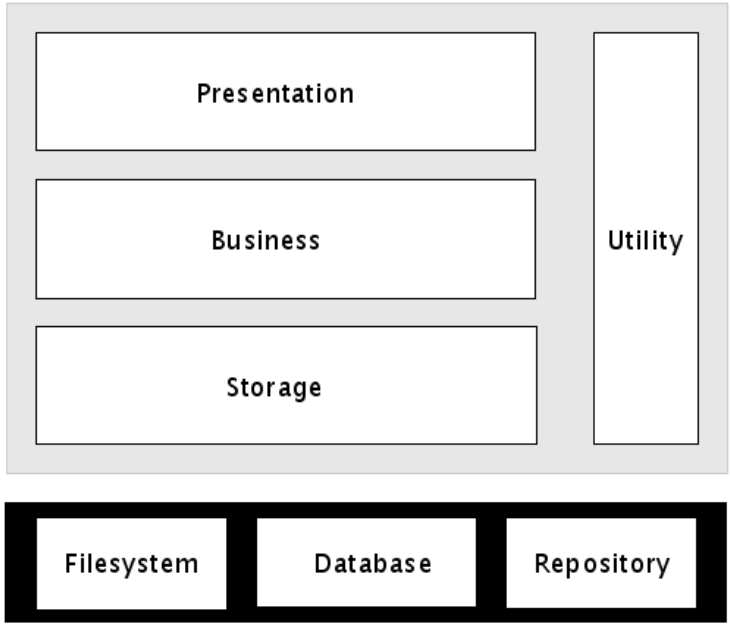

Figura 5.8: Arquitetura da Wiki/RE.

Além das três camadas principais, existem elementos globalmente visíveis, que consistem em classes de uso geral e recorrente (como manipulação de arquivos e estruturas de dados) agrupadas na camada de utilitários (**Utility**). Adicionou-se a restrição de que uma camada não pode comunicar-se com uma camada não vizinha por meio dessa camada utilitária, garantindo-se assim a coesão da arquitetura.

Essas quatro camadas definem o núcleo da arquitetura da Wiki/RE. A última camada, a de persistência, ainda utiliza alguns serviços do sistema, como sistema de arquivos (**Filesystem**), sistemas gerenciadores de bancos de dados (**Database**) e sistemas de controle de versão (**Repository**). A próxima subseção descreve como se comporta a interação entre a camada de persistência e os serviços do sistema.

5.3.1 Camada de Persistência

Uma das características mais desejáveis de documentos é que possua um longo tempo de vida. Idéias e conceitos, discutidos e ampliados durante a engenharia de requisitos, devem ser armazenados para posterior utilização no desenvolvimento de software. De maneira análoga aos livros, em que várias edições são criadas, cada qual com um conteúdo, de modo evolutivo, cada versão do documento de requisitos deve ser armazenada permanentemente. Somente assim pode-se garantir a preservação do conhecimento utilizado na produção de outros artefatos pela engenharia do software (rastreabilidade retroativa).

As soluções de persistência geralmente não se preocupam em armazenar a história dos objetos, classificando-os em transientes e persistentes. Em algumas implementações, existe ainda o estado desconectado. O ciclo de vida do objeto se comporta como esquematizado no diagrama de estados da figura 5.9.

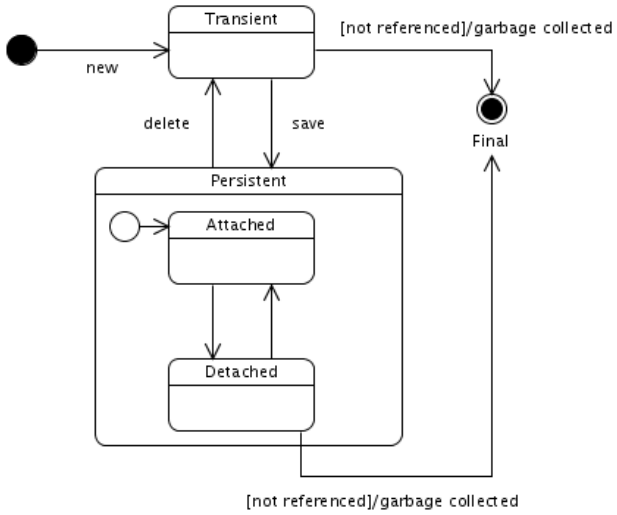

Figura 5.9: Estados de um objeto persistente.

Os significados dos estados que o objeto pode assumir, conforme evidenciado na figura 5.9, são:

Transiente: um objeto cujo estado estará disponível apenas pelo tempo em que ele, o objeto, permanecer ativo.

Persistente: um objeto cujo estado será armazenado por um tempo além daquele em que ele permanecer ativo. Como um objeto persistente, ele pode ser:

Conectado: um objeto persistente está em um contexto em que a persistência está ativa e, portanto, qualquer alteração em seu estado é automaticamente preservada.

Desconectado: um objeto persistente que é utilizado em um contexto em que as alterações feitas em seu estado não são armazenadas. Posteriormente, se esse objeto reingressar em um contexto em que a persistência é possível, seu estado atual pode ser armazenado definitivamente. No entanto, se ele não se reconectar a um contexto com persistência, seu estado será perdido ao final de seu ciclo de vida (quando ele for removido da memória por um *garbage collector*, por exemplo).

Uma implementação, livre e de qualidade, de persistência nos moldes apresentados é o Hibernate (KING et al., 2005a). Ele utiliza uma base de dados para o armazenamento dos atributos dos objetos, acessando-a via *Java DataBase Connectivity API* (JDBC) (ELLIS; HO; FISHER, 2001). As alterações são realizadas em transações gerenciadas pelo sistema gerenciador de banco de dados, por uma implementação da *Java Transaction API* (CHEUNG; MATENA, 2002) (JTA) ou por um mecanismo implementado pelo usuário. Na Wiki/RE, utilizam-se transações gerenciadas pelo SGBD.

A questão da preservação das diversas edições do documento ainda está presente. O Hibernate fornece uma solução para controle de versões como parte de sua estratégia de reserva otimista (*optimistic locking*), mas esta cria uma dependência indesejada a uma implementação específica de mecanismos de persistência (caso, no futuro, deseje-se substituir o Hibernate, realizar tal alteração seria uma tarefa complexa). O ideal seria desenvolver um meio diferenciado para armazenar as diferentes versões.

Os sistemas de controle de versão são consagrados na engenharia de software. Em software livre, tamanha é sua importância que eles desempenham um papel essencial no processo (REIS, 2003). A maioria das *wikis* suportam controle de versão de suas páginas (SILVA, 2005), seja por mecanismos *ad hoc* ou pela utilização de ferramentas já consagradas, como o RCS (TICHY, 1982) e o CVS (GRUNE et al., 1986). A Wiki/RE suporta sistemas de controle de versão que operam com arquivos (tal como na maioria das outras *wikis*). Para isso, ela implementa uma camada de controle de versão, abstraindo assim que implementação será utilizada. A figura 5.10 mostra o modelo conceitual das classes internas a essa camada. Resumidamente, o único elemento público necessário para a aplicação é o `RepositoryTransaction`, obtido pelo `RepositoryTransactionFactory`. Atualmente, apenas o

Subversion é suportado[5] (`SubversionRepositoryTransaction`), mas seria possível utilizar qualquer solução que implemente a API proposta.

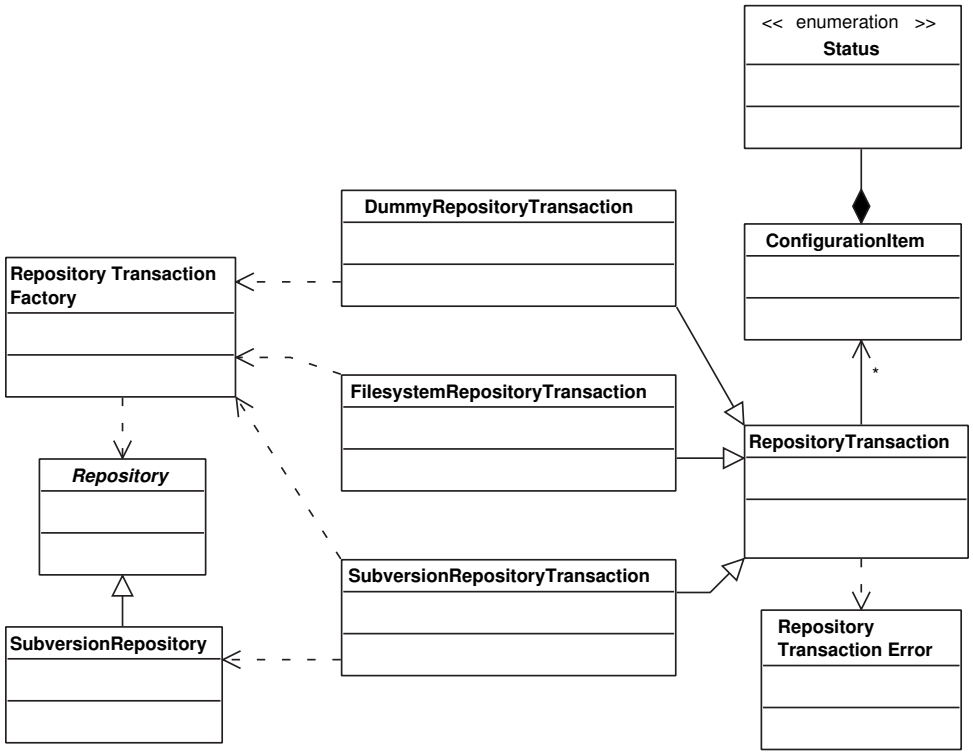

Figura 5.10: Esquema conceitual do interior da camada de controle de versões da Wiki/RE.

A Wiki/RE utiliza sistemas que definem arquivos como itens de configuração. Porém, até o presente momento, os dados são armazenados apenas na base de dados. Torna-se necessário, portanto, criar uma representação em arquivo dos objetos armazenados. Para isso, escolheu-se transformar cada objeto em um documento XML. Esses mesmos documentos serão utilizados, posteriormente, para apresentar uma visão do documento de requisitos.

Analisaram-se duas tecnologias para representar os objetos *Plain Old Java Object* (POJO), controlados pelo Hibernate, em fontes de dados XML: serialização e mapeamento (*binding*). A serialização consiste em transformar os atributos de um objeto em uma seqüência de bytes, a partir dos quais poderá, posteriormente, ser restaurado um outro objeto com o mesmo estado daquele serializado. Essa transformação garante o isomorfismo, entre o objeto e os dados armazenados na seqüência, apenas no instante em que a operação é executada. O mapeamento, por sua vez, define uma relação entre o objeto e sua representação, garantindo sempre a consistência entre eles.

[5] O `DummyRepositoryTransaction` é uma prova de conceito, utilizado principalmente para testes da API, e o `FilesystemRepositoryTransaction` encontra-se parcialmente implementado.

Em um primeiro momento, os mapeamentos são alternativas mais interessantes. As implementações atuais, no entanto, impõem restrições quanto ao seu uso na Wiki/RE. Foram estudadas duas soluções, o *XMLBeans* (ANDREI et al., 2005) e a implementação de referência do *Java API for XML Binding* (JAXB) (FIALLI; VAJJHALA, 2003), fornecida pela SUN em seu *Java Web Services Developer Pack* (JWSDP) (SUN Microsystems, 2004). Em ambos, o mecanismo é semelhante: o mapeamento, diferentemente do Hibernate, é realizado em tempo de compilação. A partir de um documento *XML Schema Definition* (XSD), geram-se interfaces e classes Java que representam o tipo do documento definido no XSD. Essas não podem ser estendidas pela aplicação, que precisa empregar padrões como o *decorator* para adicionar novos comportamentos. A criação de mapeamentos para estas classes no Hibernate também torna-se uma tarefa inviável. De fato, a representação é adequada para documentos XML, mas não para modelos relacionais.

Os mecanismos de serialização primam pela simplicidade. Dado um objeto qualquer, é possível gerar uma representação na forma de um documento XML e posteriormente restaurá-lo. Como os documentos XML são utilizados na Wiki/RE apenas para apresentação e controle de versão, em instantes bem definidos, a serialização demonstra-se mais adequada. Para sua implementação, escolheu-se o XStream (WALNES et al., 2005).

Definido o mecanismo a ser utilizado para a geração dos documentos XML, torna-se necessário definir os instantes em que tais documentos devem ser criados e terem seus estados atualizados no repositório. Foram identificados os seguintes estados:

- Criação de um novo objeto: Serializa o objeto criado, armazenando-o na cópia de trabalho e inserindo-o no repositório.
- Carregamento de um objeto: Carrega o documento XML correspondente, do repositório para a cópia de trabalho.
- Remoção de um objeto: Remove o documento XML da cópia de trabalho e o marca para remoção no repositório.
- Alteração de um objeto: Serializa o objeto alterado, substituindo o arquivo da cópia de trabalho, provocando a atualização do arquivo no repositório.

Para executar essas operações, utiliza-se a infra-estrutura de eventos do Hibernate. Ela estabelece, para cada operação realizada nos objetos, a geração de um evento. Este, por sua vez, pode ser "percebido" por um objeto Java que implemente a interface correspondente ao evento gerado, podendo então operar com os dados gerados pelo evento.

Os eventos suportados pelo Hibernate possuem uma granularidade fina, das quais apenas uma fração é utilizada pela Wiki/RE. Uma lista completa dos eventos gerados pelo Hibernate encontra-se na documentação de sua API (KING et al., 2005b), na seção referente ao pacote `org.hibernate.event`. Para a Wiki/RE, bastam os seguintes:

- `post-insert`, atendido pela classe `safe.wikire.event.InsertAction`;
- `post-load`, atendido pela classe `safe.wikire.event.LoadAction`;
- `post-update`, atendido pela classe `safe.wikire.event.UpdateAction`.

Cada uma das ações disparadas pelos eventos são efetuadas não diretamente no repositório e sim em uma cópia de trabalho. Desta maneira, mantém-se o isolamento entre operações simultâneas no documento de requisitos. Ao concluir uma transação do Hibernate, enviam-se as alterações realizadas na cópia de trabalho para o repositório.

Nesse modelo, nunca existirão conflitos nessa atualização do repositório, dado que as transações do Hibernate garantem o isolamento necessário entre operações simultâneas nos objetos. Isto é, considerando-se que o repositório é utilizado unicamente pelo Hibernate. Como existe a prevalência dos arquivos criados pelo Hibernate, se um documento do repositório é alterado por uma aplicação externa, na próxima alteração realizada pela Wiki/RE no objeto referente ao documento, este será sincronizado com o estado do objeto, perdendo-se as alterações realizadas pela aplicação externa.

5.3.2 Camada de Negócio

A camada de negócio é responsável pela manutenção do hiperdocumento de requisitos, oferecendo uma interface de alto nível para manipulação do modelo do documento de requisitos. A comunicação entre a camada de negócio e a camada de apresentação realiza-se segundo o modelo *Model View Controller* (MVC) (KRASNER; POPE, 1988). As interfaces gráficas oferecidas para o usuário representam um estado do modelo de negócio da aplicação: são as visões. O controlador recebe estímulos do usuário, geralmente conseqüências da interação do usuário com a visão oferecida da aplicação, e realiza alterações no modelo e gera novas visões, cumprindo assim um papel de intermediador entre as visões (usuário) e o modelo de negócio (aplicação). O modelo representa as classes que implementam o modelo de negócios da aplicação. O diagrama de classes da figura 5.11 representa esses três elementos e seus relacionamentos.

O controlador, na Wiki/RE, é implementado com o *framework* Struts (The Apache Software Foundation, 2001). A classe do *framework* que exerce a função de controlador, a `RequestProcessor`, foi especializada, criando-se a `WikiRequestProcessor`. Ela registra automaticamente o histórico de navegação, controla a internacionalização dos recursos da *wiki*, gerencia a sessão do usuário, inicializando e concluindo transações, e delega a um controlador de caso de uso o tratamento da requisição.

Os controladores de caso de uso são implementados através de especializações da classe `WikiAction`, que, por sua vez, é uma especialização da classe `DispatchAction` do Struts. Existe uma `WikiAction` para cada recurso suportado pela Wiki/RE e cada uma dessas classes

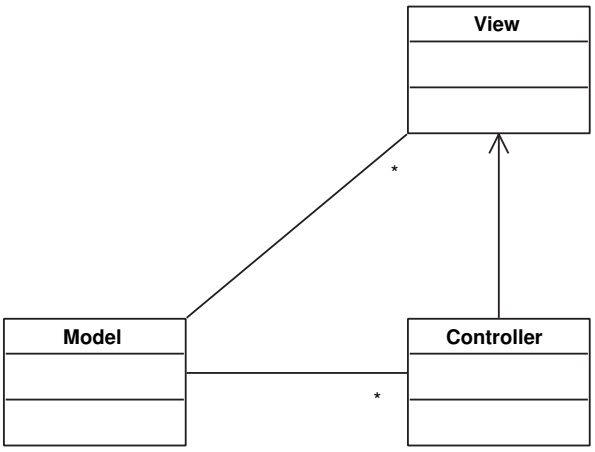

Figura 5.11: Diagrama de classes do Padrão Visão Modelo Controlador (MVC).

contém um método que representa um caso de uso. Por exemplo, o recurso `Wikipage` é tratado pela `WikipageAction` que contém os métodos `View`, `Edit` e `Create`.

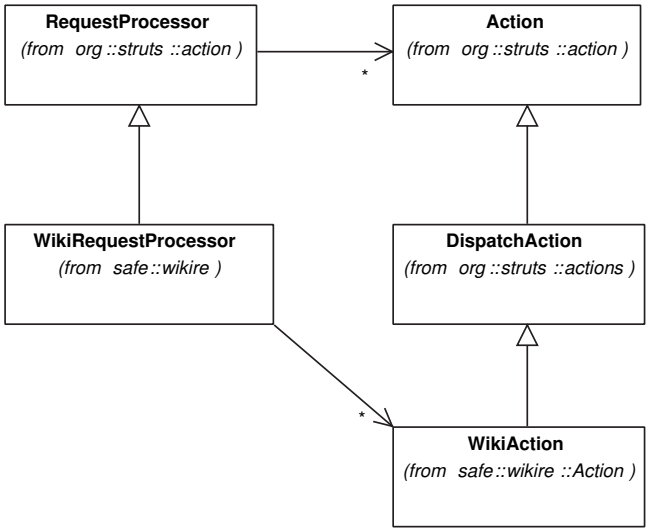

Figura 5.12: Diagrama de classes da parte dos controladores da implementação do padrão MVC na Wiki/RE.

O diagrama de classes da figura 5.12 fornece uma visão de alto nível da implementação da parte dos controladores do padrão MVC na Wiki/RE. Importante observar que, apesar das novas atribuições da classe `WikiRequestProcessor`, ele permite a delegação do controle para qualquer classe que o `RequestProcessor` original do Struts suporte.

A configuração do controlador `WikiRequestProcessor` e a quem ele delegará o controle são configurados no arquivo `struts-config.xml`. A cada `WikiAction` associa-se um caminho, que o `WikiRequestProcessor` utilizará para decidir qual a `WikiAction` apropriada para atender a requisição. A listagem 5.1 demonstra a declaração da configuração do controlador no `struts-config.xml`, definindo o `WikiRequestProcessor` como principal controlador,

que delegará a uma instância do `WikipageAction` toda requisição cuja *Uniform Resource Locator* (URL) combina com a expressão regular definida no atributo `path`.

Fragmento de código 5.1: Fragmento do `struts-config.xml`, exemplificando a configuração do controlador.

```
<controller
        processorClass = "safe.wikire.WikiRequestProcessor"
/>
<action
        path = "/projects/*/Wikipage"
        type = "safe.wikire.action.resource.WikipageAction"
        scope = "request"
        parameter = "action"
        attribute="WikipageForm"
        name="WikipageForm"
>
        <forward name="view" path="/wikipage.jsp" />
        <forward name="create" path="/createWikipage.jsp" />
        <forward name="edit" path="/editWikipage.jsp" />
        <forward name="rename" path="/editWikipage.jsp" />
        <forward name="refactor" path="/editWikipage.jsp" />
</action>
```

Os controladores são os responsáveis pela validação dos parâmetros enviados pelo usuário (verificação e conversão de tipo dos dados recebidos em cada requisição) e o acionamento das classes do modelo de negócios da aplicação. Na Wiki/RE, existe ao menos uma `WikiAction` para cada `Resource` e as classes especializadas de `Resource` (como a `Wikipage`) são as principais classes de negócio que interagem com os controladores. O diagrama da figura 5.13 demonstra essa relação entre os controladores e o modelo na Wiki/RE.

Após a realização de operações pelo controlador, uma nova visão é criada. Para cada caso de uso, existe, pelo menos, uma visão. Definem-se também visões globais, utilizadas pelo controlador `WikiRequestProcessor` para tratamento de erros e autenticação.

O diagrama da figura 5.14 modela as classes e os relacionamentos de controle e visão. As visões são arquivos da aplicação, disponibilizados para acesso pela Web, ou endereços para recursos externos (definidos por uma URL). Na Wiki/RE, as visões são sempre páginas *Java Server Pages* (JSP). O fragmento de código 5.1 demonstra como são definidas as visões para cada controlador: o atributo `name` contém o identificador da visão (nome este único no controlador em questão) e o atributo `path` a página JSP que será utilizada. O controlador, para enviar uma determinada visão ao usuário, precisa apenas especificar o nome da visão: o `RequestControler`, nesse momento, reassume o controle da operação e encaminha ao usuário[6] o arquivo definido em `path`.

[6] Na verdade, o arquivo é encaminhado ao servidor de aplicações, que realiza o processamento do arquivo (no caso de uma página JSP ou de um programa *Common Gateway Interface* (CGI)) e envia o resultado ao usuário.

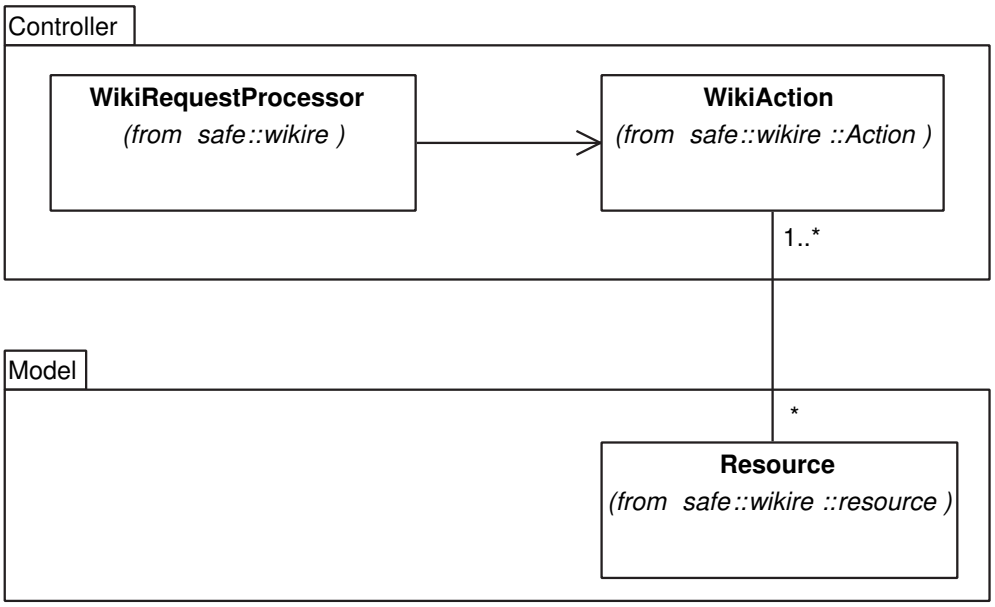

Figura 5.13: Diagrama de classes e pacotes da parte de controladores e modelo da implementação do padrão MVC na Wiki/RE.

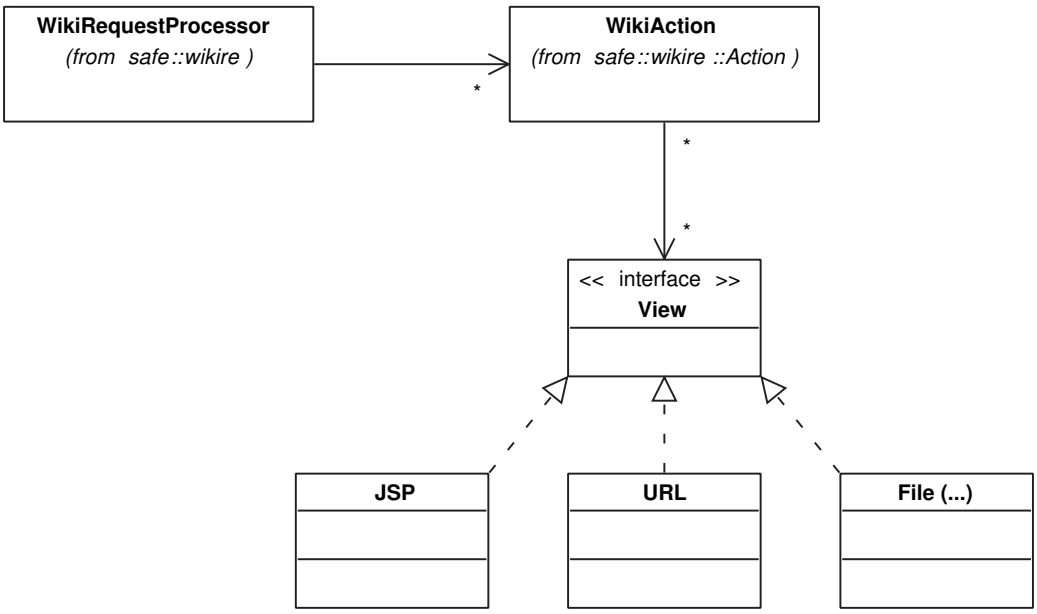

Figura 5.14: Diagrama de classes da parte de visão da implementação do padrão MVC na Wiki/RE.

As visões obtém seus dados do modelo a partir de objetos *Value Object* (VO) (FOWLER, 2002) preparados pelo controlador de caso de uso, evitando-se o acesso direto às classes do modelo de negócio para a obtenção dos dados. Essa prática foi adotada não apenas para

facilitar posteriores alterações no modelo e nas visões, mas também para desencorajar a realização de operações nas visões (mais precisamente nas páginas JSP), deixando-as a cargo apenas da apresentação do modelo (e não pela lógica). Na prática, realizar o preparo dos dados na classes de controle (quando os dados provenientes das classes de negócio não estão no formato requerido pelas visões) é mais simples do que fazê-lo nas classes de visões.

Devido a uma restrição de aplicações Web, as visões oferecidas do modelo são obtidas a pedido do usuário (modelo *push*) e não atualizadas automaticamente a cada alteração do modelo[7]. Pelo padrão MVC, alterações no modelo deveriam refletir automaticamente nas visões. Felizmente, essa característica não prejudica a utilização de ferramentas *wikis* como a Wiki/RE.

5.3.3 Camada de Apresentação

As visões são criadas a partir de estímulos enviados à camada de negócios e processados pelo controlador (padrão MVC). As visões dos diferentes tipos de recursos disponíveis na Wiki/RE possuem liberdade para a apresentação de seus dados, principalmente para operações de alteração. A única operação que é comum a todos os recursos é a apresentação do conteúdo do recurso (caso de uso "View wikipage").

Todo recurso possui um arquivo XML, que o representa, conforme mencionado na seção 5.3.1. Esse documento é utilizado para a geração do formato de saída desejado pelo usuário, utilizando-se transformações XSLT:

1. Obtêm-se todos os documentos XML do projeto correspondentes à revisão do documento XML alvo da renderização.

2. Aplica-se a transformação XSLT, de acordo com o tipo de saída especificado pelo usuário (HTML, PDF, etc).

O primeiro passo talvez pareça controverso. Na verdade, dado que vários documentos XML podem ser alterados ao longo do tempo, recuperar apenas o documento alvo na revisão desejada levaria a possíveis inconsistências, não mostrando um retrato fiel. Por exemplo, considere-se uma *wikipage* que cite o caso de uso "Transferir dinheiro", com a opção para mostrar o resumo do mesmo ativada, em sua versão 1. Posteriormente, altere-se a meta satisfeita pelo caso de uso "Transferir dinheiro", ocasionando a alteração de sua descrição (criando um novo documento XML para esse caso de uso). Denomine-se esta versão do modelo de 2. Assim, se solicitada a versão 1 da *wikipage* e recuperar-se apenas a versão 1 da mesma, o resultado será inconsistente, dado que o documento do caso de uso que será utilizado estará na versão 2 e a *wikipage* utiliza os dados desse documento. Logo, torna-se

[7] Seria possível atualizar as visões automaticamente caso fossem utilizados aplicações em Flash (Macromedia, 2000) ou *applets* Java.

necessário recuperar a versão 1 do caso de uso para manter-se a consistência dos dados da *wikipage* na versão solicitada. Uma notificação de que uma versão mais recente do caso de uso encontra-se disponível deve ser emitida, indicando que seria desejável a atualização do conteúdo da *wikipage* (o que manteria a consistência entre os dados mais atualizados).

Em resumo, quando apresentado um documento, todos os elementos utilizados devem ser da mesma versão. Neste ponto, a utilização do *SubVersion*, cujo sistema de controle de versão institui versões globais, resume o problema em obter uma cópia do repositório na revisão desejada. No caso do CVS, tal comportamento seria simulado através de marcações (*tags*), uma tarefa custosa se comparada com o SubVersion.

5.4 Projeto

O modelo de hiperdocumento da Wiki/RE, apresentado na figura 5.15, assemelha-se ao proposto na seção 4.5 do capítulo 4. Ele é constituído de vários objetos do tipo `Resource`. Um `Resource` é uma abstração para os vários tipos de conteúdos, que podem ser armazenados em uma *wiki* (páginas de texto simples, figuras, arquivos binários, etc) e são identificados unicamente pelo nome e o tipo concreto. Desse modo, estabelece-se um espaço de nomes global, que reúne todos os nomes de todos os `Resources`, e um espaço de nomes para cada tipo concreto de `Resource`. Para os espaços de nomes de cada tipo concreto, não se admite nomes repetidos. O espaço de nomes global, no entanto, admite-os. O mecanismo de resolução de nomes de uma `Wiki`, conseqüentemente, pode fornecer vários recursos (de tipos distintos) quando inquirido por um nome.

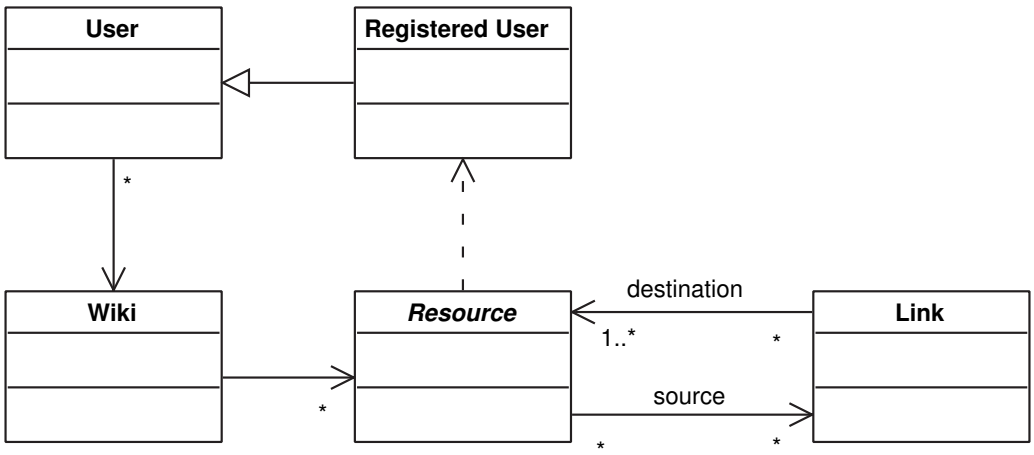

Figura 5.15: Diagrama conceitual do modelo de hiperdocumento da Wiki/RE.

O ciclo de vida de um `Resource` inicia-se pela sua criação por um `Registered User`. A esse usuário atribui-se o direito autoral (*copyright*) sobre o recurso. Futuramente, poder-

se-á implementar um mecanismo de controle de direitos autorais, preservando a natureza livre da informação[8]. Os recursos podem ser acessados por qualquer usuário, mas alterados apenas por usuários registrados a fim de evitar as adulterações e os vandalismos nos recursos, problemas esses descritos nos casos de mau uso da seção 5.2.4.

Os `Resources` podem ser associados com outros `Resources` através de ligações (`Links`). As âncoras-origem, quando existem, são definidas nos `Resources`, mas não é possível designar uma âncora-destino. Se um trecho do `Resource` é importante o suficiente para ser citado, o trecho do `Resource` deve ser transformado em um `Resource` distinto.

O modelo de um projeto de software (figura 5.16) apresenta a mesma simplicidade do modelo de hiperdocumentos. Todo projeto, representado por `Project`, é constituído por vários modelos (modelo de caso de uso, classes, etc). Cada modelo requer, para sua construção, vários elementos (`Model Element`), que são criados e alterados por engenheiros de software (`Software Engineer`).

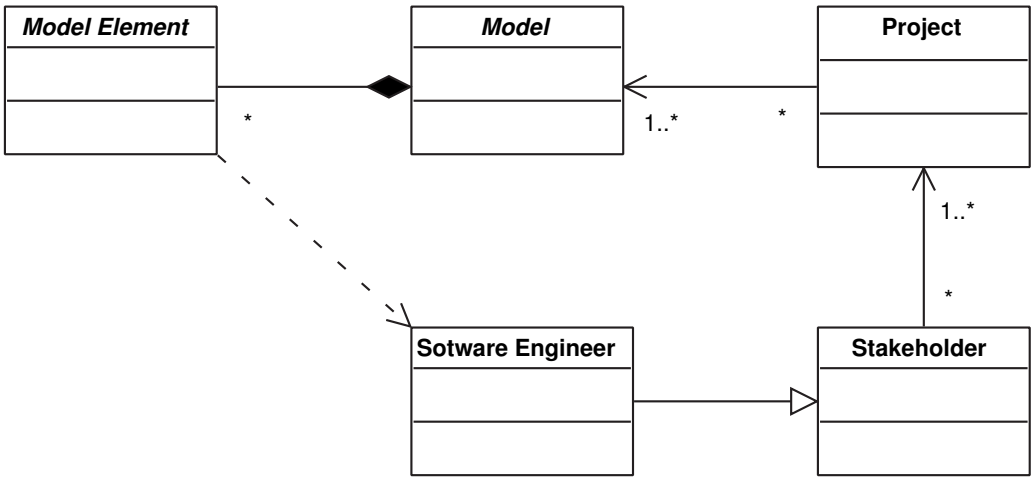

Figura 5.16: Diagrama conceitual de um projeto de software na Wiki/RE.

A Wiki/RE permite a criação de hiperdocumentos de requisitos que são uma representação dos modelos de requisitos que constituem o projeto de software. O diagrama da figura 5.17 associa os elementos de ambos os modelos, hiperdocumentos e projeto de software, evidenciando a capacidade para a completa documentação dos artefatos de requisitos de um projeto de software.

O objetivo da paridade entre elementos de ambos os modelos é permitir que a documentação dos elementos dos modelos de um software usufrua das mesmas facilidades disponíveis no desenvolvimento de hiperdocumentos *wiki*: geração incremental, unificação e convergência.

[8] Licenças para documentos já começaram a se difundir na comunidade de software livres.

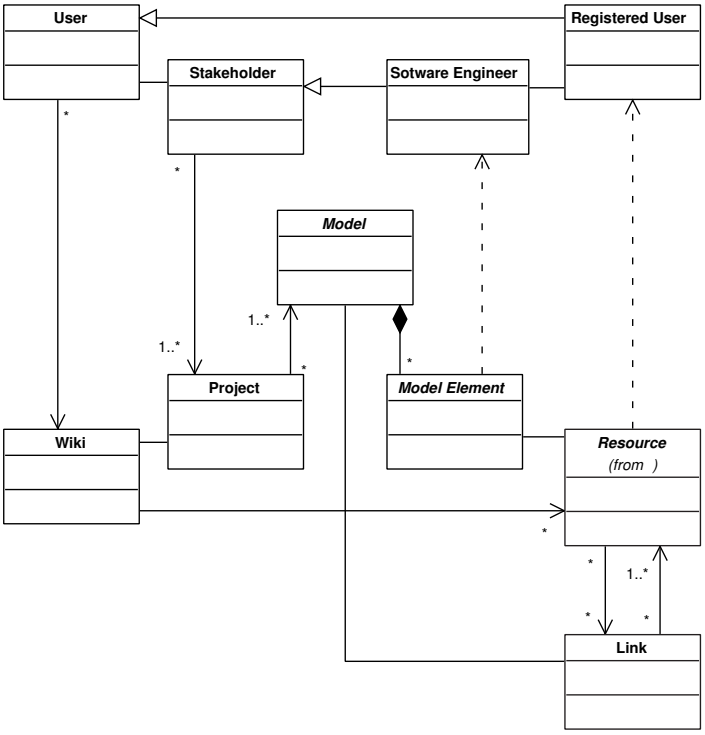

Figura 5.17: Diagrama conceitual do mapeamento entre hiperdocumentos e projetos de software.

5.5 Implementação

A camada de persistência utiliza os serviços do Hibernate para armazenar o estado dos objetos. O Hibernate apresenta a estrutura observável na figura 5.18. O `SessionFactory` é o responsável pelo mapeamento objeto-relacional. Ele é criado durante a inicialização da Wiki/RE, a partir de um objeto `Configuration`, que contém as especificações dos mapeamentos, a configuração de acesso ao servidor de banco de dados e a escolha do gerenciador de transações. Todas as alterações do estado de um objeto ocorrem no contexto de transações (`Transaction`), controladas pelo gerenciador de transações, que pode ser o do servidor de banco de dados (`SBGD`) ou um que atenda às especificações JTA (CHEUNG; MATENA, 2002) (geralmente disponível em servidores da aplicações J2EE). As transações iniciam-se de uma sessão (`Session`), criada a partir do `SessionFactory`. Objetos do tipo `Query` são utilizados para recuperar objetos armazenados na base de dados.

O controle de versões (diagrama da figura 5.19) realiza-se por `RepositoryTransactions`. Ela é uma classe pública, porém abstrata. Classes concretas do tipo `Repository-`

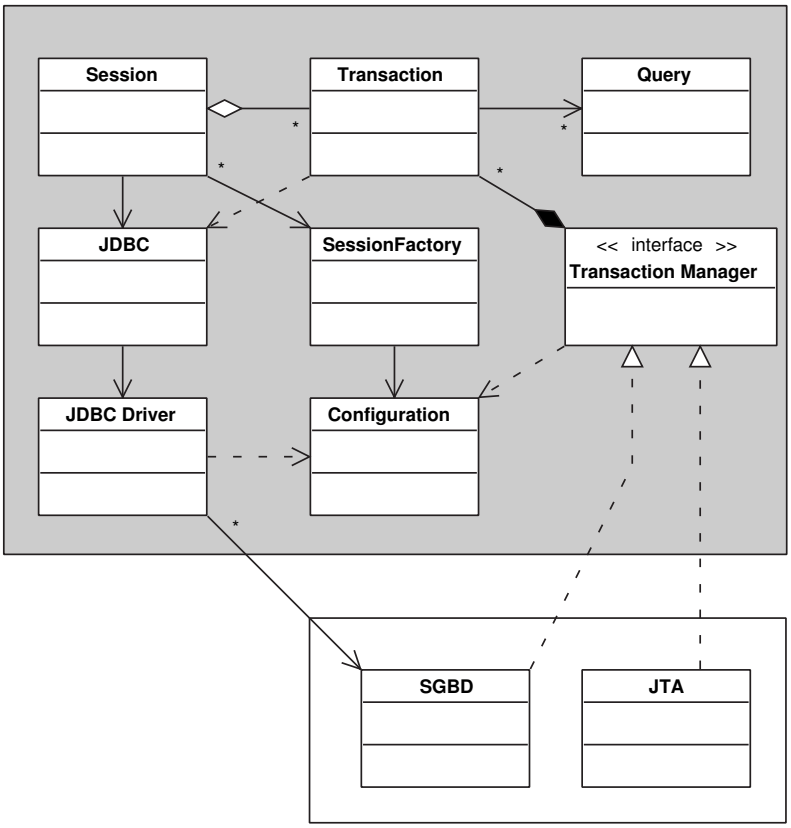

Figura 5.18: Diagrama do interior da camada de persistência da Wiki/RE sob a perspectiva do Hibernate.

`Transaction` são obtidas do `RepositoryTransactionFactory`. A implementação de `RepositoryTransaction` é escolhida a partir da análise do `Repository` em uso (uma classe abstrata). A Wiki/RE suporta o controle de versões através do Subversion, disponibilizando as respectivas classes `SubversionRepository` e `SubversionRepositoryTransaction` (as classes `DummyRepositoryTransaction` e `FilesystemRepositoryTransaction` são utilizadas interinamente a título de testes).

A camada de utilitários, conforme o diagrama de classes da figura 5.20, contém a estrutura de dados `CompressedLinkedList` (uma lista dinâmica comprimida com o algoritmo RLE) e métodos estáticos para manipulação de arquivos (`IOUtil`) e internacionalização (`LocaleUtil`).

A implementação da camada de negócio (figura 5.21) é essencialmente idêntica àquela definida na seção Projeto (5.4), diferenciando-se pela definição de classes concretas de `Resource` e a identificação dos recursos relativos a elementos do modelo de projeto de software através da implementação da interface `Model Elements` (seguindo o padrão "Marcação").

O controle da aplicação é determinado pelo `WikiRequestProcessor` (figura 5.22). Ele é iniciado pelo `ActionServlet`, um *servlet*, definido pelo Struts, encarregado de receber as requisições HTTP dos usuários e, principalmente, configurar o *framework* MVC. O `WikiRe-`

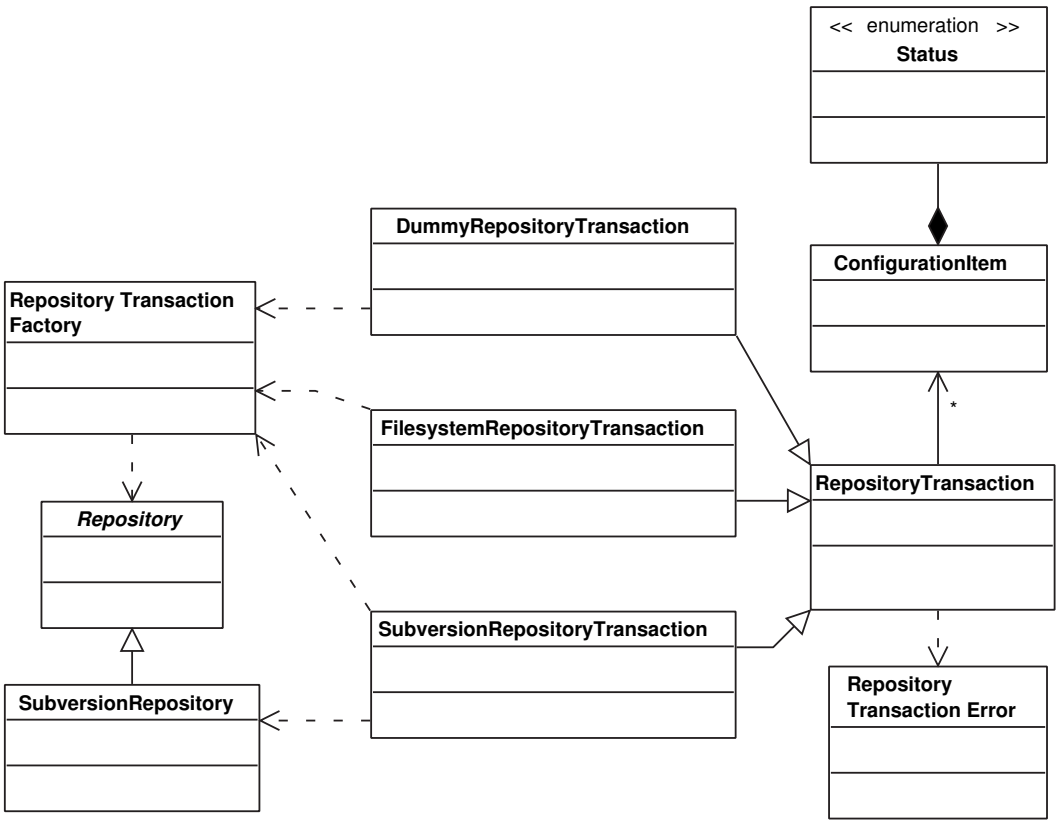

Figura 5.19: Diagrama das classes que implementam e coordenam o controle de versões na camada de persistência da Wiki/RE.

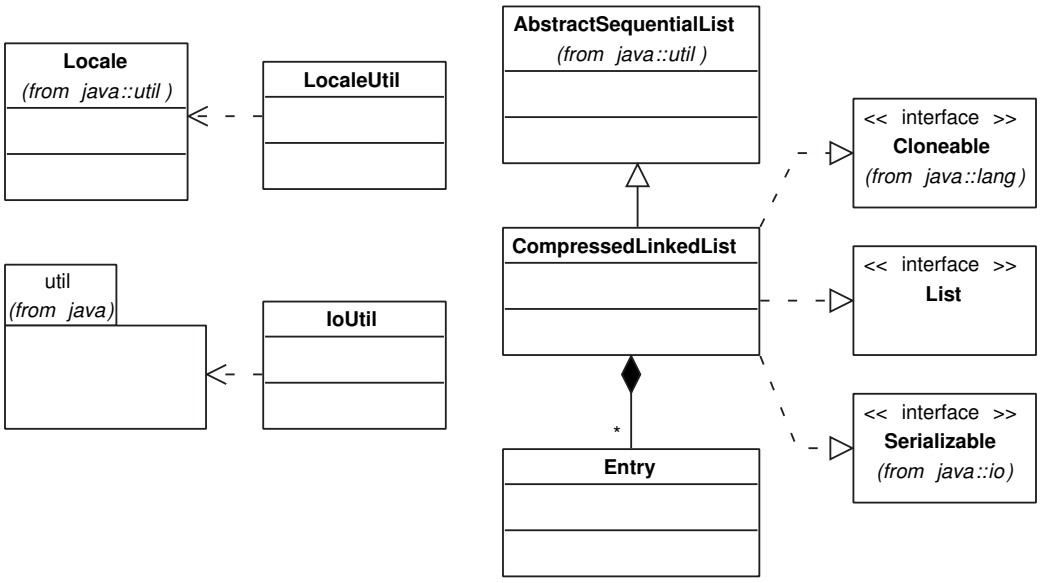

Figura 5.20: Diagrama das classes que implementam a camada de utilitários da Wiki/RE.

`questProcessor` é incumbido de delegar, de acordo com os parâmetros da requisição e sua configuração, o processamento à implementação apropriada de `Action`.

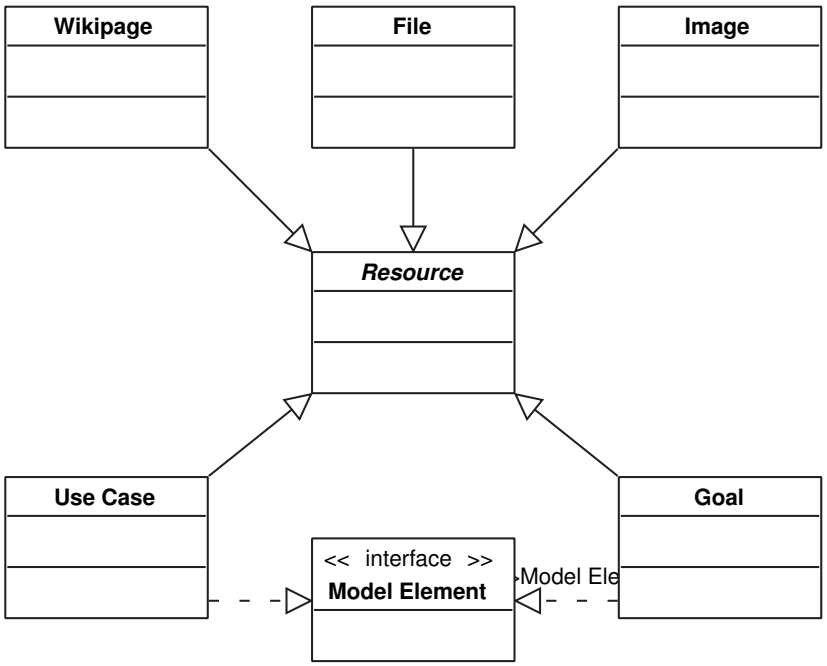

Figura 5.21: Diagrama das classes de recurso da Wiki/RE.

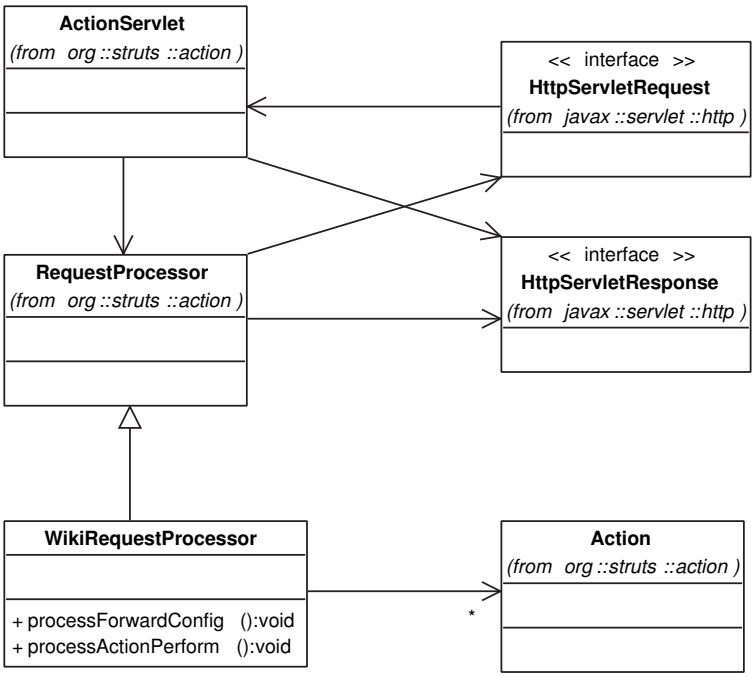

Figura 5.22: Diagrama das classes que implementam o controlador (do padrão MVC) da Wiki/RE.

O `WikiRequestProcessor` inicia, automaticamente, uma sessão HTTP para o usuário. A essa sessão, associa-se uma sessão de aplicação, a `WikiSession`, iniciando-se a interação com o modelo de negócio da Wiki/RE (figura 5.23). Para cada operação a ser realizada, cria-se uma transação (`WikiTransaction`), que encapsula as transações do Hibernate (`Transaction`)

e do repositório (`RepositoryTransaction`). Instâncias de `WikiAction`, no contexto de uma `WikiTransaction`, realizam as operações no domínio de negócio.

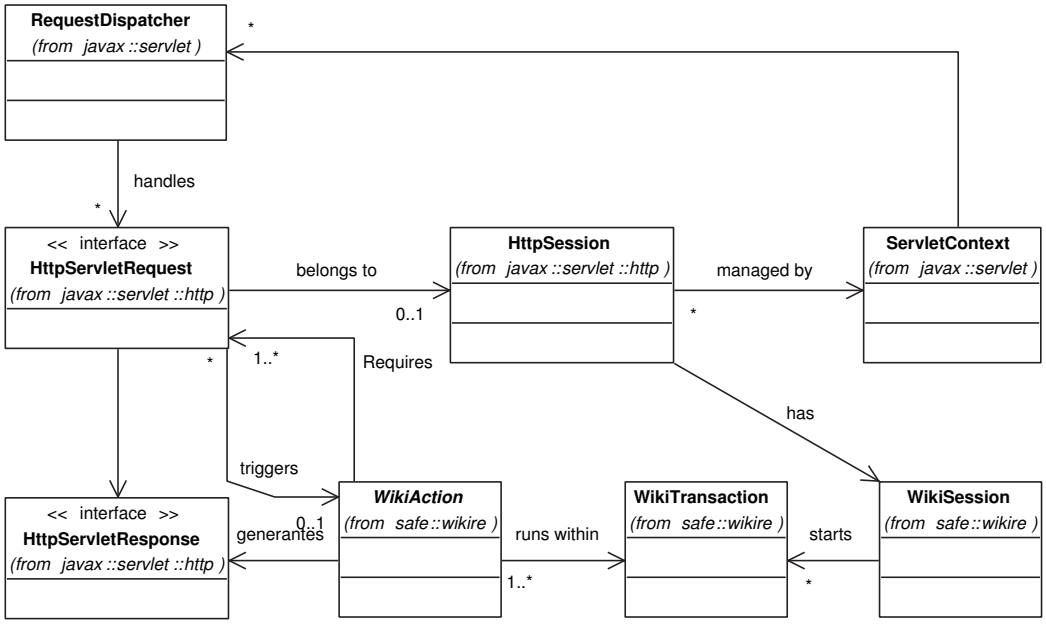

Figura 5.23: Diagrama das classes que controlam a interação entre a camada de negócio e de apresentação da Wiki/RE.

Existe uma hierarquia de classes derivadas `WikiActions` servindo a diferentes propósitos (diagrama de classes da figura 5.24). Para manipulação de objetos da camada de negócio, de modo geral, a própria `WikiAction` pode ser utilizada. Ela provê acesso direto ao `WikiSession`, do qual pode-se obter qualquer objeto (`Resource`) do modelo.

Para sistemas Web, métodos como o OOHDM (SCHWABE; ROSSI; BARBOSA, 1996) utilizam especialização de diagramas de estado. O conceito demonstra-se conveniente para a manipulação de `Resources` na Wiki/RE. Cada requisição HTTP atua como um estímulo que altera os estados. Em cada estado, um conjunto de operações são executadas e prossegue-se ao próximo estado ou espera-se um novo estímulo. O diagrama de máquina de estados, na figura 5.25, ilustra o funcionamento de um `StateAction` criado para a realização de operações CRUD em um recurso `Repository`. Uma requisição ativa o estado ortogonal de `RepositoryAction`. Determina-se, a partir de um parâmetro da requisição, qual o estado que será ativado (apenas um dos estados aninhados pode ser ativado). No caso de `Resources`, o argumento de guarda é o tipo de ação que se deseja executar.

Os operações do padrão CRUD, no entanto, não são implementadas com o `StateAction` na Wiki/RE. A classe `ResourceAction` implementa as operações básicas de manipulação

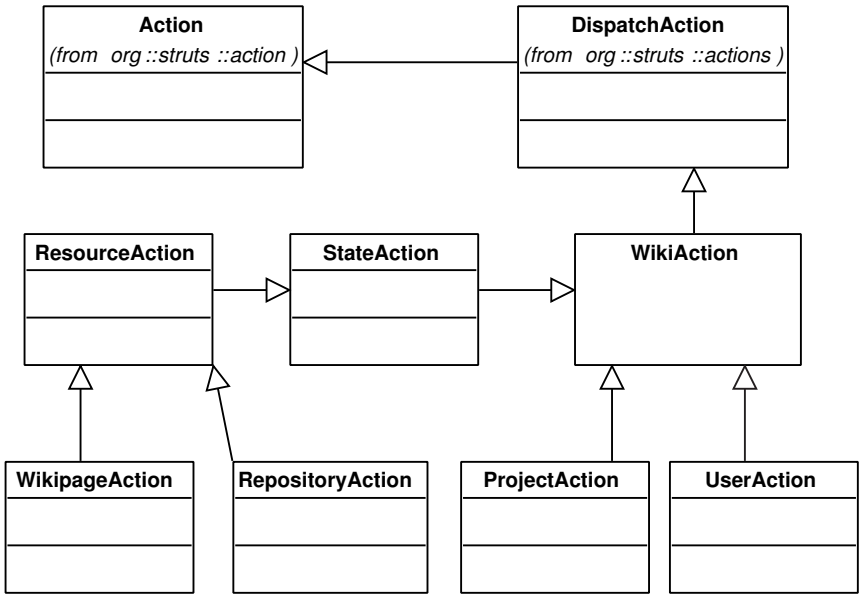

Figura 5.24: Diagrama das classes do tipo `Action` da Wiki/RE.

de `Resource`, facilitando, assim, a criação de novos tipos de `Resources` e melhorando a capacidade de extensibilidade da Wiki/RE.

5.6 Testes e Garantia de Qualidade

Implementaram-se casos de teste unitários, para as principais classes que constituem a Wiki/RE, com o *framework* JUnit (GAMMA; BECK, 2001). A execução dos casos de testes, ao final de cada iteração, permitiu, além da detecção de falhas na implementação das funcionalidades desejadas para a iteração, a detecção de regressões no código-fonte do projeto. Utilizou-se o Ant (Apache Software Foundation, 2000) para a execução dos casos de teste ao final de cada iteração (com o objetivo de detectar regressões) e o Eclipse (Eclipse Foundation, 2000) durante o desenvolvimento (o Eclipse oferece recursos de depuração e alteração de código integrados à execução das unidades de teste em JUnit, o que torna o processo mais ágil).

Além dos testes, executaram-se, periodicamente, revisões automatizadas do código, utilizando ferramentas de análise estática: Hammurapi (VLASOV et al., 2004), PMD (COPELAND et al., 2002) e Checkstyle (BURN, 2001). Elas verificavam desde regras de formatação de código até práticas não recomendadas e erros comuns de programas escritos em Java.

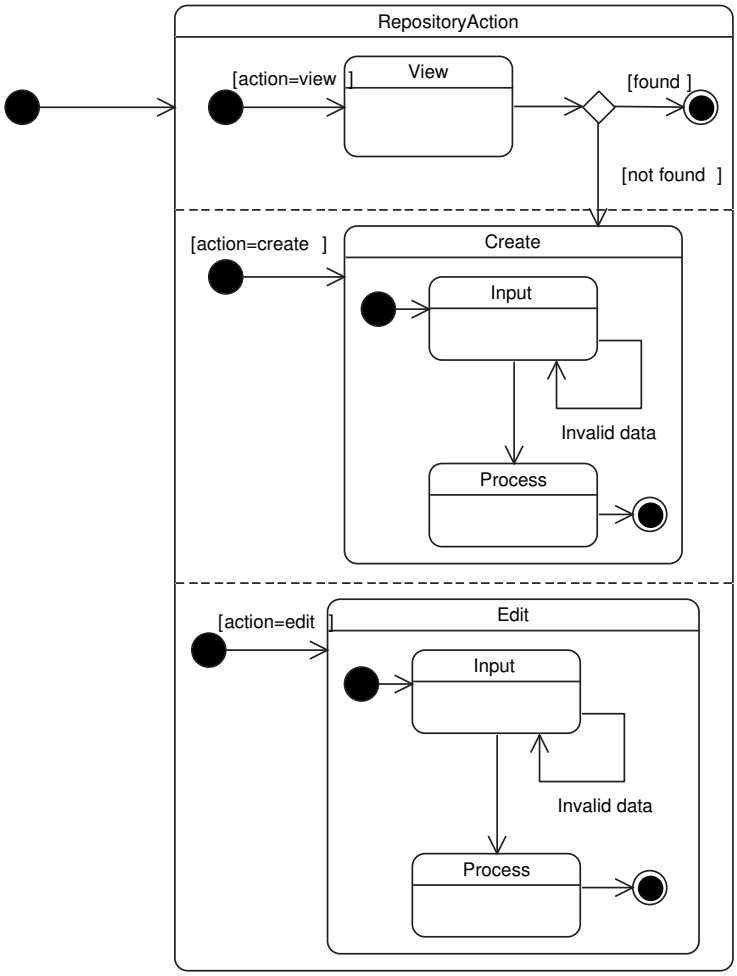

Figura 5.25: Diagrama de estados da atividade `RepositoryAction`.

5.7 Implantação

A Wiki/RE é uma aplicação cliente/servidor. Ela atua como um servidor para agentes Web (navegadores Web, *crawlers*), atendendo, simultaneamente, várias requisições de vários clientes. A Wiki/RE também age como um cliente, necessitando de um servidor de banco de dados e de um ou mais servidores de repositórios de controle de versão. O diagrama de implantação (figura 5.26) representa as dependências da Wiki/RE e demais componentes computacionais.

5.8 Considerações Finais

A ferramenta Wiki/RE foi desenvolvida com uma adaptação, simplificada e acadêmica, do *Unified Process*. A natureza iterativa do processo refletiu-se na evolução da documentação, que foi armazenada em uma *wiki*, por meio da ferramenta CoTeia. A execução das atividades

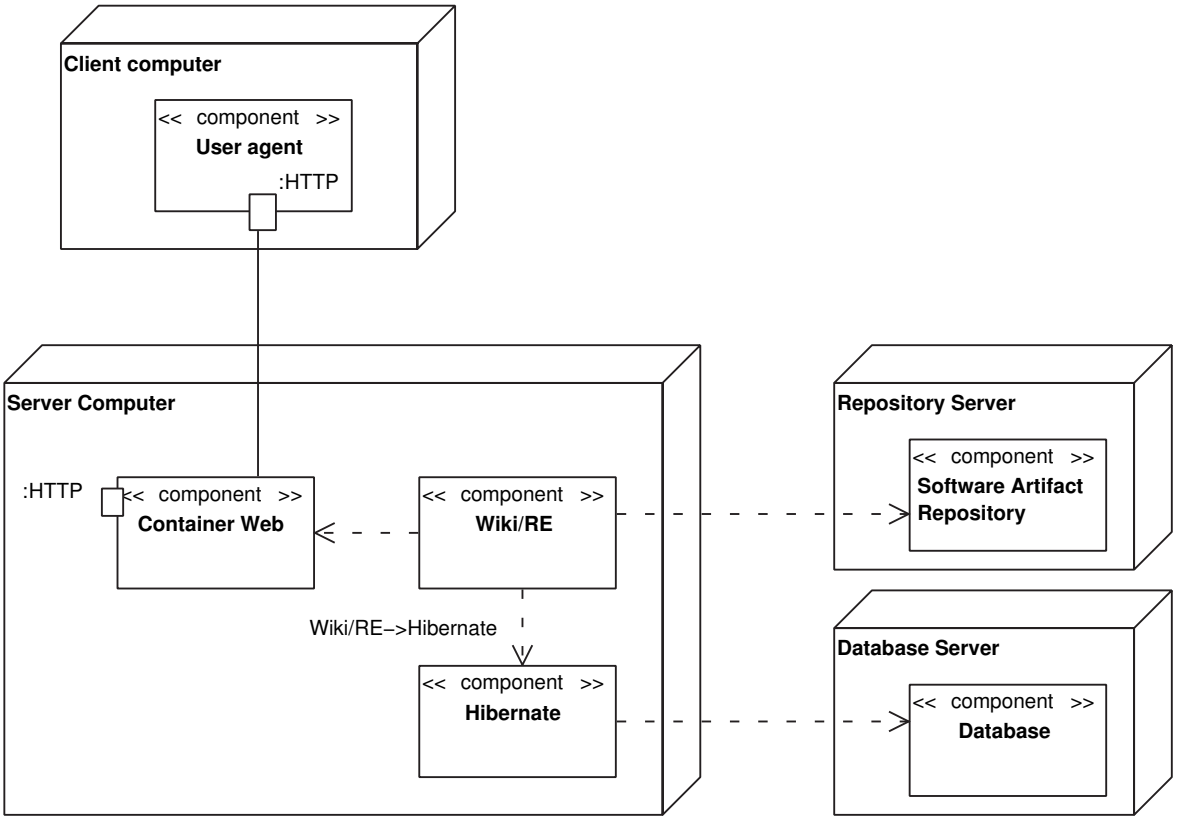

Figura 5.26: Diagrama de implantação da Wiki/RE.

de engenharia de software e a evolução do documento ocorreu de modo natural: documentar deixou de ser uma tarefa tediosa e passou a ser uma ação corriqueira.

O uso de uma ferramenta *wiki* para a documentação do projeto inteiro permitiu uma rica investigação de rastreabilidade, desde dados do domínio do sistema (do qual iniciou-se a elicitação de requisitos) até referências a projetos. Esse mesmo uso também detectou deficiências no modelo, como a perda de rastreabilidade quando páginas eram renomeadas[9] ou divididas. Esses e outros pontos, como o suporte a vários tipos de dados, foram cuidadosamente analisados e soluções adequadas foram elaboradas e expostas neste capítulo. Questões de suma importância ao público alvo da ferramenta, a comunidade de software livre, foram observadas durante todo o processo, visando à construção de um produto de qualidade: segurança, internacionalização, respeito a direitos autorais, utilização de apenas ferramentas livres no desenvolvimento, controle de qualidade.

Traduzindo a Wiki/RE em alguns números, atualmente ela possui 8.677 LOC de código-fonte em Java, 6.351 linhas de comentários (notas de *copyright* e documentação da API no formato suportado pelo Javadoc) e 819 LOC de código-fonte em JSP.

[9] Não existia uma operação para renomear uma página *wiki*. Essa atividade simulava-se através da criação de uma nova página, com o novo nome desejado, e da cópia dos dados da página antiga para a nova.

Capítulo

6

Conclusão

A adequada documentação de requisitos é essencial ao desenvolvimento de software de qualidade. Inúmeras dúvidas existem quanto ao processo de engenharia de requisitos em projetos de software livre. Não se observa a existência de especificações de requisitos, como aquelas encontradas em projetos comerciais não livres. Seria, portanto, uma contradição a produção de software de qualidade por tais projetos.

Algumas pesquisas esclarecem pontos importantes nesse processo de engenharia de requisitos tão nebuloso. Reis (2003) confirma dois pontos. O primeiro é a importância de especificações abertas e de aplicações que sirvam de exemplo (ou contra-exemplo) à desejada. Elas reúnem um abrangente e correto conhecimento do sistema, o que resume a elaboração dos requisitos.

O segundo ponto que Reis comprova é o envolvimento do desenvolvedor não apenas como um funcionário realizando um trabalho, mas como alguém que é um usuário efetivo, e geralmente o principal, dos projetos em que participa. De longa data, sabe-se a importância do envolvimento dos usuários e demais interessados no desenvolvimento do software, principalmente durante o desenvolvimento dos requisitos. Os desenvolvedores reúnem, em uma única pessoa, as duas figuras: um especialista no domínio do negócio e um especialista no desenvolvimento de software. Uma conseqüência direta é que os problemas notórios de comunicação que existem entre desenvolvedores e usuários deixam de existir: eles compreendem o que estão falando e a troca de informações é instantânea.

Em um cenário como esse, uma das conclusões de Scacchi (2002) é plenamente compreensível: a exposição dos requisitos apenas após a sua implementação. O código-fonte e a

satisfação do desenvolvedor para com seu trabalho é um indicador de que os requisitos que ele especificou realmente estavam corretos.

Entretanto, há ocasiões de conflito entre os requisitos, a partir do momento em que eles, o requisito e o código-fonte, são publicados. Outros desenvolvedores, que participam do projeto, passam a observar e analisar aspectos até então ignorados: o impacto das alterações no software como um todo, o cumprimento de práticas seguras de programação, os algoritmos utilizados, a necessidade ou não da funcionalidade proposta, o respeito à propriedade intelectual, o correto funcionamento em situações extremas, etc. Nesse instante, inicia-se uma discussão para tratar dos conflitos, geralmente por listas de email. As mensagens contêm não apenas alterações no código, mas também o raciocínio responsável por elas, o que, em muitos casos, reflete-se em requisitos. A consolidação da discussão (e dos requisitos), em diversos casos, ocorre em documentos simples, armazenados no mesmo repositório do código-fonte. Esses arquivos, conforme atestam Yamauchi et al. (2000), encontram-se dispersos, mas são uma importante fonte de consulta para os desenvolvedores, principalmente os principiantes.

O que se observa em projetos de software livre é que existe documentação de requisitos, porém ela se encontra dispersa e fragmentada. São os requisitos e respectivo código-fonte, os emails, os arquivos no repositório, as especificações abertas. Reunir, de maneira ordenada, esses artefatos não é uma tarefa simples.

Em 1989, Shneiderman e Kearsley definiram as três regras de ouro (*The Three Golden Rules*) que estabelecem as condições às quais um sistema hipertexto é apropriado para a tarefa de organização de informações:

1. Existência de muita informação.

2. Informações compostas de pedaços menores de informação.

3. Os relacionamentos entre os pedaços de informação podem ser definidos e representados.

O volume de informações envolvido no desenvolvimento de software é inegável. Em software livre, todo esse contingente deve ser público, livre para redistribuição. Não existem grandes documentos que reúnam todas as informações: elas são geradas incrementalmente, em um esforço colaborativo em um ambiente distribuído coordenado por ferramentas díspares. Entretanto, todas elas corroboram em um único produto.

A coincidência entre as "três regras de ouro de hipertextos" e a disposição dos artefatos dos projetos de software livre não pode ser desprezada. E não é. A maioria dos projetos possuem *sites* Web, sistemas hipermídias bem simples, que tentam, se não reunir, ao menos indicar a localização das informações desejadas.

A manutenção de tais *sites*, no entanto, não acompanha o ritmo de desenvolvimento do software. Eles são desprovidos de capacidades de edição colaborativa, centralizando a

responsabilidade pela atualização a um grupo restrito de pessoas. Para contornar esse problema, diversos projetos de software livre começaram a adotar sistemas *wikis* para realizar essa tarefa.

As *wikis* resolvem satisfatoriamente a questão de atualização das informações, mas deixam outras questões ainda em aberto. Elas são mecanismos genéricos de documentação e os tipos de dados que suportam restringem-se a textos, figuras, tabelas, etc. Não existe a representação de objetos do domínio de negócio: casos de uso, requisitos não-funcionais, documentação de APIs. A representação dos artefatos de engenharia de requisitos como tais permitiria o uso mais eficiente de *wikis* e possibilitaria o tratamento diferenciado das relações entre as informações, como, por exemplo, na forma de rastreabilidade entre os artefatos.

Neste trabalho, aperfeiçoou-se o conceito de *wikis*, relacionando-se o modelo de hipertexto, intrínseco das mesmas, ao modelo de engenharia de requisitos. A união do caráter incremental e convergente das *wikis* permitiram a criação de hiperdocumentos de requisitos enxutos, adequados ao processo de software livre.

A Wiki/RE, ferramenta desenvolvida neste trabalho, permite a elaboração de hiperdocumentos de requisitos de acordo com esse modelo, utilizando a estratégia incremental de *wikis* para permitir que a rastreabilidade, tão desejada em documentos de requisitos, seja obtida sem custos adicionais. Em outras palavras, a rastreabilidade é um bem-vindo efeito colateral do uso de *wikis*.

Uma parcela significativa da documentação gerada durante o desenvolvimento da Wiki/RE foi criada em um sistema hipermídia tipo *wiki*, a ferramenta CoTeia. A construção desse hiperdocumento evidenciou as capacidades de rastreabilidade provenientes de documentos *wiki*, mas também destacou algumas deficiências na aplicação de *wikis* ao domínio de Engenharia de Requisitos:

- a perda de rastreabilidade ao dividir uma página *wiki*;
- a dificuldade imposta para renomear uma página *wiki*;
- um único espaço de nome, independente do tipo do dado inserido;
- os poucos tipos de dados suportados.

As vantagens de rastreabilidade, obtidas pelo uso de *wikis*, seriam perdidas após períodos longos de desenvolvimento caso prevalecessem esses problemas. A Wiki/RE elabora soluções que permitem a manutenção da rastreabilidade sem ônus aos seus usuários.

6.1 Contribuições Deste Trabalho

A principal contribuição deste trabalho foi o desenvolvimento de uma ferramenta *wiki*, a Wiki/RE, especializada para o domínio de Engenharia de Requisitos e voltada para o processo de software livre.

A criação da ferramenta não seria possível sem a análise realizada sobre *wikis* e suas implementações, elementos essenciais à criação de modelos que representam adequadamente os conceitos de *wikis*. A utilização de metas e *soft-goals*, outro fruto dessa análise, guiaram as decisões quanto às alterações a serem realizadas no modelo de hiperdocumento utilizado na Wiki/RE.

A manutenção e evolução da ferramenta *wiki* CoTeia, com o seu lançamento como software livre, sob a licença GPL, contribuiu não apenas para a definição das características, específicas à engenharia de requisitos, do hiperdocumento, mas também para a comunidade de usuários da CoTeia.

6.2 Trabalhos Futuros

Duas vertentes de trabalhos futuros são facilmente identificáveis. Uma é a evolução da ferramenta Wiki/RE. No presente momento, ela não explora todo o seu potencial para apoiar o processo de engenharia de requisitos. O objetivo, neste ciclo, foi permitir a rastreabilidade dos artefatos gerados pelas diversas técnicas de Engenharia de Requisitos, o que exigiu mudanças importantes no modelo de *wikis*. Essa capacidade permitiu não apenas a construção de documentos rastreáveis, mas ofereceu as condições necessárias para evoluir o documento e manter a rastreabilidade automaticamente. No entanto, outros aspectos da ferramenta, tais como auxílio na criação de relacionamentos, usabilidade e verificação de atributos de qualidade do documento, não foram explorados.

A segunda vertente de trabalho é o estudo sobre o processo de engenharia de requisitos em softwares livres. São poucas e insuficientes as pesquisas sobre a engenharia de requisitos nos projetos de software livre, abreviando a compreensão sobre seus mecanismos. Essa ignorância traz inconvenientes, econômicos e tecnológicos, tanto para empresas que desejam utilizar softwares livres em seus produtos, quanto para a comunidade que os mantém.

6.2.1 Ferramenta Wiki/RE

Segundo (REIS, 2003), o processo de software livre caracteriza-se por quatro estágios: (1) criação, (2) lançamento público, (3) crescimento e organização e (4) maturidade. O desenvolvimento da Wiki/RE encontra-se no início do segundo estágio. Um dos trabalhos futuros

é, logicamente, a continuidade da ferramenta, seguindo o caminho natural da evolução de um software livre.

O primeiro trabalho futuro, e que deverá ser realizado o mais rapidamente possível, é o lançamento público da Wiki/RE. Para isso, é necessária a criação de um *site*, para a disponibilização dos arquivos, e o anúncio da ferramenta perante a comunidade. Para disponibilizar os arquivos, o uso de *sites* hospedeiros de projetos de software livre, como o *Sourceforge*[1] e a Incubadora FAPESP[2], se apresentam como a melhor opção. A Incubadora FAPESP diferencia-se pela sua localização no Brasil, boa infra-estrutura e a utilização de repositórios Subversion (o mesmo atualmente em uso pela Wiki/RE), sendo, portanto, uma boa alternativa para hospedar a Wiki/RE. O anúncio do lançamento da Wiki/RE realizar-se-ia em diretórios de software livre, tal como o *freshmeat.net*[3].

Concluído o segundo estágio do processo de software livre, a Wiki/RE caminharia para o crescimento e organização. Nesse momento, o processo utilizado para o desenvolvimento da ferramenta se alteraria, assumindo caráter distribuído. Apesar do processo adotado até este instante, descrito no início do capítulo 5, possuir uma importante influência do processo de software livre, não é possível garantir a sua execução pelo restante da equipe de desenvolvimento nesse novo contexto. Apesar disso, é possível incentivar o uso do processo ou, pelo menos, das práticas essenciais do mesmo, de modo a garantir a qualidade do software. Para isso, é essencial que o processo seja documentado e publicamente disponibilizado, com a precisa descrição das tarefas a serem realizadas e a sugestão de boas práticas para a execução das mesmas, em local de fácil acesso.

Um segundo item importante nesse estágio do processo é indicar o que as pessoas interessadas podem fazer pelo projeto. Essas listas de afazeres (*todo*), como apontam Yamauchi et al. (2000), são um importante ponto de partida para trabalhos espontâneos em projetos de software livre. Algumas das tarefas que poderiam ser desenvolvidas para a Wiki/RE são[4]:

- criação semi-automática de relacionamentos, nos moldes do trabalho de Dag et al. (2005) (talvez até utilizando a ferramenta, livre, que eles desenvolveram, a ReqSimile[5]);
- implementar um glossário utilizando o modelo *Asynchronous JavaScript Technology and XML* (AJAX);
- avaliar a usabilidade da ferramenta, considerando diferentes perfis de usuários (clientes, desenvolvedores principiantes e desenvolvedores experientes);

[1] `http://www.sourceforge.net`
[2] `http://incubadora.fapesp.br`
[3] `http://www.freshmeat.net`
[4] Os itens foram descritos como em uma lista de afazeres real, de modo direto e sem muitos detalhes. O propósito é fornecer apenas um mapeamento de idéias, ao invés das instruções completas para a realização das tarefas(SCHMIDT, 1997), oferecendo aos indivíduos liberdade para inovação durante a implementação das mesmas (CUSUMANO; SELBY, 1997).
[5] `http://reqsimile.sourceforge.net`

- melhorar a usabilidade da ferramenta (com base na avaliação anterior);
- verificar automaticamente atributos de qualidade do requisito e do documento (por exemplo, analisando o *Glossary Circularity* (GLC) e o *Minimality of Vocabulary* (MOV));
- implementar um mecanismo de pesquisa (por exemplo, utilizando o Lucene (The Apache Foundation, 2000)).

Essa é uma breve lista de idéias para melhorias da Wiki/RE que podem ser implementadas a curto prazo. A médio prazo, deve-se buscar a integração da Wiki/RE com outras ferramentas de engenharia de software, o que permitiria um melhor e mais completo gerenciamento dos requisitos (HOFFMANN et al., 2004). No contexto do projeto SAFE, ao qual a Wiki/RE pertence, essa integração poderá ser estudada, realizada e avaliada, permitindo, assim, que os projetos de software livre usufruam, pela primeira vez, de uma solução, livre, completa para apoio ao desenvolvimento de software.

Finalmente, espera-se que futuros trabalhos integrem, à Wiki/RE, suporte a outras técnicas de engenharia de requisitos. Seria desejável que todos os métodos e técnicas descritos no capítulo 2 estivessem disponíveis na ferramenta. Acredita-se que as características da ferramenta e sua disponibilização como um software livre encorajem a sua utilização em futuras pesquisas.

6.2.2 Engenharia de Requisitos em Software Livre

A popularização e a boa qualidade dos softwares livres atraíram a atenção de empresas, que começaram a lançar, recentemente, os primeiros produtos de consumo em massa[6] equipados com softwares livres (celulares, aparelho de DVD, etc). Obviamente, a produção desses produtos não é tão simples quanto escolher um ou dois programas e inseri-los na memória dos aparelhos. São necessárias alterações nos softwares, adaptando-os às características intrínsecas do produto alvo. Para realizar essas modificações, requer-se a compreensão do software: requisitos, projeto, código. O primeiro problema é justamente que, apesar do código estar disponível, a documentação de projeto é escassa e a de requisitos praticamente inexistente. Apesar de ser um problema significativo, em termos de custo e tempo para a empresa, provavelmente ele apresenta um preço menor do que o desenvolvimento completo de um programa equivalente ou do que a compra de uma solução de terceiros.

O segundo problema, no entanto, trata-se do retorno das alterações das modificações para o software livre original. Deve-se lembrar que a maioria das licenças de software livre requer a distribuição do código alterado. Trata-se de uma salvaguarda que permite não apenas garantir a qualidade do software, mas também de evoluir o software original, o que contribui para o avanço do software para toda a comunidade, e não apenas para a empresa.

[6] Produtos para nichos específicos, como roteadores de rede, já se encontravam no mercado há vários anos.

No entanto, não existe uma obrigatoriedade da empresa em integrar suas alterações: essa é uma responsabilidade da comunidade. Algumas empresas simplesmente disponibilizam o código-fonte, sem identificar as alterações realizadas ou explicar o raciocínio por trás delas.

Contudo, essa prática não é interessante para as empresas. Softwares livres estão em constante evolução, seja para a adição de novas funcionalidades quanto para correção de erros. Diferente de um programa de computador, que geralmente não oferece garantias quanto a sua utilização para seus usuários, os produtos comercializados pelas empresas precisam respeitar diversas regras para receber a permissão de comercialização, sendo o desrespeito delas (como, por exemplo, pelo mau funcionamento) penalizados (multas e até mesmo retirada do produto do mercado). Nesse cenário, é interessante que a empresa mantenha a sua versão do software em sincronia com a versão mantida pela comunidade. Para isso, ela pode: (1) integrar suas alterações no software da comunidade (o que a livraria do ônus de mantê-lo) ou (2) sincronizar, freqüentemente, as alterações, do software da comunidade, em sua versão do programa. A opção (1), certamente, é a mais barata. Mesmo para casos em que existem questões de proteção intelectual que a impedem de integrar seu código (uma patente, por exemplo), é possível modularizar o programa de modo que seja integrado, no software da comunidade, um módulo que facilite a manutenção da sincronia do módulo protegido, da versão do software dela[7].

Todavia, integrar código "alheio" ao software da comunidade não é uma tarefa trivial. Muitos requisitos de software livre são implícitos, de conhecimento tácito dos desenvolvedores. No momento da integração, as alterações precisam atender a esses requisitos. O fato é que, geralmente, elas não atendem, o que exige modificações do código por parte da empresa. Se a integração tornar-se cara, como a empresa não é obrigada a integrar o código, ela pode optar por apenas liberar o código, deixando o trabalho a cargo de terceiros. Esses, por sua vez, por não terem os requisitos e demais documentação de desenvolvimento gerados pela empresa, também encontram dificuldades na realização da tarefa. O resultado final é que o projeto de software livre original deixa de ganhar funcionalidades, talvez importantes, por um problema que poderia ser evitado.

Se a preferência da empresa é integrar suas alterações ao código, seria desejável diminuir os obstáculos para isso, fornecendo, de alguma maneira, a documentação necessária dos softwares livres. De conhecimento anterior dos requisitos, ela poderia desenvolver seus produtos de modo fiel ao requerido pela comunidade de software livre. Uma possível solução seria extrair as informações a partir da análise dos comentários inseridos no código-fonte, do sistema de controle de alteração, de controle de versão e lista de emails. A execução de tal análise é factível, alguns trabalhos já exploram a análise de código-fonte e controle de alteração. Entretanto, o custo, dependendo do volume de informações utilizado, pode ser

[7] Necessita-se lembrar que o código-fonte é protegido pelas leis de direitos autorais. É possível ter um software livre que viola uma patente (e esse é um dos motivos para a criação da próxima versão da licença GPL, a GPLv3).

elevado, tanto em termos de processamento quanto de tempo (para obter os dados dos vários *sites* da Internet, considerando que todos os dados do projeto estão disponíveis na Web).

Uma segunda solução seria incentivar a documentação do processo de engenharia de requisitos nos projetos de software livre. A comunidade costuma adotar práticas de engenharia que são bem sucedidas: controle de versões (principalmente graças ao CVS) e controle de alterações (Bugzilla). Se uma alternativa para engenharia de requisitos, de fácil execução e com resultados visíveis e práticos, fosse disponibilizada, existiria uma boa chance dela ser utilizada nos projetos.

Assim, um trabalho futuro, desafiador, seria desenvolver essa alternativa: um processo de engenharia de requisitos de softwares livres. Sabe-se que a existência de uma ferramenta, que apóie o processo, é essencial. A Wiki/RE foi desenvolvida para atender esse possível processo, com base nas informações encontradas na literatura e na convivência e experiência do autor deste trabalho no desenvolvimento e estudo de software livre. Resta elucidar o processo utilizado atualmente, definir um processo de engenharia de requisitos adequado para a comunidade de software livre, testá-lo e divulgá-lo (certamente não é uma tarefa trivial).

Capítulo

7

Referências Bibliográficas

ACHOUR, M. et al. Php manual: Using register globals. 2005. Disponível em: `http://www.php.net/register_globals`.

AIKEN, P. Advanced technology for command and control systems engineering. In: _____. *Advanced Technology for Command and Control Systems Engineering*. [S.l.]: AFCEA International Press, 1990. cap. Hypermedia-based Requirements Engineering.

ALEXANDER, I. Misuse cases: Use cases with hostile intent. *IEEE Software*, v. 20, n. 1, p. 58–66, jan 2003.

ANDREI, C. et al. *XMLBeans*. 2005. Programa de Computador. Disponível em: `http://xmlbeans.apache.org/`.

Apache Software Foundation. *Ant*. jan. 2000. Programa de Computador. Disponível em: `http://ant.apache.org/`.

Apple Computer. *Mac OS*. jan 1984. Programa de Computador. Disponível em: `http://www.apple.com/macosx/`.

ATKINSON, B. *Hypercard*. ago. 1987. Programa de Computador. Disponível em: `http://en.wikipedia.org/wiki/HyperCard`.

BECK, K. *Extreme Progamming Explained*. [S.l.]: Addison-Wesley, 1999.

BERNERS-LEE, T. *Information Management: A Proposal*. mar. 1989.

BOMPANI, L.; CIANCARINI, P.; VITALI, F. Sophisticated hypertext functionalities for software engineering. In: *3rd International Workshop on Software Engineering over the Internet*. Limerick, Ireland: ACM Press, 2000. p. 67–79.

BORGSTRÖM, J. et al. *Trac*. fev. 2004. Programa de Computador. Disponível em: `http://www.edgewall.com/trac/`.

BREITMAN, K. K.; LEITE, J. C. S. do P. Managing user stories. In: EBERLEIN, A.; LEITE, J. C. S. do P. (Ed.). *Proceedings of the International Workshop on Time Constrained Requirements Engineering*. Essen, Germany: [s.n.], 2002.

BROOKS, J. F. P. No silver bullet — essence and accidents of software engineering. In: *International Federation of Information Processing (IFIP) Congress '86*. Dublin, Irlanda: [s.n.], 1986. p. 1069–1076.

BURN, O. *CheckStyle*. 2001. Programa de Computador. Disponível em: `http://checkstyle.sourceforge.net/`.

BUSH, V. As we may think. *The Atlantic Monthly*, jul. 1945. Disponível em: `http://www.ps.uni-sb.de/~duchier/pub/vbush/vbush-all.shtml`.

CHEESMAN, J.; DANIELS, J. *UML Components - A Simple Process for Specifying Component-Based Software*. [S.l.]: Addison-Wesley, 2001. (Component Software Series).

CHEUNG, S.; MATENA, V. *Java Transaction API*. nov 2002. JCP Specification.

CHUNG, L. et al. From information system requirements to designs: a mapping framework. *Information Systems*, v. 16, n. 4, p. 429 – 461, 1991.

CLELAND-HUANG, J.; ZEMONT, G.; LUKASIK, W. A heterogeneous solution for improving the return of investment of requirements traceability. In: *IEEE International Requirements Engineering Conference*. Kyoto, Japão: [s.n.], 2004. p. 230–239.

CollabNet. *Subversion*. jun. 2000. Programa de Computador.

CONALLEN, J. *Building Web Applications with UML Second Edition*. 2. ed. [S.l.]: Addison Wesley, 2002.

COPELAND, T. et al. *PMD*. 2002. Programa de Computador.

CRONIN, G. *VeryQuickWiki*. 2000. Programa de Computador. Disponível em: `http://veryquickwiki.croninsolutions.com/`.

CROWSTON, K.; ANNABI, H.; HOWISON, J. Defining open source software project success. In: *Proc. of International Conference on Information Systems (ICIS 2003)*. [S.l.: s.n.], 2003.

CROWSTON, K. et al. Towards a portfolio of floss project success measures. In: *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Enginnering (ICSE 2004)*. Edinburgh, Scotland: [s.n.], 2004.

CUNNIGHAM, W. *WikiWikiWeb*. mar. 1995. Programa de Computador. Disponível em: `http://c2.com/cgi/wiki`.

CUNNINGHAM, W. Wiki design principles. 2005. Disponível em: `http://c2.com/cgi/wiki?WikiDesignPrinciples`.

CUSUMANO, M. A.; SELBY, R. W. How Microsoft builds software. *Communications of the ACM*, v. 40, n. 6, p. 53–61, 1997.

DAG, J. N. och et al. Speeding up requirements management in a product software company: Linking customers wishes to product requirements through linguistic engineering. In: *International Requirements Engineering Conference*. [S.l.: s.n.], 2004. p. 1–12.

DAG, J. N. och et al. A linguistic-engineering approach to large-scale requirements management. *IEEE Software*, v. 22, n. 1, p. 32–39, jan 2005.

DAMIAN, D. E. H. et al. Using different communication media in requirements negotiation. *IEEE Software*, v. 17, n. 3, p. 28–36, out. 2000.

DARDENNE, A.; LAMSWEERDE, A. van; FICKAS, S. Goal-directed requirements acquisition. *Science of Computer Programming*, v. 20, n. 1-2, p. 3–50, 1993. Disponível em: `citeseer.nj.nec.com/dardenne93goaldirected.html`.

DAVIS, H.; LEWIS, A.; RIZK, A. OHP: A draft proposal for a standard open hypermedia protocol. In: *Hyptertext'96: the 2nd Workshop on Open Hypermedia Systems*. Washington DC: ACM Press, 1996. p. 27–53.

Departament of Defense. *MIL-STD-498 - DI-IPSC-81433 - Software Requirements Specification*. dez. 1994.

DEROSE, S. J.; DURAND, D. G. *Making Hypermedia Work: A User's Guide to HyTime*. 1. ed. Estados Unidos: Klewer Academic Publishers, 1994.

DURÁN, A. *A Methodological Framework for Requirements Engineering of Information Systems*. Tese (Doutorado) — University of Seville, 2000. Em espanhol.

DURÁN, A.; RUIZ, A.; TORO, M. An automated approach for verification of software requirements. In: *JIRA'2001 Proceedings*. Seville: [s.n.], 2001.

EASTERBROOK, S.; NUSEIBEH, B. Using viewpoints for inconsistency management. *IEE Software Engineering Journal*, v. 11, n. 1, p. 31–43, jan. 1996.

EBERLEIN, A. Agile requirements definition: A view from requirements engineering. In: *International Workshop on Time Constrained Requirements Engineering*. Essen, Alemanha: [s.n.], 2002.

Eclipse Foundation. *Eclipse*. 2000. Programa de Computador. Disponível em: `http://www.eclipse.org`.

ELLIS, J.; HO, L.; FISHER, M. *JDBC 3.0 Specification*. oct 2001. Proposed Final Draft 4.

ETTRICH, M. et al. *K Destop Environment*. 1996. Programa de Computador. Disponível em: `http://www.kde.org`.

FELICÍSSIMO, C. H.; LEITE, J. C. S. do P.; BREITMAN, K. K. C&l: Um ambiente para edição e visualização de cenários e léxicos. In: REIS, R. Q. (Ed.). *Sessão de Ferramentas do Simpósio Brasileiro de Engenharia de Software*. Brasília, Brasil, 2004. p. 43 – 48.

FERREIRA, A. B. de H. et al. *Aurélio Século XXI: O Dicionário da Língua Portuguesa*. 3. ed. [S.l.]: Nova Fronteira, 1999.

FIALLI, J.; VAJJHALA, S. *Java Architecture for XML Binding (JAXB) Specification 1.0*. jan 2003. Specification. Disponível em: `http://www.jcp.org/en/jsr/detail?id=31`.

FICKAS, S.; LAMSWEERDE, A. van; DARDENNE, A. Goal-directed concept acquisition in requirements elicitation. In: *6th International Workshop on Software Specification and Design*. Como, Italy: [s.n.], 1991. p. 14–21.

FINKELSTEIN, A.; KRAMER, J.; GOEDICKE, J. K. Viewpoints oriented software specification. In: IEEE COMPUTER SOCIETY. *3rd International Workshop on Software Engineering and its Applications*. Toulouse, France, 1990. p. 337–351.

FINKELSTEIN, A.; SOMMERVILLE, I. The viewpoints FAQ. *Software Engineering Journal: Special Issue on Viewpoints for Software Engineering*, v. 11, n. 1, p. 2–4, 1996.

FORTES, R. P. de M.; TURINE, M. A. S.; REIS, C. R. *Engenharia de Software Disponível a Todos - Software Engineering Available for Everyone (SAFE)*. oct 2004. Proposta de Financiamento do Projeto.

FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: Addison-Wesley Professional, 2002.

Frauhofer Institute. *Eudibamus*. 2003. Programa de Computador. Disponível em: `http://www.first.fraunhofer.de/eudibamus`.

Fraunhofer FIRST et al. *KOGITO - Knowledge Based Requirements Engineering in Software Development*. 2003. Projeto. Patrocinado pelo Federal Ministry of Education and Research da Alemanha (código do patrocínio: 01ISB01). Disponível em: `http://www.kogito.org`.

FURUTA, R.; STOTTS, P. D. Generalizing hypertext: Domains of the trellis model. *Technique et Science Informatiques (Technology and Science of Informatics)*, v. 9, n. 6, p. 493–503, nov. 1990. Disponível em: `http://www.csdl.tamu.edu/~furuta/trellis/trellis-papers.html`.

GAMMA, E.; BECK, K. *JUnit*. 2001. Programa de Computador. Disponível em: `http://www.junit.org`.

GARZOTTO, F.; PAOLINI, P.; SCHWABE, D. Hdm: a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, ACM Press, New York, NY, USA, v. 11, n. 1, p. 1 – 26, jan. 1993.

GRUNE, D. et al. *Concurrent Versions System (CVS)*. jun. 1986. Programa de Computador. Disponível em: `http://cvshome.org`.

Grupo de Engenharia de Requisitos (PUC-RJ). *C&L: Cenários e Léxicos*. 2004. Programa de Computador. Disponível em: `http://sl.les.inf.puc-rio.br/cel/`.

GUDGEIRSSON, G. *Requirements engineering and XML*. set. 2000. Disponível em http://www.raqoon.is/rqml/rqml-spec.htm. Acesso em 16 de outubro de 2003).

GUNTER, C. A. et al. A reference model for requirements and specifications. *IEEE Software*, v. 17, n. 3, p. 37 – 43, may 2000.

GUZDIAL, M. *Swiki/CoWeb*. 1999. Programa de Computador. Disponível em: `http://minnow.cc.gatech.edu/swiki`.

HAGGE, L.; LAPPE, K. Sharing requirements engineering experience using patterns. *IEEE Software*, v. 22, n. 1, p. 24 – 31, jan 2005.

HALASZ, F.; SCHWARTZ, M. The Dexter hypertext reference model. *Communications of the ACM*, v. 37, n. 2, p. 30–39, fev. 1994.

HALASZ, F. G.; SCHWARTZ, M. The Dexter hypertext reference model. In: *NIST Hypertext Standardization Workshop*. [S.l.: s.n.], 1990.

HAYES, J. H. et al. Helping analysts trace requirements: An objective look. In: *International Requirements Engineering Conference*. Kyoto, Japão: IEEE Computer Society, 2004. p. 249–259.

HOFFMAN, M. et al. Requirements for requirements management tools. In: *IEEE Internacional Requirements Engineering Conference*. Kyoto, Japão: IEEE, 2004. p. 301–308.

HOFFMANN, M. et al. Requirements for requirements management tools. In: *International Requirements Engineering Conference (RE'04)*. Kyoto, Japão: IEEE Computer Society, 2004. p. 301–308.

IBM Rational. *RequisitePro*. jun. 2003. Programa de Computador. Disponível em: `http://www-306.ibm.com/software/awdtools/reqpro/`.

IEEE Computer Society. *Software Engineering Body of Knowledge (SWEBOK)*. EUA: Angela Burgess, 2004. Disponível em: `http://www.swebok.org/`.

Institute of Electrical and Electronics Engineers. *IEEE recommended practice for software acquisition (IEEE Std 1362-1998)*. dez. 1998.

Institute of Electrical and Electronics Engineers. *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)*. jun. 1998.

Institute of Electrical and Electronics Engineers; Electronics Industry Association. *IEEE/EIA 12207 - Industry Implementation of International Standard ISO/IEC 12207 : 1995*. mar 1998. Standard.

Institute of Electrical and Electronics Engineers; Electronics Industry Association (EIA). *Trial-use standard standard for information technology software life cycle processes software development acquirer-supplier agreement (J-STD-016-1995)*. sep 1995. Standard.

International Organization for Standardization (ISO); International Electrotechnical Commission (IEC). *ISO/IEC 12207 - Standard for Information Technology - Software life cycle processes*. aug 1995. Standard.

ISAKOWITZ, T.; STOHR, E. A.; BALASUBRAMANIAN, P. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, v. 38, n. 8, p. 34 – 44, 1995.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. 2. ed. Massachusetts: Addison Wesley Longman, 1999. (Object Tecnology).

JACOBSON, I. et al. *Object-Oriented Software Engineering: A Use Case Driven Approach*. [S.l.]: Addison-Wesley, 1992.

JALKANEN, J. *JSPWiki*. jul. 2001. Programa de Computador. Disponível em: `http://www.jspwiki.org/`.

JR, C. R. E. A.; IZEKI, C. A.; PIMENTEL, M. G. C. Coteia: Uma ferramenta colaborativa de edição baseada na web. In: *Workshop de Ferramentas e Aplicações do VIII SBMIDIA*. [S.l.: s.n.], 2002.

JUGEL, M. L.; SCHMIDT, S. J. *Snipsnap*. set. 2002. Programa de Computador. Disponível em: `http://snipsnap.org`.

KAINDL, H. Active tool support for requirements engineering through reth. In: *International Requirements Engineering Conference*. [S.l.: s.n.], 2004. p. 362 – 363.

KAINDL, H.; KRAMER, S. *Semiautomatic Generation of Dictionary Links in Hypertext*. 1995.

KING, G. et al. *Hibernate - Relational Persistence for Idiomatic Java*. may 2005. Software. Disponível em: `http://www.hibernate.org`.

KING, G. et al. *Hibernate API Documentation*. jun 2005. Web. Disponível em: `http://www.hibernate.org/hib_docs/v3/api/`.

KNABBEN, F. C. *FCKEditor*. 2005. Programa de Computador. Disponível em: `http://fckeditor.wikiwikiweb.de`.

KOTONYA, G.; SOMMERVILLE, I. *Requirements Engineering With Viewpoints*. Reino Unido, 1995.

KOTONYA, G.; SOMMERVILLE, I. *Requirements engineering : processes and techniques*. New York: EUA: J. Wiley, 1998. (Worldwide series in computer science).

KRASNER, G. E.; POPE, S. T. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, v. 1, n. 3, p. 26 – 49, aug 1988.

KRUCHTEN, P. *The Rational Unified Process*. [S.l.]: Addison Wesley, 1999.

LAMSWEERDE, A. van. Goal-oriented requirements engineering: A guided tour. In: *5th International Symposium on Requirements Engineeing*. Toronto, EUA: [s.n.], 2001. p. 249–263.

LANG, M.; FITZGERALD, B. Hypermedia systems development practices: A survey. *IEEE Software*, v. 22, n. 2, p. 68– 75, mar. 2005.

LEE, H.; LEE, C.; YOO, C. A scenario-based object-oriented methodology for developing hypermedia information systems. In: *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*. [S.l.: s.n.], 1998. v. 2, p. 47 – 56.

LEITE, J. C. do P. Viewpoint analysis: A case study. *ACM J. Software Engineering Notes*, v. 14, n. 3, p. 111–119, 1989.

LEITE, J. C. S. do P. *Slides do Minicurso de Engenharia de Requisitos*. 2001. V Semana da Computação - ICMC/USP.

LEITE, J. C. S. do P.; FRANCO, A. P. M. A strategy for conceptual model acquisition. In: *Proceedings of IEEE International Symposium on Requirements Engineering 1993*. [S.l.: s.n.], 1993. p. 243–246.

LEITE, J. C. S. do P.; FREEMAN, P. A. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, v. 17, n. 12, p. 1253–1269, dez. 1991.

LENNON, J. A. *Hypermedia Systems and Applications: World Wide Web and Beyond*. Alemanha: Springer, 1997.

LOCONSOLE, A. Empirical studies on requirements management measures. In: *Doctoral Symposium, International Conference on Software Engineering*. [S.l.]: IEEE Computer Society, 2004. p. 42–44.

Macromedia. *Flash*. 2000. Programa de Computador. Disponível em: `http://www.macromedia.com/platform/`.

MANSKE, M. *MediaWiki*. 2002. Programa de Computador. Disponível em: `http://wikipedia.sourceforge.net/`.

MARSCHALL, F.; SCHOENMAKERS, M. Towards model-based requirements engineering for web-enabled b2b applications. In: *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03)*. Huntsville, AL, EUA: IEEE Computer Society, 2003. p. 312–320.

MCLAUGHLIN, L. European union struggles with new rules for software patents. *IEEE Software*, v. 21, n. 5, p. 101–104, sep 2004.

MCORMOND, R. *A Review Of Software Patent Issues*. [S.l.], 2003. Disponível em: `http://www.flora.ca/patent2003/software-patent2003.shtml`.

MILLARD, D.; DAVIS, H.; MOREAN, L. Standardizing hypertext: Where next for OHP? In: *6th International Workshop on Open Hypermedia Systems*. [S.l.: s.n.], 2000.

MILLARD, D.; DAVIS, H.; MOREAN, L. Standardizing hypertext: Where next for OHP? *Lecture Notes on Computer Science*, Springer-Verlog, v. 1903, p. 3 – 12, maio 2000.

MILLER, R.; MILLER, R. *Myst*. 1993. Programa de Computador. Disponível em: `http://www.myst.com/myst_home.html`.

Moxiecode Systems AB. *TinyMCE*. 2005. Programa de Computador. Disponível em: `http://tinymce.moxiecode.com/`.

MULLERY, G. P. A method for controlled requirements specifications. In: IEEE COMPUTER SOCIETY. *4th International Conference on Software Engineering*. [S.l.], 1979. p. 126–135.

MYLOPOULOS, J.; CHUNG, L.; YU, E. From object-oriented to goal oriented requirements analysis. *Communications of the ACM*, v. 42, n. 1, p. 31 – 37, jan 1999.

NA, J.-C.; FURUTA, R. Dynamic documents: Authoring, browsing, and analysis using a high-level petri net-based hypermedia system. In: *Proceedings of the ACM Symposium on Document Engineering (DocEng '01)*. Atlanta, GA: ACM Press, 2001. p. 38–47.

NARAYANASWAMY, K.; GOLDMAN, N. Lazy consistency: A basis for cooperative software development. In: *International Conference on Computer-Supported Cooperative Work*. Toronto, Ontario, Canada: [s.n.], 1992. p. 257–264.

NELSON, T. *Ted Nelson on Hypertext from Hyperland*. set. 1990. Vídeo (. Entrevista extraída do documentário Hyperland, distribuído pela da BBC Television (canal BBC2) e produzido por Douglas Adams. Disponível em: `http://xanadu.com.au/AV/hypertext.mpg`.

NELSON, T. H. A file structure for the complex, the changing and the indeterminate. In: *ACM 20th national conference*. [S.l.]: ACM, 1965. p. 84–100.

NIELSEN, J. *Multimedia and Hypertext: The Internet and Beyond*. Estados Unidos: Academic Press, 1995.

Object Mentor, Inc. *Fitnesse*. mar. 2003. Programa de Computador. Disponível em: `http://fitnesse.org/`.

PAIVA, D. M. B.; FORTES, R. P. d. M. Design rationale in software engineering: A case study. In: *Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2005)*. Taipei, China: [s.n.], 2005. p. 14–16.

PAPAIOANNOU, V.; THEODOULIDIS, B. Here: Hypermedia environment for requirements engineering. In: *7ty Workshop on the Next Generation of CASE Tools (NGCT'96)*. Heraklion, Crete: [s.n.], 1996. p. 20–21.

PRESSMAN, R. S. *Software engineeering: a practioner's approach*. 5. ed. Boston, EUA: McGraw-Hill, 2000.

RAMESH, B.; JARKE, M. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, v. 27, n. 1, p. 58–93, jan 2001.

RASCH, C. *A Brief History of Free/Open Source Software Movement*. dez. 2000. Web site. Disponível em: `http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html`.

RAYMOND, E. S. *The Cathedral and the Bazaar*. 1. ed. [s.n.], 1997. Disponível em: `http://www.catb.org/~esr/writings/cathedral-bazaar/`.

RAYMOND, E. S. *The Cathedral and the Bazaar*. 1. ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc, 2001. Disponível em: `http://www.catb.org/~esr/writings/cathedral-bazaar/`.

REIFER, D. J. Requirements management: The search for nirvana. *IEEE Software*, v. 17, n. 3, p. 45–47, may 2000.

REIS, C. R. *Caracterização de um Modelo de Processo para Projetos de Software Livre*. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, São Paulo, jun. 2003.

RIDAO, M.; DOORN, J.; LEITE, J. C. S. do P. Uso de patrones en la construcción de escenarios. In: *WER 2000*. [S.l.]: WER, 2000. p. 140–157.

ROSS, D. T.; SCHOMAN, K. E. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, v. 3, n. 1, p. 6–15, 1977.

SCACCHI, W. Understanding the requirements for developing open source software systems. In: *IEE Proceedings - Software*. [s.n.], 2002. v. 149, n. 1, p. 24–39. Disponível em: `mit.edu/papers/Scacchi.pdf`.

SCHMIDT, K. Of maps and scripts — status of informal constructs in cooperative work. In: *ACM Conference on Supporting Group Work*. [S.l.: s.n.], 1997. p. 138–147.

SCHOMAN, K.; ROSS, D. T. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, v. 3, n. 1, p. 6–15, 1977.

SCHWABE, D.; ROSSI, G. Building hypermedia applications as navigational views of information model. In: *28th Hawaii International Conference on Information Systems*. Maui, Hawaii: [s.n.], 1995. p. 231–240.

SCHWABE, D.; ROSSI, G.; BARBOSA, S. D. J. Systematic hypermedia application design with OOHDM. In: *UK Conference on Hypertext*. [S.l.: s.n.], 1996. p. 116–128.

SHNEIDERMAN, B.; KEARSLEY, G. *Hypertext Hands-On! An introduction to a New Way of Organizing and Accessing Information*. [S.l.]: Addison-Wesley, 1989.

SHULL, F.; RUS, I.; BASILI, V. How perspective-based reading can improve requirements inspections. *IEEE Software*, v. 33, n. 7, p. 73–79, jul 2000.

SILVA, M. A. G. *CoTeia*. fev. 2004. Programa de Computador. Disponível em: `http://incubadora.fapesp.br/projects/coteia`.

SILVA, S. *Controle de versões - um apoio à edição colaborativa na Web*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação de São Carlos, Universidade de São Paulo, jul. 2005.

SOMMERVILLE, I. *Software Engineering*. 5. ed. Massachussets: Addison Wesley, 1996.

SOMMERVILLE, I. *Software Engineering*. 6. ed. New York, USA: Addison-Wesley, 2001. (International computer science series).

SOMMERVILLE, I.; SAWYER, P. Viewpoints: principles, problems and a pratical approach to requirements engineering. *Annals of Software Engineering*, v. 3, p. 101–130, 1997.

STALLMAN, R. M. et al. *GNU Compiler Collection*. nov. 1987. Programa de Computador. Disponível em: `http://gcc.gnu.org`.

STOTTS, P. D.; FURUTA, R. Petri-net-based hypertext: Document structure with browsing semantics. *ACM Transactions on Information Systems*, v. 7, n. 1, p. 3–29, 1989.

SUN Microsystems. *Java Web Services Developer Pack*. 2004. Programa de Computador. Disponível em: `http://java.sun.com/webservices/jwsdp/index.jsp`.

Telelogic. *DOORS*. 2005. Programa de Computador. Disponível em: `http://www.telelogic.com/products/doorsers/doors/index.cfm`.

The Apache Foundation. *Lucene*. mar. 2000. Programa de Computador. Disponível em: `http://lucene.apache.org/`.

The Apache Software Foundation. *Struts*. jun. 2001. Programa de Computador. Disponível em: `http://struts.apache.org/`.

The Institute of Electrical and Eletronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology*. set. 1990. IEEE Standard.

THOENY, P. *TWiki*. jul. 1998. Programa de Computador. Disponível em: `http://twiki.org`.

TICHY, W. *Revision Control System (RCS)*. 1982. Software. Disponível em: http://www.cs.purdue.edu/homes/trinkle/RCS/.

TORO, A. D. *Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*. Tese (Doutorado) — Universidad de Sevilla, set. 2000. Disponível em: http://www.lsi.us.es/~amador/publicaciones/tesis.pdf.zip.

TORO, A. D. *REM (REquisite Management)*. 2004. Programa de Computador. Disponível em: http://www.lsi.us.es/descargas/descarga_programas.php?id=3.

TORO, A. D. et al. Identificación de patrones de reutilización de requisitos de sistemas de información. In: *WER2000*. [S.l.: s.n.], 2000.

TORO, A. D. et al. A requirements elicitation approach based in templates and patterns. In: *Workshop em Engenharia de Requisitos*. Buenos Aires, Argentina: [s.n.], 1999. p. 17–29.

TORVALDS, L. et al. *Linux*. ago. 1991. Programa de Computador. Disponível em: http://www.kernel.org.

TROYER, O. M. F. D.; LEUNE, C. J. WSDM: a user centered design method for Web sites. In: *Computer Networks and ISDN Systems*. [S.l.: s.n.], 1998. v. 30, p. 85–94.

United Nations Educational, Scientific and Cultural Organization. *Free Software History*. 2001. Web site. Disponível em: http://www.unesco.org/webworld/portal_freesoft/open_history.shtml.

VLASOV, P. et al. *Hammurapi*. jul. 2004. Programa de Computador. Disponível em: http://www.hammurapi.org,http://sourceforge.net/projects/hammurapi/.

W3C. *World Wide Web Consortium*. 1994. Http://www.w3c.org.

WALNES, J. et al. *XStream*. abr. 2005. Programa de Computador. Disponível em: http://xstream.codehaus.org/.

WANG, Q.; LAI, X. Requirements management for the incremental development model. In: *Asia-Pacific Conference on Quality Software*. [S.l.]: IEEE, 2001. p. 295–301.

WEBSTER, D. Mapping the design information representation terrain. *IEEE Computer*, v. 21, n. 12, p. 8–23, dez. 1988.

WEISSMAN, T. et al. *Bugzilla*. 1998. Programa de Computador. Disponível em: http://www.bugzilla.org.

WHEELER, D. A. *SLOCCount*. jan. 2001. Programa de Computador. Disponível em: http://www.dwheeler.com/sloccount/.

WHITEHEAD, K. *Component-Based Development: Principles and Planning for Business Systems*. 1. ed. [S.l.]: Addison-Wesley Professional, 2002.

WIERINGA, R. J. *Requirements Engineering: Framework for understanding*. [S.l.]: Wiley, 1995.

YAMAUCHI, Y. et al. Collaboration with lean media: How open-source software succeeds. In: *Computer Supported Cooperative Work*. Philadelphia, EUA: [s.n.], 2000. p. 329–338.

YU, E. S. K. Why agent-oriented requirements engineering. In: *REFSQ'97 - Requirements Engineering: Foundation of Software Quality*. Barcelona, Espanha: Presses Universitaires de Namur, 1997.

ZANLORENCI, E. P.; BURNETT, R. C. REQAV: Modelo para descrição, qualificação, análise e validação de requisitos. In: *IDEAS2000: Jornada Ibero Americana de Ingeneria de Requisitos Y Ambientes de Software*. [S.l.: s.n.], 2000. p. 61–72.

ZYLBERMANN, D.; COHEN, Y.; GOLDIN, L. The road to requirements maturity. In: *IEEE Internacional Conference on Software - Science, Technology & Engineering (SwSTE´03)*. [S.l.]: IEEE Computer Society, 2003. p. 71–78.

Apêndice. HyperCard

O HyperCard foi criado por Atkinson (1987). Ele é citado por Cunnigham como uma das fontes de inspiração para a criação da WikiWikiWeb. Entender o por quê da escolha do HyperCard seria, no mínimo, uma curiosidade.

O primeiro passo é compreender o sucesso que o HyperCard alcançou no exterior, principalmente nos Estados Unidos. Ele não era apenas uma das primeiras ferramentas hipermídias disponíveis comercialmente na época, mas também a com maior base instalada. O HyperCard era distribuído, gratuitamente, com o sistema operacional Mac OS (Apple Computer, 1984), presente em todos os computadores Macintosh fabricados pela Apple Computer[1].

O funcionamento do HyperCard é relativamente simples. Sua janela possui sempre um tamanho fixo, sem barra de rolagens (os nós hipertextos são apresentados como quadros, e não páginas). São os cartões (*cards*). Cada aplicação HyperCard é um conjunto de cartões, que se denomina "*stack card*". Define-se um pano de fundo comum a todos os cartões: ele será apresentado, independentemente do cartão apresentado. Cada cartão possui textos e elementos gráficos (*widgets*). As ligações entre os cartões são definidas por *scripts*, especificados em uma linguagem de programação de alto nível, a HyperTalk (Dan Winkler, 1987). Os *scripts* são associados aos *widgets*. Ao ativar um desses elementos gráficos, o *script* é executado. Por exemplo, para associar um *widget* a um cartão denominado "Memex", seria utilizado um script como o apresentado na listagem abaixo:

```
goto "Memex"
```

A figura 1 mostra uma aplicação Hypercard. Tipicamente, os *widgets* são botões, como o botão "*Definition of HyperText*" ou quadros ao redor de termos chaves do texto.

[1] Os computadores Macintosh não conseguiram uma parcela significativa do mercado no Brasil. Questões como preço e, principalmente, barreiras como a reserva de mercado, impediram sua inserção no país.

A figura 2 apresenta o cartão da figura 1, agora decomposto em pano de fundo (quadro à direita), texto (quadro do meio) e *widgets* que acionam as ligações (quadro da esquerda).

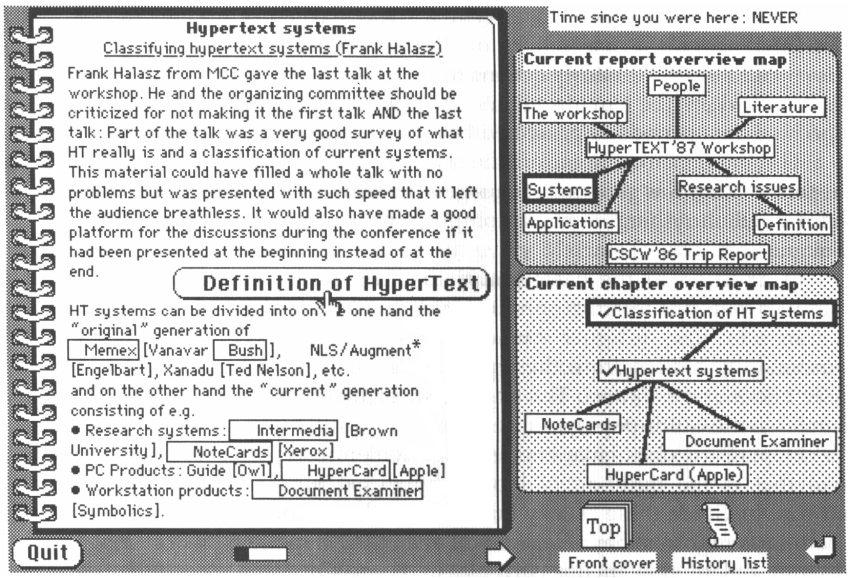

Figure 1. Exemplo de um cartão de uma aplicação HyperCard. Fonte: Nielsen (1995, p. 60).

Esta é uma descrição simplificada dos recursos do HyperCard. A linguagem HyperTalk permite a construção de sistemas mais complexos[2], mas eles não são necessários para identificar as principais similaridades entre o HyperCard e as *wikis*:

- nós de tamanho reduzido (cartões em HyperCard, páginas curtas em *wikis*);
- nós identificados por um nome;
- ligações geralmente criadas com base no nome dos nós.

Uma wiki pode ser visualizada como uma simplificação do HyperCard. Eliminou-se a linguagem de alto nível por uma linguagem de marcação simples, restringiram-se as âncoras das ligações aos nomes dos cartões (as páginas *wikis*). Finalmente, adicionou-se o conceito de autoria incremental do hiperdocumento, em um ambiente próximo daquele utilizado na navegação (o HyperCard possui um ambiente distinto para criação dos cartões).

[2] Um exemplo seria o jogo Myst (MILLER; MILLER, 1993).

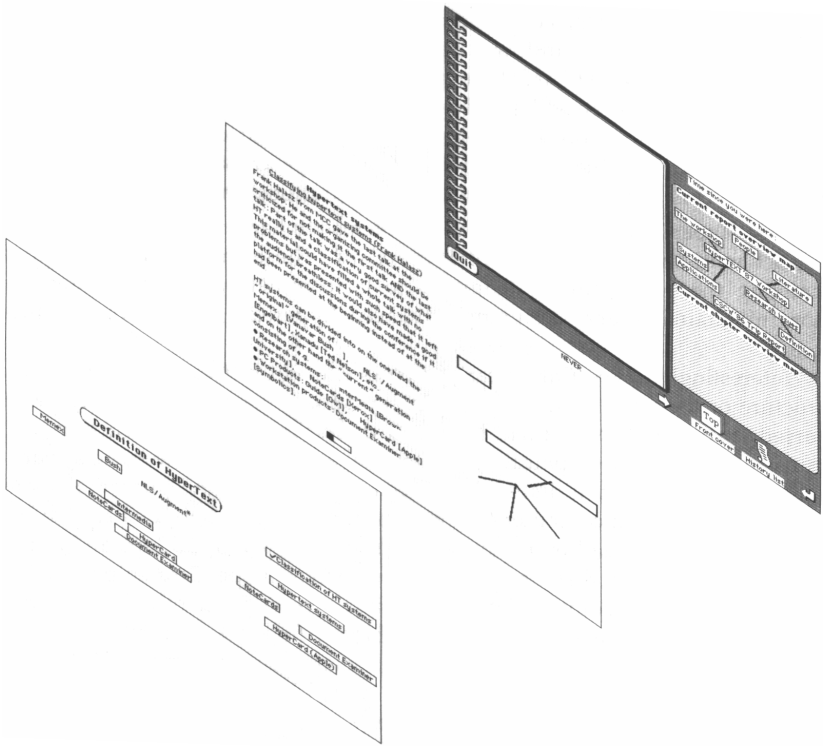

Figure 2. Exemplo de um cartão, decomposto em pano de fundo, texto e *widgets*, de uma aplicação HyperCard. Fonte: Nielsen (1995, p. 61).

Glossário

Árvore Termo utilizado, pela comunidade de software livre, para designar um repositório (controle de versão). A árvore pode ser criada vazia, a partir de arquivos cujas versões não são controladas, ou duplicando, total ou parcialmente, um outro repositório. Neste caso, mantém-se a relação entre os repositórios (como se fosse uma nova ramificação do repositório origem), mas em repositórios separados e independentes. As alterações realizadas no novo repositório não afetam o repositório do qual ele se originou. Posteriormente, é possível enviar as alterações feitas de volta para a árvore original.

Artefato Objeto gerado ou alterado pela aplicação de uma técnica.

Bug Erro encontrado no software. O termo "bug" também é utilizado na ferramenta Bugzilla para nomear pedidos de alteração ou relato de falhas ou erros de um software.

Crawler Programa que acessa os *sites* da Internet, indexando todas as informações encontradas. São geralmente utilizados por *sites* de busca (Google, Yahoo, etc).

Documento XML

Arquivo escrito na linguagem XML. Um documento XML pode ser bem-formado e válido. Um documento bem-formado é aquele que obedece a todas as regras estabelecidas pela linguagem XML. Já um documento válido, além de ser bem-formado, precisa declarar um DTD ou um ou mais XSD aos quais seus elementos obedecem.

DTD DTD é um documento que descreve uma estrutura e o conteúdo de documentos XML, definindo assim um tipo de documento XML.

HEAD Versão mais recente dos artefatos disponíveis em um sistema de controle de versão. É um termo usualmente empregado para controle de versão de arquivos de código-fonte.

HTTP Protocolo de comunicação orientado a conexão e sem estado, construído acima do protocolo TCP.

Internet Rede IP de alcance mundial. Nasceu de um projeto acadêmico, financiado pela DARPA, alcançando fins comerciais décadas mais tarde. Hoje é a rede mais utilizada no mundo, interligando milhões de pessoas.

IP Protocolo de rede não confiável, sem conexão, baseado em pacotes. Geralmente utilizado pelos protocolos UDP e TCP, formando assim a base da Internet.

Java Linguagem de programação orientada a objetos desenvolvida pela *Sun Microsystem*. Maiores informações podem ser encontradas em `http://java.sun.com`.

JAXP API para processamento de documentos XML (SAX, DOM, XSLT), independente de implementações específicas de uma empresa (padrão de projeto *Factory*).

Licença Permissão concedida pelo detentor do direito de cópia de um trabalho original. Essa permissão define os direitos e deveres dos pessoas que obtêm uma cópia do trabalho.

PDF Formato de arquivo, desenvolvido pela Adobe, que permite a representação de documentos de modo fiel ao documento original, independentemente do dispositivo utilizado para visualização (tela, impressora, etc).

Perl Linguagem de programação de propósito geral criada por Larry Wall. Maiores detalhes podem ser encontrados em `http://www.perl.org`.

PHP Linguagem de programação de propósito geral, voltada para o desenvolvimento de aplicações Web. Maiores informações podem ser obtidas em `http://www.php.net`.

Prioridade Classificação atribuída pelo desenvolvedor a um requisito. Geralmente analisam-se a severidade e complexidade do requisito, dentre outros fatores presentes no contexto do requisito, para a determinação da prioridade.

Python Python é uma linguagem de programação orientada a objeto. Maiores informações podem ser encontradas em `http://www.python.org`.

Ramo Termo utilizado, no contexto de controle de versões, pela comunidade de software livre, para denominar uma nova linha de desenvolvimento em um mesmo repositório, criada a partir de uma linha principal (por exemplo, a HEAD). As alterações realizadas nela não afetam a linha da qual ela se originou. Posteriormente, é possível unir as alterações feitas no ramo em sua linha original.

Severidade Grau de gravidade ou importância de um requisito. Não confundir com prioridade.

Site Local onde os recursos disponibilizados na WWW estão armazenados.

SOAP Tecnologia para comunicação entre objetos. O SOAP estabelece um protocolo baseado em XML cuja troca de dados é feita por chamadas remotas (RPC) ou mensagens (XML-RPC).

tag Uma *tag*, ou marcador, é uma palavra que identifica um conteúdo representado em um documento XML.

Universo de Discurso
Contexto definido pelo processo de engenharia de sistemas.

URI Identifica universalmente um recurso em uma rede (geralmente a Internet).

URL Endereço de um determinado recurso na rede (geralmente a Internet).

Usenet Sistema distribuído de grupos de discussão disponível na Internet. Trata-se de um dos mais antigos serviços disponibilizados na Internet, datado de meados de 1980. Nele usuários trocam mensagens (denominadas artigos) em grupos, estes organizados em hierarquias de acordo com o tema abordado em cada grupo.

Web Services
Conjunto de especificações e convenções para a construção de aplicações distribuídas complexas. Utiliza SOAP como meio de integração entre objetos, UDDI para serviços de registro, descoberta e nomes, e WSDL para descrever os serviços.

WWW A World Wide Web é o principal serviço disponibilizado na Internet. Ela forma um complexo sistema hipermídia distribuído pelos vários nós da rede, sistema este com fins pessoais, acadêmicos e comerciais. Atualmente ela é dirigida pela W3C, que cria normas e especificações (por exemplo, o HTML) que organizam o uso.

XBase Cria uma URI base a ser utilizada implicitamente em ligações XLink.

XInclude Permite a inclusão de documentos XML inteiros ou de trechos de documentos XML (trechos estes determinados por expressões XPointer) em um documento XML.

XLink Permite a criação de relacionamentos entre documentos XML.

XML A XML é uma linguagem de marcação derivada da SGML. Ela é utilizada na criação de documentos que são um importante meio para armazenamento e recuperação de dados em ambientes heterogêneos.

XML Schema

Cumpre papel semelhante ao DTD, mas acrescenta suporte a vários espaços de nomes em um mesmo documento, além de regras mais ricas para definição da estrutura e conteúdo de documentos XML.

XML-FO Linguagem de transformação voltada para a produção de arquivos binários, com instruções otimizadas para definição de layout e apresentação. Um uso típico de XML-FO é na transformação de documentos XML em arquivos PDF.

XML-RPC Protocolo de comunicação entre objetos remotos baseado em troca de mensagens. As mensagens são definidas como documentos XML e geralmente transportadas utilizando-se o protocolo HTTP (existem projetos para a utilização de outros meios de transporte, inclusive emails).

XPath Linguagem para recuperação de elementos de um documento XML

XPointer Linguagem para recuperação de partes de elementos de um documento XML.

XQuery Linguagem de consulta para documentos XML.

XSL Uma folha de estilo XSL contém um conjunto de instruções que permite a transformação de um documento XML em outro documento de saída, que pode ou não ser um documento XML (por exemplo, pode-se produzir arquivos binários utilizando XML-FO).