Miriam Sayão

Verificação e Validação em Requisitos: Processamento da Linguagem Natural e Agentes

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do título de Doutor pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Julio Cesar Sampaio do Prado Leite

Rio de Janeiro

Abril de 2007

Miriam Sayão

Verificação e Validação em Requisitos: Processamento da Linguagem Natural e Agentes

Tese apresentada como requisito parcial para obtenção do título de Doutor pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

prof. Julio Cesar Sampaio do Prado Leite
Orientador
PUC-Rio

prof. Marco Antonio Casanova
PUC-Rio

profa. Karin Koogan Breitman
PUC-Rio

prof. Jorge Luis Nicolas Audy
PUC-RS

profa. Vera Maria Benjamim Werneck
UERJ

prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico - PUC-Rio

Rio de Janeiro, 18 de abril de 2007

Miriam Sayão

Graduada em Estatística pela Universidade Estadual de Campinas (UNICAMP) em 1975. Obteve o título de Mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS) em 1986. Atua como docente na Pontifícia Universidade Católica do Rio Grande do Sul, na Faculdade de Informática.

Ficha Catalográfica

Sayão, Miriam

Verificação e validação em requisitos: processamento da linguagem natural e agentes / Miriam Sayão ; orientador: Julio César Sampaio do Prado Leite. – 2007.

205 f. : il. ; 30 cm

Tese (Doutorado em Informática)–Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2007.

Inclui bibliografia

1. Informática – Teses. 2. Análise de Requisitos. 3. Geração de visões de requisitos. 4. Agrupamento de requisitos. 5. Verificação e Validação em requisitos. 6. Omissões em requisitos não funcionais. 7. Processamento da linguagem natural em requisitos. 8. Agentes e gerenciamento de requisitos. I. Leite, Julio César Sampaio do Prado. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD : 004

Dedico aos meus pais, Antonio Rachid Sayão e Iracema Fischer Sacconi Sayão,
pelos ensinamentos da vida

Agradecimentos

Agradeço a todos os amigos que colaboraram de alguma forma, nesta tese. São muitos, alguns até colaboraram sem nem mesmo saber que o faziam....

A Julio César S. P. Leite, meu orientador, cujos questionamentos e revisões colaboraram para a qualidade deste trabalho.

Aos funcionários da PUC-Rio e do DI, em especial à Ruth, Débora, Emanuelle, Wagner e Rosane; a Zé Carlos e equipe de suporte. Aos professores Arndt von Staa, Carlos J. Pereira de Lucena, Ruy Milidiú, pelos ensinamentos. Ao Akeo Tanabe, pela imensa disposição em apoiar.

À minha família de origem: Iracema & Antonio, Ivette, Rosely & Fábio & Camila, Silvia, Yara & Celso & Marco, Filó, pelo carinho, sempre. Ao Aluízio, pelo afeto e apoio nesta jornada.

À minha "família" carioca (???): Karin, Carol, Lyrene, Roberta, Uirá, Eduardo, Karla, Pádua, Guga, Dani, Daflon, Léo, Mauren, Adéle, Silvana, Roberto,.... pela amizade e companheirismo.

Aos amigos de todos os lugares: Malu & Fleck, Sônia & Mário, Carlão, Eldo, Dai, Denise, Schüler, Olandina, Kleyna & Eduardo, Deise .... pela amizade que supera tempo e distâncias.

Ao Aluízio e ao Rafael, pelo apoio com a plataforma MIDAS. À Karin Breitman, pelo imenso apoio e amizade, sempre.

À Vera Lima e Jorge Audy, pelo grande incentivo nesta caminhada. À Faculdade de Informática da PUCRS, por todo o apoio recebido.

À Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS), à CAPES e à Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) pelo financiamento e apoio financeiro concedidos, sem os quais este trabalho não poderia ter sido realizado.

Resumo

Sayão, Miriam; Leite, Julio César S. P. (orientador). **Verificação e Validação em Requisitos: Processamento da Linguagem Natural e Agentes.** Rio de Janeiro, 2007. 205p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

No processo de desenvolvimento do software, atividades relacionadas ao Processo de Requisitos envolvem elicitação, modelagem, verificação e validação dos requisitos. O uso da linguagem natural no registro dos requisitos facilita a comunicação entre os participantes do processo, além de possibilitar que clientes e usuários validem requisitos sem necessitar de conhecimento extra. Por outro lado, na economia globalizada atual, o desenvolvimento de software por equipes geograficamente distribuídas está se tornando uma norma. Nesse cenário, atividades de verificação e validação de requisitos para um software de média ou alta complexidade podem envolver o tratamento de centenas ou milhares de requisitos. Com essa ordem de complexidade é importante que o engenheiro de software tenha apoio computacional para o desempenho adequado das atividades de aferição de qualidade. Neste trabalho estamos propondo uma estratégia que combina técnicas de processamento da linguagem natural (PLN) e agentes de software para apoiar as atividades de análise dos requisitos. Geramos visões textuais ou gráficas de grupos de requisitos relacionados; visões apóiam a análise de completude, a identificação de duplicidades e de dependências entre requisitos. Utilizamos técnicas de análise de conteúdo para apoiar a identificação de omissões em requisitos não funcionais. Também propomos uma estratégia para a construção ou atualização do léxico da aplicação, utilizando técnicas de PLN. Utilizamos agentes de software para implementar serviços que incorporam as estratégias referidas, e também para atuar como representantes dos participantes do projeto em desenvolvimento.

Palavras-chave

Análise de Requisitos, geração de visões de requisitos, agrupamento de requisitos, verificação e validação em requisitos, omissões em requisitos não funcionais, processamento da linguagem natural em requisitos, agentes e gerenciamento de requisitos.

Abstract

Sayão, Miriam; Leite, Julio Cesar S. P. (Advisor). **Requirements Verification and Validation: Natural Language Processing and Software Agents.** Rio de Janeiro, 2007. 205p. Ph. D. Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In software development process, initial activities can involve requirements elicitation, modeling and analysis (verification and validation). The use of natural language in the register of the requirements facilitates the communication among stakeholders, besides offering possibilities to customers and users to validate requirements without extra knowledge. On the other hand, in the current global economy, software development for teams geographically distributed is becoming a rule. In this scenario, requirements verification and validation for medium or high complexity software can involve the treatment of hundreds or even thousand requirements. With this complexity order it is important to provide computational support for the software engineer execute quality activities. In this work we propose a strategy which combines natural language processing (NLP) techniques and software agents to support analysis activities. We have generated textual or graphical visions from groups of related requirements; visions help completeness analysis, identification of duplicities and dependences among requirements. We use content analysis techniques to support the identification of omissions in nonfunctional requirements. Also, we propose a strategy to construct the lexicon, using NLP techniques. We use software agents to implement web services that incorporate the related strategies, and also agents to act as personal assistants for stakeholders of the software project.

Keywords

Requirements analysis, requirements clustering, requirements visions, requirements verification and validation, lack of nonfunctional requirements, requirements and natural language processing, software agents and requirements management.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

No processo de desenvolvimento do software, atividades relacionadas ao Processo de Requisitos envolvem atividades de elicitação, modelagem, verificação e validação dos requisitos [Leite94]. Tais atividades, pela sua própria natureza, são mais intensivas em comunicação que as demais atividades em um processo de desenvolvimento de software.

No contexto do processo de requisitos, tomemos como exemplo especificamente as atividades de verificação e validação (V&V). O processo de validação envolve diferentes atores, como por exemplo o engenheiro de requisitos, representantes de clientes e usuários e o coordenador ou gerente do processo de desenvolvimento. Nas atividades de verificação, por outro lado, interagem diferentes atores: inspetores, secretário, coordenador, autor do documento. Cada um desses atores participa visando atingir seus próprios objetivos, e a colaboração entre eles é necessária para atingir a meta maior associada a cada processo. Na verificação, a meta envolve obter respostas à pergunta "Estamos construindo o produto corretamente?". Na validação, a pergunta a ser respondida é "Estamos construindo o produto desejado pelos clientes e usuários?".

Dificuldades relacionadas às atividades de comunicação e coordenação de projetos são exacerbadas em ambientes distribuídos de desenvolvimento de software [Carmel99] [Paré99] [Bianchi02] [Damian03] [Herbsleb03]. Diferenças culturais implicam em dificuldades de compreensão da linguagem natural utilizada nos documentos de requisitos e diferenças em fusos horários podem dificultar a comunicação síncrona entre os interessados. Nesse cenário, praticamente inexistem os encontros informais entre desenvolvedores, que muitas vezes resultam no compartilhamento do conhecimento relacionado ao projeto ou mesmo em auxílio na resolução de problemas. Encontros virtuais ou troca síncrona de informações não substituem o relacionamento face a face, e desta forma a confiança entre os interessados e o espírito de grupo são afetados.

As restrições e dificuldades hoje encontradas em ambientes distribuídos de desenvolvimento têm levado muitas empresas a deslocar equipes para interagir diretamente com os clientes e usuários, evitando parte das dificuldades simplesmente pela eliminação da distribuição das atividades do processo de requisitos [Zowghi02] [Damian03a] [Lopes05].

A orientação a agentes tem sido apresentada como adequada para tratar problemas complexos ou de natureza intrinsecamente distribuída [Jennings00] [Message01] ou ainda sistemas nos quais atores desempenham diferentes papéis e interagem visando atingir diferentes objetivos [Cysneiros03]. O uso de agentes tem sido visto também na solução de problemas relacionados ao processo de desenvolvimento de software [Dellen96] [Gaeta02] [Grundy05] [Wongthongtham06].

Documentos de requisitos e outros artefatos manipulados no processo de requisitos costumam ser escritos em linguagem natural. A utilização de uma linguagem formal para registro dos requisitos traria a facilidade de uma verificação automática, mas dificultaria o processo de validação, pois clientes e usuários precisariam conhecer a linguagem formal utilizada. O uso da linguagem natural facilita a comunicação entre os participantes do processo, além de possibilitar que clientes e usuários validem o documento de requisitos sem necessitar de conhecimento extra. Técnicas de processamento da linguagem natural podem ser aplicadas a documentos de requisitos e outros artefatos, apoiando atividades relacionadas aos processos de verificação e validação de requisitos.

Propomos neste cenário a utilização do paradigma de agentes em apoio aos interessados[1] em atividades relacionadas ao Processo de Requisitos, tratando especificamente da verificação e validação dos requisitos. Nossa abordagem inclui também identificação de aspectos de dependência entre requisitos, e compartilhamento efetivo de informações relacionadas ao sistema em desenvolvimento. Utilizamos técnicas da área de Processamento da Linguagem Natural (PLN) para implementação dos serviços de apoio às atividades de V&V, disponibilizados através de agentes de software [Sayão05].

[1] denominamos de interessados todos aqueles que possuem algum interesse no sistema em desenvolvimento (clientes, usuários e profissionais de software e/ou hardware)

Nosso principal objetivo é explorar positivamente as possibilidades trazidas pela aplicação de técnicas de processamento da linguagem natural, implementadas como serviços de agentes pró-ativos de software, para apoio aos processos de verificação e validação de requisitos, no contexto do processo de definição de requisitos. Com isso visamos à obtenção de um documento de requisitos que atenda às necessidades dos usuários (validação) e aos requisitos de qualidade esperados (verificação), respeitando a distribuição de atividades entre os participantes do projeto.

1.1. Desenvolvimento Distribuído de Software (DDS): desafios para o Processo de Requisitos

Diversos relatos envolvendo estudos e experimentos identificaram nas dificuldades de comunicação os maiores desafios enfrentados pelo pessoal envolvido nas tarefas e atividades associadas à etapa de requisitos. Carmel [Carmel99] considera que os impactos trazidos pela distribuição do desenvolvimento estão relacionados à distância, diferenças culturais e de fusos horários entre interessados. Essas dimensões são discutidas no contexto do Processo de Requisitos em [Zowghi02], para quem distância e fusos horários impactam diretamente na comunicação entre interessados, afetando as atividades de elicitação, negociação e priorização de requisitos. Dentre as diferenças culturais, aquelas relacionadas a normas de comportamento, linguagem e costumes podem levar à falta de confiança entre equipes e tendem a impactar negativamente atividades do processo de requisitos.

Em [Prikladnicki04] são relatados os principais resultados de um estudo envolvendo duas organizações e vinte e dois desenvolvedores utilizando DDS (Desenvolvimento Distribuído de Software); os participantes consideraram que o maior desafio no desenvolvimento distribuído é a Engenharia de Requisitos. A distância entre os participantes dificulta a convergência de idéias e impacta na estabilidade dos requisitos; dificuldades relacionadas à comunicação e falta de confiança entre equipes são menores quando há um padrão bem definido a ser seguido, tanto para o processo de desenvolvimento em si quanto para os artefatos gerados.

[Damian03a] apresenta resultados de um estudo de caso abrangendo quatro

sites situados em diferentes países, com a participação de vinte e quatro interessados. Dificuldades de comunicação face-a-face tornam a interação entre grupos dependente das características das ferramentas disponíveis, e interferem diretamente nas várias etapas do Processo de Requisitos. Sem um adequado compartilhamento da informação, a confiança entre equipes é atingida, os encontros virtuais necessários para atividades como priorização e negociação de requisitos tendem a não ser efetivos. A diversidade cultural, em especial diferenças lingüísticas, afeta a compreensão comum dos requisitos e a convergência entre diferentes interesses. *Delays* de tempo devidos aos diferentes fusos horários impactam nas atividades de priorização e negociação, se não devidamente gerenciados. Uma das principais contribuições desse trabalho é discutir e apresentar de forma esquemática as dimensões de comunicação, conhecimento, cultura e diferenças temporais no contexto do desenvolvimento distribuído, bem como identificar claramente as atividades afetadas.

Nossa proposta de trabalho está diretamente relacionada às dificuldades de comunicação entre interessados, e a aspectos do processo de requisitos que são afetados por esses problemas.

Comunicação é fator crítico de sucesso em projetos distribuídos: já em 1996 Gorton e Matwani [Gorton96] apresentaram resultados de um experimento que aponta como sendo de 22% o tempo utilizado em comunicação entre os membros de uma equipe. A maior parte desse tempo, cerca de 17%, foi utilizado em comunicação assíncrona, com uso de e-mail e de uma ferramenta de *groupware* desenvolvida especialmente para a empresa. Um outro estudo, feito por Cherry e Robillard [Cherry04], identificou que atividades de *cognitive synchronization* são responsáveis por aproximadamente 29% do tempo de desenvolvimento de um software. Por *cognitive synchronization* entende-se atividades de comunicação entre dois ou mais desenvolvedores, visando apenas confirmar que eles compartilham o mesmo conhecimento ou a mesma representação do objeto em questão. Outros estudos relacionados em [Cherry04] apontam para atividades de comunicação consumindo de 15 a 41% do tempo total do desenvolvimento.

A proposta de [Lopes05] para o Processo de Requisitos de certa forma corrobora o que foi apontado em [Cherry04]. Nessa proposta, uma equipe co-localizada aos usuários elabora o Documento de Requisitos (SRS ou Software Requirements Specification) e o repassa à equipe que executará o

desenvolvimento. Esta última equipe, após analisar os requisitos relacionados na especificação, identifica as possíveis fontes de problemas devidos por exemplo à ambigüidade ou falta de clareza na definição dos requisitos. Após a reescrita, o documento de requisitos é objeto de negociações entre as duas equipes até que a equipe de desenvolvimento tenha confiança no seu entendimento acerca do estabelecido no documento de requisitos.

Nossas observações junto a duas empresas que utilizam DDS e vários relatos mostram que é bastante comum a situação de uma equipe elaborando o documento de requisitos e posteriormente repassando esse documento para a equipe de desenvolvimento [Zowghi02] [Damian03a] [Prikladnicki03] [Audy04a]. Também encontramos a situação onde a equipe responsável pelo processo de requisitos se desloca até os clientes e usuários para a elicitação dos requisitos, retorna e trabalha este documento junto com a equipe de implementação, diminuindo desta forma os problemas decorrentes dos fatores culturais e lingüísticos para os demais participantes do desenvolvimento. Nestas situações, não há distribuição geográfica nas atividades do processo de requisitos. Acreditamos que isto se deve em parte à inexistência de ferramentas para dar suporte adequado à distribuição dos interessados e em parte para evitar os problemas de comunicação decorrentes da distância.

1.2. Agentes nas Atividades de Verificação e Validação de Requisitos no Desenvolvimento Distribuído de Software

O paradigma de orientação a agentes para desenvolvimento de software tem sido tópico de trabalhos desde o início da década de 80 [Faltings00]. A literatura aponta para diversas e às vezes conflitantes definições para o termo *agentes*, mas a propriedade de *autonomia* é apontada como fundamental para muitos autores [Wooldridge99a]. O termo *autonomia* é utilizado para indicar que agentes podem agir sem necessitar da intervenção de usuários humanos ou outros sistemas. Aplicações utilizando agentes são encontradas em diferentes domínios, e recentemente no domínio da própria Engenharia de Software [Himmelspach03] [Dhavachelvan04] [Grundy05].

Wooldridge define agente como *um sistema de computação situado em algum ambiente e capaz de ações autônomas sobre este ambiente de forma a*

atingir os objetivos desejados [Wooldridge99a]. O agente monitora o ambiente através de sensores, e pode disparar ações que irão atuar sobre esse mesmo ambiente; a forma como isso ocorre aparenta não-determinismo. Um agente também é visto como um elemento interativo, adaptativo e autônomo que possui seu próprio ciclo de vida.

Em 2001 Jennings e Wooldridge [Jennings01] apontavam que a tecnologia de desenvolvimento orientada a agentes ainda estava num estágio inicial, mas que sua aceitação mostrava um futuro significativo na engenharia de software. Trabalhos posteriores confirmaram essa previsão, incluindo aplicações no próprio processo de desenvolvimento de software: [Gaeta02] utilizou agentes no contexto do desenvolvimento distribuído de software, com técnicas de *workflow* e de gerenciamento de documentos para apoiar a comunicação e o gerenciamento do projeto. Em atividades associadas à evolução de requisitos, [Chang01] mostra o uso de agentes móveis para controle da evolução de requisitos em ambientes distribuídos. [Wongthongtham06] utilizou agentes no contexto do desenvolvimento distribuído, com agentes manipulando um repositório controlado que armazenava uma ontologia de termos da própria engenharia de software.

Acreditamos que o uso de agentes é apropriado no contexto do Processo Distribuído de Requisitos [Chang01], porque o uso de agentes em sistemas de software propicia:

autonomia: agentes podem monitorar o ambiente, detectando modificações significativas ou ocorrência de eventos e comunicá-las aos envolvidos. No contexto em pauta, agentes podem monitorar mudanças em artefatos de requisitos, disparando atividades do processo de verificação e validação;

flexibilidade: agentes podem ser dinamicamente inseridos ou retirados do ambiente de execução, de forma transparente. Como agentes pessoais podem ser utilizados, e o conjunto de participantes no processo de desenvolvimento de um software evolui ao longo do processo de requisitos, o conjunto de agentes deve refletir essa evolução;

interatividade: agentes podem se comunicar através de mensagens; atividades de comunicação entre interessados são essenciais às atividades do gerenciamento de requisitos e agentes podem realizar parte dessas comunicações;

agregação do conhecimento distribuído: agentes podem colaborar e cooperar nas atividades que exigirem agregação de conhecimento para tomada de

decisão. O conhecimento relacionado ao sistema está distribuído entre interessados geograficamente dispersos, e agentes podem colaborar nessa agregação para a tomada de decisões;

acesso controlado à informação: definir restrições ao acesso de agentes à informação é tarefa facilmente realizável. No contexto de desenvolvimento de software, as restrições aos diferentes interessados com diferentes direitos de acesso às informações podem ser atendidas pelos agentes;

continuidade temporal: agentes podem estar ativos de forma contínua, monitorando os ambientes, detectando eventos e tomando decisões de forma independente, o que é essencial no caso do desenvolvimento *follow-the-sun*, quando os interessados geograficamente dispersos estarão ativos desempenhando diferentes tarefas durante as 24 horas do dia.

Nossa proposta visa repassar a agentes de software parte das tarefas e das comunicações que atualmente ocorrem entre agentes reais - os participantes do processo de requisitos - focando em atividades de verificação e validação. Visualizamos agentes de software como representantes dos diferentes atores nesses processos, propiciando a efetiva distribuição do trabalho no contexto do processo de requisitos e diminuindo os problemas de comunicação entre os interessados reais.

No processo de **verificação**, uma atividade possível é a inspeção do documento de requisitos. O processo de inspeção caracteriza-se pela utilização de uma técnica de leitura aplicável a um artefato, buscando a localização de defeitos ou inconsistências no mesmo, segundo um critério pré-estabelecido. Algumas das verificações estão relacionadas à omissão ou à duplicação de requisitos: o conjunto de requisitos é verificado em relação à completude (inexistência de omissões) e em relação à existência de duplicidade dos requisitos.

No processo de inspeção os diferentes atores envolvidos no processo de desenvolvimento vêem os artefatos sob diferentes pontos de vista. Tomemos como exemplo o documento de requisitos: ele será analisado através da visão de cada participante. O projetista vê esse documento considerando que pode utilizá-lo como base para a criação da arquitetura do software, considerando as funcionalidades e restrições descritas. O especialista em testes vê o documento como fonte primeira para a geração dos testes, que devem assegurar que o mesmo

atende às necessidades (funcionais e não funcionais) registradas. O engenheiro de requisitos vê o documento buscando identificar se ele atende aos requisitos de qualidade esperados.

A complexidade destas tarefas é amplificada se o número de requisitos a ser verificado ou inspecionado é da ordem de centenas ou mesmo milhares. O agrupamento de requisitos contribui não só para a organização do trabalho, através da geração de grupos menores de requisitos relacionados, mas também agiliza atividades próprias aos processos de verificação e validação, como por exemplo verificação da completude.

No processo de **validação**, engenheiro de requisitos e representantes do cliente e dos usuários avaliam a SRS e interagem com o objetivo de assegurar que os requisitos ali relacionados pelo engenheiro de requisitos correspondem ao esperado por usuários e clientes. Cada representante dos usuários traz a visão do grupo ao qual ele pertence (operacional, gerencial, decisório) e possui determinados interesses e necessidades em relação ao sistema a ser desenvolvido. Neste processo visualizamos o conjunto de requisitos dividido em agrupamentos que correspondem a diferentes características, facilitando para o cliente ou usuário o processo de identificar se as funcionalidades e restrições necessárias ao atendimento de seu trabalho estão refletidas no documento sendo avaliado.

Consideramos que as tarefas de verificação e validação podem ser parcialmente automatizadas por agentes de software. Nesse contexto, a cada ator corresponde um agente que age em seu nome, executando ações em acordo com a perspectiva de seu papel e seguindo o estabelecido em um plano pré-definido. Técnicas de processamento da linguagem natural podem ser aplicadas aos requisitos, possibilitando que parte das tarefas de V&V seja automaticamente realizada. Como exemplo, citamos a atividade de verificação de duplicidade em requisitos, que pode ser automatizada com a utilização de medidas estatísticas de similaridade.

Utilizamos a linguagem **i*** [Yu95] [Yu02] para modelar nossa proposta de uso de agentes para atividades de verificação e validação de requisitos. A Figura 1 apresenta o modelo de dependência estratégica dos processos de V&V de requisitos. Nesse modelo são mostradas as metas de cada agente e as dependências entre agentes na consecução desses objetivos ou metas. Os principais atores (cliente, desenvolvedor, engenheiro de requisitos, gerente do

projeto, inspetor) possuem relações de dependência por recursos, por *goals* (metas) ou por *softgoals*, que representam a intencionalidade dos atores.

Os principais atores envolvidos no processo de requisitos: clientes, usuários e profissionais de software, visualizados na Figura 1, ocupam diferentes posições. Profissionais de software ocupam posições que agregam responsabilidades por tarefas associadas ao gerente de projeto, ao engenheiro de requisitos e aos desenvolvedores. Clientes e usuários ocupam posições de representantes, agregando responsabilidades por tarefas de representação de diferentes perfis e interesses. O inspetor, papel que pode ser desempenhado por clientes, usuários e profissionais de software, possui responsabilidades específicas ao processo de verificação.

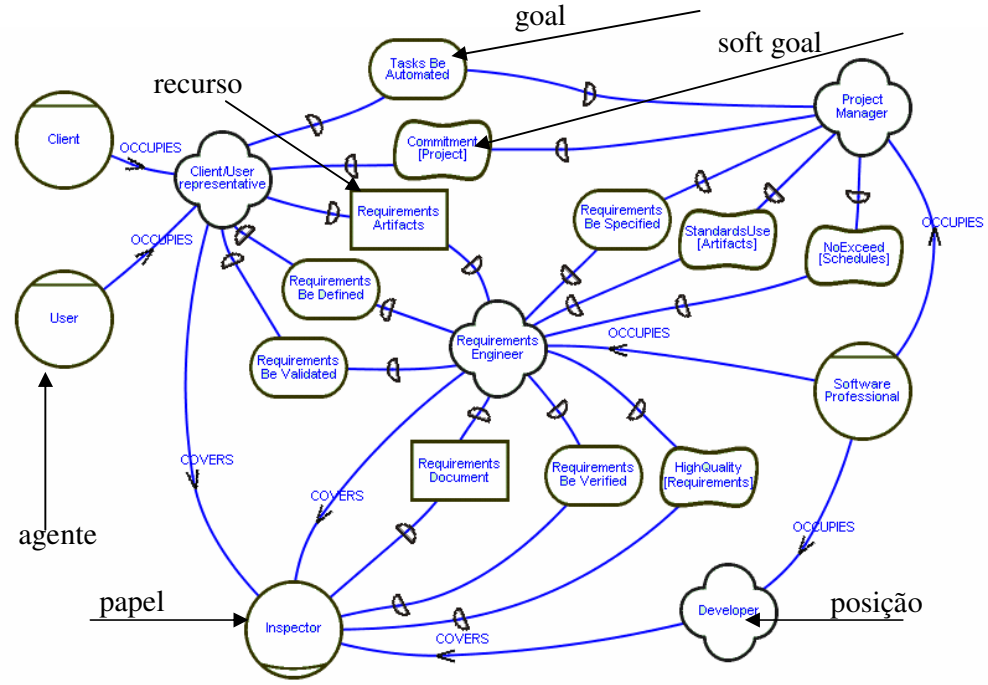


Figura 1 - Modelo de dependência estratégica para verificação e validação de requisitos

As relações entre atores reproduzem a rede de dependências existente entre eles, dependências estas relacionadas a metas, softgoals ou recursos. Numa dependência por metas um ator depende (*depender*) de outro (*dependee*) para atingir objetivos concretos. Numa dependência por *softgoals* um ator depende do outro para satisfazer desejos ou intenções; numa dependência por recursos um ator depende de outro para obter um determinado recurso.

Podemos observar na Figura 1 que para atingir a meta de automação de

tarefas, representantes de clientes e usuários dependem do gerente do projeto. Este, por sua vez, depende do engenheiro de requisitos para que os requisitos sejam especificados atendendo aos padrões vigentes na organização e dentro do prazo previsto para a tarefa. O gerente de projeto depende também do comprometimento dos representantes de clientes e usuários para as atividades do processo de requisitos. O engenheiro de requisitos depende dos representantes de clientes e usuários que os requisitos sejam definidos e validados, e dos inspetores para que os requisitos sejam verificados e atendam a critérios de qualidade. Representantes de clientes e usuários dependem do engenheiro de requisitos para obter os artefatos que serão utilizados nas atividades de V&V, e o inspetor depende do engenheiro de requisitos para obter o documento de requisitos a ser verificado.

Em nossa proposta argumentamos que agentes de software são adequados para o atendimento de algumas atividades relacionadas à V&V. A implementação desta proposta envolve agentes e componentes inter-relacionados para atendimento das seguintes características:

agência: diferentes papéis a serem desempenhados pelos agentes, na representação dos interessados reais;

serviços de comunicação: para possibilitar a troca de informações entre usuários e agentes de software, e para notificação dos interessados na ocorrência de eventos;

serviços de gerenciamento: para possibilitar aos agentes o monitoramento de modificações no ambiente e eventos significativos nesse contexto;

persistência de dados: armazenamento de informações relacionadas a projetos e usuários e uma *baseline* para artefatos de requisitos [Leite95];

serviços de autenticação: para validação dos usuários aptos a utilizar o sistema;

geração de relatórios: para impressão de relatórios e de artefatos armazenados no sistema;

interface com usuários: para possibilitar fácil interação entre usuários e sistema;

Nossa argumentação e justificativa pela adequação do paradigma de agentes às atividades do processo de requisitos, num contexto de desenvolvimento distribuído, é detalhada no capítulo 4. Nossa argumentação considera que o

paradigma de agentes vem sendo apresentado, com sucesso, como alternativa para a modelagem e implementação de sistemas complexos e distribuídos, dado que (i) agentes autônomos podem atingir seus objetivos mesmo em ambientes dinâmicos, (ii) são capazes de interações em alto nível, e (iii) conseguem operar em estruturas organizacionais flexíveis [Jennings00].

1.3. Contribuições deste trabalho

A contribuição fundamental desta tese é a criação de um conjunto de procedimentos, técnicas e ferramentas para apoio às atividades de verificação e validação de requisitos. Parte destes processos fundamenta-se em estratégias de processamento automatizado da linguagem natural, dado que artefatos do processo de requisitos costumam ser modelados com uso da linguagem natural. Agentes de software implementam serviços que automatizam parcialmente os processos propostos.

De forma mais específica, várias contribuições são observadas a partir do trabalho desenvolvido:

- especificação de um **processo para a criação de uma taxonomia da aplicação**, baseada na extração de conceitos relevantes a partir de documentos de requisitos;

- utilização de **processo baseado nessa taxonomia para categorização de requisitos**, gerando agrupamento de requisitos semanticamente relacionados;

- proposta de um **processo para geração de visões dos requisitos**. Mapas textuais e gráficos são gerados e permitem uma melhor organização do trabalho nas atividades de V&V, além de possibilitar a visualização de interdependências entre requisitos, e a navegação através dos relacionamentos existentes entre a taxonomia e requisitos relacionados;

- desenvolvimento de um **processo para automatizar a construção e atualização do léxico da organização** a partir de documentos da organização e de artefatos gerados durante o processo de requisitos. Impactos de diferenças culturais entre participantes do DDS são minimizados quando o léxico inclui também expressões idiomáticas que podem levar a diferentes interpretações devido a fatores culturais;

- elaboração de **técnicas para apoio à detecção de discrepâncias, erros e omissões no documento de requisitos**. Aplicamos medidas de similaridade entre documentos para identificar duplicidade em requisitos, e elaboramos uma forma de identificar omissões em RNF's, tendo como base um dicionário de categorias (requisitos não funcionais) e expressões associadas. O documento de requisitos é avaliado em relação a esse dicionário, fornecendo um conjunto de relatórios e tabelas para subsidiar a análise de incompletude.

- **utilização de agentes na arquitetura da ferramenta de software** para suportar os procedimentos propostos. Na ferramenta visualizada, agentes interagem e fornecem serviços. Os serviços oferecidos incluem ferramentas para diversos tipos de tratamento da linguagem natural, cálculo de medidas de similaridade, notificação de interessados na ocorrência de modificações em requisitos. Os serviços oferecidos pelos agentes destinam-se a apoiar atividades de V&V do processo de requisitos. A ferramenta também é composta por agentes pessoais, destinados a representar humanos participantes do processo de desenvolvimento, visando diminuir sua carga de trabalho.

Até onde vai nosso conhecimento da literatura pesquisada, nenhum outro trabalho apresenta a utilização de agentes de software como apoio à geração de visões textuais ou gráficas de agrupamentos de requisitos, construção do léxico da aplicação ou mesmo atividades de verificação e validação de requisitos. Nesse sentido, as propostas apresentadas neste trabalho constituem uma nova abordagem, mostrando a adequação do uso de agentes como entidades autônomas, colaborando e apoiando atividades de verificação e validação de requisitos.

A adequação do paradigma de agentes para modelagem das funcionalidades providas neste trabalho é demonstrada através de um protótipo. O protótipo criado possibilita a oferta dessas funcionalidades em uma intranet ou na web. Isto apóia o reuso de funcionalidades disponibilizadas via serviços web e agiliza a construção de soluções para problemas mais complexos através da composição de serviços. A notificação automática dos interessados na ocorrência de determinados eventos pelo agente de comunicação poderá ainda levar à diminuição dos problemas de coordenação e comunicação entre participantes do processo de requisitos em ambientes distribuídos de desenvolvimento.

Uma das linhas de trabalho que aplica técnicas de PLN a artefatos de requisitos utiliza um sub-conjunto controlado da linguagem natural com o

objetivo de reduzir a ambigüidade nos requisitos. Diferentemente dessa linha, ressaltamos que a abordagem proposta trabalha com artefatos de requisitos escritos na língua portuguesa, sem quaisquer limitações ou restrições a nível de estruturas gramaticais ou vocabulário utilizados. Nossa abordagem também permite a utilização de requisitos expressos de diferentes formas, como sentenças de requisitos, casos de uso ou estórias do usuário.

1.4. Visão geral deste documento

No capítulo 2 são apresentados os principais desafios para a Engenharia de Requisitos no contexto do Desenvolvimento Distribuído de Software, e suas implicações para atividades de validação e verificação de requisitos. Este capítulo também relaciona trabalhos desenvolvidos com técnicas de processamento da linguagem natural em requisitos de software, uso de agentes para apoio às atividades próprias do processo de desenvolvimento de software e técnicas de processamento de linguagem natural que utilizamos nesta tese. Nossa proposta está apresentada em dois capítulos: o capítulo 3 apresenta os procedimentos desenvolvidos com uso de técnicas de processamento da linguagem natural, seguindo uma perspectiva de processo. O capítulo 4 detalha uma arquitetura baseada em agentes que atende aos processos propostos e vai além, incorporando características necessárias ao atendimento de necessidades específicas do desenvolvimento distribuído de software. Os estudos de caso realizados são apresentados no capítulo 5; o capítulo 6 apresenta trabalhos relacionados, e no capítulo 7 são relacionadas as conclusões e trabalhos futuros.

2
Processo de Requisitos, PLN e Agentes

O ciclo clássico de desenvolvimento de um produto de software tem início pelo Processo de Requisitos e é nesta fase que são determinados os requisitos que o software em construção deverá atender. Este processo gera um conjunto de artefatos que constituem uma *baseline* para o registro e acompanhamento da evolução dos requisitos ao longo do ciclo de desenvolvimento, possibilitando um efetivo gerenciamento de requisitos. A qualidade é fator crítico no documento de requisitos, dado que este será a base para as demais atividades ao longo do processo de desenvolvimento. Várias são as propostas para estruturação das atividades no processo de requisitos.

2.1.
Processo de Requisitos: estruturação das atividades

Segundo Leite [Leite94], este processo é composto pelas fases de elicitação, modelagem e análise de requisitos, representadas no diagrama SADT da Figura 2.

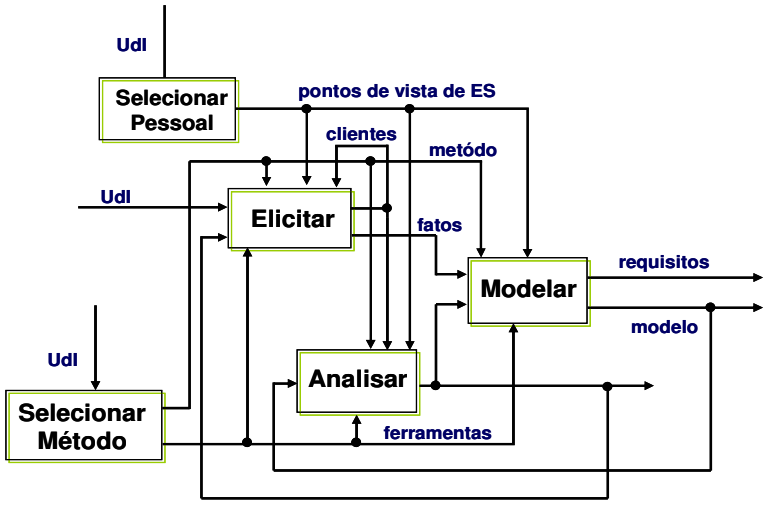


Figura 2 - Modelo SADT da Engenharia de Requisitos [Leite94]

Durante a fase de elicitação são utilizadas diferentes técnicas para a descoberta dos requisitos junto ao conjunto de interessados no sistema, com o objetivo de identificar necessidades e expectativas em relação ao sistema a ser desenvolvido. Estas necessidades e expectativas serão estruturadas e registradas de forma sistemática na fase de modelagem, seguindo um método previamente definido.

A última etapa deste processo é denominada de análise, que envolve as atividades de verificação e validação e tem por objetivo avaliar a qualidade da representação dos requisitos em relação a aspectos como consistência e completude (verificação) e identificar se os requisitos correspondem às expectativas dos clientes e usuários (validação). O contexto geral onde se insere o sistema a ser desenvolvido é denominado de Universo de Informações (UdI) e inclui todas as fontes de informação e pessoas relacionadas ao software (este contexto também é identificado como Universo do Discurso ou domínio da aplicação).

Durante a fase de elicitação a comunicação entre engenheiro de requisitos e clientes e usuários visa à identificação de funcionalidades a serem atendidas pelo sistema a ser desenvolvido. Neste processo, reconhecidamente um dos mais intensivos em comunicação no desenvolvimento de software, surgem termos próprios do domínio da aplicação. Tais termos devem ser registrados num léxico para a aplicação, possibilitando a todos os interessados o compartilhamento de uma mesma compreensão desse Universo de Informação. Várias são as técnicas utilizadas para a elicitação dos requisitos: entrevistas, reuniões, leitura de documentos, workshops e mesmo reuso de requisitos. O sistema a ser desenvolvido deverá ainda respeitar regras e padrões da organização para o qual ele está sendo construído, além de atender à legislação em vigor. Restrições derivadas do contexto de operação também devem ser consideradas e, portanto, nesta etapa não apenas os clientes e usuários devem ser ouvidos.

Na fase de modelagem os requisitos são registrados de acordo com um modelo tal como sentenças de requisitos, cenários, casos de uso, estórias do usuário. O uso da linguagem natural no registro dos requisitos facilita a comunicação entre os clientes, usuários e engenheiro de requisitos e possibilita, posteriormente, a validação desses mesmos requisitos por parte dos clientes e

usuários. O uso de modelos formais para o registro de requisitos facilita atividades de verificação, mas dificulta atividades de validação, pois clientes e usuários teriam que utilizar e compreender esses modelos formais; já o uso da linguagem natural não traz dificuldades adicionais ao processo, pois ela é a linguagem utilizada normalmente pelos interessados.

Na fase de análise os requisitos são verificados em relação ao modelo sendo utilizado e em relação ao atendimento das solicitações de clientes e usuários. Durante a verificação busca-se identificar no documento de requisitos discrepâncias, erros e omissões; este processo é realizado pelos profissionais de software. O documento é avaliado em relação ao modelo utilizado para registro dos requisitos, e uma avaliação sintática verifica se o documento de requisitos respeita a sintaxe do modelo utilizado. Também é verificado se o documento atende aos padrões estabelecidos pela organização, ou atende a modelos de qualidade. Verificações semânticas são realizadas, analisando-se o conjunto de requisitos e buscando encontrar inconsistências entre pares ou conjuntos de requisitos. A avaliação em relação à completude é parte deste processo. O objetivo desta fase é garantir a qualidade do documento que será utilizado posteriormente como guia para as demais atividades do processo de desenvolvimento do software.

Na validação o conjunto de requisitos é avaliado pelos clientes e usuários, e busca-se confirmar se suas necessidades e expectativas estão contempladas no documento analisado. Assim como na fase de elicitação, é importante que, na validação, participem usuários e clientes representando as diferentes visões dos envolvidos ou atingidos pelo sistema em desenvolvimento. Em caso de conflitos, negociações entre os interessados poderão solucionar o impasse ou levar a novas definições. Dependendo da técnica utilizada para o processo de desenvolvimento, a priorização dos requisitos junto com o cliente definirá quais os requisitos a serem trabalhados em cada incremento do software a ser liberado. Estas atividades resultam no comprometimento de clientes e usuários com os requisitos registrados no documento de requisitos.

Kotonya e Sommerville [Kotonya98, Sommerville04] estruturam as atividades do processo de requisitos nas fases de estudo de viabilidade, elicitação e análise, validação e gerenciamento de requisitos. O estudo de viabilidade utiliza regras do negócio, alguns requisitos preliminares e uma descrição de alto nível do

sistema a ser desenvolvido, apresentando a forma como o sistema deverá suportar ou apoiar processos do negócio. Este estudo resulta num parecer recomendando ou não a continuidade do projeto. Na fase de elicitação e análise de requisitos os interessados buscam obter maior conhecimento sobre o domínio da aplicação e identificar tanto os serviços que o sistema deverá prover como as restrições operacionais a respeitar.

É ainda na fase de análise que são identificados e negociados os eventuais conflitos entre requisitos, derivados das diferentes necessidades trazidas pelos interessados, e efetuada a priorização dos requisitos. As atividades relacionadas à representação dos requisitos em algum modelo ou linguagem padronizada são denominadas de especificação dos requisitos. A fase de validação busca mostrar que os requisitos correspondem ao sistema que o usuário deseja obter. Os requisitos são checados ainda em relação a aspectos como consistência, completude e testabilidade, e também quanto à viabilidade de sua implementação. A Figura 3, adaptada de [Sommerville04], apresenta esquematicamente estas fases.

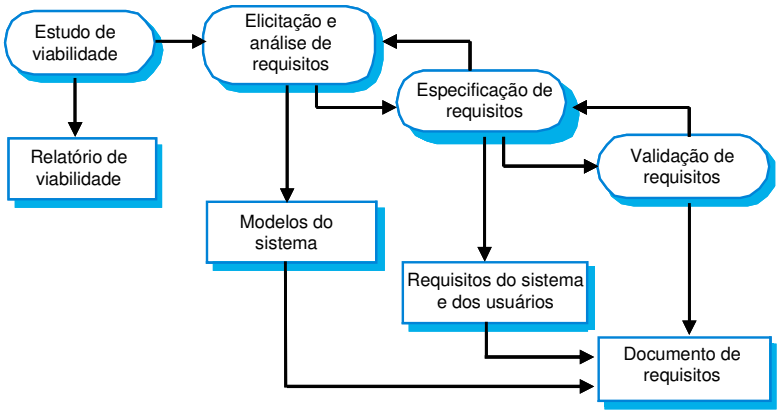


Figura 3 - Processo de Requisitos [Sommerville04]

Ao longo do Processo de Requisitos, e mesmo durante o processo de desenvolvimento do software, não é incomum que os requisitos já definidos sofram alterações devido a diferentes motivos como por exemplo mudanças no contexto onde o software está inserido, novas expectativas por parte dos clientes/usuários, negociação entre clientes e desenvolvedores, etc. Tanto a proposta de Leite quanto a de Kotonya e Sommerville consideram e acomodam a inevitabilidade dessas mudanças. A Figura 4, adaptada de [Sommerville04], mostra um modelo genérico para as etapas que envolvem elicitação, classificação e organização dos requisitos, priorização e negociação, e a documentação dos

requisitos. Como um modelo geral, ele pode ser instanciado a cada processo de desenvolvimento; a espiral mostra que estas atividades são cíclicas, e a cada execução do ciclo deve aumentar a compreensão dos interessados e a completude dos requisitos.

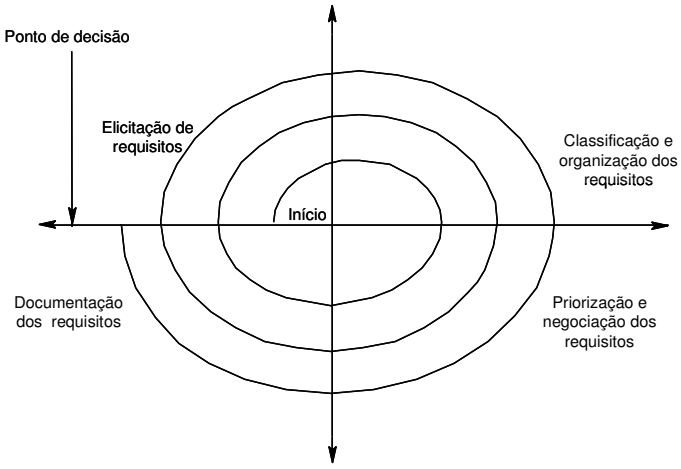


Figura 4 - Ciclo de elicitação e análise [Sommerville04]

Os vários artefatos gerados durante o processo de requisitos visam registrar as necessidades e expectativas dos interessados, facilitar a comunicação com clientes e usuários e também servir como base para o desenho e implementação do sistema. Um desses artefatos, denominado de *Documento de Requisitos*, é a base para o desenvolvimento do software; os requisitos nele registrados delimitam a abrangência do software, estabelecem funcionalidades requisitadas pelo conjunto de clientes e usuários, fornecem subsídios para o processo de verificação e validação do software construído. Não se questiona a importância do processo de requisitos para o sucesso de um projeto de software; sem um documento de requisitos de qualidade, as estimativas de custos ficam prejudicadas, o cronograma de execução passa a ser apenas uma estimativa, não há como afirmar que um projeto foi concluído (como saber se todos os requisitos foram implementados?), o processo de validação do software pelos usuários é dificultado.

Estas duas abordagens para o processo de requisitos diferem na estruturação das várias atividades, mas coincidem na essência das mesmas. O que Leite denomina de análise de requisitos, englobando verificação e validação, Kotonya e Sommerville chamam de validação. Adotaremos neste trabalho a estruturação de

Leite, dado que ela separa mais claramente as atividades a serem executadas nesta fase.

O conjunto de atores envolvido num processo de requisitos inclui representantes do cliente e dos usuários, gerente do projeto e engenheiros de requisitos, de software e de sistemas, entre outros. Cada um desses atores participa visando atingir seus próprios objetivos, e a colaboração entre eles é necessária pois existe uma meta maior associada a cada fase. Considerando especificamente as atividades de V&V: (i) na verificação é preciso determinar se um artefato de requisitos atende aos preceitos de qualidade estabelecidos pela organização; significa verificar se ele está internamente completo, consistente e correto, o que possibilita passar às etapas seguintes no processo de desenvolvimento e (ii) na validação é preciso assegurar que este mesmo artefato atende às necessidades dos clientes e usuários.

Repetindo o que já foi expresso no capítulo 1: na verificação, a meta envolve obter respostas à pergunta "Estamos construindo o produto corretamente?". Na validação, a pergunta a ser respondida é "Estamos construindo o produto desejado pelos clientes e usuários?" [Sommerville04]. Essas meta-questões serão respondidas pelos interessados, com utilização de métodos e técnicas apropriados a cada caso.

2.2. Processo de requisitos em ambientes distribuídos de desenvolvimento

Os desafios enfrentados no DDS têm sido avaliados segundo diferentes dimensões. Segundo [Paré99], times virtuais devem ser avaliados segundo as dimensões de tecnologia, contexto (ambiental e organizacional), processo e dinâmica do time (padrões de comunicação, cooperação, compartilhamento de informações) e estratégias para gerenciamento do desenvolvimento. Este *framework* [Paré99] foi utilizado como ponto de partida para um estudo em empresas utilizando DDS; os resultados desse estudo mostram que a distância dificulta o comprometimento dos times e alinhamento com propósitos assumidos. Atividades sociais ou mesmo encontros casuais também são minimizadas; tais atividades permitem o desenvolvimento do espírito de equipe e diminuem a possibilidade de ocorrência de conflitos. A distância também afeta atividades do

gerenciamento do projeto: a liderança utiliza a comunicação para manter a união da equipe e motivar para o andamento do projeto. Projetos bem sucedidos exigem planejamento detalhado, maior esforço e disciplina para seu acompanhamento, gerenciamento por objetivos e uso de métodos padronizados.

Considerando-se especificamente o Processo de Requisitos, [Damian03a] esquematiza conforme Figura 5 as dimensões derivadas da distribuição dos interessados, os desafios identificados e as atividades por eles afetadas. Nesse estudo, os problemas derivados da distribuição geográfica são vistos como associados à comunicação, conhecimento, cultura e diferenças temporais.

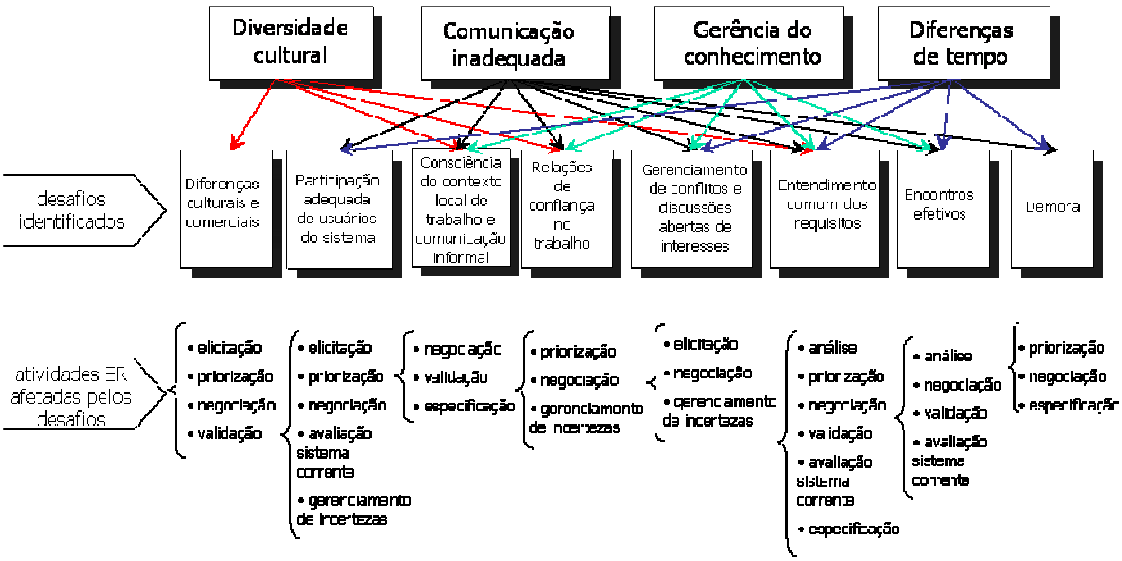


Figura 5 - Impactos do desenvolvimento distribuído no Processo de Requisitos [Damian03a]

Conforme pode ser verificado na Figura 5, atividades de verificação e validação são diretamente atingidas pelos seguintes fatores: diversidade cultural, comunicação inadequada, gerência do conhecimento e diferenças temporais. A literatura sobre o tema apresenta inúmeros trabalhos investigativos do impacto da distribuição das atividades no processo de desenvolvimento, e os resultados encontrados não são divergentes na sua essência [Carmel99] [Paré99] [Bianchi02] [Herbsleb03] [Prikladnicki04] [Cherry04].

Concluindo, como resultado da literatura pesquisada e das informações iniciais coletadas junto a empresas, os principais desafios são:

a) **comunicação e linguagem**: a linguagem natural, normalmente utilizada para registrar os requisitos, é inerentemente ambígua, e além disso, as

diferenças culturais podem levar a uma compreensão equivocada. Conflitos entre requisitos colocados por diferentes interessados também são mais difíceis de serem negociados; a convergência de idéias é comprometida. Do nosso ponto de vista, sem dúvida o fato da língua utilizada para comunicação e documentação não ser a língua nativa de todos os envolvidos é o fator de maior impacto dentre as diferenças culturais. Dos relatos obtidos junto às organizações objeto da avaliação inicial do nosso estudo de caso, mesmo quando utilizada a mesma língua podem ocorrer ambigüidades - caso do português do Brasil e de Portugal;

b) **fusos horários**: se adequadamente explorados, os diferentes fusos horários das equipes podem levar a um bom aproveitamento do tempo (caso do desenvolvimento *follow-the-sun* ou *24-hour*). Em oposição, se a localização geográfica das equipes e as diferenças de horário não possibilitarem coincidência parcial de turnos de trabalho, encontros virtuais serão possíveis apenas se uma das partes se dispuser a modificar seu horário de trabalho, antecipando ou postergando horários de entrada ou de saída do trabalho, ou mesmo de intervalos. Atividades do Processo de Requisitos que exigem negociação ou comprometimento entre interessados são diretamente impactadas pela diferença de fusos horários;

c) **fatores culturais e contextuais e distribuição do trabalho**: fatores culturais e contextuais estão associados à confiança entre equipes [Paré99] [Audy04] [Damian03] e impactam decisões gerenciais relacionadas à distribuição do trabalho. Confiança é necessária para manutenção do espírito de equipe, para troca efetiva de informações entre integrantes de equipes distantes e para realizar a atribuição das tarefas. Aspectos associados à linguagem estão colocados no item a acima;

d) **gerenciamento do conhecimento**: o volume de informações a ser compartilhado durante o desenvolvimento de um sistema envolve documentação técnica, artefatos gerados nas várias etapas do processo, registros de comunicações formais e informais trocadas entre clientes, usuários e desenvolvedores, "dicionários de viagem", ... O compartilhamento efetivo dessas informações é um desafio: mesmo com a utilização de um repositório [Gorton96] [Damian03] existem decisões associadas: utilizar repositório central, ou distribuir as informações?

replicar a base de informações? que informações devem ser compartilhadas? como manter desenvolvedores atualizados em relação a alterações importantes?

e) **coordenação do projeto**: atividades de gerenciamento e coordenação do desenvolvimento são diretamente afetadas pela distribuição geográfica: a atenção do gerente será compartilhada entre as várias equipes, exigindo maior disciplina e atenção. O gerente precisa obter o comprometimento das várias equipes, motivá-las e manter o espírito de grupo mesmo quando parte das equipes está distante. Análises de impacto motivadas por alterações em requisitos poderão implicar em modificações no trabalho das equipes distantes. Para gerar novas estimativas de prazos, o gerente necessitará ter conhecimento da carga de trabalho da equipe distante. Eventos como este aumentam as necessidades de comunicação. Ainda, se o sistema a ser desenvolvido necessita de hardware e/ou software especial, o gerente deve se certificar que todas as equipes têm acesso aos recursos necessários, evitando que as atividades de desenvolvimento sofram interrupções.

2.3. Análise de Requisitos: importância das atividades de verificação e validação

É fato conhecido que o custo de descobrir e corrigir um defeito na fase de testes de software é de 5 a 100 vezes maior que o custo de descobrir e corrigir o problema ainda no Processo de Requisitos [Boehm76] [Bohem01] [Rosenberg98] [Blackburn01]. A correção de um defeito identificado na fase de testes pode implicar em re-trabalho nos artefatos gerados nas fases anteriores: requisitos, arquitetura, projeto e implementação. Problemas decorrentes de atividades desenvolvidas ainda no processo de requisitos são reportados tanto pela literatura acadêmica como por empresas que atuam no mercado de desenvolvimento.

Após avaliar um bom conjunto de organizações, Capers Jones [Jones96] afirma que o processo de requisitos é deficiente em mais de 75% das empresas; isto mostra que obter os requisitos corretos é também uma das mais difíceis atividades de um projeto de software. Um estudo envolvendo empresas européias de desenvolvimento de software registrou que mais de 50% delas identificou

problemas na área de especificação de requisitos [El-Emam00].

Defeitos em requisitos, isoladamente, é a falha com maior freqüência em projetos de software: erros em requisitos são responsáveis por uma taxa de 70 a 85% do custo do re-trabalho [Lefingwell97] [Bohem01].Estudos relativamente recentes indicam que aproximadamente 50% das falhas detectadas na fase de testes são causadas por defeitos em requisitos [Blackburn01], e que a correção dessas falhas implica em re-trabalho muitas vezes evitável. Detectar e corrigir problemas ainda na fase de requisitos é, portanto, uma medida fundamental para contribuir para a conclusão do software dentro dos custos e cronogramas previstos.

Em relação à importância da validação, lembramos que o Chaos Report 2004 do Standish Group [Standish04], envolvendo 9.236 projetos de software, mostra que mesmo entre os projetos que foram concluídos no prazo e dentro do orçamento previsto, apenas 66% atenderam a todas as características solicitadas pelos clientes. Isto reafirma a relevância do envolvimento de clientes e usuários no processo de validação dos requisitos, para que o conjunto de requisitos definidos realmente corresponda às expectativas.

No contexto do desenvolvimento distribuído, os participantes desse processo estarão geograficamente separados. A interação face-a-face ou mesmo mediada por ferramentas de comunicação nem sempre será viável, devido à distância e à diferença de fusos horários entre clientes, usuários e engenheiros de requisitos. Nesse caso as dificuldades inerentes aos processos de verificação e validação serão amplificadas, devido às dificuldades de comunicação e diferenças culturais já relatadas.

A automação parcial de atividades associadas aos processos de verificação e validação contribuirá para a descoberta de defeitos ainda no processo de requisitos, o que, acreditamos, melhorará a qualidade do documento de requisitos e diminuirá o re-trabalho decorrente da descoberta tardia de defeitos.

2.4. Verificação e Validação: uso de PLN e Agentes de Software

Técnicas de processamento da linguagem natural têm sido aplicadas a documentos de requisitos com diferentes objetivos: melhoria da qualidade dos

requisitos, agrupamento de requisitos relacionados, identificação de abstrações, identificação de entidades relevantes. Por outro lado, agentes tem sido utilizados em diversas áreas do conhecimento, e também no contexto de desenvolvimento distribuído de software. Apresentaremos alguns desses trabalhos de forma breve, dado que eles são relacionados ao trabalho desenvolvido nesta tese.

Um trabalho pioneiro na avaliação da qualidade de documentos de requisitos escritos em linguagem natural foi desenvolvido pelo Software Assurance Technology Center (SATC) do Goddard Space Flight Center (GSFC-NASA) [Hammer96] [Wilson97] [Rosenberg98]. Uma ferramenta, denominada de ARM (Automated Requirement Measurement), extrai um conjunto de métricas. A ferramenta obtém valores indicativos de linhas de texto (indicador do tamanho do documento), imperativos (frases indicando ação), continuação (frases que introduzem a especificação de requisitos de nível mais baixo), diretivas (referências para tabelas, figuras, notas), frases dúbias (contendo palavras ambíguas), incompletos (os denominados TBD´s - *to be defined*) e escolhas/opções (palavras que mostram ausência de definição). Os valores obtidos devem ser comparados em relação à média de outros documentos da organização, apontando desvios e pontos que devem receber avaliação e talvez medidas corretivas.

Gervasi e Nuseibeh [Gervasi02] defendem a utilização de métodos formais "leves" para a validação automática de requisitos escritos em linguagem natural. Um documento de requisitos com aproximadamente 250 páginas foi analisado com a utilização de diversas ferramentas, entre elas Circe e Cico, que dão suporte à extração de modelos dos requisitos e sua validação. O trabalho realizado coletou também um conjunto de métricas sobre o documento de requisitos, o sistema nele descrito e o próprio processo de escrita do documento.

Em outra linha de trabalho, Palmer e Liang [Palmer92] buscaram técnicas para agrupar requisitos, idealizando um algoritmo que foi denominado de Two Tiered Clustering (TTC). O objetivo do algoritmo é agrupar um conjunto de M requisitos, de forma que o número N de grupos seja tal que N<<M. Os requisitos são inicialmente agrupados considerando os verbos extraídos do documento, e um thesaurus de verbos cuja construção teve por base um trabalho anterior em requisitos de um sistema operacional. Esses grupos iniciais são então subdivididos em conjuntos similares, utilizando a medida do coseno (seção 2.5.3).

O gerenciamento de processos distribuídos de engenharia de software, apoiado por agentes, foi descrito em [Gaeta02]. Desenvolvido no contexto do projeto GENESIS (GEneralised eNvironment for procEsS management in cooperatIve Software engineering), visando apoiar o gerenciamento de projetos e a comunicação entre engenheiros de software, responsabiliza agentes pela manipulação de exceções, pela sincronização de processos entre os sites distribuídos e pela monitoração e coleta de informações relacionadas a processos.

A utilização de agentes de software para atividades de gerenciamento e controle da evolução de requisitos em ambientes distribuídos de desenvolvimento foi relatado em [Chang01]: agentes móveis foram utilizados para gerenciamento e controle de versões de documentos de requisitos, distribuídos pelos diversos sites de uma organização.

Na área de testes de software, agentes foram propostos como um novo paradigma para incrementar o desenvolvimento de software numa ampla gama de aplicações [Ponnurangam05]. O sistema é composto por agentes pessoais, que interagem com o usuário e executam tarefas específicas, e agentes de serviço, que executam em *background* e atendem ao usuário de forma indireta. Os agentes de serviço ou são especializados em algum tipo de teste de software, ou são responsáveis pela distribuição de recursos de forma eqüitativa e eficiente.

Técnicas de processamento da linguagem natural e uso de agentes de software para apoio a atividades do processo de desenvolvimento de software, portanto, já têm sido apresentados na literatura. Esta proposta utiliza algumas técnicas de tratamento da linguagem natural para apoiar atividades de V&V em requisitos, e agentes de software na ferramenta de suporte.

2.5. Alguns métodos e técnicas de PLN para apoio ao processo proposto

Na proposta apresentada nesta tese as estratégias para apoio às atividades de verificação e validação de artefatos de requisitos utilizam diversos recursos da área de Processamento da Linguagem Natural. Nas seções a seguir descreveremos brevemente os métodos, técnicas e ferramentas dessa área que subsidiaram a construção da nossa estratégia de trabalho para V&V.

2.5.1.
Part-Of-Speech Tagger (POS tagger)

Em processamento da linguagem natural *taggers* são sistemas que analisam um texto e inserem etiquetas morfológicas, gramaticais ou sintáticas a cada item lexical. Um *part-of-speech tagger* é um etiquetador morfossintático, que analisa o texto e identifica as categorias gramaticais como substantivos, adjetivos, pronomes e verbos, dentre outras. Para sinais de pontuação pode ser utilizado o próprio sinal, enquanto que para palavras estrangeiras e fórmulas utiliza-se um rótulo único.

O etiquetador basicamente insere uma etiqueta do conjunto utilizado (*tagset*) junto ao texto; nesta tarefa ele pode utilizar um léxico e um conjunto de procedimentos que apóiam o processo de definir a etiqueta a ser utilizada numa determinada palavra. Estes dois componentes – léxico e conjunto de procedimentos – fazem parte do modelo da língua utilizado para a tarefa de etiquetagem [Aires00]. Alguns etiquetadores exigem que o texto de entrada esteja num formato específico - por exemplo, o etiquetador pode exigir que o texto esteja *tokenizado*, com apenas uma palavra ou caracter de pontuação por linha. A Figura 6 mostra um esquema genérico para um etiquetador morfossintático, com a fase de *tokenização* já incorporada.

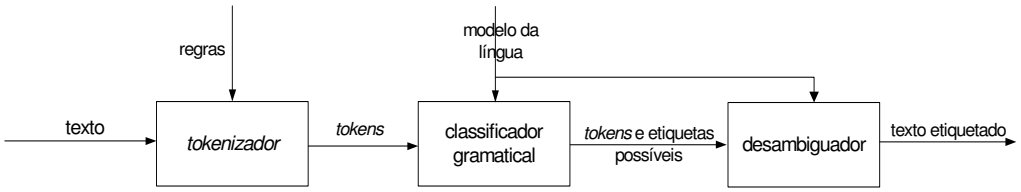


Figura 6 - Visão geral de um etiquetador morfossintático

Após o texto ser *tokenizado* a fase de classificação gramatical tem início. Para cada *token* o classificador busca no léxico as classes gramaticais possíveis. Se o *token* não é encontrado no léxico, então o etiquetador utiliza procedimentos específicos visando encontrar uma classificação do mesmo. Nos casos onde há ambigüidade, ou seja, o *token* pode receber mais de uma etiqueta, o desambiguador utiliza informações do contexto para realizar a desambiguação e definir a etiqueta mais provável para o *token* sendo analisado.

O modelo da língua utilizado pelo etiquetador pode ser baseado em regras,

em casos ou em árvores de decisão, e neste caso o etiquetador é denominado de simbólico ou lingüístico. O modelo pode utilizar representação baseada em Markov, em árvores de decisão probabilísticas ou distribuição estatística de palavras no texto, e neste caso o etiquetador é denominado de probabilístico ou estatístico. Neste trabalho foi utilizado o etiquetador QTAG, detalhado em [Mason97], e que é classificado como um etiquetador probabilístico.

O QTAG é denominado de etiquetador probabilístico porque utiliza a etiqueta mais provável para um termo específico, após identificar no modelo de linguagem todas as possíveis etiquetas (alguns termos podem ser classificados em várias categorias). Para casos onde ao termo sendo analisado podem ser atribuídas diversas categorias, a definição da etiqueta a ser utilizada considera o contexto no qual o termo está inserido. Isto significa que etiquetas de termos próximos ao termo em pauta também são avaliadas, e será utilizada a etiqueta correspondente à seqüência de maior freqüência.

O QTAG utiliza um dicionário que registra os termos da língua (o léxico) juntamente com as probabilidades associadas a cada possível etiqueta para o termo. Inicialmente o etiquetador consulta o dicionário, obtém informações sobre o termo sendo avaliado e as combina com informações do contexto, considerando as etiquetas dos dois termos anteriores. A etapa seguinte avalia ainda o termo em pauta em relação aos dois termos posteriores, e a etiqueta de maior probabilidade de ocorrência é escolhida.

Os recursos utilizados por este etiquetador são um dicionário de termos, com etiquetas e freqüências associadas, e uma matriz de seqüências de etiquetas e respectivas freqüências. Estes recursos podem ser abstraídos de um corpus anotado, o que possibilita que ele seja adaptado para trabalhar com diferentes linguagens; apesar de ter sido desenvolvido originalmente para a língua inglesa, já existem modelos para os idiomas romeno e sueco [Tufis98], além do português.

Para a língua inglesa, a taxa de precisão das etiquetas inseridas é da ordem de 96,3% [Mason97]. Para a língua portuguesa, este etiquetador foi treinado pelos pesquisadores T. Sardinha e R. Lima-Lopes, associados ao Lael/PUCSP, utilizando um corpus de textos jornalísticos composto por aproximadamente 500 mil palavras e etiquetado por lingüistas. Conforme dados experimentais, a precisão deste etiquetador para textos em língua portuguesa é da ordem de 93% [Sardinha04]. O conjunto de etiquetas para a língua portuguesa está relacionado

na Tabela 1, e a Tabela 2 apresenta o resultado do etiquetador para a frase "O modelo de língua utilizado pelo etiquetador pode ser baseado em regras.".

Tabela 1 - Etiquetas utilizadas pelo QTAG

Etiqueta	Significado
ADJ	Adjetivo
ADV	Advérbio
ARTD	Artigo definido
ARTI	Artigo indefinido
CJ	Conjunção
CPR	Contração de preposição e artigo, pronome ou advérbio
ESTR	Palavra estrangeira
IN	Interjeição
N	Substantivo
NUM	Numeral
PART	Particípio passado
PRN	Pronome
PRP	Preposição
PT	Pontuação
V	Verbo

Tabela 2 - Frase etiquetada pelo QTAG

```
<w pos="ARTD">O</w>
<w pos="N">modelo</w>
<w pos="CPR">da</w>
<w pos="N">língua</w>
<w pos="PART">utilizado</w>
<w pos="CPR">pelo</w>
<w pos="N">etiquetador</w>
<w pos="V">pode</w>
<w pos="V">ser</w>
<w pos="PART">baseado</w>
<w pos="PRP">em</w>
<w pos="N">regras</w>
<w pos="PT">.</w>
```

No contexto desta tese, este etiquetador foi utilizado como parte dos procedimentos necessários para a identificação de termos ou expressões representando atores em documentos gerados e/ou manipulados durante o processo de requisitos. A descrição completa dos procedimentos é apresentada no capítulo 3.

2.5.2. Representação de documentos: abordagem bag-of-words

Um dos problemas a ser resolvido quando visamos ao processamento automatizado de documentos escritos em linguagem natural é a escolha da estrutura a utilizar para a representação desses documentos; este problema já foi enfrentado pelas áreas de Recuperação de Informação (RI) e Mineração de Textos (MT). Em RI o problema consiste em recuperar documentos previamente armazenados num repositório, de forma a atender a uma determinada consulta realizada por um usuário. Em MT, o problema consiste em extrair conhecimento de um repositório, possivelmente utilizando algoritmos de aprendizado de máquina. Em ambos os casos impõe-se a necessidade de estruturar os documentos

analisados, visando possibilitar a sua manipulação.

Uma das possíveis abordagens para a estruturação de documentos é denominada de *bag-of-words*. Nessa abordagem, cada documento é representado como um vetor dos termos que ocorrem no documento: os vetores são derivados de matrizes termo-documento, cuja estrutura é similar à apresentada na Tabela 3. Em matrizes termo-documento, colunas representam termos e linhas representam documentos. Cada um dos *n* documentos é representado por um vetor de tamanho *t*, onde *t* corresponde ao número de termos; o vetor pode representar todos os termos existentes no conjunto de documentos, ou ainda todos os termos relacionados num dicionário. Os valores nas células registram o peso que um determinado termo representa para cada um dos documentos que faz parte do repositório ou do conjunto avaliado.

Tabela 3 - Estrutura genérica para uma matriz termo-documento

	$termo_1$	$termo_2$	$termo_3$	$termo_4$	.....	$termo_t$
doc_1	$peso_{11}$	$peso_{12}$	$peso_{13}$	$peso_{14}$	.....	$peso_{1t}$
doc_2	$peso_{21}$	$peso_{22}$	$peso_{23}$	$peso_{24}$	.....	$peso_{2t}$
.....	.....	.....	.....	.....	.....	.....
doc_n	$peso_{n1}$	$peso_{n2}$	$peso_{n3}$	$peso_{n4}$	.....	$peso_{nt}$

O peso de um termo indica a relevância desse termo para o documento em questão, e seu valor é nulo para termos que não estejam presentes no documento. Existem diversas abordagens para definição do peso de cada termo num documento, a freqüência é uma medida presente em várias fórmulas para o cálculo do peso.

Entre as abordagens mais freqüentemente utilizadas estão *boolean*, *tf (*abreviação para *term frequency*) ou *tfidf (*abreviação para *term frequency inverse document frequency*).

Na medida *boolean*, o valor do peso é zero se o termo não está presente no documento, ou 1, caso contrário. Esta é uma medida extremamente simples de ser computada, e quando utilizada em operações de RI causa o retorno de todos os documentos da coleção que contenham o termo objeto da consulta. O uso da medida *boolean* implica em considerar que todos os documentos que contenham o termo são igualmente relevantes.

Na medida *tf* o valor do peso é a própria freqüência do termo no

documento. Em operações de RI isto torna viável classificar os documentos obtidos em resposta a uma dada consulta, colocando no topo da lista aqueles que se acredita melhor respondam à consulta realizada.

A medida *tfidf* , apresentada na equação (1), é calculada ponderando a freqüência do termo por um fator que minimiza o peso de termos presentes em grande parte dos documentos, pois tais termos não são considerados discriminantes. Termos menos freqüentes ou raros terão um fator de ponderação maior. Esta medida é bastante utilizada em processos de MT.

$$tfidf(t_j, d_i) = freq(t_j, d_i) \times \log \frac{n}{d(t_j)} \quad (1)$$

Nesta fórmula, $freq(t_j, d_i)$ representa a freqüência do termo j no documento i, n representa a quantidade de documentos da coleção e $d(t_j)$ representa a quantidade de documentos que contém o termo t_j. Termos que estejam presentes em todos os documentos terão peso nulo, pois log 1 vale zero; o uso da função *log* evita que um termo que apareça em apenas um documento tenha peso duas vezes maior que outro termo que esteja presente em dois documentos da coleção.

Um dos problemas enfrentados quando se utiliza a abordagem *bag-of-words* para estruturação de documentos, considerando todos os termos presentes no conjunto de documentos, é a grande dimensão da matriz termo-documento. Cada documento é representado por um vetor de termos, e todos os termos presentes no conjunto de documentos devem ser representados nesses vetores. A matriz gerada é possivelmente uma matriz esparsa, pois mesmo termos que apareçam em um único documento deverão estar representados. A redução da dimensionalidade da matriz de termos pode ser obtida com o uso dos *stems* dos termos (vide seção 2.4.4.) ou seleção dos termos a representar.

Uma maneira de encontrar termos pouco representativos combina as leis de Zipf [Zipf49] e os cortes de Luhn [Luhn58]. A teoria de Zipf afirma que o produto da freqüência de um termo e sua classificação (*rank*) é aproximadamente constante. Obtém-se a freqüência dos termos de um conjunto de documentos, gera-se a classificação desses termos em ordem descendente de freqüência (*rank*) e gera-se um gráfico de classificação versus freqüência, como pode ser visualizado na Figura 7.

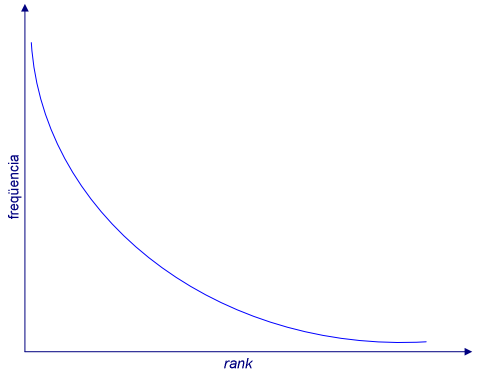


Figura 7 - Curva segundo a lei de Zipf

Esta teoria foi utilizada posteriormente por Luhn [Luhn58] que ponderou que os termos relevantes para a indexação de documentos ficam concentrados numa região delimitada por dois pontos de corte. Termos acima do limite superior e termos abaixo do limite inferior são considerados pouco discriminantes, por serem muito freqüentes ou por serem muito raros. Uma das formas de implementar o conjunto de termos que pertencem ao limiar superior é agrupar artigos, pronomes, advérbios, preposições e conjunções num conjunto denominado de *stoplist*. A Figura 8 mostra limites inferior e superior para uma curva de Zipf, segundo a abordagem de Luhn.

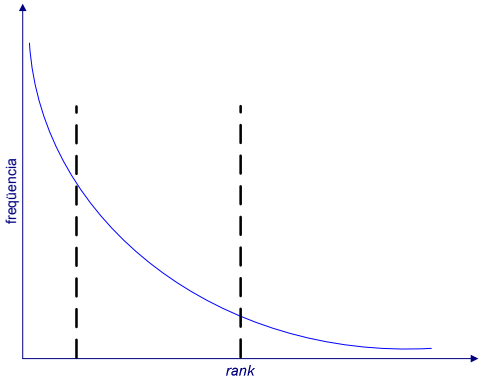


Figura 8 - Curva de Zipf com os cortes propostos por Luhn

A região que engloba os termos com maior freqüência inclui termos como artigos (o, a), preposições (de, da), conjunções (e, ou), que estão presentes em quase todos os documentos e não são significativos para representação dos documentos. A região central inclui termos como substantivos, adjetivos, verbos e que podem ser utilizados para a representação de documentos. A terceira região inclui termos com baixa freqüência de ocorrência, muitas vezes apenas uma ocorrência, e que são considerados “ruídos”.

A delimitação dessas regiões não é tarefa trivial, e à decisão sobre os limites está associado um certo grau de arbitrariedade. Para a terceira região, [Meadow00] sugere que seja utilizado o limiar de 1 ou 2 para a freqüência, [Daile96] sugere que o limiar utilizado seja 4. Para a primeira região, é comum o uso de um conjunto de *stopwords*[2] que agrega os termos da língua que são artigos, pronomes, preposições, conjunções. [Meadow00] observa que o conjunto de *stopwords* é sensível ao contexto: por exemplo, retirar o termo A num contexto de saúde pode excluir indevidamente referências à vitamina A.

No contexto desta tese a representação de documentos utilizou a abordagem *bag-of-words* para apoiar a identificação de termos relevantes em documentos gerados ou manipulados no processo de requisitos, e também para apoiar a identificação de similaridades entre requisitos.

2.5.3. Similaridade entre documentos

Um das formas de se identificar documentos similares utiliza a representação espaço-vetorial. No modelo espaço-vetorial [Salton88] são utilizadas matrizes termo-documento, já referidas na seção anterior. Cada elemento ou termo do vetor é considerado uma coordenada num espaço vetorial euclidiano t-dimensional, e a posição do documento em cada dimensão é dada pelo peso associado. Documentos localizados numa mesma região desse espaço possuem conteúdos similares, e uma forma de determinar essa similaridade está relacionada ao ângulo entre documentos representados nesse espaço vetorial. Uma representação do espaço vetorial assim definido pode ser visualizada na Figura 9.

[2] *Stopwords* são aqueles termos da língua geral ou elementos tais como preposições, artigos, conjunções e outros termos que não apresentam relevância ou valor terminológico [Teline03]; o conjunto desses termos é denominado de *stoplist*

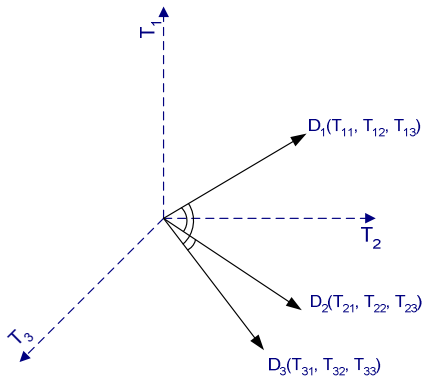


Figura 9 - Espaço vetorial para representação de documentos [Salton83]

No exemplo da Figura 9, o espaço vetorial é definido pelos eixos T_1, T_2 e T_3, que correspondem aos três termos utilizados na representação dos documentos D_1, D_2 e D_3. Os valores utilizados para a representação de cada documento, nesse espaço, são determinados pelos pesos associados aos termos T_1, T_2 e T_3 (e representados, na figura, por T_{11}, T_{12}, T_{13}, ...). Pode-se observar que o ângulo formado pelos vetores que representam os documentos D_2 e D_3 é menor que os ângulos formados por D_1 e D_3 ou D_1 e D_2, indicando um maior grau de similaridade entre eles. Métricas de similaridade baseadas nesta representação são a distância euclidiana e o coseno do ângulo formado pelos documentos.

Tomando como exemplo dois documentos quaisquer, que chamaremos de *x* e *y*, e adotando a representação espaço-vetorial para ambos, considerando um conjunto de *t* termos, o cálculo do coseno do ângulo formado pelo par de documentos é dado pela fórmula apresentada em (2a), e o cálculo da distancia euclidiana é dado pela fórmula em (2b).

$$\cos(x, y) = \frac{\sum_{i=1}^{t} x_i \, y_i}{\sqrt{\sum_{i=1}^{t} x_i^2} \sqrt{\sum_{i=1}^{t} y_i^2}} \quad (2a) \qquad \delta = \sqrt{\sum_{i=1}^{t} (x_i - y_i)^2} \quad (2b)$$

No cálculo do coseno, o numerador representa um produto vetorial, correspondendo à soma dos produtos dos pesos associados aos termos nos vetores representando *x* e *y*. Cada um dos componentes do denominador representa uma função sobre um único vetor, e seu produto é um fator de normalização para a medida. Se dois documentos são exatamente iguais, então seus vetores serão superpostos no espaço vetorial, o ângulo entre eles será nulo e o valor do coseno será 1. Como o denominador utiliza sempre valores positivos, teremos que valores

obtidos com aplicação desta fórmula resultarão sempre no intervalo [0,1]; quanto mais próximo de 1 o valor obtido, maior a similaridade entre os documentos.

O cálculo da distância euclidiana mostra quão próximos ou distantes estão dois documentos representados no espaço de documentos. Se aplicadas a vetores normalizados, os resultados dessas duas medidas serão idênticos [Maning99].

Outras formas para identificação de similaridade freqüentemente encontradas na literatura [Salton83] [Kowalski97] [Manning99] [Meadow00] são os coeficientes de Dice e de Jaccard, utilizando a mesma representação espaço-vetorial para os documentos. As fórmulas desses coeficientes estão apresentadas respectivamente nas equações (3) e (4).

$$Dice(x, y) = \frac{2\sum_{i=1}^{t} x_i \, y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2} \quad (3)$$

$$Jaccard(x, y) = \frac{\sum_{i=1}^{t} x_i \, y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i \, y_i} \quad (4)$$

No coeficiente de Dice, o produto vetorial das representações de x e y é dividido pela média dos pesos $\left(\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2\right)/2$ associados aos termos nos dois conjuntos.

No coeficiente de Jaccard, o produto vetorial das representações de x e y é dividido pela soma dos pesos que correspondem aos termos que não são comuns aos dois documentos. Para documentos iguais, estes dois coeficientes (Dice e Jaccard) também retornarão o valor 1.

As três medidas de similaridade apresentadas consideram, no numerador, os termos co-ocorrentes, pois se um determinado termo não ocorre em um dos documentos, seu peso será zero e o produto resultará também em zero. Desta forma, apenas termos que ocorrem em ambos os documentos serão considerados para efeito de cálculo do numerador dos coeficientes de similaridade.

No contexto desta tese, estas três medidas de similaridade foram utilizadas na identificação de requisitos similares, apoiando a verificação da existência de duplicidade em requisitos.

2.5.4. Stemização

O processo de stemização (do inglês *stemming*) visa à obtenção do radical de uma palavra, seja ela uma forma verbal flexionada, um substantivo, um adjetivo ou de outra classe de palavra. O processo de stemização é utilizado quando se deseja agrupar palavras com diferentes grafias, ou mesmo diferentes categorias gramaticais, mas relacionados a um mesmo conceito. Exemplo: as palavras guardar e guarda possuem o mesmo radical (guard), embora pertençam a classes diferentes de palavras: enquanto guardar é um verbo no infinitivo, guarda pode ser um substantivo ou outra forma flexionada do verbo guardar. Em processos onde a semântica deve se sobrepor à morfologia, o processo de stemização possibilita esse agrupamento.

A raiz (ou *stem*) resultante do processo de stemização não é necessariamente idêntica ao radical lingüístico, mas *servirá como uma denotação mínima não ambígua do termo. Stemming consiste em reduzir todas as palavras ao mesmo stem, por meio da retirada dos sufixos, permanecendo apenas um radical* [Chaves03].

Na área de recuperação de informação milhares de documentos são analisados, seus termos são extraídos e contabilizados, gerando matrizes do tipo termo-documento (vide seção 2.4.3). Os termos extraídos são utilizados como indexadores desses documentos, muitas vezes gerando o problema de manipular uma enorme escala de índices e gerar matrizes muito esparsas. Utilizar os radicais, e não os próprios termos, é uma forma de diminuir o número de índices, agilizar o processo de recuperação de informações e otimizar o espaço ocupado pelos índices, além de agrupar termos que apresentam similaridade morfológica e proximidade conceitual.

Um dos mais conhecidos algoritmos para o processo de obtenção do radical de uma palavra é o algoritmo de Porter [Porter80]. Este algoritmo está estruturado em cinco etapas, e cada uma delas gera uma transformação no termo sendo avaliado. Este algoritmo foi originalmente proposto para a língua inglesa, e foi adaptado para a língua portuguesa pelo próprio autor. Na fase de experimentação utilizamos diferentes implementações deste algoritmo, entre elas a implementação descrita em [Caldas01] e uma outra disponibilizada pelo software PreTexT

[Matsubara03]. Apesar de, na grande maioria das vezes essas duas implementações retornarem o radical que possibilita o agrupamento de palavras semanticamente relacionadas, em vários casos importantes para o nosso trabalho isso não aconteceu. A Tabela 4 apresenta alguns dos resultados obtidos do processo de extração do radical de palavras com uso dessas implementações, considerando um mesmo termo no singular e plural, ou vários termos remetendo a um mesmo conceito.

Tabela 4 - Resultados obtidos na aplicação de dois diferentes stemmers

Termo	Radical esperado	Radical obtido com [Caldas01]	Radical obtido com [Matsubara03]
hotel	hotel	hotel	hotel
hotéis	hotel	hoté	hot
avião	avi	aviã	avia
aviões	avi	aviõ	avio
aviação	avi	aviaçã	AviaCa
confirma	confirm	confirm	confirm
confirmação	confirm	confirmaçã	confirmaCa

Nosso primeiro experimento utilizou um documento de requisitos para a área do turismo, e pela tabela pode-se observar que nenhuma das implementações teve sucesso no agrupamento de palavras acentuadas ou com caracteres como o cedilha. Partimos então para a busca de um outro *stemmer*, e localizamos um algoritmo criado especificamente para a língua portuguesa. Esse algoritmo foi proposto por Orengo e Huick e está descrito em [Orengo01]. Este último algoritmo é um pouco mais sofisticado que o de Porter, pois considera as classes morfológicas das palavras sendo avaliadas. Ao termo em análise são aplicados oito procedimentos seqüenciais, a saber: redução do plural, redução do feminino, redução de advérbio, redução do aumentativo e do diminutivo, redução de formas nominais, redução das terminações verbais, redução da vogal temática e finalmente remoção dos acentos.

Na redução do plural busca-se retirar o sufixo indicativo de plural, normalmente a letra **s**. Exemplo de exceção a esta regra são as palavras que terminam em **s** e não representam termos no plural (exemplo: cais) ou palavras que terminam em **ns** e cujo singular termina em **m** (exemplo: itens). Na redução para o feminino substantivos e adjetivos terminados em **a** e são modificados para corresponder ao gênero masculino da palavra, trocando o **a** final por **o**. Encontramos como exceção neste caso palavras como portuguesa/português,

chinesa/chinês. Redução de advérbio retira o sufixo **mente**, a não ser que a palavra seja uma das exceções registradas, como a palavra experimente.

Redução do aumentativo e diminutivo retira sufixos, como inho ou inha (casinha, carrinho), indicativos de diminutivo, e aqueles indicativos de aumentativo, como ão e íssimo (buracão e felicíssimo). A redução de formas nominais atua sobre substantivos e adjetivos, identificando aqueles cuja terminação está relacionada numa lista com 61 sufixos; se sufixos são removidos nesta etapa, as próximas duas não serão executadas. Na redução das terminações verbais busca-se obter o radical do verbo, já que a língua portuguesa utiliza mais de 50 diferentes formas verbais para os verbos regulares. Na remoção de vogal temática busca-se obter o radical de palavras não tratadas anteriormente, como por exemplo a palavra menino, que será transformada no radical **menin**. O último procedimento, remoção dos acentos, retira acentos que poderiam diferenciar palavras com um mesmo radical, como por exemplo psicólogo e psicologia.

Cada procedimento possui um conjunto de regras, e apenas uma regra em cada procedimento é aplicada ao termo sendo avaliado. O procedimento visa remover o sufixo mais longo possível; no caso de redução do plural da palavra aluguéis será tratado o sufixo **éis**, ao invés de apenas **s**. O algoritmo utiliza um total de 199 regras, estruturadas num mesmo padrão. Esse padrão estabelece o sufixo a ser removido, o tamanho mínimo para o radical após a remoção, um sufixo a ser adicionado em substituição àquele sendo retirado e uma lista de palavras que são consideradas exceções à regra. Um exemplo de regra pode ser visualizado a seguir [Orengo01]:

“inho”, 3, “ ”, { “caminho”, “golfinho”, “padrinho”, “sobrinho”, “vizinho” }

Essa regra estabelece que palavras terminadas em *inho* (normalmente um diminutivo) terão este sufixo retirado, desde que o radical restante tenha tamanho mínimo 3 e que a palavra não seja uma das exceções relacionadas: caminho, golfinho, padrinho, sobrinho e vizinho.

Em [Orengo01] são relatados diversos experimentos comparando os algoritmos de Porter e de Orengo e Huick. Os resultados obtidos com a aplicação dos dois algoritmos a um mesmo vocabulário são resumidos a seguir. Considerando um conjunto inicial de 32.000 palavras, o *stemmer* de Orengo e

Huick apresentou redução de vocabulário de 51%, contra 44% obtida com o algoritmo de Porter. Um extrato de 1.000 palavras desse conjunto inicial foi então utilizado para verificação da correção dos *stems* obtidos, e o algoritmo de Orengo&Huick apresentou resultado correto em 96% dos casos, contra 71% obtidos pelo algoritmo de Porter. Um terceiro experimento foi realizado, desta vez considerando como entrada conjuntos de palavras morfológica e semanticamente relacionadas. Um novo conjunto de 1.000 palavras foi dividido em 170 grupos de palavras relacionadas, sendo efetuadas medidas utilizando índices de *overstemming* e *understemming*. Com o algoritmo de Orengo e Huick foram obtidos valores de 0,034 e 0,000985 para *understemming* e *overstemming*, e com o algoritmo de Porter foram obtidos os valores de 0,215 e 0,000211 para essas mesmas medidas. Novamente foram obtidos resultados melhores para o algoritmo de Orengo e Huick.

Após alguns experimentos realizados sobre o documento de requisitos já referido, e considerando as particularidades do nosso trabalho, optamos pelo uso do algoritmo proposto por Orengo e Huyck no contexto desta tese.

2.5.5. Concordanceador

Um concordanceador é um programa que avalia um documento e recupera os contextos nos quais uma determinada palavra ou expressão de busca está presente. Na Figura 10 observa-se os contextos para a palavra **concordanceador** considerando-se os dois primeiros parágrafos desta seção e uma janela de 5 palavras para cada lado da palavra **concordanceador**. A palavra ou expressão de busca geralmente é centralizada, e a concordância apresenta um número determinado de palavras à esquerda e à direita da palavra de busca. Numa leitura vertical podem-se observar padrões gramaticais e lexicais; uma leitura horizontal permite observar colocações e diferentes sentidos.

Um	**concordanceador**	é um programa que avalia
os contextos para a palavra	**concordanceador**	considerando-se os dois primeiros parágrafos
No ensino de línguas o	**concordanceador**	é utilizado para que o

Figura 10 - Contextos para a palavra concordanceador

As concordâncias ou contextos são exibidos e podem ser manipulados para

uso com diferentes finalidades, por exemplo para identificar colocações (palavras que aparecem freqüentemente próximas num texto), para identificar qualificadores para um substantivo, para apoio à tradução automática de textos, para identificação de expressões idiomáticas. No ensino de línguas, o concordanceador é utilizado para que o aprendiz possa identificar em que contextos uma determinada palavra costuma ser utilizada.

A forma mais comum de uso de concordanceador é denominada de KWIC (Key Words In Context), e é utilizada para a geração de índices remissivos em livros e outros documentos.

Um dos mais famosos usos de concordâncias está relacionado aos Manuscritos do Mar Morto. Esses manuscritos, descobertos na década de 40, foram colocados sob os cuidados de um grupo internacional de pesquisadores, que por muito tempo os manteve sob sigilo; até o início da década de 90 apenas um terço desses manuscritos havia sido liberada para outros estudiosos ou mesmo para o público em geral. Em 1991 um estudante, Martin Abegg, reconstruiu quase que integralmente o texto dos manuscritos tendo por base aproximadamente cinqüenta mil cartões com extratos dos manuscritos e um código de referências das palavras e sua posição nos manuscritos (indicadores de contexto ou **kwic**). Confrontado por esse trabalho, o grupo de pesquisadores então liberou os microfilmes dos manuscritos, e uma edição fac-símile dos manuscritos originais foi posteriormente publicada (veja em http://www.pennandteller.com/sincity/penn-n-teller/pcc/deadsea.html ou em http://www.byaaronhoward.com/index.php?action=details&record=7).

Dado um tema (palavra ou expressão), o concordanceador é utilizado para que sejam extraídos os contextos (*concordances*) para avaliar a existência de termos que co-ocorrem com o tema. No contexto desta tese, o concordanceador foi utilizado como parte do processo para a obtenção de colocações de temas relevantes de documentos de requisitos, visando à criação de uma taxonomia para a classificação dos requisitos.

2.5.6. Análise de conteúdo

Nas ciências sociais e humanas, duas linhas de trabalho para análise de

textos são análise do discurso e análise de conteúdo. Análise do discurso é uma metodologia qualitativa, interpretativa e construcionista para análise de fenômenos sociais [Bardin77] [Hardy04], e incorpora um conjunto de técnicas para conduzir uma investigação qualitativa e estruturada de textos. O objetivo da análise do discurso é descobrir a maneira como a realidade social existente é produzida; ela envolve o estudo sistemático de textos para encontrar evidências de seu significado e de como este significado se traduz numa realidade social. Isto inclui identificar características da linguagem utilizada pelos atores, as categorias utilizadas na estruturação ou organização de seu universo e ainda as metáforas ou analogias utilizadas na descrição dessas categorias [Lowe04].

Análise de conteúdo diverge da análise do discurso no sentido de ser fortemente baseada em métodos quantitativos. Historicamente, a análise de conteúdo evoluiu da avaliação de textos tendo por base a simples análise de freqüência de determinados termos para métodos e técnicas mais sofisticados, baseados em conceitos e em relações semânticas entre eles. O texto é analisado não apenas pelos conceitos explicitados, mas também por aqueles implícitos de forma proposital ou não pelo autor do mesmo. Enquanto a identificação de termos explícitos é relativamente simples de realizar, a identificação de termos implícitos pode estar sujeita à interpretação do avaliador. Uma forma de diminuir a subjetividade de diferentes pontos de vista envolve a utilização de dicionários especializados.

A análise de conteúdo, na prática, envolve o desenvolvimento de categorias analíticas que permitem verificar em que medida tais categorias estão ou não presentes no texto ou conjunto de textos analisado. Análise de conteúdo pode ser caracterizada como sendo objetiva, sistemática e quantitativa. Ela é objetiva na definição precisa das categorias de forma que diferentes investigadores possam aplicá-las e obter resultados equivalentes; é sistemática na utilização de regras claras e bem definidas para a inclusão ou exclusão de categorias, e quantitativa no sentido de gerar resultados tratáveis por técnicas estatísticas [Hardy04]. Análise do discurso e análise de conteúdo de alguma forma avaliam também o contexto onde se insere o texto ou documento, mesmo que isso seja apenas presente na escolha dos termos utilizados para registro das categorias escolhidas. Métodos de análise de conteúdo são baseados em dicionários e classificados como modelos de variáveis latentes.

Em tais modelos, variáveis não observáveis diretamente (que chamaremos de *x*) originam efeitos observáveis (que chamaremos de *y*). O processo de inferência num modelo de variáveis latentes implica na avaliação da presença de *x* quando um particular *y* é observado. Podemos associar a variável *x* a um conceito que se deseja avaliar se está ou não presente no documento e *y* às características observáveis como termos e suas freqüências. Análise de conteúdo especifica o mapeamento de *x* para *y* através da construção de um dicionário de termos e expressões; o dicionário define como um determinado conceito (*x*) é expresso através de termos e expressões (*y*).

Análise do discurso e análise de conteúdo convergem na utilização de um conjunto de conceitos, chamados de categorias pela análise do discurso. Esse conjunto de conceitos é estruturado num dicionário, e a cada conceito é associado um conjunto de termos ou expressões a ele relacionados, em alguma medida. O processo de análise de conteúdo é baseado nesse conjunto de categorias e está estruturado em cinco etapas: preparação das informações, divisão do texto em unidades de análise, categorização das unidades, descrição e interpretação dos resultados [Moraes99]. A análise de conteúdo propriamente dita tem início apenas depois que os objetivos da análise estão claros, e as categorias (ou conceitos) estão definidas.

Preparação dos dados envolve definição da amostra a ser utilizada para o processo de análise, considerando os objetivos propostos; também significa aplicar as transformações necessárias que permitirão o trabalho com o texto. Unitarização implica em definir a unidade de análise, dividir o texto em unidades e identificar cada unidade - uma unidade pode ser uma palavra, uma expressão idiomática, um termo, uma frase ou um documento. Uma unidade deve ter um significado, deve ser completa em si mesma. A categorização pode utilizar critérios sintáticos ou semânticos, e, neste último caso, as categorias são denominadas de temáticas. A etapa de descrição, numa análise quantitativa, compreende a organização das categorias, freqüências e percentuais computados em tabelas e visualizados em gráficos ou outras formas de representação. A última etapa é a de interpretação de resultados, onde se buscam inferências que permitam generalizações a partir dos resultados obtidos na amostra ou documento analisado. A interpretação busca a compreensão sobre os conteúdos explícitos no texto, e também sobre aqueles não explicitados, latentes, não verbalizados pelos autores.

O processo de criação do conjunto de categorias a ser utilizado é central ao processo de análise de conteúdo, e pode tanto utilizar categorias extraídas do próprio texto ou derivar essas categorias de um referencial teórico relacionado à área de estudo. As categorias devem ser válidas em relação aos objetivos da análise, em relação à natureza do material avaliado e às questões que se busca responder por meio do trabalho de análise [Moraes99]. O conjunto de categorias deve ser exaustivo, no sentido de possibilitar que toda unidade de análise de conteúdo significativo seja classificada numa das categorias presentes no dicionário. Isto envolve escolha criteriosa das categorias e dos conjuntos de termos semanticamente relacionados a cada uma delas.

No contexto desta tese, a análise de conteúdo foi utilizada para a identificação de omissões em requisitos não funcionais, conforme apresentado no capítulo 3.

2.5.7. Agrupamento ou clusterização de documentos

Atividades de recuperação da informação e de mineração de dados estão muitas vezes associadas ao trabalho com um grande volume de documentos, ou corpora de informações textuais. Um dos desafios envolve agrupar documentos com características semelhantes, ou relacionados a uma mesma área. Nesses contextos, clusterização é definida como o processo de agrupar documentos similares tendo por base as palavras ou conceitos presentes nos documentos [Wives04]. A literatura pesquisada aponta muitas técnicas para o agrupamento de documentos [Kowalski97] [Manning99] [Meadow00] [Witten00].

Na área de recuperação de informação o agrupamento de documentos possibilita o armazenamento de documentos similares em áreas próximas da base de dados, agilizando o processo de recuperar todo um grupo quando uma consulta solicita documentos relacionados a um determinado tema. Para a descoberta do conhecimento na mineração de textos os grupos agilizam a identificação de associações entre palavras, facilitando o processo de criação de dicionários e de tesauros [Wives04].

Para o processo de clusterização não existe informação conhecida, a priori, sobre o número de grupos que resultará do processo de agrupamento, e os

métodos classificam documentos em grupos de acordo com um critério pré-determinado. Estes critérios são baseados em medidas de semelhança ou de diferença entre documentos, medidas estas obtidas a partir de representações estruturadas dos documentos. Uma representação freqüente é a matriz termo-documento, descrita na seção 2.5.2, e para o processo de agrupamento, utiliza-se uma seleção dos termos, que são denominados de atributos.

Na área de mineração de textos o uso de algoritmos de particionamento iterativo tem sido bastante freqüente [Wives04]. Dentre estes, um dos mais utilizados é o *k-means*, ou k-médias. Este algoritmo é um método iterativo para particionamento de conjuntos de dados onde o número *k* de grupos é indicado pelo usuário. Um conjunto inicial de *k* objetos é obtido pelo algoritmo de forma aleatória ou de acordo com alguma heurística; esses *k* objetos correspondem aos elementos centrais (ou centróides[3]) dos *k* agrupamentos. Em seguida é analisada a distância ou similaridade de cada documento aos centróides; utiliza-se como medida de similaridade a distância euclidiana. Cada documento é alocado ao agrupamento de centróide com a menor distância (ou maior similaridade) e o centróide desse agrupamento é então recalculado. O processo é repetido até que os centróides não mudem mais de posição.

Como o passo inicial deste algoritmo é aleatório, diferentes agrupamentos podem resultar de diferentes execuções do *k-means*, devido à escolha aleatória dos *k* centróides iniciais. É possível também que os agrupamentos resultantes da aplicação deste algoritmo não resultem em agrupamentos razoáveis [Witten00]. O problema também pode decorrer da escolha de um valor *k* não adequado ao conjunto de documentos que se deseja agrupar. O primeiro problema a resolver então está relacionado à definição do número *k* de agrupamentos que se deseja obter.

Uma boa estimativa para *k* pode ser obtida através de um outro algoritmo de particionamento iterativo, denominado de EM (Expectation-Maximization) [Manning99]. Este algoritmo, da mesma forma que o *k-means*, é iterativo e pode ser executado sem que seja informado o número de agrupamentos. Neste caso, o algoritmo divide os documentos em dois grupos e calcula um conjunto de valores que inclui média e desvio padrão para cada atributo, além do coeficiente

[3] centróide é a média do conjunto de características representativas de cada agrupamento

LogLikelihood[4], que é um coeficiente que mede a similaridade global. O número de grupos é incrementado e os demais procedimentos são repetidos até se obter um coeficiente ótimo para o coeficiente LogLikelihood. Nesse momento não ocorrem mais iterações, e o número ideal de grupos é aquele que otimizou a verosimilhança dos grupos encontrados.

No contexto desta tese técnicas de clusterização foram experimentadas, tendo como entrada documentos de requisitos. Utilizamos para os experimentos a implementação disponibilizada pela ferramenta WEKA [Witten00].

2.5.8. Recuperação de informações: medidas *recall* e *precision*

Na recuperação de informações as medidas recall e precision são utilizadas para avaliar a qualidade da estratégia utilizada. Conforme [Manning99], as medidas de *recall* e *precision* são definidas como:

$$Recall = \frac{tp}{tp + fn} \quad (5) \qquad Precision = \frac{tp}{tp + fp} \quad (6)$$

A utilização do diagrama apresentado na Figura 11 facilita a compreensão do significado dessas medidas. Nas fórmulas (5) e (6), *fp* indica a quantidade de falsos positivos, *fn* indica a quantidade de falsos negativos, *tp* indica a quantidade de positivos corretos (*true positives*) e *tn* indica a quantidade de negativos corretos (*true negatives*). Portanto, *recall* é a proporção dos objetos corretos retornados em relação ao total de objetos buscados (alvo), e *precision* é a proporção de itens corretos no conjunto de objetos recuperados. A Tabela 5 complementa essa definição.

Tabela 5 - Medidas utilizadas para cálculo de recall e precision

Objetos	*alvo (buscados)*	*não alvo*
selecionados	tp	fp
não selecionados	fn	tn

[4] o coeficiente Likelihood é também denominado de coeficiente de verosimilhança

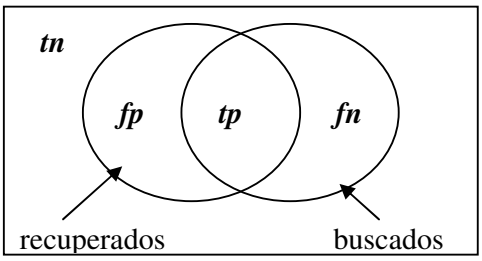


Figura 11 - Diagrama ilustrativo das medidas *precision* e *recall* [Manning99]

2.6. Processo de Requisitos, PLN e Agentes

Este capítulo condensou os principais conceitos do Processo de Requisitos (seções 2.1, 2.2 e 2.3), visando apresentar ao leitor os fundamentos necessários para uma boa compreensão do trabalho desenvolvido nesta tese. Também estão registrados os métodos e técnicas do Processamento da Linguagem Natural (seção 2.5) utilizados para atingir os objetivos propostos, dado que trabalhamos com requisitos expressos em linguagem natural. Como a ferramenta de suporte à abordagem proposta utiliza agentes de software, apresentamos também os conceitos básicos da área (seção 2.4), possibilitando ao leitor acesso à base conceitual necessária ao entendimento deste trabalho.

3
Proposta para Análise de Requisitos utilizando PLN

A proposta aqui apresentada está fundamentada no uso de técnicas de processamento da linguagem natural para apoio às atividades associadas à verificação e validação de requisitos expressos em linguagem natural. Na medida do possível, abstraimos aspectos da implementação de cada estratégia proposta.

Os principais processos desenvolvidos são apresentados brevemente a seguir, e detalhados no decorrer deste capítulo.

Geração de visões dos requisitos: desenvolvemos uma estratégia para extração semi-automática de uma taxonomia a partir do próprio documento de requisitos. A taxonomia obtida é utilizada para agrupar requisitos associados a determinadas características do sistema. Os agrupamentos também podem ser definidos a partir de solicitações dos interessados que, na representação de um certo grupo de usuários do futuro sistema, têm interesse em avaliar o conjunto de requisitos associados a uma propriedade ou característica específica do sistema.

Uma vez obtidos os agrupamentos, visões textuais ou gráficas são oferecidas aos participantes dos processos de verificação e validação (V&V). A visão gráfica possibilita a identificação rápida das associações e dependências entre requisitos; a explicitação de tais dependências é fundamental para atividades de evolução do sistema, principalmente na análise de impacto que uma determinada alteração deverá provocar.

Construção ou atualização do léxico da aplicação: nossa proposta envolve a identificação de termos relevantes do domínio da aplicação, através da análise de documentos relacionados ao sistema em desenvolvimento. A estratégia proposta busca: a) termos próprios do domínio da aplicação, através da identificação de termos não incluídos em dicionários, e b) a identificação de atores e de objetos manipulados no sistema, através de uma estratégia baseada em sintagmas nominais.

Esta estratégia também é útil na reengenharia de sistemas, possibilitando a obtenção de informações a partir de manuais de usuário, de documentos originais

de requisitos ou de solicitações de alteração do sistema.

Detecção de discrepâncias, erros e omissões em requisitos: utilizamos duas abordagens, uma delas visando à identificação de duplicidades em requisitos, e a outra visando à indicação de requisitos não funcionais ausentes do documento de requisitos. A primeira abordagem está baseada em medidas de similaridade entre documentos, e a segunda utiliza a metodologia de análise de conteúdo.

A seguir cada um desses processos é apresentado em detalhe; a arquitetura de suporte a esta proposta incorpora os processos detalhados neste capítulo, e vai além, utilizando agentes pessoais e apoiando o trabalho dos participantes nas atividades de V&V. A arquitetura está detalhada no próximo capítulo.

3.1. Geração de visões dos requisitos

Documentos de requisitos muitas vezes são escritos em linguagem natural. Ali estão registrados características e propriedades importantes do sistema a ser desenvolvido, bem como necessidades e expectativas dos usuários.

O processo de análise de requisitos [Leite94] inclui atividades de verificação e validação. Muitas técnicas foram propostas e estão em uso corrente na indústria para atividades de V&V em documentos de requisitos [Fagan86] [Leite91] [Davis93] [Porter95] [Wilson97] [Shull00]. No entanto, aspectos tais como competitividade e novos paradigmas para o processo de desenvolvimento de software têm gerado maior pressão pela diminuição dos prazos para estas atividades. Pesquisadores tanto na academia quanto na indústria têm buscado técnicas e ferramentas que possibilitem agilizar o processo de desenvolvimento, sem perda da qualidade do produto gerado.

Para sistemas com grande número de requisitos, as atividades de V&V são difíceis de realizar devido ao volume de informações a verificar ou validar. Verificação da incompletude, por exemplo, é uma tarefa árdua se o conjunto de requisitos a avaliar é da ordem de centenas ou mesmo milhares de requisitos. Se conseguirmos trabalhar com grupos menores, a complexidade destas tarefas tende a diminuir. Requisitos não podem ser agrupados ao acaso: precisamos de uma estratégia que nos auxilie a encontrar conjuntos de requisitos que realmente contenham características comuns.

Agrupamentos de requisitos com características similares, ou visões que possam ser associadas a características ou propriedades do sistema tornam-se necessários [Palmer92] [Gruenbacher01]. Diferentes visões do documento de requisitos, geradas a partir de critérios previamente definidos, possibilitam que o participante escolha um conjunto de requisitos para analisar. No contexto deste trabalho, denominamos visão dos requisitos a um conjunto de requisitos com características em comum, apresentado de forma textual ou gráfica. A visão é obtida pela aplicação de uma estratégia que combina técnicas de processamento da linguagem natural e uso de taxonomias.

A necessidade de visões de requisitos é maior em ambientes distribuídos de desenvolvimento, onde os participantes estão geograficamente distantes e sujeitos às dificuldades de comunicação, dentre outras. Diferentes visões de requisitos podem ser atribuídas a diferentes participantes, para os processos de verificação e validação. Acreditamos que a identificação de características relevantes da aplicação, associadas a visões textuais ou gráficas dos requisitos relacionados, possa auxiliar também o trabalho do gerenciamento de requisitos, principalmente em ambientes distribuídos de desenvolvimento. Dificuldades inerentes a este ambiente podem tornar mais trabalhosas as atividades relacionadas ao processo de verificação e validação de requisitos.

A identificação de grupos de requisitos relacionados pode também ser utilizada nas fases posteriores do processo de desenvolvimento, como por exemplo para apoio às tarefas de alocação de requisitos a componentes, ou na definição de casos de testes de aceitação do sistema pelos usuários. Agrupamentos de requisitos relacionados também podem ser utilizados para apoiar a alocação de requisitos a componentes ou mesmo a incrementos, no caso de desenvolvimento incremental.

A construção de visões é fundamentada na identificação de grupos de requisitos relacionados entre si ou a uma determinada característica do sistema. O agrupamento está fundamentado no princípio de que agrupamentos de requisitos relacionados a uma determinada característica do sistema facilitam a identificação de erros relacionados a conflitos, inconsistências, incompletude e ambigüidade em documentos de requisitos, conforme proposto por Palmer e Liang [Palmer92]. Desta forma, a determinação automática de agrupamentos de requisitos associados a características comuns apóia a verificação e a validação, pois permite focalizar

em conjuntos menores e possivelmente menos complexos de requisitos.

3.1.1. Categorização de requisitos e geração de visões

A indústria costuma utilizar documentos de requisitos escritos em linguagem natural, e um dos modelos mais comuns é o que utiliza sentenças numeradas e algumas vezes classificadas segundo uma taxonomia geral (por exemplo: requisitos funcionais, requisitos inversos e requisitos não-funcionais [Leite01]). Outras representações, como casos de uso, estórias do usuário e cenários também utilizam linguagem natural. Nosso desafio inicial está relacionado tanto à definição do número de agrupamentos a identificar, quanto à escolha do processo a adotar para reunir requisitos relacionados. O processo definido deve ainda ser independente dos modelos utilizados para o registro dos requisitos.

Para o número de agrupamentos a utilizar, seguimos um preceito da área de classificação de documentos. É comum que documentos sejam referidos através de palavras, expressões ou frases que caracterizam o tema ou o conteúdo principal de um documento [Baeza-Yates99]. De maneira geral, espera-se que poucos temas sejam suficientes para referir assuntos relevantes num documento, independente do fato da escolha de temas ou conceitos significativos ter sido feita de forma manual ou de forma semi-automática através de ferramentas de manipulação da linguagem natural. Para o nosso trabalho, definimos inicialmente que o número de grupos seria um valor próximo de cinco, conforme discutido na seção 3.1.2.

Um primeiro experimento para gerar esses agrupamentos considerou a identificação de termos significativos no documento de requisitos, através de medidas estatísticas para identificar termos relevantes. Foram utilizadas medidas como *tf* (*term frequency*), *tfidf* (*term frequency inverse document frequency*) e *relevância* [Gonzales05], com resultados similares. Para gerar os agrupamentos, identificamos requisitos relacionados utilizando um procedimento determinístico: para cada um dos termos obtidos pela aplicação das medidas referidas obtinha-se seu *stem* ou radical e varria-se o documento de requisitos procurando pela ocorrência desse radical. Requisitos contendo o radical eram agrupados e esse

grupo identificado pelo termo utilizado.

Os agrupamentos obtidos desta forma mostraram-se consistentes com relação à característica identificada pelo termo, porém um aspecto não atendido por esta abordagem envolveu expressões ou termos relacionados a requisitos não funcionais, como por exemplo termos relativos à segurança e à confiabilidade. Expressões ou termos relacionados a estes requisitos tendem a ter pouca presença no texto, sendo portanto preteridos no momento da escolha dos termos a utilizar. De maneira geral podemos dizer que requisitos não funcionais não foram adequadamente atendidos por esta abordagem, que se mostrou inadequada para os objetivos que buscávamos atingir.

Uma outra tentativa realizada envolveu a utilização de algoritmos de clusterização para a identificação dos grupos de requisitos. Neste experimento foi utilizado o pacote *Weka* disponibilizado pela Universidade de Waikato [Witten00]. Trabalhamos com o *k-means*, um dos mais utilizados algoritmos de particionamento iterativo. Este algoritmo utiliza, como medida de similaridade entre documentos, a distância Euclidiana, que mede a similaridade entre documentos considerando os atributos em comum [Wives04].

Utilizamos o *k-means* aplicando-o a um documento de requisitos de uma aplicação da área do turismo. Este documento passou por um processo de preparação que incluiu a separação dos requisitos em arquivos do tipo texto puro e a geração de vetores termo-documento. Estes vetores registram, para cada documento, a freqüência dos *t* termos definidos para avaliação. Os termos costumam ser aqueles mais relevantes, ou aqueles de maior freqüência.

Para a execução do algoritmo *k-means*, o número de agrupamentos é um dos argumentos fornecidos pelo usuário. Apesar de já termos uma definição inicial para este número, resolvemos pesquisar uma alternativa que pudesse confirmar ou não nossa escolha. Utilizamos então o algoritmo EM (Expectation-Maximization) [Manning99].

Este algoritmo, EM, inicialmente agrupa os documentos em dois grupos e calcula um conjunto de valores que inclui média e desvio padrão dos termos utilizados para representação desses documentos. A implementação utilizada também calcula o coeficiente *LogLikelihood*, que mede a similaridade global. Na segunda iteração o número de grupos é incrementado e os demais procedimentos são repetidos até obter um coeficiente ótimo para os indicadores utilizados. Nesse

momento a execução é encerrada e o número ideal de grupos é fornecido.

No nosso caso, a execução do algoritmo EM para o documento já referido indicou cinco grupos, o que corroborou nossa definição inicial para o número de grupos. Utilizamos como atributos todos os substantivos com freqüência por documento maior ou igual a quatro [Daile96], e uma etapa de pré-processamento gerou um conjunto de vetores que, para cada requisito individual, trazia a freqüência dos termos selecionados. Trabalhamos com substantivos pois eles podem expressar funções gramaticais como sujeito e objeto, funções semânticas como agente e instrumento ou funções retóricas como tópico ou tema [Thrane80].

Definido o número ideal de grupos, continuamos o experimento executando o *k-means* para então obter os agrupamentos de requisitos. O documento utilizado, já referido, era oriundo de uma aplicação na área do turismo e possuía cento e sessenta e nove requisitos. Dos cinco grupos resultantes do processo de clusterização, um apresentava apenas dois requisitos, enquanto outro apresentava cento e cinco. A este último grupo, particularmente, não se conseguiu identificar claramente um tema que perpassasse todos os requisitos do grupo.

Realizamos duas outras tentativas com o *k-means*, modificando, a cada vez, os atributos considerados: ora utilizamos como atributos aqueles termos próximos da média de freqüência no conjunto de documentos [Manning99] [Luhn58], ora limitando o conjunto de atributos aos mais freqüentes. Não obtivemos resultados significativamente melhores que o anterior.

Uma das premissas que nos guiou nesta investigação considera que requisitos podem estar associados a mais de um grupo, ou seja, os agrupamentos a serem obtidos não serão necessariamente disjuntos. Um exemplo claro desta característica resultou da análise dos agrupamentos do documento de requisitos da área do turismo; este documento continha requisitos que deveriam ser manipulados por usuários de diferentes áreas da organização, no processo de V&V. Alguns requisitos deveriam ser avaliados por pessoal da área de atendimento ao cliente, por tratarem de aspectos da venda de um pacote turístico, mas também interessavam ao pessoal da área financeira, por tratarem de questões relacionadas à forma de pagamento utilizada pelo cliente.

Optamos então por investigar alternativas com o uso de taxonomias. Taxonomias têm sido utilizadas para classificação de documentos. Bibliotecas constituem um excelente exemplo de uso de taxonomias para classificação de

diferentes tipos de documentos. Na área de requisitos, encontramos um exemplo do uso de taxonomias na classificação de requisitos organizacionais que impactam no projeto de produtos [Gershenson99].

Nossa estratégia está baseada no uso de uma taxonomia estruturada em dois níveis. Um conjunto mínimo de temas relevantes da aplicação compõe o nível mais alto da taxonomia; esta taxonomia é a base para identificação automática, no documento de requisitos, de termos associados que serão utilizados para a categorização dos requisitos. Estes termos irão compor o segundo nível da taxonomia, e sua seleção combina a descoberta de associações (através da análise do contexto no qual constam os temas do primeiro nível da taxonomia) e a sua avaliação através de medidas estatísticas como o escore T e a informação mútua. Associações são descobertas através de colocações, que são grupos de palavras próximas mas não necessariamente contíguas, que co-ocorrem no texto numa freqüência maior que o esperado pelo acaso [Manning99] [Sardinha04].

A estratégia proposta é esquematizada utilizando uma perspectiva de processo (um actigrama SADT [Ross77]) conforme mostra a Figura 12. Esse processo é composto por 3 atividades: a) identificação de termos relevantes para enriquecer a taxonomia; b) categorização dos requisitos com base na taxonomia enriquecida pelos termos e c) geração das visões dos requisitos.

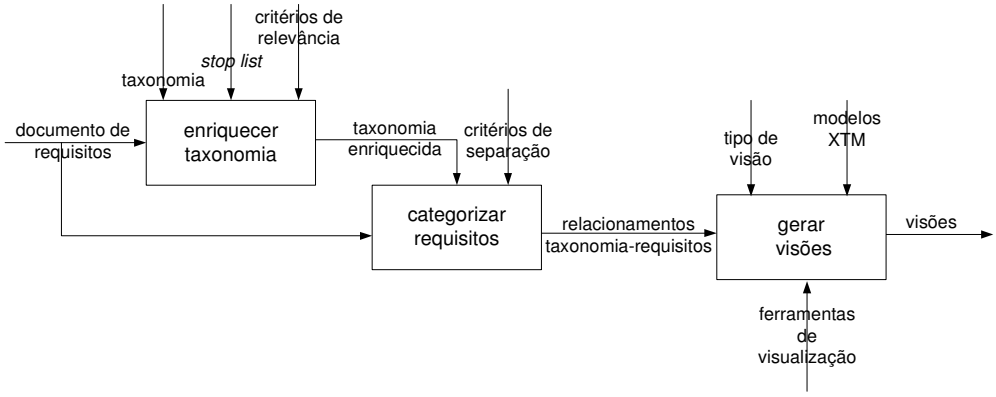


Figura 12 - Visão geral do processo para categorização dos requisitos e geração de visões

A taxonomia enriquecida é utilizada para categorização dos requisitos, gerando agrupamentos de requisitos relacionados. Estes conjuntos de requisitos associados à taxonomia possibilitam diferentes visões dos requisitos aos participantes do processo. Essas visões serão utilizadas nos processos de análise

de requisitos que se quer apoiar. A seguir descreveremos cada uma das etapas do processo proposto.

3.1.2. Identificação de termos para enriquecimento da taxonomia

Como apresentado na Figura 12, o processo pode ser dividido em 3 grandes atividades, ou sub-processos. Primeiro, usando tecnologias de tratamento de linguagem natural, procuramos identificar termos para compor nossa taxonomia. Em seguida procede-se a categorização dos requisitos de acordo com a taxonomia enriquecida e finalmente utilizamos os agrupamentos para compor as visões e apresentá-las tanto textualmente como graficamente.

Os temas que irão compor o primeiro nível da taxonomia definem a quantidade de agrupamentos para os requisitos. Uma referência forte para esta definição refere-se ao número inicial de características para guiar o processo de elicitação proposto pelo método PreView [Sommerville98]. Nesse método, características representam necessidades de alto nível para a aplicação e influenciam os requisitos derivados de cada ponto de vista. Fundamentado no uso de pontos de vista e de características para elicitação de requisitos, o PreView estipula que o número de características deve ser baixo, da ordem de cinco.

Outra referência utilizada consta no trabalho de Hoskinson [Hoskinson05], que trata da extração de conceitos de textos para posterior geração de taxonomias. Hoskinson estabelece que o número de conceitos para os nodos de mais alto nível da taxonomia deve estar no intervalo de três a oito. Adotamos este último critério, pois ele é mais geral que a nossa definição inicial e abarca o proposto pelo método PreView. Acreditamos, porém, que o engenheiro de requisitos deva usar o valor aqui apresentado apenas como guia, avaliando a cobertura dos requisitos em relação aos temas selecionados e trabalhando o número de temas de mais alto nível da taxonomia de forma a atingir cobertura completa do conjunto de requisitos, ou de sua maior parte.

Para a identificação dos termos do segundo nível, consideramos que a frase (ou contexto) que apresenta um tema da taxonomia inicial deve conter outros termos que são relacionados a ele e, portanto, também são relevantes. O processo de busca de contexto procura pelo tema da taxonomia inicial, e retira para análise

uma sub-sentença que é composta pelo tema considerado e pelas palavras imediatamente à sua esquerda e à sua direita. Para cada um dos temas da taxonomia inicial vamos então obter um conjunto de sub-sentenças que será analisado para a extração de colocações, ou de termos para o segundo nível da taxonomia. Este processo pode ser visualizado na Figura 13.

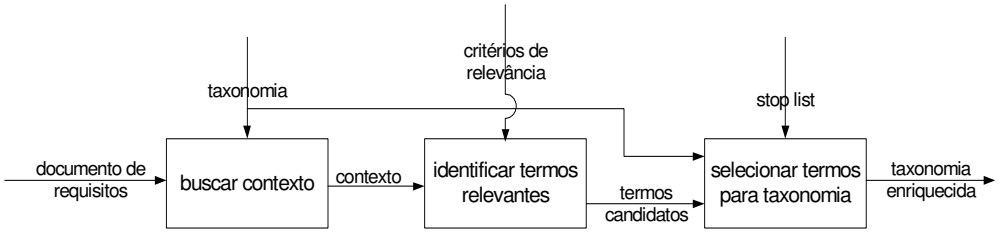


Figura 13- Processo para enriquecer a taxonomia

A busca de contexto trabalha com cada um dos termos da taxonomia inicial, varrendo o texto do documento de requisitos e buscando identificar a presença do termo. Quando isto acontece, é extraída uma sub-sentença composta do termo, das cinco palavras à sua esquerda e das cinco palavras à sua direita. Esta quantidade pode ser menor se, por exemplo, for encontrado o final do parágrafo antes que esse número seja atingido. Procedimento similar é executado em relação ao início do parágrafo. A Figura 14 mostra um exemplo de sub-sentença contendo o termo **reserva**.

a	proposta	e	confirma	a	**reserva**	porém	ainda	não	efetua	nenhum

Figura 14 - Sub-sentença apresentando contexto para o termo "reserva"

Na identificação das sub-sentenças é utilizado o radical da palavra de busca, então para o termo reserva será utilizado o radical **reserv**. As sub-sentenças identificadas compõem um novo documento. Esse documento será analisado buscando a identificação de colocações, que serão avaliadas de acordo com medidas de associação para identificar se a colocação é relevante ou se sua presença no texto é devida ao acaso. Utilizamos como medidas de associação o escore T e a Informação Mútua. A forma de cálculo para o escore T é apresentada na equação (1), e a fórmula para a informação mútua é apresentada na equação (2).

$$T(x,y) = \frac{freq(x,y) - \frac{freq(x) * freq(y)}{N}}{\sqrt{freq(x,y)}} \quad (1)$$

$$im(x,y) = Log_2\left(\frac{P(x,y)}{P(x) * P(y)}\right) \quad (2)$$

Nestas medidas, *x* e *y* representam os termos em análise. Em (1), *freq(x,y)* corresponde à freqüência conjunta de *x* e *y*, *freq(x)* indica a freqüência do termo *x*, *freq(y)* a freqüência do termo *y* e *N* indica quantidade de termos. Em (2), *P(x,y)* indica a probabilidade da ocorrência conjunta de *x* e *y*, *P(x)* e *P(y)* indicam respectivamente a probabilidade de ocorrência do termo *x* ou do termo *y* no documento, e N corresponde ao total de termos do documento.

A informação mútua compara a probabilidade da ocorrência conjunta de x e y com a probabilidade de observar x e y independentemente - ela é dada pela razão entre o valor observado e o esperado. Se houver uma associação genuína entre x e y, a probabilidade conjunta P(x,y) será muito maior do que P(x)*P(y), e conseqüentemente *im(x,y)*≈0. Se não houver nenhum relacionamento interessante entre x e y, então P(x, y)≈P(x)*P(y), e assim, *im(x;y)*≈0. A probabilidade conjunta, P(x,y), é estimada normalizando-se o valor de *freq(x,y)*, ou seja, dividindo *freq(x,y)* por N; as probabilidades P(x) e P(y) são calculadas também normalizando-se as freqüências.

O escore T considera freqüências não normalizadas, e portanto é um indicador absoluto das colocações - indica pares que são por si só muito freqüentes. Valores de informação mútua tendem a ser maiores quando as palavras analisadas aparecem sempre próximas no texto, ou seja, a presença de uma delas é forte indicador da presença da outra (as palavras tem pouca presença independente). Valores maiores do escore T são indicativos de maior número de co-ocorrência das palavras analisadas.

Para que as colocações sejam consideradas significativas é necessário observar o patamar mínimo para cada uma delas. Os valores de corte aceitos e normalmente utilizados para estas medidas de associação são respectivamente três para a informação mútua e dois para o escore T [Sardinha04]. Assim, todas as colocações cujas medidas de associação sejam iguais ou maiores que os patamares definidos são consideradas relevantes para enriquecimento da taxonomia. Na identificação das colocações também trabalhamos com o radical das palavras.

Os termos obtidos neste processo podem ser compostos de até três palavras (trigramas). Esses termos são denominados *termos candidatos*, e são analisados num processo de seleção que desconsidera termos que iniciem ou terminem por palavra relacionada na lista de *stopwords*. Finalmente, as *stopwords* eventualmente presentes nos termos selecionados são também retiradas, gerando

um padrão que permitirá que expressões como *reserva de vôo*, *reserva do vôo*, *reservar o vôo* ou *reservar os vôos* sejam consideradas similares para fins do processo de categorização e correspondam a um mesmo padrão.

A participação do engenheiro de requisitos no processo de definição dos termos a serem incluídos na *stoplist* é muito importante, dado que é provável que termos técnicos da área da TI e mesmo do domínio da aplicação também devam ser desconsiderados no processo de seleção de termos candidatos.

3.1.3. Categorizar requisitos

A identificação dos requisitos relacionados aos termos da taxonomia é centrada no processo de categorização, que utiliza a taxonomia enriquecida. O processo utilizado para categorização é determinístico e trabalha com o reconhecimento de padrões. Consultas por padrões possibilitam a identificação de documentos que atendem a propriedades pré-especificadas; expressões regulares, padrões estendidos e radicais estão entre os tipos de padrão mais utilizados na recuperação de informações [Baeza-Yates99]. A Figura 15 esquematiza o processo de categorização.

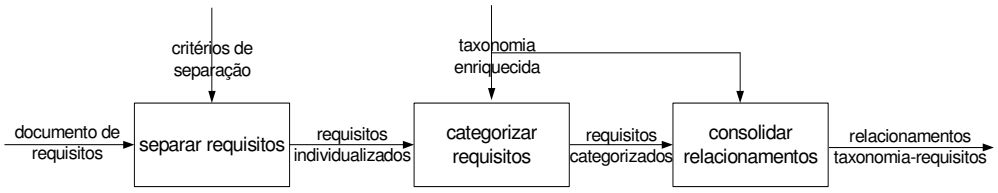


Figura 15 - Processo para categorização de requisitos

Separar requisitos consiste em individualizar requisitos que estão reunidos num único documento. Se não existir uma delimitação clara do conjunto de frases que descrevem um requisito, a própria identificação do requisito pode ser utilizada para esta finalidade. Esta etapa na verdade é uma preparação ao processo de categorização, e pode ser dispensada se os requisitos já estiverem individualizados.

Para categorizar requisitos utilizamos a taxonomia enriquecida, e a base para este processo utiliza os termos associados identificados pelas colocações relevantes (veja seção 2.5.5). Para cada um destes termos associados é aplicada a técnica de reconhecimento de padrões ao texto dos requisitos; é considerado como padrão o radical das palavras que compõem o termo. Se o requisito analisado

atende o padrão, então sua identificação é incluída na lista de relacionamentos para o termo sendo avaliado.

O uso do radical das palavras para compor o padrão propicia a recuperação de termos no singular ou no plural, no masculino ou no feminino, de palavras de diferentes categorias gramaticais e também de expressões similares. Observamos que como o *stemmer* utilizado é dirigido para a língua portuguesa, termos em outras línguas serão mantidos como apresentados, ou seja, na íntegra. Isto deve ser observado pois é comum, em documentos de requisitos, encontramos termos da área de tecnologia da informação que são utilizados na língua inglesa, mesmo que exista uma tradução para a língua portuguesa. Este é o caso de termos como *password*, *logon*, *logoff* e *home page.*

Ao final desta atividade cada termo estará associado a uma lista de requisitos relacionados, e a última etapa consolida as listas correspondentes a cada um dos temas de alto nível da taxonomia, pois um mesmo requisito poderá atender a diversos padrões. A consolidação irá gerar um conjunto de requisitos associados a cada tema de alto nível da taxonomia.

Pelo processo utilizado, esse conjunto reflete aqueles requisitos cuja associação foi determinada pelo fato do texto do requisito conter ao menos um dos padrões considerados relevantes para aquele tema da taxonomia. Como um mesmo requisito poderá estar associado a mais de um ramo da taxonomia, o número de associações será maior que o número de requisitos do sistema. Estas associações, se representadas graficamente, resultarão num grafo onde as interdependências entre requisitos estarão explicitadas. Essa representação é descrita a seguir.

3.1.4. Gerar visões

Diferentes apresentações das visões dos requisitos devem ser propiciadas, para que cada um dos interessados possa escolher aquela mais apropriada às suas necessidades. As visões podem ser apresentadas textualmente ou estruturadas num grafo, possibilitando uma visão mais abrangente de conjuntos de requisitos relacionados. A Figura 16 apresenta o processo para geração das visões.

A atividade de expandir as listas de relacionamentos gera conjuntos <termo-

da-taxonomia> <requisitos relacionados e atributos>. Os atributos a serem utilizados são selecionados dentre aqueles que estão sendo registrados no projeto em desenvolvimento, por exemplo origem do requisito, data de inserção e *rationale*. A descrição dos requisitos é essencial para as visões a serem apresentadas, e portanto ao menos este atributo é utilizado.

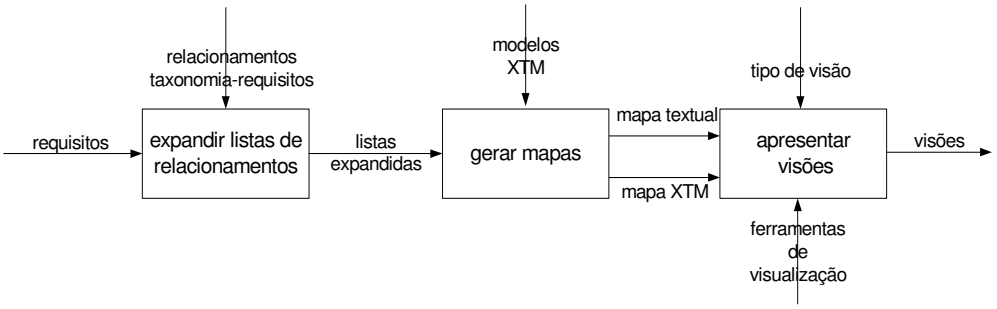


Figura 16 - Processo para geração das visões dos requisitos

As listas expandidas serão utilizadas na geração de mapas que são a base para as visões textuais e gráficas dos requisitos. O mapa textual é estruturado na forma de uma tabela, onde a cada tema da taxonomia são registrados os requisitos associados e seus atributos. O mapa XTM (XML *Topic Maps*) é necessário para a apresentação das visões gráficas e é gerado com utilização dos modelos apresentados logo a seguir.

Visões textuais são apresentadas através de um visualizador para textos, e serão utilizadas em atividades tais como identificação de duplicidades entre requisitos e verificação de completude. Visões gráficas foram construídas porque possibilitam percepção visual das interdependências entre requisitos. Se esta visão permitir ainda a navegação entre os nodos, será possível explorar as associações entre requisitos e entre requisitos e taxonomia.

O modelo que escolhemos para apresentação das visões gráficas está baseado na utilização de *topic maps*, ou mapas de tópicos. Mapas de tópicos podem ser caracterizados como um padrão para descrever estruturas de conhecimento e associá-las a recursos de informação [Pepper00]. As entidades básicas presentes em tais mapas são tópicos de informação, associações e ocorrências. Um tópico pode representar uma entidade ou um objeto; uma associação relaciona dois tópicos, de acordo com um critério, e uma ocorrência pode representar um atributo de tópicos ou de associações.

No contexto deste trabalho, itens da taxonomia e requisitos foram caracterizados como tópicos, e associações foram utilizadas para registrar as ligações de cada termo da taxonomia aos requisitos relacionados. Denominamos relacionado_a à associação entre termos da taxonomia e requisitos associados. No contexto de requisitos, associações também podem ser utilizadas para registro da rastreabilidade e consultadas para análise de impacto em eventos de mudanças: se o item representa, por exemplo, um requisito não funcional, os requisitos funcionais a ele associados também devem ser avaliados quando ocorre uma alteração na política ou procedimentos correlatos ao requisito não-funcional.

Mapas de tópicos podem ser representados com uso do XML, e a linguagem XTM (XML *Topic Maps*) foi criada para facilitar a troca de *Topic Maps* entre aplicações [Pepper01]. Estruturas previamente definidas podem ser utilizadas para definição dos tópicos e suas associações, agilizando a criação dos mapas e a geração das visões. O modelo (*template*) utilizado para os itens da taxonomia pode ser visualizado na Figura 17, o modelo para requisitos é apresentado na Figura 18 e o modelo para associações na Figura 19 (os modelos aqui apresentados estão simplificados).

```
<topic>
  <instanceOf> <topicRef xlink:href="#termos"/> </instanceOf>
  <baseName>
         <baseNameString> termo da taxonomia </baseNameString>
  </baseName>
</topic>
```

Figura 17 - Modelo XTM para termos da taxonomia

```
<topic>
  <instanceOf> <topicRef xlink:href="#req_func"/> </instanceOf>
  <baseName>
         <baseNameString> id_do_requisito </baseNameString>
  </baseName>
</topic>
```

Figura 18 - Modelo XTM para requisitos funcionais

```
<association>
  <instanceOf> <topicRef xlink:href="#relaciona"/> </instanceOf>
  <member>
         <topicRef xlink:href="#id_requisito"/> </member>
  <member>
         <topicRef xlink:href="#id_termo"/>  </member>
</association>
```

Figura 19 - Modelo XTM para associações entre temas e requisitos

A escolha de uso de uma ferramenta de mapas de tópicos para a visualização das associações entre requisitos e a taxonomia deveu-se principalmente às características dinâmicas oferecidas por estas ferramentas. A visualização dos agrupamentos, por si só, já é uma característica importante para a

compreensão das interdependências entre requisitos [Baniassad04] [Gruenbacher01], mas o uso destas ferramentas possibilita também a navegação entre requisitos relacionados. Esta última característica foi apontada como necessária [Dag01], pois possibilita uma forma visual e ágil para a percepção das interdependências entre requisitos.

Exemplo de um mapa de tópicos identificando um conjunto de requisitos associado ao tema pagamento pode ser visualizado na Figura 20.

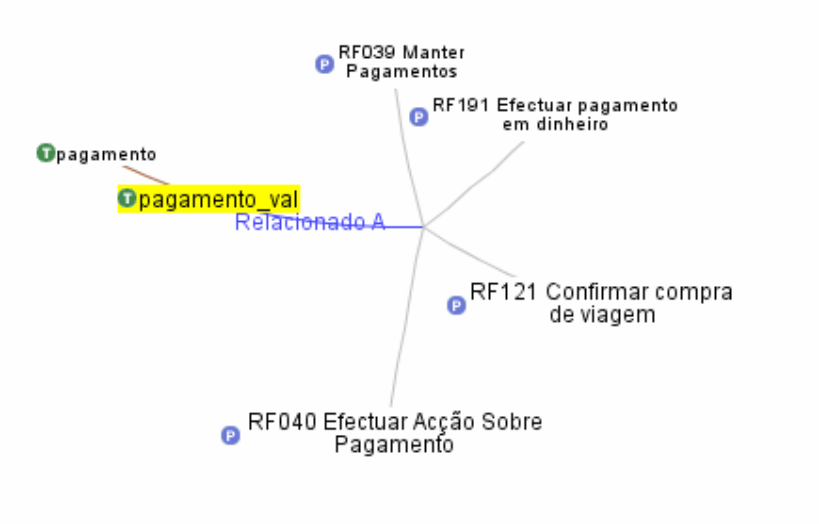


Figura 20 - Mapa visual mostrando requisitos relacionados ao tema **pagamento**

3.2. Construção do léxico da aplicação

Atividades de apoio à construção ou atualização do Universo de Informações são baseadas na identificação de palavras ou expressões próprias do domínio da aplicação. Nosso desafio está relacionado à identificação de símbolos que irão compor o léxico da aplicação e contribuir para a construção do vocabulário do domínio da organização.

Processos de desenvolvimento de software, como por exemplo o RUP (Rational Unified Process) definem glossários como um dos artefatos a serem gerados no processo de requisitos. Um glossário pode ser entendido como uma forma simplificada de léxico, estruturado de forma linear e contendo termos e suas definições. Os termos a serem inseridos num glossário são aqueles utilizados pelos participantes do processo para fazer referências à características da aplicação, visando facilitar o entendimento entre eles. O glossário deve ser

construído durante a elaboração do modelo de negócios ou modelo de domínio. Já o Léxico Ampliado da Linguagem, ou LAL [Leite93], é uma forma mais elaborada de registro de termos próprios do domínio da aplicação, fornecendo mais informações que simplesmente a definição de um termo. Detalharemos o LAL, segundo proposta apresentada em [Leite93].

Termos registrados no LAL são tipificados, e esta é a primeira diferença em relação a um glossário de termos do domínio da aplicação. Termos inseridos no LAL representam símbolos característicos do domínio da aplicação, e correspondem a um de quatro tipos: sujeito, objeto, verbo ou estado. Símbolos do LAL possuem noção e denotação. A noção de um símbolo é o que o define, e a d2enotação registra os impactos que o símbolo provoca ou recebe, no domínio considerado. A Tabela 6 registra os quatro tipos de símbolos e impactos correspondentes.

Tabela 6 - Símbolos do LAL, noção e impactos [Leite90]

Tipo do símbolo	Noção	Impacto
Sujeito	quem é o sujeito	ações que executa
Verbo	quem realiza, quando acontece e quais os procedimentos	quais os reflexos das ações no ambiente e novos estados decorrentes
Objeto	definir o objeto e identificar outros objetos com os quais ele se relaciona	ações que podem ser aplicadas ao objeto
Estado	o que indica e ações que levaram a esse estado	identificar outros estados que podem ocorrer a partir do estado que se descreve

Sujeitos correspondem a entidades ativas, atores com papel relevante para a aplicação; um sujeito pode ser um ator, um componente ou um outro sistema com o qual deverá ocorrer interação. Verbos registram ações ou funcionalidades a serem desempenhadas pelos sujeitos ou pelo sistema em desenvolvimento, com algum impacto ou reflexo no ambiente operacional. Objetos são entidades passivas utilizadas ou necessárias a uma ação ou conjunto de ações, e estados são caracterizados por atributos significativos que registram valores em diferentes momentos da execução do sistema. A Tabela 7 apresenta exemplo de entrada para o léxico de um sistema para bibliotecas de uma universidade.

Tabela 7 – Exemplo de símbolo de um léxico do tipo LAL

Léxico Ampliado da Linguagem - Sistema de Bibliotecas
Usuário tipo do símbolo: sujeito noção: pessoa que pode utilizar a biblioteca; pode ser um aluno, professor ou funcionário da universidade impactos: usuário é cadastrado no sistema usuário é retirado do cadastro de usuários usuário retira obras da biblioteca usuário devolve obras anteriormente retiradas usuário renova datas para devolução de obras anteriormente retiradas

Optamos pela utilização do léxico Ampliado da Linguagem [Leite90] para representação do léxico das aplicações, e definimos uma estratégia para a avaliação de documentos da organização e extração de termos para compor o léxico. Nossa proposta utiliza duas diferentes abordagens para a construção do léxico da organização: a primeira está baseada na identificação de termos ou expressões não-dicionarizados, e a segunda utiliza extração de sintagmas nominais.

3.2.1. Termos ou expressões não incluídos em dicionários

O processo para extração de termos não-dicionarizados baseia-se na premissa que termos próprios do domínio da aplicação não deverão estar presentes em dicionários usuais da língua. Exemplos de termos incluem acrônimos para outros sistemas, palavras técnicas de tecnologia de informação em língua inglesa e até mesmo diferentes denominações para representar um mesmo objeto. Este último caso é especialmente interessante quando os participantes do processo de requisitos compartilham uma mesma língua, mas diferenças culturais devidas às diferentes localizações geográficas provocam o uso de vários termos para um mesmo objeto ou sujeito (por exemplo, celular e telemóvel).

Termos que atendem a uma dessas características têm forte indicação para serem registrados no léxico da aplicação, evitando dificuldades na leitura de documentos originadas por desconhecimento de termos ou por ambigüidades. O processo para extração de termos não dicionarizados está representado na Figura 21.

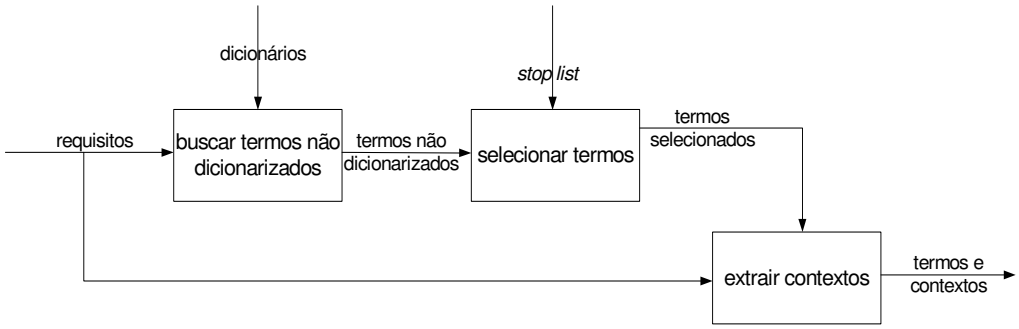


Figura 21 - Processo para extração de termos não-dicionarizados

3.2.1.1. Buscar termos não incluídos em dicionários

Nesta etapa o texto é *tokenizado* e os *tokens* são utilizados como expressão de busca num conjunto de dicionários para a língua portuguesa. Estes dicionários não são dicionários no sentido estrito do termo, pois não contêm definições ou sinônimos para os termos. Eles podem ser pensados como uma lista de termos, ordenados em ordem alfabética; cada dicionário é restrito a uma única classe gramatical de termos. O conjunto inclui dicionários específicos para categorias como substantivos, adjetivos, verbos, abreviaturas, pronomes, artigos, conjunções e interjeições.

Os dicionários utilizados foram criados e cedidos por Akeo Tanabe, pesquisador associado ao LES/DI-PUC-Rio, e estão descritos em [Tanabe06]. Esses dicionários abrangem aproximadamente dezoito mil adjetivos, mais de cinco mil substantivos e trezentas abreviaturas. Termos não relacionados nos dicionários são extraídos dos requisitos, pois são candidatos a comporem o léxico da aplicação.

3.2.1.2.Selecionar termos

Os termos não dicionarizados são agora avaliados em relação a uma *stop list*, que deverá conter os termos do léxico, se este já existir, e quaisquer outros termos considerados irrelevantes pelos participantes do processo.

O processo de seleção considera todos os termos, independente da freqüência com que eles possam estar presentes no texto. Isto significa que qualquer termo explicitado no documento e que não conste dos dicionários ou

stoplist utilizados não é um termo corrente da língua. Para evitar que esses termos possam dar margem a diferentes interpretações nos processos de V&V, eles deverão ser relacionados no léxico da aplicação.

Apenas termos não dicionarizados e que não constem da *stoplist* específica desta etapa deverão ser processados pela próxima etapa, que trata da extração de contextos.

3.2.1.3. Buscar noção e impactos

A busca de contexto trabalha com cada um dos símbolos selecionados, varrendo os requisitos e buscando identificar a presença do termo. Este processo utiliza um concordanceador ligeiramente modificado para extrair não apenas uma sub-sentença que contenha o termo, mas extrai o parágrafo completo.

O parágrafo extraído pode corresponder à definição ou a impactos desse termo no contexto da aplicação. Os parágrafos extraídos serão agrupados junto ao símbolo candidato, e utilizados posteriormente pelo engenheiro de requisitos para inserção no léxico da aplicação.

3.2.2. Identificação de sujeitos e objetos

Sintagmas nominais são definidos como (a) classe gramatical com comportamento sintático de sujeito, de objeto direto e, se precedido de preposição, de adjunto adnominal ou de objeto indireto [Perini98] [Vieira01]; (b) enunciado que representa um conceito ou uma entidade (abstrata ou concreta) identificada por nomes próprios ou sintagmas nominais descritivos; pode ainda representar um papel [Liberato97].

No contexto da engenharia de software, Booch et al [Booch00] afirmam que um ator representa um papel que um ser humano, um dispositivo de hardware ou mesmo um outro sistema desempenha [Booch00]: ele é o sujeito da ação. Em documentos técnicos, atores são registrados através de identificadores que incluem nomes próprios, profissões e papéis. Em termos lingüísticos podemos associar atores e sintagmas nominais. Recursos correspondem a objetos que ocupam um espaço no mundo real ou virtual e são utilizados ou gerados por

ações. Recursos e objetos podem ser identificados por substantivos e, portanto também podem ser lingüisticamente identificados por sintagmas nominais.

O processo geral de extração de símbolos proposto é baseado na extração de sintagmas nominais e é apresentado na Figura 22.

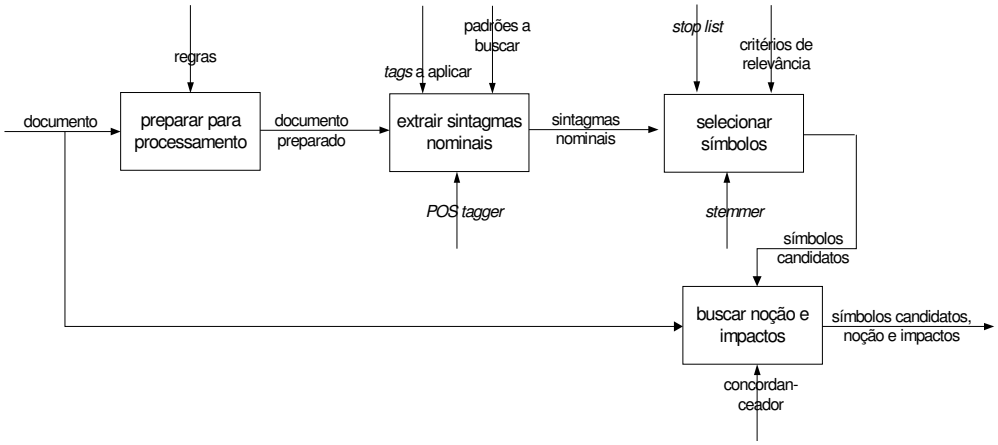


Figura 22 - Processo geral para extração de símbolos

O processo de extração de símbolos está estruturado em quatro atividades ou sub-processos. Inicialmente o documento é preparado para posteriormente ser manipulado por um *Part of Speech tagger* (*POS tagger*). Após a inserção das *tags* (etiquetas) são extraídos sintagmas nominais que correspondem a padrões pré-definidos; estes sintagmas nominais passarão por um processo de seleção, que irá considerar apenas aqueles que atendem a critérios de relevância estabelecidos. A etapa final extrai, para cada um dos sintagmas, contextos que possam vir a ser utilizados como noção e impactos. A seguir detalharemos cada um desses sub-processos.

3.2.2.1. Preparar para processamento

O trabalho de preparação dos documentos é necessário, pois as ferramentas utilizadas trabalham com arquivos do tipo texto puro. Os documentos da organização podem estar em diferentes formatos, como por exemplo, *pdf* (Portable Document Format), *doc* (Microsoft Word document) ou *html* (HyperText Markup Language). Documentos podem ainda incluir figuras e tabelas, que não serão processadas pelas ferramentas de tratamento de texto. O processo de preparação inicialmente retira figuras, transforma tabelas em texto,

retira possíveis *tags* de formatação e gera texto puro, ou seja, arquivos do tipo *txt*.

Após a obtenção do arquivo em formato *txt*, faz-se necessário a *tokenização* do documento, ou seja, a colocação de um *token* por linha (inclusive dos caracteres de pontuação). A separação dos *tokens* é uma exigência do etiquetador utilizado e poderia ser dispensada caso a ferramenta utilizada não exigisse este formato. O documento de requisitos, agora já no formato texto puro, é *tokenizado* e pode então ser trabalhado para a extração de sintagmas nominais.

3.2.2.2. Extrair sintagmas nominais

O processo de extração de sintagmas nominais é baseado na utilização do etiquetador morfossintático QTAG, que analisa o texto pré-processado e associa uma etiqueta a cada um dos *tokens*. Após a etiquetagem são extraídos e contabilizados os sintagmas nominais que atendem a padrões pré-estabelecidos, e que corresponderão a atores e objetos registrados no documento avaliado.

Sintagmas nominais possuem uma estrutura bem definida. Perini [Perini98] coloca que sintagmas nominais possuem duas estruturas básicas: a estrutura à esquerda do núcleo do sintagma é composta por posições que podem ser ocupadas por determinantes, possessivos, quantificadores e outras classes de palavras. A estrutura à direita do núcleo é composta por modificadores, que por sua vez podem ser classes abertas ou mesmo outros sintagmas nominais. Neste trabalho não utilizamos a estrutura à esquerda, pois a linguagem de documentos de requisitos é objetiva, não deve utilizar figuras de linguagem e neste contexto o que nos interessa é o núcleo do sintagma nominal.

Para a extração dos sintagmas utilizamos alguns padrões pré-definidos, pois nem todo sintagma nominal presente no texto é interessante, no contexto deste trabalho.

Extração de Atores ou sujeitos: padrões utilizados

Na língua portuguesa funções ou papéis desempenhados por pessoas ou entidades são identificados por substantivos com terminações específicas. Alguns exemplos com terminação *ente*: gerente, presidente; terminação *or*: gestor, diretor, trabalhador; terminação *ário*: usuário, funcionário. Na extração de sintagmas

nominais correspondendo a atores/sujeitos, utilizamos um conjunto de 82 padrões, pois para cada terminação consideramos o singular, o plural, feminino e masculino, na prática multiplicando cada terminação por quatro. Exemplos de padrões utilizados são apresentados na Tabela 8; os anexos B e C trazem a relação completa.

Tabela 8 - Padrões para extração de sujeitos e atores

N*ente PRP* N* / N*entes PRP* N* / N*enta PRP* N* / N*entas PRP* N*
N*or PRP* N* / N*ores PRP* N* /N*ora PRP* N* /N*oras PRP* N*
N*eiro PRP* N*/ N*eiros PRP* N*/ N*eira PRP* N*/ N*eiras PRP* N*
N*ente N*/N*entes N* /N*enta N*/N*entas N*
N*or N*/N*ores N*/N*ora N*/N*oras N*

Nos padrões relacionados acima o símbolo * indica qualquer quantidade de caracteres. Como exemplo, o padrão *N*ente PRP N** deverá extrair sintagmas compostos por um nome seguido por preposição e outro nome, sendo que o primeiro nome deverá ter a terminação *ente*. Os padrões foram definidos tendo por base um dicionário de terminações da língua portuguêsa e revisados após avaliação de textos etiquetados nos nossos experimentos iniciais.

Extração de Recursos ou Objetos: padrões utilizados

A extração de recursos/objetos utilizou padrões mais genéricos que os utilizados na etapa de extração de atores/sujeitos. Foram utilizados nove padrões, relacionados na Tabela 9.

Tabela 9 - Padrões para extração de objetos e recursos

N* PRP* N* PRP* N*	N* PRP* N*	N* N*
N* PRP* N* N*	N* PRN* N*	N* PART*
N* CPR* N* N*	N* CPR* N*	N* ADJ*

Os padrões utilizados foram definidos empiricamente após avaliação de diferentes tipos de documentos gerados ou manipulados no processo de requisitos. As principais fontes para este trabalho foram documentos de requisitos de diferentes domínios de aplicação e manuais de usuário. Buscamos identificar padrões gerais, mas certamente o conjunto utilizado deve ser revisto e modificado para atender especificidades de uma determinada organização de domínio de aplicação.

3.2.2.3. Selecionar símbolos

O processo de seleção dos símbolos a partir dos sintagmas candidatos desconsidera sintagmas que contenham termos relacionados na *stoplis*t, que é composta por termos da língua geral ou mesmo do domínio, mas que não são relevantes para o léxico. É importante observar que a *stoplist*, neste contexto, não é formada por artigos, pronomes, preposições, e outros normalmente incluídos em listas desse tipo.

A etapa seguinte realiza a *stemização* dos sintagmas extraídos. Isto é necessário pois, nos nossos experimentos, identificamos a necessidade de agrupar singular e plural, feminino e masculino. Por exemplo, em nossos estudos de caso, foram separadamente extraídos e contabilizados os sintagmas nominais *gestor de escala* e *gestor de escalas*, o mesmo ocorrendo com os sintagmas *cessionária* e *cessionário*.

Os sintagmas candidatos são então agrupados e contabilizados após o processo de *stemização*, e adotamos o masculino singular para a representação, quando fosse o caso. A última atividade do processo de seleção descarta os sintagmas cuja freqüência seja menor que quatro, conforme procedimento usual na área de extração de terminologia [Daile96].

3.2.2.4. Buscar noção e impactos

A busca de contexto trabalha com cada um dos símbolos selecionados, varrendo o documento buscando identificar a presença do símbolo, e para esta tarefa foi utilizado um concordanceador modificado. Quando a ferramenta identifica a presença de um símbolo, é extraído o parágrafo completo, pois este pode corresponder à definição ou a impactos do símbolo.

Na identificação das sentenças é utilizado o radical (*stem*) do símbolo de busca; os parágrafos extraídos serão agrupados junto ao símbolo candidato, e utilizados posteriormente pelo engenheiro de requisitos para inserção no léxico da aplicação. Nos estudos de caso observamos que a identificação da noção (ou definição) dos símbolos é sensível ao contexto da aplicação ou mesmo a padrões da organização.

3.3. Detecção de discrepâncias, erros e omissões em requisitos

O apoio à detecção de discrepâncias, erros e omissões em requisitos é apoiado em parte pelo agrupamento de requisitos, que possibilita aos participantes do processo de verificação ou validação trabalhar em conjuntos menores de requisitos. Isto agiliza o processo: por exemplo, para identificar incompletude em requisitos relacionados a um determinado tema manipula-se apenas o conjunto de requisitos associado a esse tema, e não todo o conjunto de requisitos do sistema.

Outra formas de apoio que estamos propondo estão relacionadas à automação da identificação de duplicidade em requisitos e à identificação de omissões em RNF´s.

Em ambientes distribuídos de desenvolvimento, usuários, clientes e engenheiros de requisitos estão dispersos geograficamente. Se o processo de elicitação ocorre em diferentes locais, usuários distantes geograficamente podem passar a diferentes engenheiros de requisitos informações que irão gerar requisitos similares. Isto só será percebido quando os requisitos forem colocados num único documento, mas esta detecção poderá ser dificultada se houver um grande número de requisitos. Se a verificação de duplicidade é automatizada, a descoberta de requisitos candidatos à duplicidade pode ser feita rapidamente.

Alguns modelos de registro de requisitos, como casos de uso, são mais voltados à especificação de funcionalidades do sistema. Organizações que usem um modelo de especificação de requisitos baseadas em casos de uso costumam registrar requisitos não funcionais numa especificação de requisitos suplementares. Mesmo com diversos catálogos de requisitos não funcionais disponíveis para uso, a prática mostra que o registro dos requisitos não funcionais não é tarefa adequadamente atendida em muitas organizações de desenvolvimento de software.

Nossa proposta para identificação de duplicidade em requisitos e de identificação de omissões em requisitos não funcionais está descrita a seguir.

3.3.1.
Duplicidade em requisitos

A identificação de duplicidade nos requisitos é aplicada nos agrupamentos de requisitos, e está baseada em medidas de similaridade. Entre medidas de similaridade aplicáveis a documentos, optamos por utilizar os coeficientes do coseno, Jaccard e Dice para identificar requisitos candidatos à verificação de duplicidade. A abordagem utilizada neste processo para representação dos documentos é a conhecida por *bag-of-words*, e a estrutura do processo para identificação de requisitos similares é apresentada na Figura 23.

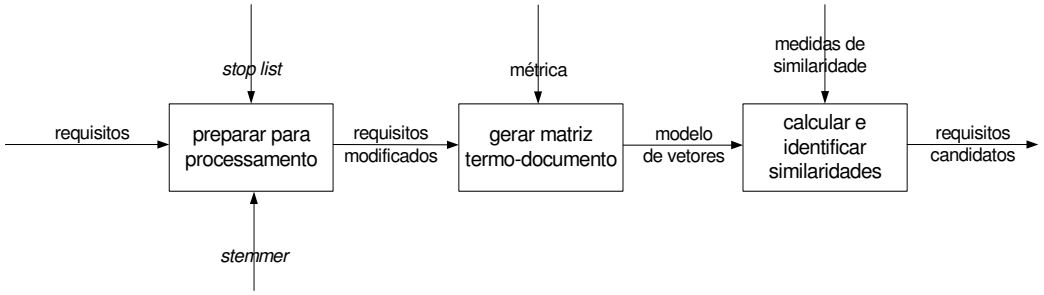


Figura 23 - Processo para identificação de requisitos similares

Para o cálculo das similaridades é utilizada uma matriz termo-documento: as linhas representam os requisitos da aplicação e as colunas os termos utilizados na definição desses requisitos. As células de tais matrizes contém uma métrica relacionada à freqüência do termo no documento, e para o nosso caso utilizamos a freqüência pura. Tais matrizes podem apresentar problemas de alta dimensionalidade (quantidade de termos) e conter dados esparsos (células vazias ou com valor zero). Para tratar esse problema, utilizamos uma etapa de pré-processamento descrita a seguir.

3.3.1.1.
Preparar para processamento

O pré-processamento dos documentos está baseado na utilização de uma *stoplist* e de um *stemmer* para obtenção dos radicais (*stems*) dos termos. A *stop list* utilizada inclui um conjunto de composto por artigos, preposições, conjunções, advérbios. Termos que façam parte da *stoplist* serão retirados do

texto dos requisitos. O uso do stemmer propicia o agrupamento de termos com proximidade conceitual.

A aplicação deste processo diminui o número de termos únicos nos documentos de requisitos, contribuindo para redução da dimensão da matriz termo-documento e melhor desempenho dos algoritmos para cálculo da similaridade. Os termos que compõem a *stop list*, conforme já visto no capítulo 2, costumam estar presentes em grande parte dos documentos e possuem baixo poder discriminatório em relação aos demais.

3.3.1.2. Gerar matriz termo-documento

Para a geração da matriz cada um dos requisitos é analisado, os *stems* são contabilizados e também colocados num vetor comum a todos os requisitos. Este processo é repetido para todos os requisitos, e ao final obteremos uma matriz onde cada requisito estará representado por um vetor de termos, e o peso associado a cada termo será obtido pela freqüência do termo no requisito correspondente. Cada um dos *n* requisitos estará representado por um vetor de tamanho *t* (t correspondendo ao número total de termos únicos).

Como o objetivo nesta etapa é identificar similaridades entre requisitos, são desconsiderados apenas os termos relacionados na *stoplist*, não importando a freqüência com que eles possam estar presentes no conjunto de requisitos.

3.3.1.3. Calcular e identificar similaridades

O cálculo dos coeficientes de similaridade considera pares de requisitos: cada requisito é avaliado contra cada um dos demais requisitos do grupo. Utilizamos os coeficientes de Dice, do coseno e de Jaccard, conforme discutido no capítulo 2. Os coeficientes calculados retornam sempre valores entre 0 e 1; quanto mais próximo de 1 o valor, mais similar é o par de requisitos. Quanto mais próximo de 0, menos similar é o par de requisitos.

No nosso trabalho, ao estruturarmos os requisitos na forma de matrizes termo-documento, utilizamos uma *stoplist* composta apenas por artigos, pronomes, advérbios, conjunções e outras palavras sem valor relevante. Não

foram incluídos termos técnicos ou do domínio da aplicação.

Nosso cálculo correlaciona os valores obtidos para a obtenção de um índice de similaridade único, considerando-se a média aritmética desses coeficientes. Os requisitos candidatos à análise de duplicidade são então apontados, considerando o limiar de 0,90, ou seja, todos os pares cujo índice de similaridade seja igual ou maior que 0,90 serão avaliados para efeitos de identificação de duplicidade. Este limiar pode ser ajustado para atender a particularidades de uma dada aplicação.

Para a definição do limiar foram executados diversos experimentos, considerando documentos de requisitos de diferentes domínios de aplicação. O limiar definido foi aquele que apresentou, considerando o conjunto de experimentos, maiores taxas de acerto na identificação de requisitos em duplicidade e menor taxa de falsos positivos.

3.3.2. Omissões em requisitos

Uma das tarefas mais difíceis em processos de verificação de requisitos é a identificação de omissões. Como identificar aquilo que não está expresso nos requisitos? É muito difícil viabilizar esta verificação de forma automática para requisitos funcionais, expressos em linguagem natural, dado que cada aplicação possui características e necessidades que a diferenciam de qualquer outra. Mesmo sem termos realizado uma avaliação sistemática e qualitativa, podemos inferir que os conjuntos de requisitos funcionais em documentos de requisitos de diferentes domínios diferem de forma significativa. Não há um conjunto de referência que possa ser utilizado para a verificação de omissões em requisitos funcionais.

Por outro lado, sistemas de informação costumam utilizar um conjunto reduzido de requisitos não funcionais. Acreditamos que a identificação da presença ou ausência de requisitos não funcionais seja viável, desde que o processo utilizado consiga resolver ou contornar dificuldades inerentes ao uso da linguagem natural no documento de requisitos.

Um dos problemas a ser resolvido está diretamente relacionado ao grau de detalhamento aplicado no registro dos requisitos. Para sistemas com muitas funcionalidades a serem atendidas, ou cujos clientes e usuários estão geograficamente separados, é possível supor o cenário onde um grupo de

engenheiros de requisitos atua, buscando identificar junto a clientes e usuários as características e funcionalidades a serem atendidas pelo sistema a ser desenvolvido. As diferentes capacidades lingüísticas dos engenheiros de requisitos podem levar ao registro de requisitos em diferentes estilos, e então teremos que lidar com a avaliação de requisitos expressos em diferentes graus de abstração ou mesmo utilizando diferentes termos para um mesmo conceito.

Outro problema a ser abordado está na própria classificação geral dos requisitos: como requisitos não funcionais serão implementados através de funcionalidades no sistema, alguns engenheiros de requisitos os classificam (incorretamente) como funcionais. Nos documentos de requisitos que analisamos, foi freqüente o caso de requisitos não funcionais de controle de acesso serem classificados como requisitos funcionais. Avaliar então a ausência de requisitos não funcionais implica na avaliação de todo o conjunto de requisitos, dado que eles podem estar classificados incorretamente.

Como avaliar de forma objetiva o conjunto de requisitos, resultando em informações claras sobre a presença ou ausência de requisitos não funcionais no documento avaliado? Não existem variáveis concretas a serem medidas, nem mesmo um vocabulário padrão para uso no registro dos requisitos. A solução encontrada utiliza a metodologia da análise de conteúdo- para avaliar um documento de requisitos e identificar possíveis omissões, considerando requisitos não funcionais.

Como foi visto no capítulo 2, métodos de análise de conteúdo são baseados na utilização de um dicionário de categorias; estas categorias representam idéias ou conceitos que se deseja investigar se estão ou não presentes no texto.

No contexto de requisitos, dicionários manipulados para a análise de conteúdo podem ser construídos a partir dos catálogos públicos de requisitos não funcionais. Cada entrada no dicionário é composta por um requisito não funcional; a cada entrada são associados os termos e expressões usados no contexto da organização para o registro daquele requisito não funcional.

O processo completo para a identificação de omissões em RNF´s no documento de requisitos é apresentado na Figura 24 e inclui a etapa de criação do dicionário, a instanciação para uma dada organização e a execução da análise de conteúdo do documento de requisitos, gerando um relatório sobre omissões relacionadas a RNF´s.

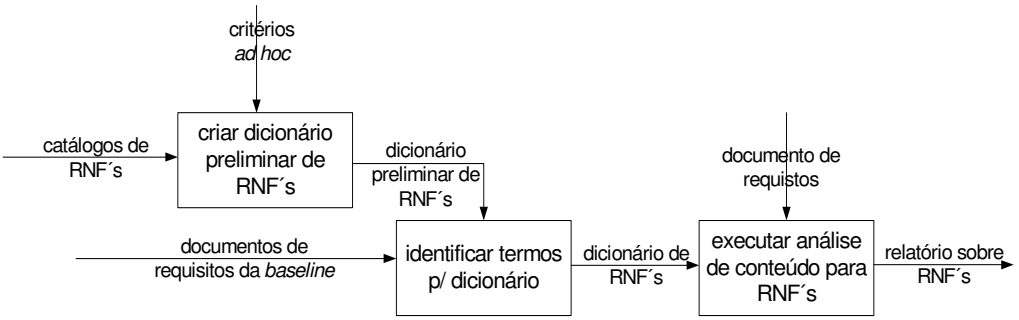


Figura 24 - Processo para análise de conteúdo para RNF´s

3.3.2.1. Criar dicionário preliminar de RNF´s

O dicionário de RNF´s a ser utilizado para análise do documento de requisitos gerado nesta etapa não pretende ser um dicionário completo; o objetivo na sua construção é registrar, para uma dada organização, os RNF´s necessários à gama de aplicações visualizada para a organização. Tendo portanto como entradas catálogos públicos de RNF´s, como por exemplo aqueles extraídos de [Sommerville04] [IEEE98] [SWEBOK04], uma avaliação conduzida por especialistas da organização deverá selecionar os RNF´s que deverão constar no dicionário. Neste processo deverão ser considerados aspectos relacionados a padrões de qualidade da organização e aos ambientes de execução dos sistemas, entre outros.

É importante ter em mente a utilização deste dicionário: ele deverá ser utilizado em atividades de verificação e validação, e não em atividades de elicitação. Essa diferença é enfatizada, pois catálogos de RNF´s para uso na etapa de elicitação ou modelagem de requisitos devem ser abrangentes, possibilitando a quem os utiliza um mapeamento amplo dos RNF´s possíveis para uma vasta gama de aplicações. A Figura 25 apresenta o dicionário inicial, fortemente baseado em [Sommerville04] [IEEE98] [SWEBOK04].

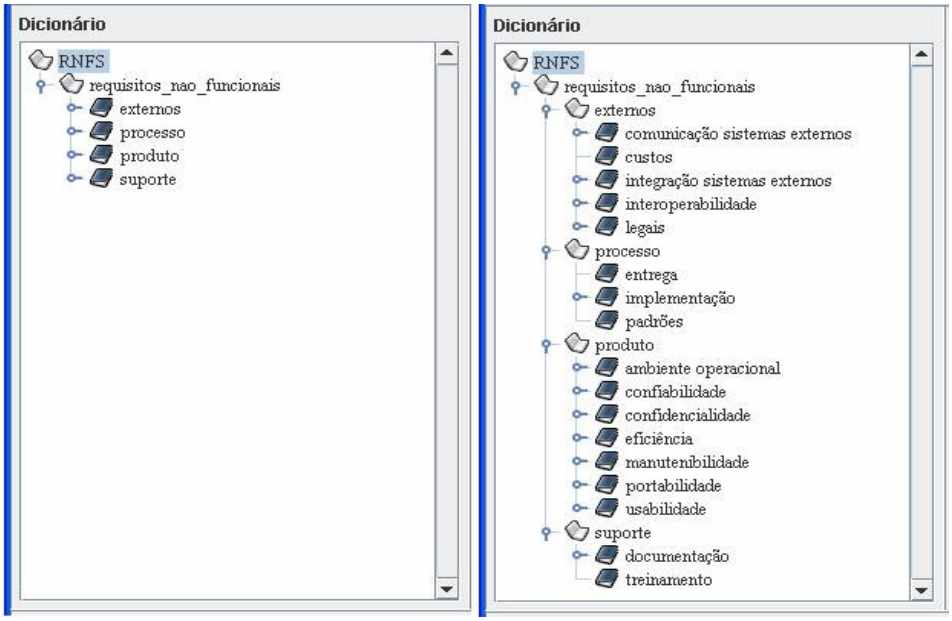


Figura 25 - Estrutura do dicionário de RNF´s para identificação de omissões

Na Figura 25, à esquerda, é exibido o primeiro nível de categorias do dicionário criado, e à direita o detalhamento de cada uma dessas categorias iniciais. Os requisitos não funcionais que serão avaliados estão agrupados nas categorias externos, processo, produto e suporte. Cada uma dessas categorias está sub-dividida em várias outras, para permitir uma boa estruturação do dicionário. A categoria externos agrega o conjunto de termos que está relacionada a requisitos externos ao sistema, como por exemplo requisitos legais, de comunicação com sistemas já existentes, de custos. A categoria processo agrega o conjunto de termos relacionados ao processo de desenvolvimento do software, como restrições de implementação, padrões a serem utilizados nas atividades e artefatos do processo de desenvolvimento, requisitos de entrega do software. A categoria produto agrega termos que estão associados a propriedades ou características que o sistema deve apresentar, como portabilidade, usabilidade, confiabilidade, confidencialidade e outros. E a categoria de suporte abriga termos relacionados à documentação que deverá acompanhar o software e ao treinamento necessário para o grupo de usuários.

A saída desta atividade é um dicionário preliminar de RNF´s; as entradas do dicionário conterão apenas o nome e uma breve descrição de cada RNF.

3.3.2.2. Identificar termos para o dicionário de RNF´s

Uma vez obtido o dicionário preliminar de RNF´s, deve-se agora instanciá-lo para uma dada organização. Isso significa inserir, para cada um dos RNF´s do dicionário, entradas que registrem termos ou expressões já utilizados em documentos de requisitos da *baseline* da organização. Este parte do processo é feita de forma semi-automatizada: os participantes analisam documentos da baseline da organização e identificam termos que são usuais no registro de requisitos não funcionais. Este processo deve ser suportado pelo léxico da organização. O ideal é que os participantes desta atividade sejam profissionais já com boa experiência em processos de requisitos, com uma visão ampla da área de tecnologia da informação da empresa, e com bom desempenho em atividades do processo de requisitos.

Os termos e expressões selecionados serão inseridos nas categorias definidas na etapa anterior. O suporte de software para esta atividade deve ter flexibilidade para permitir o registro de palavras simples e de expressões. Para possibilitar que uma única entrada no dicionário possa representar um conjunto de termos, é desejável que se possa também utilizar o caracter * para indicar uma ou múltiplas ocorrências de caracteres alfabéticos. Uma única inserção deve representar singular, plural, masculino, feminino, entre outras variações.

Exemplificando, na ferramenta implementada o termo conec* permitirá a representação de palavras como conecta, conectar, conectando, conectou. O caracter * também poderá estar presente substituindo uma palavra interna ao termo, como por exemplo em *password * acesso*, expressão que irá representar password de acesso, password para acesso, password do acesso. Os termos podem ser registrados tanto em maiúsculas como em minúsculas.

Ao final desta atividade, teremos um dicionário onde cada entrada corresponde a um RNF, sua descrição e um conjunto de termos, que será utilizado com a finalidade de identificar a presença ou ausência desse RNF em documentos de requisitos.

3.3.2.3. Executar análise de conteúdo para RNF´s

A etapa de análise de conteúdo para identificar presença ou ausência de RNF´s em documentos de requisitos deve ser executada a cada modificação no documento de requisitos. O documento de requisitos é analisado, buscando-se identificar a presença de cada uma das expressões registradas no dicionário de RNF´s. Os resultados deste processo são expressos em forma de tabelas, a mais simples uma tabela de freqüência, conforme pode ser visualizado na Figura 26.

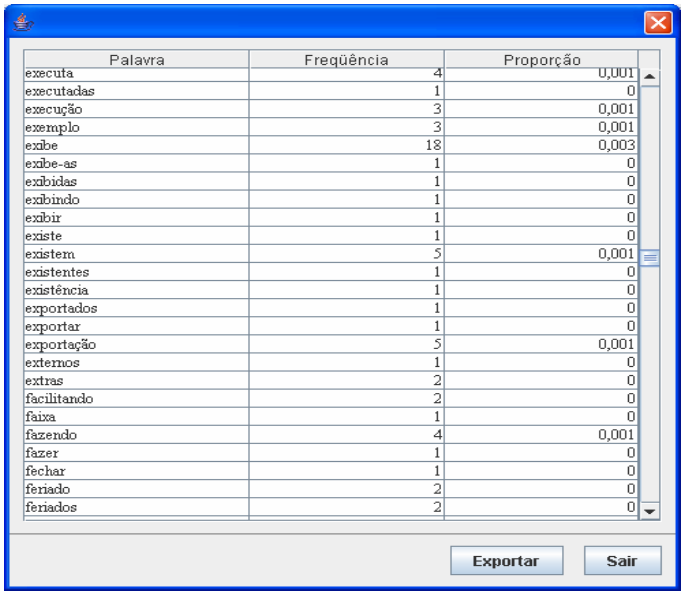
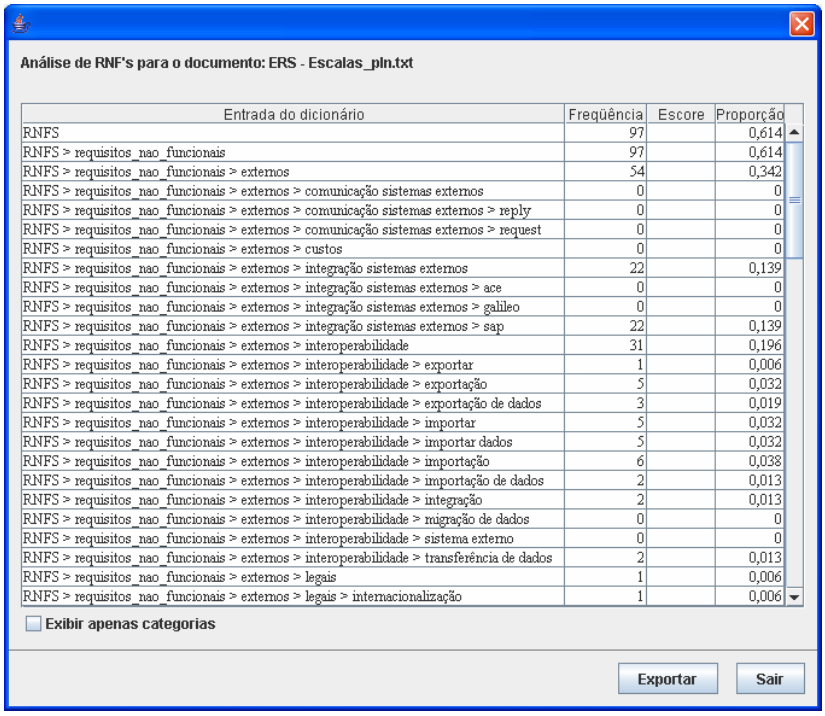


Figura 26 - Freqüência das palavras no documento de requisitos

É gerado um relatório apresentando, para cada uma das categorias, o detalhamento de freqüência de cada expressão registrada e o escore correspondente. O relatório consolida as informações de acordo com a hierarquia associada a cada um dos RNF´s, conforme apresentado na Figura 27.



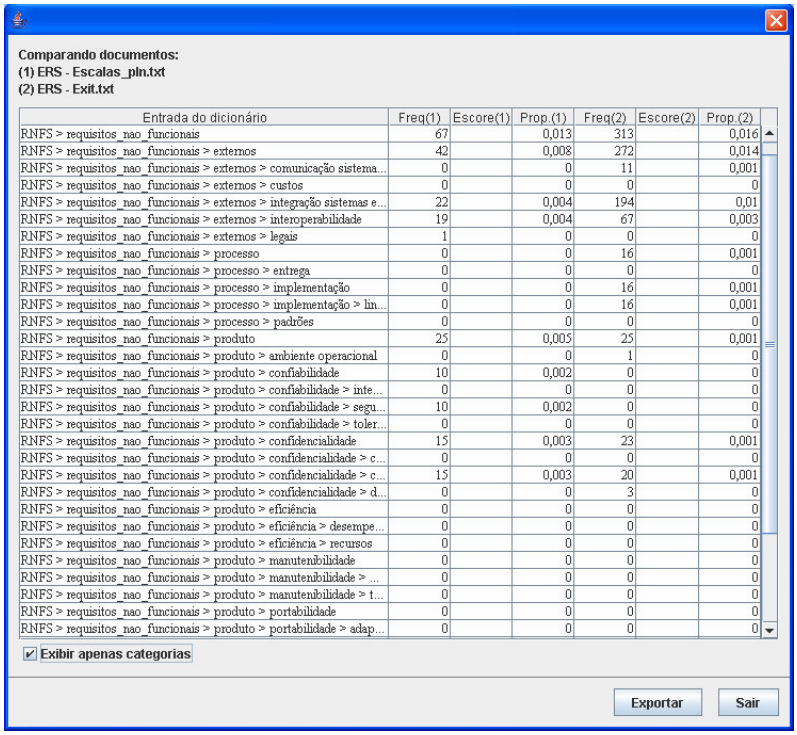
Entrada do dicionário	Freqüência	Escore	Proporção
RNFS	97		0,614
RNFS > requisitos_nao_funcionais	97		0,614
RNFS > requisitos_nao_funcionais > externos	54		0,342
RNFS > requisitos_nao_funcionais > externos > comunicação sistemas externos	0		0
RNFS > requisitos_nao_funcionais > externos > comunicação sistemas externos > reply	0		0
RNFS > requisitos_nao_funcionais > externos > comunicação sistemas externos > request	0		0
RNFS > requisitos_nao_funcionais > externos > custos	0		0
RNFS > requisitos_nao_funcionais > externos > integração sistemas externos	22		0,139
RNFS > requisitos_nao_funcionais > externos > integração sistemas externos > ace	0		0
RNFS > requisitos_nao_funcionais > externos > integração sistemas externos > galileo	0		0
RNFS > requisitos_nao_funcionais > externos > integração sistemas externos > sap	22		0,139
RNFS > requisitos_nao_funcionais > externos > interoperabilidade	31		0,196
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > exportar	1		0,006
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > exportação	5		0,032
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > exportação de dados	3		0,019
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > importar	5		0,032
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > importar dados	5		0,032
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > importação	6		0,038
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > importação de dados	2		0,013
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > integração	2		0,013
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > migração de dados	0		0
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > sistema externo	0		0
RNFS > requisitos_nao_funcionais > externos > interoperabilidade > transferência de dados	2		0,013
RNFS > requisitos_nao_funcionais > externos > legais	1		0,006
RNFS > requisitos_nao_funcionais > externos > legais > internacionalização	1		0,006

Figura 27 - Freqüência das categorias e expressões no documento de requisitos

Caso a organização disponha de um documento de requisitos considerado que seja considerado como modelo em termos de RNF's, esse documento poderá ser base para comparações, conforme visualizado na Figura 28.

Esses relatórios subsidiam o engenheiro de requisitos na avaliação das omissões no documento de requisito e elaboração de outras análises, entre as quais destacamos:

- coerência entre interoperabilidade com sistemas externos x desempenho
- coerência nos RNF´s de cadastrar usuários, conectar e desconectar
- ausência de RNF´s de documentação
- ausência de RNF´s de processo



Entrada do dicionário	Freq(1)	Escore(1)	Prop.(1)	Freq(2)	Escore(2)	Prop.(2)
RNFS > requisitos_nao_funcionais	67		0,013	313		0,016
RNFS > requisitos_nao_funcionais > externos	42		0,008	272		0,014
RNFS > requisitos_nao_funcionais > externos > comunicação sistema...	0		0	11		0,001
RNFS > requisitos_nao_funcionais > externos > custos	0		0	0		0
RNFS > requisitos_nao_funcionais > externos > integração sistemas e...	22		0,004	194		0,01
RNFS > requisitos_nao_funcionais > externos > interoperabilidade	19		0,004	67		0,003
RNFS > requisitos_nao_funcionais > externos > legais	1		0	0		0
RNFS > requisitos_nao_funcionais > processo	0		0	16		0,001
RNFS > requisitos_nao_funcionais > processo > entrega	0		0	0		0
RNFS > requisitos_nao_funcionais > processo > implementação	0		0	16		0,001
RNFS > requisitos_nao_funcionais > processo > implementação > lin...	0		0	16		0,001
RNFS > requisitos_nao_funcionais > processo > padrões	0		0	0		0
RNFS > requisitos_nao_funcionais > produto	25		0,005	25		0,001
RNFS > requisitos_nao_funcionais > produto > ambiente operacional	0		0	1		0
RNFS > requisitos_nao_funcionais > produto > confiabilidade	10		0,002	0		0
RNFS > requisitos_nao_funcionais > produto > confiabilidade > inte...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > confiabilidade > segu...	10		0,002	0		0
RNFS > requisitos_nao_funcionais > produto > confiabilidade > toler...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > confidencialidade	15		0,003	23		0,001
RNFS > requisitos_nao_funcionais > produto > confidencialidade > c...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > confidencialidade > c...	15		0,003	20		0,001
RNFS > requisitos_nao_funcionais > produto > confidencialidade > d...	0		0	3		0
RNFS > requisitos_nao_funcionais > produto > eficiência	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > eficiência > desempe...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > eficiência > recursos	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > manutenibilidade	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > manutenibilidade > ...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > manutenibilidade > t...	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > portabilidade	0		0	0		0
RNFS > requisitos_nao_funcionais > produto > portabilidade > adap...	0		0	0		0

Figura 28 - Comparação de documentos tendo por base o dicionário de RNF's

O protótipo utilizado nesta etapa do trabalho pode ser utilizado para explorar outras características do documento de requisitos. Por exemplo, os termos ou expressões relacionados no dicionário podem ser destacados através de uma opção da ferramenta, e o texto do documento de requisitos pode ser percorrido, com destaque para o termo escolhido.

4
Proposta de uma Arquitetura de Agentes

A abordagem de desenvolvimento centralizado de software tem sido amplamente utilizada por grandes e médias equipes de desenvolvimento. O desenvolvimento tradicional de software tem ocorrido em ambientes onde os artefatos utilizados ou gerados no desenvolvimento de software residem em um servidor local disponível aos interessados através de uma rede local. Entretanto, na economia globalizada atual, o desenvolvimento colaborativo de software envolvendo várias equipes distribuídas em locais diferentes está se tornando uma norma, ao invés de uma exceção [Wongthongthan06].

Freqüentemente, encontramos equipes de desenvolvimento distribuídas por diferentes cidades, regiões, e até mesmo em diversos continentes. Os modelos e metodologias atuais de desenvolvimento não contemplam as questões de projeto e desenvolvimento colaborativo de software [Wongthongthan06].

Um ambiente colaborativo de projeto é um ambiente automatizado que habilita pessoas (incluindo projetistas, engenheiros, administradores, clientes e usuários) a colaborar e interagir no desenvolvimento de novos projetos, considerando sua localização geográfica e os meios de interação [Wu06]. Wu declara também que projetos colaborativos de sistemas envolvem equipes multidisciplinares de projeto e várias ferramentas de engenharia de software; alguns dos principais problemas encontrados neste contexto incluem a dificuldade de troca de informações entre ferramentas devido aos diferentes padrões adotados, e a quase inexistência de mecanismos para auxiliar os usuários em suas tarefas colaborativas.

A arquitetura descrita neste capítulo tem por objetivo dar suporte ao processo de verificação e validação de requisitos em ambientes distribuídos de desenvolvimento. Este cenário requer que aplicações possam ser configuradas dinamicamente, e que arquiteturas flexíveis e habilitadas a evoluir sejam utilizadas para dar suporte à dinâmica destes sistemas, à constante evolução dos requisitos e às mudanças na equipe envolvida. A arquitetura proposta envolve o

uso de agentes de software.

O uso de agentes em ambientes distribuídos de desenvolvimento é relatado em [Li03], que propõe uma ferramenta para gerenciamento de dados utilizando uma abordagem baseada em agentes. Agentes de software também são utilizados para dar suporte ao gerenciamento de uma ontologia de engenharia de software destinada a apoiar uma equipe de desenvolvimento num ambiente geograficamente distribuído [Wongthongthan06]. Vários outros relatos apontam soluções para ambientes distribuídos utilizando agentes de software [Gaeta02] [Chang01], este último especificamente numa ferramenta de suporte ao processo de requisitos num ambiente distribuído.

Nos experimentos realizados identificamos que diferentes técnicas para tratamento de linguagem natural podem ser compostas e associadas a outras de forma a prover um novo tratamento para os dados. Desta forma, um conjunto inicial de tratamento da informação destinado a apoiar atividades de verificação e validação pode ser rapidamente incrementado, reutilizando algumas das técnicas já implementadas. Web Services provêem um paradigma apropriado para o desenvolvimento de sistemas com estas características porque são independentes de aplicações, plataformas e provedores.

4.1. Adequação de Web services e agentes

Tipicamente, em ambientes distribuídos de projeto, os recursos de informação são dinâmicos e muitas vezes heterogêneos. Além disso, estes ambientes de computação normalmente são dinâmicos, onde recursos de informação podem ser conectados e desconectados a qualquer momento. Agentes de software capturam e implementam serviços como funcionalidades e papéis. Da perspectiva de agentes, serviços Web são simplesmente entidades programáticas que podem ser chamadas para executar uma determinada atividade, tipicamente uma função unitária. Para que serviços Web possam trabalhar juntos com agentes, é necessário agregar propriedades comportamentais e de agenciamento, como colaboração e interação. É necessário dispor de uma arquitetura para que os agentes possam interagir, colaborar, compor e gerenciar serviços.

Diferentes de *sites* Web e aplicações *desktop*, serviços Web não são

projetados para interação direta com agentes humanos; eles operam no nível do código, são chamados e trocam informações com outros softwares através dos padrões estabelecidos para a Web. Um serviço Web pode ser usado quando o construtor de uma aplicação deseja expor alguma operação reativa, expressa como uma função, com ou sem parâmetros, que pode retornar ou não uma resposta. Na essência, um serviço Web funciona como uma invocação remota de um método, usando mensagens encapsuladas em XML sobre HTTP [Li03a].

Web Services são apropriados para incorporar regras de negócios em ambientes distribuídos [Li03a], enquanto agentes de software são apropriados em tais serviços para implementar atividades de interação e coordenação. Desta forma, propomos utilizar uma arquitetura orientada a serviços que utiliza agentes de software para compor a solução.

Pesquisadores da área de PLN tem disponibilizado ferramentas *open source* ou mesmo *free* para a comunidade em geral. Uma das motivações para a escolha da plataforma a ser utilizada na implementação da nossa proposta é relacionada, portanto, à habilidade de encapsular ferramentas já disponíveis na forma de serviços Web, possibilitando sua utilização quase que imediata.

4.2. Plataforma utilizada: MIDAS

Para demonstrar o funcionamento e viabilidade da solução, uma plataforma para a execução do modelo deve ser utilizada. Pesquisando as atuais plataformas para o desenvolvimento de SMAs para a Web, podem ser encontradas várias alternativas, tais como JADE [Bellifemine01], AgentScape [Overainder04], MIDAS [Haendchen05] [Haendchen07], dentre outras. Para os propósitos deste trabalho, alguns requisitos assumem um papel fundamental na escolha da plataforma. Primeiro, a plataforma deverá prover as funcionalidades básicas para a integração entre Web Services e agentes. Além destas funcionalidades, é necessário que existam mecanismos para dar suporte ao modelo de comunicação entre os agentes.

O *blackboard* tem sido utilizado para este propósito: a arquitetura baseada em *blackboards* é uma das mais utilizadas em sistemas multi-agentes cognitivos, e foi inicialmente criada para reconhecimento de voz no sistema HearsayII

[Ferber99]. Neste modelo de arquitetura, a comunicação entre agentes é possibilitada através do compartilhamento de informações armazenadas no *blackboard*. Isto significa que agentes podem ser inseridos dinamicamente no sistema, sem que sua identidade seja conhecida a priori pelos agentes já ativos no ambiente. Da mesma forma, um agente pode ser removido do ambiente sem que isto cause postergação indefinida em agentes que com ele trocavam informações de forma síncrona – outros agentes do ambiente podem assumir o seu papel, ou executar as atividades pelas quais o agente removido era responsável. Podemos dizer então que o uso de *blackboard* facilita atividades de interação e comunicação entre agentes, permitindo uma boa representação do ambiente de desenvolvimento de software, no qual humanos podem ser incluídos ou retirados de um determinado projeto de desenvolvimento de software.

Esta propriedade assume importante papel para os propósitos deste trabalho. Considerando as características abertas das aplicações a serem geradas, as facilidades obtidas pela utilização e possibilidade de reuso de *Web Services* e a necessidade de utilizar um modelo de comunicação onde os agentes possam se comunicar de forma anônima, a plataforma MIDAS foi escolhida para implementar a arquitetura proposta. Ela atende aos principais requisitos discutidos: propicia reuso em larga escala de serviços distribuídos e utiliza o *blackboard* como um meio de comunicação entre os agentes.

A arquitetura de MIDAS aplica os padrões definidos pela arquitetura de referência WSA, sendo baseada na coexistência de vários *containers*, cada um executando uma JVM (Java Virtual Machine). Cada máquina virtual provê um ambiente completo de execução, onde aplicações baseadas em agentes podem executar de forma concorrente no mesmo *host*. A Figura 29 mostra a arquitetura genérica do sistema, detalhando os elementos da arquitetura do servidor de front-end (FES) e dos containers de agentes (AC).

Na Figura 29 o *front-end* da plataforma Midas e o *container* de agentes e componentes já são apresentados na forma como estão instanciados para o nosso trabalho. Da plataforma Midas apresentaremos brevemente apenas características importantes no contexto deste trabalho, e detalharemos os aspectos dos agentes e componentes da aplicação.

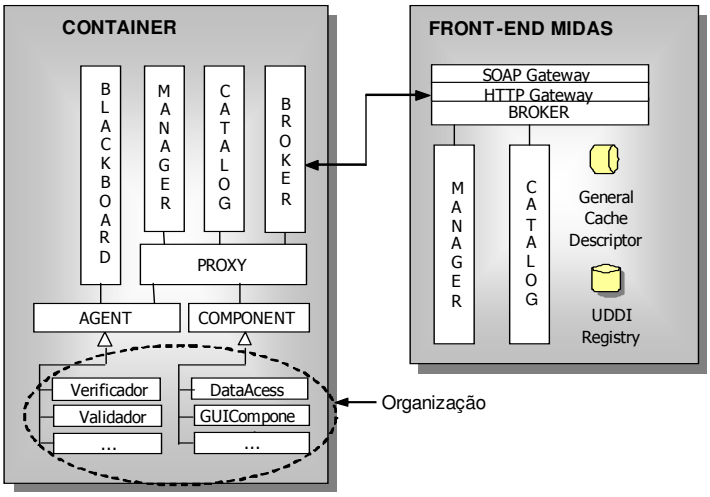


Figura 29 - Arquitetura da plataforma Midas

O *front-end* Midas possui agentes internos que provêem serviços de infra-estrutura à própria plataforma; na Figura 29 eles são representador por BROKER, MANAGER e CATALOG. Estes três agentes provêem serviços de comunicação, gerenciamento do ciclo de vida dos agentes e manutenção do catálogo de serviços para toda a plataforma. A comunicação ocorre através de duas portas de comunicação, via http ou via SOAP, esta última necessária para *web services.*

No container estão presentes agentes de infra-estrutura do container: BROKER, MANAGER, CATALOG e PROXY. Os três primeiros são instâncias dos correspondentes no *front-end* MIDAS, agora responsáveis pela comunicação, gerenciamento e catálogo de serviços do próprio container. O agente PROXY possibilita que os agentes de aplicação possam invocar serviços de forma transparente, sem necessitar conhecer a localização do serviço. O BLACKBOARD mantém informações do ambiente de execução, possibilitando que os agentes da aplicação monitorem as mudanças neste ambiente e tomem as ações apropriadas a cada caso.

Os agentes da aplicação são instanciados estendendo a classe abstrata AGENT. Esta classe abstrata contém os métodos comuns a todos os agentes da aplicação, como métodos para controle do ciclo de vida do agente e as interfaces para monitoração do *blackboard.* A classe abstrata COMPONENT representa componentes puramente reativos, e deve ser estendida para implementar, por exemplo, objetos de acesso a bancos de dados, interfaces de comunicação com usuários e objetos que encapsulam regras de negócio específicas do domínio da

aplicação. Agentes e componentes da aplicação compõem uma organização.

Neste ambiente, componentes são entidades puramente reativas, e implementam aspectos característicos do domínio da aplicação como, por exemplo, interface com o usuário e mecanismos para acesso aos dados.

4.3. Aplicação: características, agentes e modelos de papéis

A ferramenta de suporte às técnicas propostas foi visualizada para atender às seguintes características:

- repositório de artefatos de requisitos e de informações sobre projetos
- servidor mantendo repositórios e provendo serviços de comunicação
- comunicação entre sites via protocolo http
- acesso remoto via *browser* e protocolo de comunicação *http*
- autenticação de usuários
- perfis de usuários, com diferentes direitos de acesso
- controle de modificações em requisitos
- serviços de PLN e outros disponibilizados via *web services*
- agentes como assistentes pessoais, como provedores de serviços, monitores de modificações.

4.3.1. Perfil de usuários

Definimos três tipos básicos de perfis para os participantes do processo de requisitos na plataforma: gerente de projeto, engenheiro de requisitos e representante de clientes e usuários. Ao gerente de projeto são delegados acesso a todas as funcionalidades incluindo manutenção de artefatos de requisitos, agendamento das atividades de V&V e definição dos participantes para os processos de V&V. O engenheiro de requisitos não tem acesso ao agendamento das atividades de V&V, mas pode efetuar manutenções nos artefatos do repositório, e o representante de clientes e usuários participa de atividades de V&V, mas tem acesso apenas para leitura aos artefatos do repositório.

4.3.2.
Uso do blackboard

O *blackboard* é uma metáfora ao meio ambiente onde os agentes desempenham seus papéis. Todos os agentes monitoram o ambiente, identificam a ocorrência de eventos neste ambiente e decidem se devem executar alguma ação em função do ocorrido. Os principais eventos de interesse dos agentes são chegada de mensagens, solicitações de serviços e modificações no estado do documento de requisitos.

A comunicação entre agentes se processa através deste mecanismo: o agente envia uma mensagem, no formato apropriado, que é colocada no *blackboard*. Os agentes monitoram este ambiente e detectam a modificação ocorrida. Cada um dos agentes então verifica se ele é o destinatário da mensagem e, em caso positivo, trata a mensagem recebida de acordo com o seu papel.

O *blackboard* é utilizado também para manter registro do estado do documento de requisitos. No estado *aberto* requisitos podem ser inseridos, excluídos ou modificados; nos estados de *pré-verificação* ou de *verificação* não são mais aceitas modificações. Após a conclusão da verificação o estado retorna para *aberto*. De maneira análoga, no estado de *pré-validação* e de *validação* não são aceitas modificações em requisitos.

Solicitações de serviços são feitas também através do *blackboard*: o agente que requisita o serviço coloca a requisição no *blackboard*, cada agente que monitora o ambiente detecta a chegada de uma solicitação e verifica se é ele que deve atender à solicitação. A solicitação de serviços tem um protocolo padrão, que registra o nome do serviço e parâmetros de entrada e saída.

4.3.3.
Repositório de artefatos e de informações sobre projetos

A organização deve manter um repositório de artefatos relacionados ao projeto, entre eles o documento de requisitos. Requisitos são armazenados individualmente e, eventualmente reunidos num único documento. Cada requisito possui um conjunto de atributos que o identifica e provê outras informações.

Para ter acesso aos requisitos e demais artefatos (utilizados ou gerados na fase de requisitos), os participantes do projeto são cadastrados. Uma importante

informação, além do perfil do participante, é o endereço eletrônico. Através deste endereço o participante recebe informações relativas a modificações em requisitos, ao agendamento de atividades de V&V, a resultados parciais de atividades de verificação em requisitos.

4.3.4. Agentes: modelo de papéis

A aplicação desenvolvida é um sistema que combina agentes e componentes. Agentes, nesta plataforma, possuem papéis bem definidos, descritos a seguir.

4.3.4.1. Agente Manager

O agente Manager é um assistente pessoal do gerente do projeto, e responsável por monitorar atividades do gerente do projeto, detectar o agendamento de atividades de V&V no documento de requisitos, notificar os interessados e o agente Verificador/Validador sobre o evento. A Figura 30 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Manager**	**Colaboradores**
Responsabilidades ▪ Identificar agendamento da verifição ou da validação: – monitorar atividades do gerente humano e identificar o agendamento de uma data para a verificação ou validação – modificar estado do documento de requisitos de aberto para pre-verificação ou pré-validação ▪ Enviar notificações: – notificar Comunicador da agenda definida para a verificação/validação de requisitos – notificador Verificador/Validador	▪ Comunicador ▪ Verificador ▪ Validador

Figura 30 - Responsabilidades e colaborações do agente Manager

4.3.4.2. Agente Analisador Léxico (Lexical)

Este agente centraliza os serviços de tratamento léxico a ser efetuado sobre o documento de requisitos. Fornece serviços para vários agentes da plataforma,

atendendo requisições para conversão de documentos, extração do radical de palavras, obter contextos, identificar colocações, categorizar requisitos, identificar termos não dicionarizados. A Figura 31 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Analisador Léxico**	**Colaboradores**
Responsabilidades ▪ Converter documentos: – conversão de documentos de requisitos armazenados em formato doc, pdf ou html para formato texto puro ▪ Extrair requisitos: – analisar documento de requisitos, extraindo e armazenando em arquivos independentes cada um dos requisitos ▪ Fornecer stem de palavras: – a partir de uma palavra, retornar o radical ou *stem* da mesma ▪ Obter contextos (Concordanceador): – analisar o documento de requisitos e retornar as sub-sentenças onde constem o termo em análise ▪ Categorizar requisitos: – para cada item de mais alto nível da taxonomia, gerar e consolidar os padrões de segundo nível – identificar requisitos associados a cada um dos padrões ▪ Gerar matriz termo-documento: – gerar matrizes termo-documento, para uso em serviços que utilizem a abordagem *bag-of-words*	▪ Verificador ▪ Gerador de Visões ▪ Construtor do Léxico ▪ Estatístico

Figura 31 - Responsabilidades e colaborações do agente Analisador Léxico

4.3.4.3. Agente Construtor do Léxico (LexicalConstructor)

O agente Construtor do Léxico é responsável pela manutenção do léxico da aplicação, visando à atualização da base de conhecimentos para o domínio da organização. A Figura 32 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Construtor do Léxico**	**Colaboradores**
Responsabilidades ▪ Criar ou atualizar o léxico da aplicação: – Identificar termos não dicionarizados – identificar atores e recursos, via análise do documento de requisitos e extração de sintagmas nominais relevantes – em interação com o agente Léxico, buscar contextos para cada um dos terrmos não dicionarizados	▪ Analisador Léxico

Figura 32 - Responsabilidades e colaborações do agente Construtor do Léxico

4.3.4.4. Agente Gerador de Visões (Mapper)

O agente Gerador de Visões é responsável pelo enriquecimento de uma taxonomia básica fornecida pelos usuários, categorizar os requisitos tendo por referência esta mesma taxonomia e gerar visões textuais e gráficas dos agrupamentos obtidos. A Figura 33 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Gerador de Visões**	**Colaboradores**
Responsabilidades ▪ Gerar uma taxonomia: – utilizar termos fornecidos pelos usuários, que correspondem ao nível mais alto da taxonomia – em interação com o agente Léxico, identificar colocações e obter padrões léxicos que irão compor o segundo nível da taxonomia ▪ Categorizar requisitos: – em interação com o agente Léxico, identificar requisitos associados a cada um dos ramos da taxonomia ▪ Gerar visões: – gerar visão textual dos agrupamentos identificados – gerar mapa XTM para possibilitar visões gráficas ▪ Enviar notificações: – notificar agente verificador/validador e demais interessados, via agente Comunicador	▪ Analisador Léxico ▪ Comunicador ▪ Verificador ▪ Validador

Figura 33 - Responsabilidades e colaborações do agente Gerador de Visões

4.3.4.5. Agente Estatístico (Statistics)

O agente Estatístico é responsável pelo cálculo do mapa de freqüência e métricas como escore T e informação mútua, trabalhando sobre os contextos extraídos do documento de requisitos e relacionados a um determinado tema. A Figura 34 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Estatístico**	**Colaboradores**
Responsabilidades ▪ Calcular freqüências: – recebe um conjunto de sub-sentenças (concordances) relacionadas a um determinado tema e calcula frequencias do nodo (o termo principal) e dos dems termos presentes ▪ Calcular métricas para colocações: – calcula os valores de escore T e informação mutua para os termos (colocações), a partir dos valores de frequencia já obtidos. Apenas as colocações que possirem score T >=2 e informação mútua > 3 serão selecionadas	▪ Analisador Léxico ▪ Gerador de Visões

Figura 34 - Responsabilidades e colaborações do agente Estatístico

4.3.4.6. Agente Verificador

O agente verificador é responsável por atividades de verificação dos requisitos, como identificar requisitos em duplicidade e omissões em requisitos não funcionais. Para identificação de requisitos em duplicidade inicialmente é gerada a matriz termo-documento e são calculados os índices de similaridade de Dice, Jaccard e do coseno. Para omissões em requisitos não funcionais é utilizada uma taxonomia de RNF´s instanciada para a organização. A essa taxonomia são agregados termos em linguagem natural que são normalmente utilizados para fazer referência a esses requisitos não-funcionais. A Figura 35 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Verificador**	**Colaboradores**
Responsabilidades ▪ Identificar requisitos em duplicidade: – em interação com o Agente Léxico, gerar a matriz termo-documento para o conjunto de requisitos – executar análise estatística sobre pares de documentos, retornando medidas de similaridade (índices de similaridade de Dice, Jaccard e cos-seno) entre eles – gerar matriz de similaridades, e extrair pares de requisitos cujo índice seja maior que o limiar definido ▪ Identificar omissões em RNF´s: – utilizando uma taxonomia de requisitos não funcionais instanciada para a organização, identificar omissões e gerar relatório com análise dos requisitos emr elação a omissões e inconsistências detectadas ▪ Enviar mensagens: – notificar interessados, via agente Comunicador, dos resultados obtidos nas atividades de pré-verificação	▪ Analisador Léxico ▪ Comunicador

Figura 35 - Responsabilidades e colaborações do agente Verificador

4.3.4.7. Agente Validador

O agente Validador é responsável pela coleta e distribuição dos artefatos para a validação, envio de notificação aos envolvidos e consolidação parcial do relatório. A Figura 36 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Validador**	**Colaboradores**
Responsabilidades ▪ Organização da validação: – utilizar termos fornecidos pelos usuários, que correspondem ao nível mais alto da taxonomia – em interação com o agente Léxico, identificar colocações e obter padrões léxicos que irão compor o segundo nível da taxonomia ▪ Gerar visões: – gerar visão textual dos agrupamentos identificados – gerar mapa XTM para possibilitar visões gráficas ▪ Enviar notificações: – notificar interessados, via agente Comunicador	▪ Analisador Léxico ▪ Gerador de Visões ▪ Comunicador

Figura 36 - Responsabilidades e colaborações do agente Validador

4.3.4.8. Agente Comunicador

O agente Comunicador do Léxico é responsável pelo envio de mensagens aos participantes do processo de desenvolvimento, notificando-os em relação a eventos de seu interesse, como definição de agendas para os processos de verificação/validação ou modificações ocorridas nos requisitos do sistema. A Figura 37 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Comunicador**	**Colaboradores**
Responsabilidades ▪ Envio de mensagens e notificações: – responsável pelas comunicações entre agentes e participantes do processo de requisitos	▪ Analisador Léxico ▪ Verificador ▪ Validador ▪ Construtor do léxico ▪ Observador

Figura 37 - Responsabilidades e colaborações do agente Comunicador

4.3.4.9. Agente Observador

O agente Observador é responsável pelo monitoramento de operações de escrita nos artefatos de requisitos e, em caso de alteração, é o responsável por notificar os interessados. A Figura 38 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Observador**	**Colaboradores**
Responsabilidades ▪ Identificar mudanças nos artefatos controlados: – Manter controle sobre documento de requisitos, identificando modificações ▪ Notificação de interessados: – Em colaboração com o agente Comunicador, notificar interessados sobre modificação ocorrida	▪ Comunicador

Figura 38 - Responsabilidades e colaborações do agente Observador

4.3.4.10. Agente Rastreador

O agente Rastreador é responsável pela construção de matrizes de rastreabilidade, e pela verificação destas matrizes com aquelas fornecidas pelo engenheiro de requisitos, apontando as divergências e apresentando a opção de tratar ou não cada uma das divergências apontadas. A Figura 39 ilustra as principais responsabilidades e colaborações deste agente.

Agente **Rastreador**	**Colaboradores** ▪ Comunicador
Responsabilidades ▪ Construção de matriz de rastreabilidade para dependencias entre requisitos: – Em colaboração com o agente Léxico, identificar dependências entre requisitos funcionais e não funcionais – Gerar matriz de rastreabilidade RF x RNF ▪ Notificação de interessados: – em colaboração com o agente Comunicador, notificar interessados após construção da matriz	

Figura 39 - Responsabilidades e colaborações do agente Rastreador

4.4. Uma visão da implementação

A visão de alto nível da plataforma de suporte ao sistema é apresentada na Figura 40: um servidor central que é ao mesmo tempo responsável por prover a comunicação entre os diferentes sites da organização e também mantém o repositório de informações sobre projetos. A comunicação entre as máquinas clientes, servidor local e servidor central é realizada via protocolo *http*,

possibilitando que qualquer máquina conectada à Internet possa acessar as informações armazenadas no servidor central com a utilização de um *browser*.

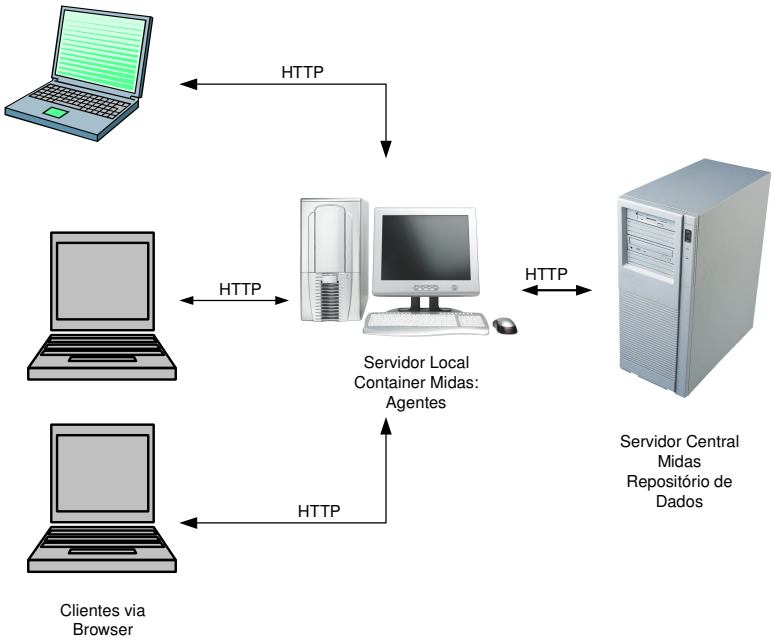


Figura 40 - Visão de alto nível da plataforma

A plataforma Midas provê uma interface, denominada Agent Server Manager, a qual possibilita o gerenciamento do ciclo de vida dos agentes. A Figura 41 apresenta a visão do sistema multi-agente criado, com os agentes referidos na seção anterior, através desta interface. É possível também a visualização dos serviços disponibilizados pelos agentes, como pode ser observado no detalhamento do agente Lexical.

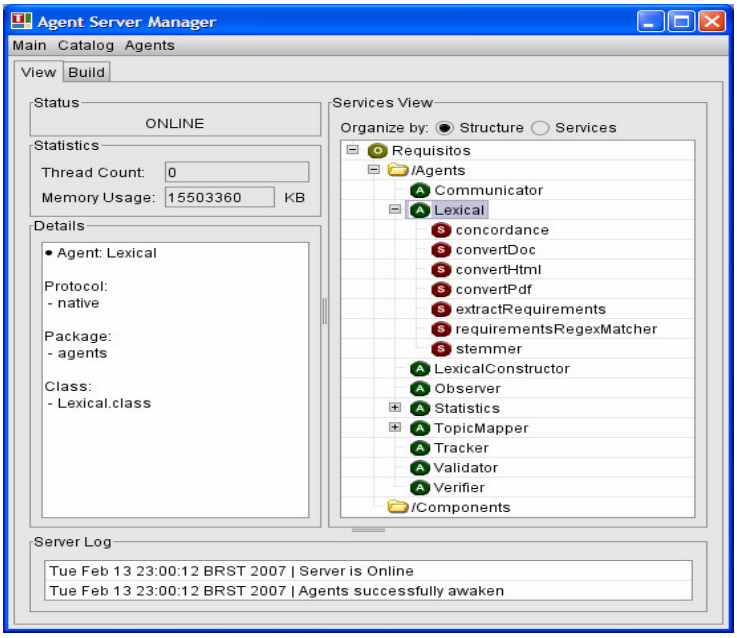


Figura 41 - Visão do sistema multi-agente

A arquitetura da aplicação multi-agente e é apresentada a seguir, com descrição e características principais.

4.4.1. Estrutura da aplicação multi-agente

A aplicação está estruturada em 3 subsistemas, a saber: *core*, *web* e *midas*. Para a implementação foi utilizada a linguagem Java, mas os diferentes softwares incorporados como serviços dos agentes estão desenvolvidos nas linguagens Perl, C e AWK. Descrevemos a seguir cada um desses subsistemas.

O subsistema *core* agrega componentes responsáveis pelos agentes e seus serviços e pelo acesso e manutenção do repositório de informações sobre projetos. É composto pelos pacotes *requisitos.agents* (agentes que compõem a aplicação), *requisitos.business.entities* (entidades da aplicação), *requisitos.dao*, *requisitos.dao.hibernate* e *requisitos.dao.jdbc* (responsáveis pelas funcionalidades de acesso aos repositórios de dados) e *requisitos.util* (alguns serviços implementados na linguagem Java, como envio de mensagens, cálculo de similaridade entre requisitos e stemização).

O sub-sistema *web* agrega componentes responsáveis pelas funções de interface com os usuários do sistema, incluindo validação de acesso. É composto pelos pacotes *requisitos.web.adapter* (processos da aplicação como mudanças em

requisitos, geração de mapas visuais e processos de verificação e validação), *requisitos.web.controller* (responsável pelas comunicações entre usuário/midas/repositório), *requisitos.web.servlets* e *requisitos.web.user* (responsáveis pelas operações de login/logout na aplicação) e *requisitos.web.util* (montagem de strings *http* para comunicação).

O subsistema *midas* é constituído pela própria plataforma de agentes que fornece a infra-estrutura de comunicação, possibilita o gerenciamento do ciclo de vida dos agentes e a manutenção do catálogo de serviços.

4.5. Interface com o usuário: processos abordados

A aplicação oferece interfaces para três processos: manutenção de requisitos, geração de mapas e processo de verificação e validação.

Manutenção de documentos de requisitos: nos estudos de caso que fizemos utilizamos documentos de requisitos cedidos por uma organização que utiliza DDS; tais documentos eram versões ainda não verificadas e validadas, mas em princípio deveriam ser completas no sentido de abarcar todas as funcionalidades e atender as restrições colocadas para o sistema. Para que efetivamente a plataforma possa ser utilizada em ambientes distribuídos, é necessário que os requisitos possam ser submetidos por engenheiros de requisitos localizados em ambientes geograficamente distantes. A Figura 42 apresenta a interface para a manutenção do documento de requisitos: requisitos podem ser incluídos, modificados ou excluídos, via Web.

Geração de visões (mapas): no processo de V&V os participantes podem desejar obter uma particular visão dos requisitos, buscando avaliar o conjunto de requisitos associados a um determinado tema. Para esse tipo de solicitação a plataforma provê uma interface onde o usuário seleciona o projeto, informa o tema e solicita as associações relevantes. A plataforma identifica as associações e as avalia através das medidas de informação mútua e score T; o usuário pode então solicitar a geração do mapa visual correspondente. Isto pode ser visualizado na Figura 43.

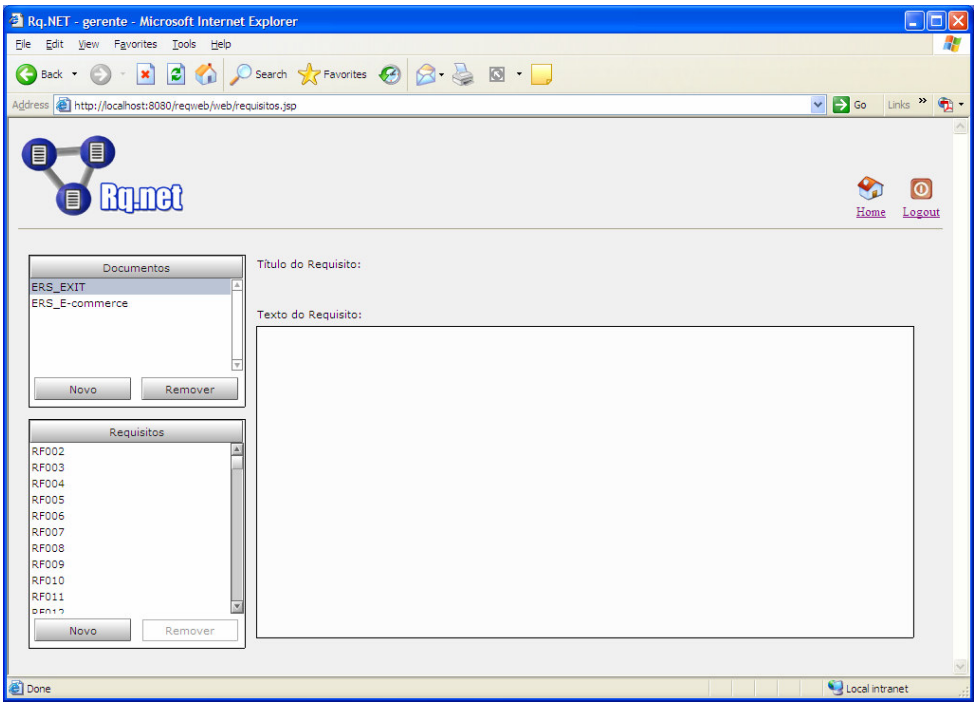


Figura 42 - Interface para manutenção de documentos de requisitos

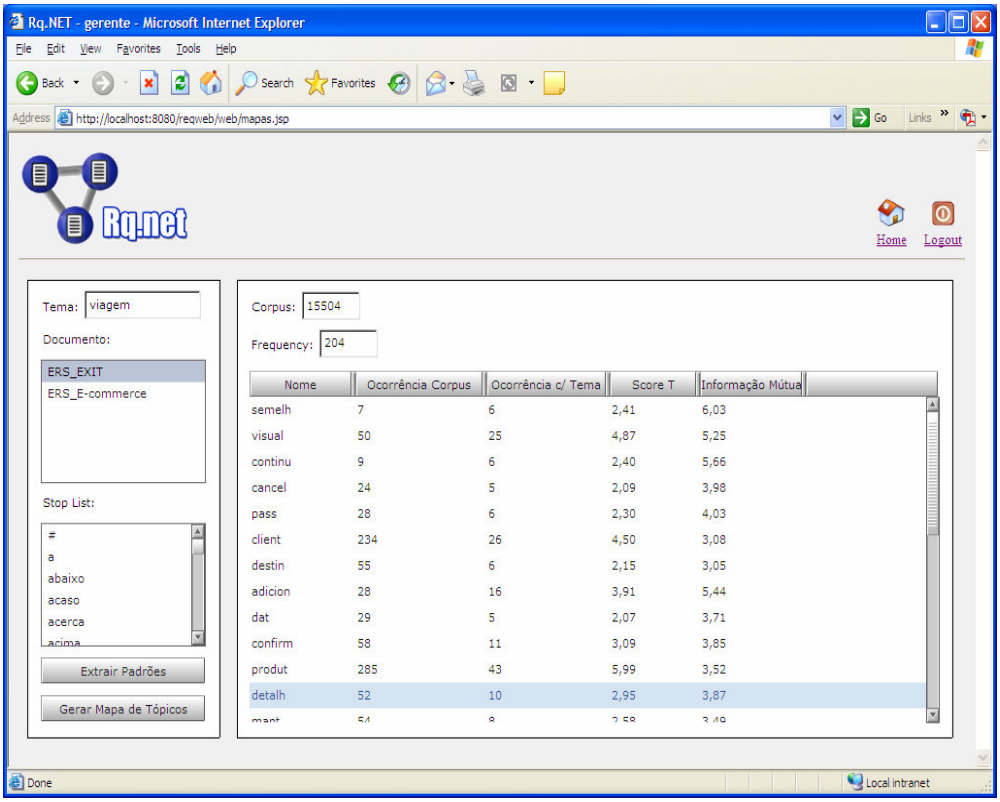


Figura 43 - Interface para identificação de requisitos associados a um tema

Processo de V&V: é responsabilidade do gerente do projeto escolher uma data para o processo de verificação do documento de requisitos. Nesse momento o seu agente pessoal, em colaboração com os demais agentes da plataforma, se encarrega de enviar mensagem aos participantes informando da data definida para a verificação. O estado do documento de requisitos é alterado, e não são mais aceitas inserções ou atualizações no documento. São realizadas algumas verificações, como a busca pela duplicidade em requisitos, e os resultados são enviados aos participantes também por mensagem eletrônica. O diagrama de atividades para este processo é mostrado na Figura 44.

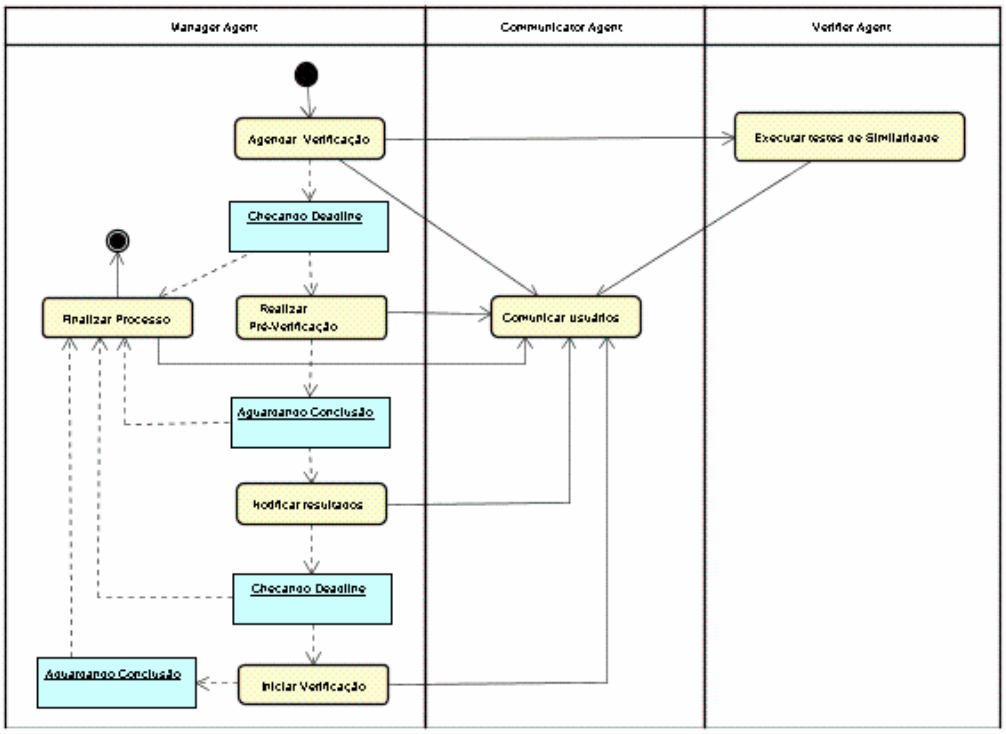


Figura 44 - Diagrama de atividades para o processo de Verificação

O diagrama de atividades para o processo de validação é equivalente ao diagrama das atividades de verificação, à exceção da verificação de similaridades, que não tem correspondente na validação.

4.6. Algumas considerações sobre a aplicação

A implementação da plataforma multi-agente para a aplicação apresenta as principais características e propriedades: uso de agentes pessoais, encapsulamento e reuso de aplicações completas como serviços de agentes (*web services*),

notificação dos interessados em eventos específicos no processo de V&V. Nem todos os serviços dos agentes estão disponíveis ainda através de interface com o usuário, pois a aplicação ainda está em fase de construção. O código fonte dos agentes aqui apresentados está disponível no Anexo D deste volume.

Os serviços e funcionalidades implementados por softwares disponibilizados pelas comunidades de PLN e incorporados e/ou utilizados na plataforma são:

1. software: Yoshikoder converter
    - finalidade: conversão de documentos dos formatos *doc*, *html* e *pdf* para txt
    - serviços ConvertDoc, ConvertHtml e ConvertPdf
    - agente: Analisador Léxico
    - obs: foram necessárias modificações para correção de bugs
2. software: Q-TAG
    - finalidade: POS Tagger: colocação de etiquetas gramaticais aos tokens
    - serviço: tagger
    - agente: Construtor do Léxico
    - obs: fornecido apenas o executável e arquivo de treinamento
3. software: qtoken
    - finalidade: divisão do texto em tokens
    - serviço: tokenizer
    - agente: Construtor do Léxico
    - obs: fornecido apenas o executável
4. software: pretex
    - finalidade: geração de matrizes termo-documento
    - serviço:
    - agente: Verificador
    - obs: implementado em Perl, o *stemmer* foi substituído por uma implementação baseada em [Orengo01]
5. software: yoshikoder
    - finalidade: geração de relatórios para análise do discurso

- serviço:
- agente: Verificador
- obs: modificado para permitir utilização de expressões (e não apenas palavras simples) e colocação dos menus em português. Ainda não incorporado à plataforma.

6. software: TMNAV
   - finalidade: exibição de mapas de tópicos
   - serviço:
   - agente: Verificador e Validador
   - obs: ainda não incorporado à plataforma.
7. software: analisador
   - finalidade: identificação de termos não dicionarizados
   - serviço:
   - agente: Construtor do Léxico
   - obs: implementado em C e AWK pelo pesquisador Akeo Tanabe, do LES/DI/PUC-Rio

A Tabela 10 relaciona agentes, Web services associados, software utilizado e estado atual da implementação, permitindo ao leitor uma visão geral dos resultados obtidos até o momento na ferramenta de suporte à estratégia utilizada.

Tabela 10 - Estado atual da implementação da ferramenta de suporte

AGENTE	SERVIÇO WEB	SOFTWARE	ESTADO
Comunicador			implementado
Analisador Léxico	conversão de formatos	Yoshikoder Converter	modificado e encapsulado
	Concordanceador		implementado
	individualizar requisitos		implementado
	identificador de padrões		implementado
	stemizador		implementado e encapsulado
Construtor do Léxico	tagger	QTAG	encapsulado*
	tokenizer	qtoken	encapsulado*
		analisador	ainda não encapsulado
Manager			implementado

AGENTE	SERVIÇO WEB	SOFTWARE	ESTADO
Observador			implementado
Estatístico	frequencyMap		implementado e encapsulado
	metrics		implementado e encapsulado
Gerador de Visões	build		implementado e encapsulado
	setAssociations		implementado e encapsulado
		TMNav	ainda não encapsulado
Rastreador			não implementado
Validador			não implementado
Verificador	TermDocument	pretext	encapsulado
	calculateSimilarity		implementado e encapsulado
	contectAnalysis	Yoshikoder	modificado ainda não encapsulado

5
Estudos de caso

Os vários estudos de caso apresentados neste capítulo foram utilizados na exemplificação dos processos propostos no capítulo 3. Nos estudos de caso foram utilizados diversos artefatos de requisitos, originados do desenvolvimento de aplicações reais em organizações ativas no mercado nacional. Uma destas organizações, que cedeu a maior parte dos documentos utilizados neste capítulo, utiliza desenvolvimento distribuído de software.

As técnicas propostas foram aplicadas aos documentos referidos, e os resultados obtidos são apresentados e discutidos a seguir. Buscamos utilizar na estruturação de cada estudo de caso a seguinte seqüência: (i) agrupamento e geração de visões dos requisitos; (ii) construção do léxico das aplicações e (iii) identificação de discrepâncias, erros e omissões no documento de requisitos, apresentados nas seções a seguir.

5.1.
Estudo de caso: documento Exit

Este documento, denominado de Exit, é um documento preliminar de requisitos, para um sistema de informação na área do turismo. O documento segue uma estrutura definida pela organização, com separação dos requisitos funcionais daqueles não-funcionais; no total, são cento e sessenta e nove requisitos (quatro dos quais não-funcionais), e o total de palavras atinge vinte e nove mil, quinhentos e cinqüenta e quatro palavras. Requisitos, neste documento, são definidos por uma ou mais sentenças, sendo ainda encontradas figuras ilustrativas nas descrições dos requisitos. Na média, cada requisito é descrito por aproximadamente cento e oitenta palavras, distribuídos em quinze linhas. O documento está escrito na língua portuguesa, parte dele na variante do Brasil. A tabela 6, apresentada na seção 5.1.3, mostra exemplos de requisitos desse documento. Documento cedido pela Tlantic Sistemas de Informação.

Este documento passou incialmente por uma etapa de pré-processamento

que o converteu do formato *doc* para o formato *txt*, excluindo as figuras. Sempre que possível, as tabelas existentes no documento foram transformadas para texto. Exemplos de requisitos desse documento podem ser vistos na Tabela 16, apresentada na seção 5.1.1.3. O documento também foi submetido ao processo de individualização dos requisitos, para uso nas etapas que trabalham com um requisito por documento.

5.1.1. Geração de visões

A geração de visões seguiu as etapas propostas no capítulo 3: identificação de termos para a taxonomia, categorização dos requisitos e geração e apresentação das visões.

5.1.1.1. Identificação dos termos para enriquecimento da taxonomia

A taxonomia inicial foi composta pelos temas **comunicação**, **pacote**, **viagem, reservas**, **pagamento**, **sistemas externos** e **segurança.** A escolha desses termos seguiu os passos:

1. foram criadas 3 listas de termos, de acordo com os indicadores freqüência, LogLokelihood e realizado o teste χ^2
2. dos termos relacionados nas listas foram selecionados 30 de maior relevância, e excluídos aqueles não-dicionarizados. A Tabela 11 apresenta parcialmente termos e valores obtidos.
3. as listas foram comparadas para extração dos termos para compor o primeiro nível da taxonomia, considerando os indicadores apresentados.

Tabela 11 - Lista parcial de termos selecionados e índices

Tema	Freq.	χ^2	LogLikelihood
CLIENTE	224	24290,66	1858,29
PRODUTO	196	4803,79	938,70
VIAGEM	175	14557,81	1317,45
RESERVA	99	8278,97	747,50
PAGAMENTO	83	3403,69	485,81
VIAGENS	50	3039,56	338,20
PACOTE	49	3242,35	341,60
COMUNICAÇÃO	43	272,20	108,48
CONFIRMAÇÃO	36	2952,66	271,57
AVIAÇÃO	35	3985,98	298,39
VISUALIZAR	33	2620,52	246,30
RESERVAR	29	3896,10	265,70
DESTINOS	28	2628,79	222,46
VOO	20	3199,56	205,30

É interessante observar, na Tabela 11, que as listas apontavam a presença dos termos *reserva/reservar*, e *viagem/viagens*, que remetem a um mesmo conceito.

Na elaboração das listas, trabalhamos com termos de uma só palavra. Foi verificado, no entanto que alguns dos termos não-dicionarizados eram acrônimos de outros sistemas já existentes na organização (GDS e ACE, por exemplo), e com os quais haveria troca de informações. Ao mesmo tempo, o termo *sistema* constou na lista inicial (primeiros 30 termos), e o termo *externos* era o 41º na lista obtida utilizando o $\chi 2$. O termo *sistemas externos* foi incluído na lista final em função dessas observações. Já o termo *segurança* não constava das listas, mas foi incluído por representar um requisito não funcional importante para aplicação em pauta.

Detalharemos a seguir aspectos da obtenção dos termos associados ao tema de alto nível *reservas*; procedimentos similares foram executados para todos os demais temas de alto nível da taxonomia inicial. Para os temas *sistemas externos* e *segurança* foram fornecidos também alguns termos situados no segundo nível da taxonomia.

No documento de requisitos, iniciamos pela busca de contexto utilizando o radical da palavra **reserva** (*reserv*). Nesta busca, obtivemos um total de 144 sub-sentenças, que foram analisadas para identificação das colocações. As colocações obtidas foram *stemizadas*, classificadas por ordem decrescente de freqüência, e nesse momento desconsideradas todas as colocações com freqüência igual a um. A seguir, calculamos os valores das medidas de associação: o escore T e a informação mútua (conforme Tabela 12) e então selecionamos as colocações que atendiam aos critérios definidos, ou seja, com $T \geq 2$ e $im \geq 3$.

Finalmente, cada um dos termos candidatos foi avaliado em conjunto com o tema buscado, e foram descartados aqueles que iniciam ou terminam por *stopwords*. Do conjunto de exclusão constaram também termos da língua geral e termos usuais na área de TI, mas sem significado específico, como as palavras sistema, dado, etc. A Tabela 13 apresenta o conjunto final de padrões a ser utilizado em conjunto com o tema *reserva*, que é um dos integrantes do mais alto nível da taxonomia em construção.

Tabela 12 - Termos candidatos (parcial)

termos	freq	T	inf. mútua
Produt	62	7,5245	4,4937
Ser	26	4,6072	3,3739
Confirm	17	4,0619	6,0732
Efectu	16	3,9222	5,6832
Cancel	14	3,6854	6,0561
confirmaça	13	3,5215	5,4232
produt de aviaç	9	2,2022	6,0446

Tabela 13 - Padrões para o termo reserva

Tema principal: reserva		
Padrões		
reserv produt	reserv confirm	reserv aviac
efectu reserv	confirm reserv	reserv estad
cancel reserv	respons reserv	reserv voo
realiz reserv	list reserv	reserv funcion
aguard reserv	deleg reserv	produt reserv

Outra observação a ser considerada é a importância do stemmer a ser utilizado: este estudo de caso foi realizado antes de termos encontrado o *stemmer* descrito em [Orengo01], e o *stemmer* aqui utilizado não conseguiu identificar que a palavra confirmação possui o mesmo radical que palavras como confirmar, confirma, confirmada, ocasionando dupla entrada para "confirm reserv" e "confirmaç reserv". Esses dois padrões foram substituídos por um único, dado que "confirm reserv" engloba "confirmaç_reserv". Vários outros termos foram substituídos por padrões mais gerais.

5.1.1.2. Geração de visões: categorização dos requisitos com uso da taxonomia enriquecida

O processo de categorização dos requisitos teve por base a taxonomia enriquecida. A cada um dos termos da taxonomia foram associados os requisitos que continham ao menos um dos padrões identificados na etapa anterior. Os conjuntos assim obtidos possuíam diferentes cardinalidades, registradas na Tabela 14. O tema parametrização, não relacionado inicialmente, foi incluído para ampliar a cobertura dos requisitos em relação à taxonomia. Parametrização está relacionada a informações freqüentemente modificáveis no sistema, por exemplo data de envio das comunicações semanais com clientes e categorias de carro para alugar.

A análise de cobertura taxonomia/requisitos identificou que foram relacionados 166 requisitos à taxonomia, com um total de 312 ligações. Como

parte dos requisitos foi relacionada a mais de um termo da taxonomia, a Tabela 15 apresenta o número de requisitos por quantidade de ligações.

Tabela 14 - Taxonomia e requisitos relacionados

Termo	Requisitos relacionados
viagem	51
pagamento	31
reserva	32
pacote	46
comunicação	40
segurança	31
sistemas externos	69
parametrização	9

Tabela 15 - Número de requisitos por ligações

Ligações	Requisitos
0	5
1	72
2	57
3	22
4	10
5	2
6	0
7	1

Pode ser observado que muitos requisitos foram relacionados a diversos termos da taxonomia, o que explicita a interdependência entre requisitos. Por exemplo, os requisitos funcionais identificados por RF41 - Cancelar Reserva e RF42 - Confirmar Reserva foram relacionados aos termos **reserva** e **sistemas externos**, mostrando como este método também pode auxiliar na identificação de características entrelaçadas ainda no documento de requisitos. O grande número de requisitos associados ao tema **sistemas externos** explica-se, neste caso, pois a organização já possui um conjunto consolidado de outras aplicações que são reutilizadas para determinadas funcionalidades.

5.1.1.3. Geração e apresentação das visões

A Tabela 16 apresenta uma visão textual de requisitos associados ao tema **reserva** (foram omitidos detalhes não significativos das sentenças). Esta visão foi complementada por visões gráficas obtidas com o mapa de tópicos gerado. Utilizamos os modelos previamente definidos em XTM, sendo gerados tópicos para os itens da taxonomia e para cada um dos requisitos.

Tabela 16 - Visão textual de requisitos associados ao tema reserva (parcial)

reserva	RF31 Manter Viagem de Cliente ...disponibiliza as principais informações da viagem do cliente. A partir desta tela o funcionário pode executar todas as acções relacionadas ao processo de reserva. As principais funcionalidades que poderão ser executadas são: ..... Modificar o funcionário **respons**ável pela **reserv**a (**Deleg**ar **Reserv**a).....
	RF33 Consultar Produtos da Viagem Este caso de uso deve exibir uma lista com os produtos vinculados à viagem, disponibilizando informações básicas de cada um deles, tais como o tipo do produto (Avião, Hotel, etc.), o fornecedor, a situação da reserva, as datas, a descrição, os custos, dentre outras. Um produto pode ter as seguintes situações de reserva: **Aguard**ando **Reserv**a, Aguardando **Confirm**ação de **Reserv**a, **Reserv**a **Confirm**ada, **Reserv**a não **Confirm**ada, **Reserv**a **Cancel**ada.....
	RF34 Efectuar Acção Sobre Produto .... acção sobre o produto da viagem. As acções sobre produtos são as seguintes: * Aguardar Pagamento. Quando o cliente aceita a proposta e **confirm**a a **reserv**a porém ainda não efectua nenhum pagamento, assim é colocado o estado do pedido em aguardando pagamento....
	RF37 Adicionar Produto à Viagem ..... vai ser realizado quando o funcionário seleccionar um **produt**o para **reserv**á-lo. O sistema irá adicionar o produto à Viagem do cliente. ...

Em mapas de tópicos as associações são estabelecidas entre pares de tópicos e, no nosso caso, cada associação relaciona um tema da taxonomia e um requisito relacionado. Para o tema reserva foram geradas 32 associações. A Figura 45 apresenta, à esquerda, a taxonomia de turismo, e à direita, visão gráfica do tema reserva e requisitos relacionados; uma outra visão, apresentada na figura 7, mostra que o RF17, relacionado ao tema pagamento, também está relacionado aos temas viagem e pacote. Para melhorar a visibilidade das figuras, está sendo exibida apenas parte dos requisitos associados aos termos da taxonomia; as figuras foram obtidas com uso do software TMNav, versão 0.2.8 (alpha).

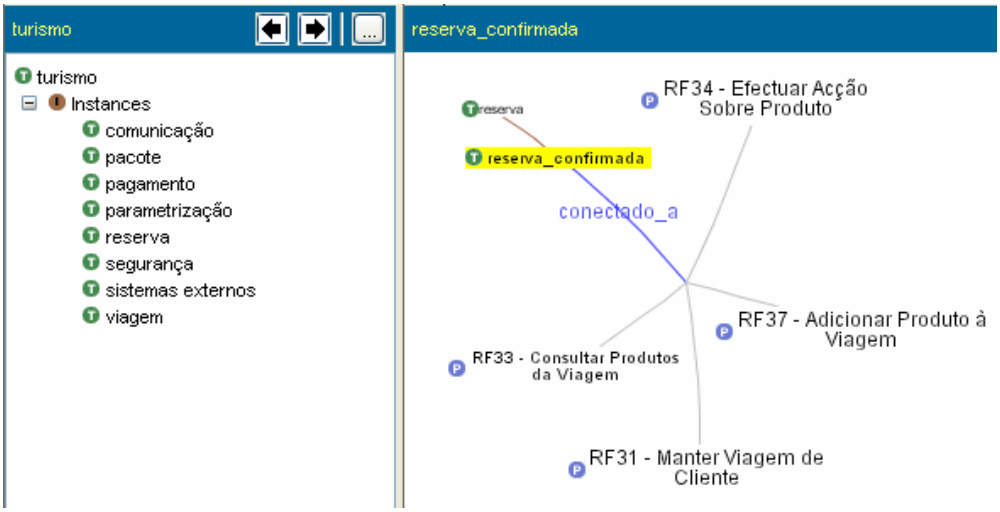


Figura 45 - Visão da taxonomia e do tema reserva e requisitos associados

Os componentes apresentados numa visão estão associados a outros através de *hiperlinks*, o que possibilita, no exemplo da Figura 46, verificar que um requisito associado a um determinado tema participa também em outras associações. Estas diferentes visões auxiliam os interessados a escolher um determinado conjunto de requisitos para focar seu trabalho, e também permitem a visualização das associações existentes entre requisitos e temas.

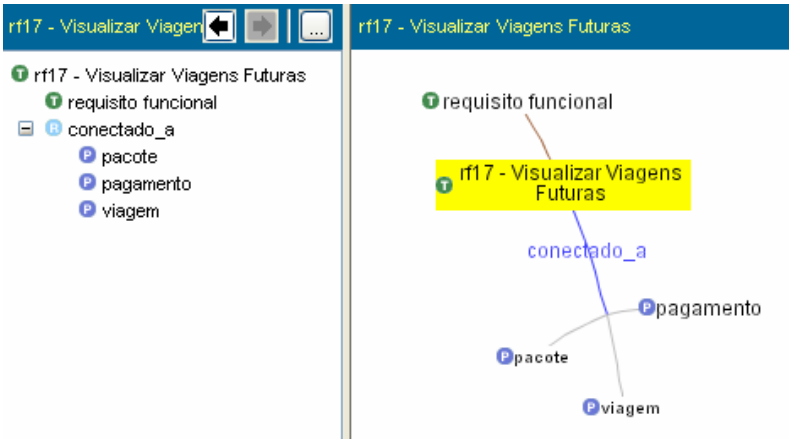


Figura 46 - O requisito funcional 17 e temas associados

5.1.2. Construção do léxico da aplicação

Os resultados obtidos após a aplicação da técnica proposta ao documento Exit são muito interessantes, pois este documento utiliza parcialmente a variante de Portugal e contém, portanto, muitos termos ou expressões que mostram explicitamente as diferenças culturais e lingüísticas entre Portugal e Brasil. No desenvolvimento distribuído isto é de grande importância, pois é largamente sabido na área de requisitos que termos ambíguos podem levar a erros ou inconsistências não só na fase de requisitos, mas em todo o processo de desenvolvimento. Ter estas diferenças documentadas em "dicionários de viagem" ou no léxico das aplicações diminui a probabilidade da ocorrência de erros derivados de interpretação incorreta. A Tabela 17 apresenta as expressões não dicionarizadas detectadas pelo processo definido na Seção 3.2.1.

Tabela 17 - Termos não-dicionarizados do documento Exit (parcial)

call centre	99	star	29	travel	13
tour	65	sibs	25	web	13
actual	63	mbnet	22	actualizar	12
back	58	restel	21	biztalk	12
front end	57	newsletter	20	cofidis	12
office	57	selecciona	19	villas	12
efectuar	50	acções	18	directamente	11
rent-a-car	40	multibanco	18	server	11
ecrã	35	password	18	projecto	10
gds	34	acção	13	secção	10
planeada	34	dossier	13	template	10
galileo	29	facto	13	page	9

Uma análise cuidadosa dos termos obtidos através deste processo mostrou que parte destes correspondem a termos específicos do domínio da aplicação ou de sistemas correlatos a ele e, portanto devem estar necessariamente registrados no léxico da aplicação. Outra parte dos termos extraídos registra claramente as diferenças culturais existentes, mesmo trabalhando numa área técnica como TI. Acreditamos que tais termos devam ser registrados num "dicionário de viagem", que deverá estar disponível para todos os participantes do processo de desenvolvimento, principalmente para desenvolvedores que iniciam o trabalho na organização e não foram conscientizados dessas diferenças.

Como exemplo registramos na Tabela 18 algumas expressões utilizadas em Portugal, e a correspondente no Brasil. Uma delas, especificamente, poderia trazer sérias consequências ao processo de desenvolvimento caso seja compreendida incorretamente, pois a expressão "por defeito" não seria entendida como um padrão a assumir em caso de ausência de informação, mas sim como a forma de tratar um defeito na informação manipulada.

Tabela 18 - Expressões registrando diferenças culturais e lingüísticas

por defeito	por default
aceder	acessar
sinalética	ícone
multibanco	tipo de pagamento
ecrã	monitor ou tela
villas	*resort*
utilizador	usuário
telemóvel	telefone celular
ficheiro	pasta ou diretório de arquivos

A extração dos contextos possibilita ao engenheiro de requisitos a inserção dos símbolos no léxico da aplicação, e com sua descrição e impactos. Como

exemplo, a Tabela 19 apresenta contextos para a palavra *markup*:

Tabela 19 - Um termo e seus contextos

markup	metodologia de markup
	O markup é um valor que deve ser aplicado ao preço de custo de um produto para calcular seu preço de venda. O markup representa a margem de lucro da agência de viagens sobre o produto vendido.
	A manutenção de markup será utilizada no novo sistema apenas para produtos externos
	O valor de markup pode ser em euros ou em percentual.

5.1.3. Detecção de discrepâncias, erros e omissões em requisitos

Ao documento Exit foram aplicadas as técnicas de identificação de similaridade em requisitos, e detecção de discrepâncias, erros e omissões em requisitos ao funcionais.

5.1.3.1. Duplicidade em requisitos

Para a identificação de duplicidade em requisitos foram calculados os valores para a matriz de similaridade, considerando os coeficientes de Dice, Jaccard e coseno entre pares de requisitos. O limiar para a extração de candidatos a duplicidade foi estabelecido empiricamente em 0,90; os resultados são apresentados na Tabela 20, com destaque para os valores iguais ou maiores ao limiar definido.

Tabela 20 - Matriz de similaridade de requisitos: exibição parcial

	RF159	RF16	RF160	RF161	RF162	RF163	RF164	RF165	RF166	RF167	RF168	RF169	RF17	RF170
RF158	0,15	0,06	0,17	0,17	0,14	0,18	0,12	0,12	0,13	0,12	0,12	0,12	0,08	0,06
RF159	1,00	0,05	0,79	0,86	0,73	0,74	0,74	0,72	0,73	0,71	0,70	0,23	0,08	0,16
RF16		1,00	0,06	0,06	0,05	0,05	0,05	0,05	0,05	0,05	0,05	0,09	0,26	0,02
RF160			1,00	0,91	0,76	0,81	0,57	0,61	0,60	0,58	0,57	0,25	0,07	0,13
RF161				1,00	0,80	0,85	0,62	0,62	0,64	0,62	0,60	0,28	0,08	0,13
RF162					1,00	0,74	0,52	0,53	0,56	0,61	0,52	0,27	0,08	0,13
RF163						1,00	0,53	0,55	0,58	0,56	0,60	0,29	0,08	0,12
RF164							1,00	0,94	0,96	0,93	0,92	0,42	0,11	0,11
RF165								1,00	0,96	0,93	0,92	0,44	0,11	0,10
RF166									1,00	0,97	0,93	0,49	0,12	0,10
RF167										1,00	0,90	0,48	0,11	0,10
RF168											1,00	0,42	0,12	0,10
RF169												1,00	0,08	0,06
RF17													1,00	0,05
RF170														1,00

Os requisitos apontados como candidatos a duplicidade estão reproduzidos na Tabela 21; observe que enquanto o par RF160/RF161 pode ser entendido como um resultado falso positivo, o conjunto de requisitos RF164, RF165, RF166, RF167 e RF168 apresentam inconsistências e devem ser revisados pelo engenheiro de requisitos ou até mesmo reescritos.

Tabela 21 - Requisitos candidatos à avaliação de duplicidade

RF160 Realizar Pesquisa por Produto – Hotel Requisito responsável por realizar a pesquisa de hotéis junto aos sistemas externos que provém este serviço, além de realizar a busca em base própria do Novo SISTEMA EXEMPLO. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “RF134 Realizar Busca por Hotéis na Restel”.
RF161 Realizar Pesquisa por Produto – Rent-a-Car Requisito responsável por realizar a pesquisa de rent-a-car junto aos sistemas externos que provém este serviço, além de realizar a busca em base própria do Novo SISTEMA EXEMPLO. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “RF128 Realizar Busca por Produto no GDS”.
RF164 Reservar Produto – Aviação Requisito responsável por realizar a reserva de produtos de aviação junto aos sistemas externos que provém este serviço. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “RF130 Reservar Produto no GDS”.
RF165 Reservar Produto – Hotel Requisito responsável por realizar a reserva de produtos de aviação junto aos sistemas externos que provém este serviço. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “RF136 Reservar Produto na Restel”.
RF166 Reservar Produto – Rent-a-Car Requisito responsável por realizar a reserva de produtos de aviação junto aos sistemas externos que provém este serviço. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “Reservar Produto em Base Própria”.
RF167 Reservar Produto – Fins de Semana Requisito responsável por realizar a reserva de produtos de aviação junto aos sistemas externos que provém este serviço. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “Reservar Produto em Base Própria”.
RF168 Reservar Produto – Pacote de Férias Requisito responsável por realizar a reserva de produtos de aviação junto aos sistemas externos que provém este serviço. O requisito irá utilizar outros casos de uso para realizar a pesquisa como, por exemplo, o requisito “RF139 Reservar Produto no Mundo Vip”.

A Tabela 22 apresenta medidas de avaliação para os resultados obtidos com a aplicação das medidas de similaridade para identificação de duplicidade em requisitos:

Tabela 22 - Valores de recall e precision para duplicidade

	ec1: exit
Recall	100%
precision	72%

5.1.3.2. Omissões em requisitos

Para identificar omissões em documentos de requisitos inicialmente trabalhamos na construção do dicionário de RNF´s, e para esta tarefa utilizamos o conjunto de documentos identificados por Exit, Escalas, E-commerce/vendas e E-commerce/entrega. Parte destes registra os requisitos em forma de sentenças, parte utiliza o modelo de casos de uso; todos estes documentos são oriundos da mesma organização. Buscamos identificar de forma semi-automatizada termos e expressões característicos de requisitos não funcionais. Geramos a matriz termo-documento para identificar termos comuns ao conjunto de documentos, e esta foi analisada manualmente para a extração de termos característicos a requisitos não funcionais. Estes termos foram inseridos no dicionário de RNF´s utilizado para o processo de identificação de omissões em requisitos. Na inserção desses termos, buscamos colocá-lo junto ao conceito que melhor representasse o requisito não funcional que o termo expressava.

5.1.3.2.1. Instanciar dicionário de RNF´s para a organização

O dicionário instanciado não pretende ser um catálogo de requisitos não funcionais. Como foi construído a partir de documentos da própria organização, ele pode ser inadequado se utilizado num outro contexto. Procuramos evitar lacunas, solicitando à equipe de desenvolvimento da organização que revisasse a estrutura criada, completando lacunas existentes. A Figura 47 apresenta a estrutura utilizada para a verificação de omissões em requisitos do tipo RNF´s, resultado desse trabalho. O conjunto de categorias está relacionado no anexo D.

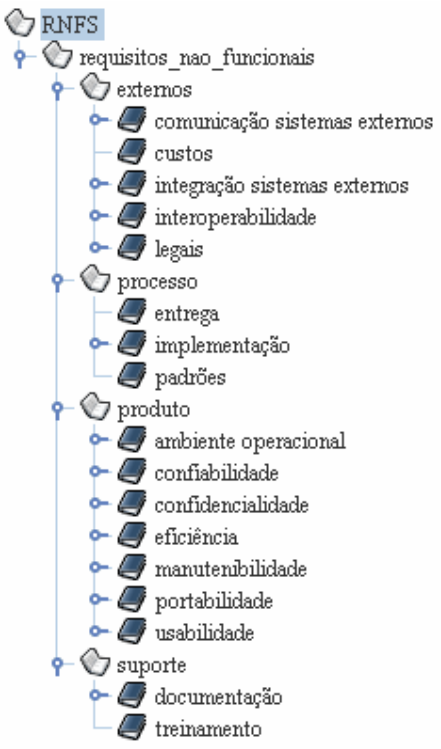


Figura 47 - Estrutura do dicionário de RNF´s para identificação de omissões

5.1.3.2.2. Executar análise de conteúdo para RNF´s

A análise foi aplicada ao documento, e os resultados obtidos são visualizados na Figura 48.

RNFS - SISTEMA EXIT	
TOTAL DE REFERENCIAS A RNFS	323
> externos	280
> externos > comunicação sistemas externos	11
> externos > custos	0
> externos > integração sistemas externos	190
> externos > interoperabilidade	79
> externos > legais	0
> processo	16
> processo > entrega	0
> processo > implementação	16
> processo > implementação > linguagens	16
> processo > padrões	0
> produto	27
> produto > ambiente operacional	1
> produto > confiabilidade	0
> produto > confiabilidade > integridade dos dados	0
> produto > confiabilidade > segurança	0
> produto > confiabilidade > tolerância a falhas	0
> produto > confidencialidade	25
> produto > confidencialidade > cadastrar usuários	0
> produto > confidencialidade > conectar/autenticar usuários	22
> produto > confidencialidade > desconectar	3
> produto > eficiência	0
> produto > eficiência > desempenho	0
> produto > eficiência > recursos	0
> produto > manutenibilidade	0
> produto > manutenibilidade > modifiabilidade	0
> produto > manutenibilidade > testabilidade	0
> produto > portabilidade	0
> produto > portabilidade > adaptabilidade	0
> produto > usabilidade	1
> produto > usabilidade > flexibilidade	0
> produto > usabilidade > operabilidade	0
> produto > usabilidade > web	1
> suporte	0
> suporte > documentação	0
> suporte > treinamento	0

Figura 48 - Resultados para o documento *exit* - Sistema Exit

A análise do relatório gerado a partir das informações registradas no dicionário de RNF´s para o Sistema Exit indica que devem ser verificadas as seguintes discrepâncias e/ou omissões:

- foram detectados 16 referências a expressões relacionadas à linguagem na categoria processo, mas não houveram referências a outras categorias relacionadas a processo. Deve ser verificada a ausência de referencias à utilização de padrões no processo de desenvolvimento;
- há um número significativo de referências a sistemas externos - 87% - indicando intensa troca de informações entre o sistema em desenvolvimento e outros. Deve ser verificado o impacto em relação ao desempenho esperado do sistema;
- não há referências explícitas a RNF´s de documentação. Novamente, este aspecto deverá ser reavaliado, pois trata-se de empresa certificada por modelos de qualidade;

- há apenas uma referência a expressões relacionadas à usabilidade: verificar este aspecto no documento de requisitos.

5.2. Estudo de caso: documento Escalas

Este documento, denominado de Escalas, é um documento preliminar de requisitos para uma aplicação na área de gerenciamento de recursos humanos, especificamente para gestão de escalas de trabalho. O documento segue padrões internos à organização, modelando requisitos com casos de uso e especificações suplementares. O total de casos de uso, neste documento, é de sessenta, com cinco especificações suplementares (estas relacionadas a características como documentação, treinamento e usabilidade). O documento está escrito na língua portuguesa, apresentando diversos termos na variante de Portugal (por exemplo, os termos *usuário* e *utilizador* são usados indiscriminadamente para fazer referência aos usuários do sistema). O total de palavras, neste segundo documento, é de seis mil, seiscentos e sessenta e cinco palavras. Documento cedido pela Tlantic Sistemas de Informação.

5.2.1. Construção do léxico da aplicação: termos ou expressões não dicionarizadas

A Tabela 23 apresenta as expressões não dicionarizadas detectadas pelo processo definido no capítulo 3, seção 3.2.1.

Tabela 23 - Expressões não-dicionarizadas

termo	freq.	termo	freq.
Sap	22	seebeyond	2
Dw	19	buffer	1
Log	10	checklist	1
Job	7	datawarehouse	1
Batch	4	eai	1
Login	4	logs	1
Dop	2	shell	1
help-desk	2	sos	1
Retek	2	tng	1
sapxdw	2	urps	1

A análise dos resultados deste estudo de caso mostrou a utilização de várias expressões do vocabulário de TI; tais termos, embora não sejam específicos do domínio da aplicação, devem ser evitados no registro dos requisitos, ou então colocados num dicionário à parte, pois não se pode exigir que eles sejam conhecidos e compreendidos corretamente pelos usuários da aplicação. Exemplos de tais termos são: *log*, *job*, *batch*, *buffer*. Também identificamos termos em comum com o estudo de caso anterior, pois os dois documentos são oriundos da mesma organização. No caso, parte desses termos refere-se a siglas de sistemas já existentes; haverá troca de informações entre eles e o sistema em desenvolvimento. Consideramos portanto que deverá ser avaliada a possibilidade da criação de um léxico para o domínio da organização, que será atualizado a cada nova aplicação. Exemplos de termos deste tipo são: SAP, RETEK, DW e SeeBeyond.

Neste segundo estudo de caso identificamos também que palavras com baixa freqüência não devem ser desconsideradas, pois podem representar termos específicos do domínio da aplicação.

5.2.2. Construção do léxico da aplicação: extração de sujeitos e objetos

Este estudo de caso utilizou o documento Escalas. Foram extraídos quarenta e um sintagmas nominais, conforme padrões definidos na seção 3.2.2.2. Esses sintagmas indicam termos candidatos a símbolos do tipo sujeito, e após o processo de seleção que exclui aqueles com freqüência menor que quatro ou aqueles relacionados numa *stoplist* obtivemos vinte e três símbolos do tipo sujeito. De maneira análoga, foram extraídos oitocentos e quarenta e sete sintagmas que, após o processo de seleção, resultaram em apenas trinta e dois candidatos a símbolos do tipo objeto. A Tabela 24 apresenta parte do conjunto de sujeitos e objetos selecionados, e a Tabela 25 apresenta os contextos identificados para alguns dos símbolos candidatos extraídos do documento.

Tabela 24 - Símbolos do tipo sujeito e objeto para o sistema de gestão de escalas

Sujeito	Padrão	Freq.
gestor	N*or, N*ores	101
usuário	N*ário	88
gestor de escalas	N*or PRP* N*	68
usuário final	N*ário N*	58
funcionário	N*ário, N*ários	39
administrador	N*or	28
colaborador	N*or, N*ores	22
administrador central	N*or N*	12
leitor	N*or	8
empresa	N*esa	7
administrador local	N*or N*	7

Objeto	Padrão	Freq.
banco de dados	N* PRP* N*	11
cálculo de estimativas	N* PRP* N*	10
sistema operacional	N* N*	8
postos de serviços	N* PRP* N*	7
parâmetros de pesquisa	N* PRP* N*	7
solicitações de processamento	N* PRP* N*	6
lista de postos de serviços	N* PRP* N* PRP* N*	6
sistema administrador	N* N*	5
relatório de ocorrencias	N* PRP* N*	5
log das ocorrencias	N* CPR* N*	5
habilita relatório	N* N*	5

Tabela 25 - Contextos para símbolos do sistema de gestão de escalas

administrador local	**Administrador Local: r**esponsável em inserir os parâmetros específicos de sua unidade. Também é responsável pela atribuição de perfis aos funcionários da unidade: Leitor Local e Gestor de Escalas
	O usuário com este perfil será o responsável em designar o **administrador local** do Gestão de Escalas nas lojas. Este usuário também deverá elencar quais são os funcionários que possuirão o perfil de Leitor Central
	Administrador Local solicita a tela de associação
	Administrador Local seleciona o perfil desejado, informa a funcionalidade e as operações que deseja habilitar.
	Este caso de uso permitirá que o **administrador local** ou central designe senhas de acesso a utilizadores do sistema, como gestores, leitores, etc. Apenas usuários que possuem senhas podem acessar o sistema.

5.2.3. Omissões em requisitos: análise de conteúdo para RNF´s

A análise foi aplicada aos cinco documentos de requisitos referidos no início deste capítulo, e os resultados estão apresentados a seguir. A Figura 49 apresenta os resultados obtidos com o documento *escalas*, referido no início deste capítulo.

RNFS - SISTEMA GESTOR DE ESCALAS	
TOTAL DE REFERENCIAS A RNFS	67
> externos	42
> externos > comunicação sistemas externos	0
> externos > custos	0
> externos > integração sistemas externos	22
> externos > interoperabilidade	19
> externos > legais	1
> processo	0
> processo > entrega	0
> processo > implementação	0
> processo > implementação > linguagens	0
> processo > padrões	0
> produto	25
> produto > ambiente operacional	0
> produto > confiabilidade	10
> produto > confiabilidade > integridade dos dados	0
> produto > confiabilidade > segurança	10
> produto > confiabilidade > tolerância a falhas	0
> produto > confidencialidade	15
> produto > confidencialidade > cadastrar usuários	0
> produto > confidencialidade > conectar e autenticar usuários	15
> produto > confidencialidade > desconectar	0
> produto > eficiência	0
> produto > eficiência > desempenho	0
> produto > eficiência > recursos	0
> produto > manutenibilidade	0
> produto > manutenibilidade > modifiabilidade	0
> produto > manutenibilidade > testabilidade	0
> produto > portabilidade	0
> produto > portabilidade > adaptabilidade	0
> produto > usabilidade	0
> produto > usabilidade > flexibilidade	0
> produto > usabilidade > operabilidade	0
> produto > usabilidade > web	0
> suporte	0
> suporte > documentação	0
> suporte > treinamento	0

Figura 49 - Resultados para o documento escalas - Sistema Gestor de Escalas

A análise do relatório gerado a partir das informações registradas no dicionário de RNF´s para o Sistema Gestor de Escalas aponta para as seguintes discrepâncias e/ou omissões:

- não foram detectados requisitos relacionados ao processo de desenvolvimento. Como a organização é certificada pelo CMMI nível 2, requisitos relacionados à utilização de padrões no processo de desenvolvimento deveriam ter sido registrados;
- há um número significativo de referências a sistemas externos, indicando troca de informações entre o sistema em desenvolvimento e outros já existentes. Como este é um fator que muito provavelmente terá impactos no desempenho do sistema, esperava-se que houvessem referências a RNF´s de desempenho, o que não aconteceu;
- não há referências explícitas a RNF´s de documentação: não estão

previstas documentação para usuários e documentação técnica que possa servir como apoio a processos futuros de evolução. Novamente, este aspecto deverá ser reavaliado, pois trata-se de empresa certificada por modelos de qualidade;

- há inconsistências relacionadas aos requisitos de confidencialidade: enquanto existem quinze referências para o item de conexão e autenticação de usuários, não existem referências para desconexão ou cadastramento de usuários do sistema.

5.3. Estudo de caso: documentos E-commerce/entrega e E-commerce/vendas

O documento denominado de E-commerce/entrega é um documento de requisitos para um sistema de controle de entregas de mercadorias vendidas *on line*. Em relação à linguagem utilizada e à representação de requisitos, o documento possui as mesmas características do documento *Escalas*, já referido na seção anterior. O total de casos de uso, neste documento, é de vinte e seis, com cinco especificações suplementares relacionadas a requisitos não funcionais. O total de palavras, neste segundo documento, é de quatro mil, quinhentos e trinta e nove. O documento *E-commerce/vendas* também é caracterizado como documento preliminar de requisitos para um sistema de vendas *on line*. Parte desses casos de uso está descrita de forma sucinta, em poucas linhas, para posterior detalhamento. Em relação à linguagem utilizada e à representação de requisitos, ele possui as mesmas características do documento Escalas. Neste documento, o total de casos de uso, é de duzentos e seis, com sete especificações suplementares relacionadas a requisitos não funcionais. O total de palavras, neste segundo documento, é de trinta e dois mil, trezentos e trinta e oito palavras. Ambos os documentos foram cedidos pela Tlantic Sistemas de Informação

Este experimento foi realizado visando à utilização do dicionário de categorias como base para comparação de documentos. Isto pode ser útil quando a organização dispõe de um documento considerado padrão em relação a RNF's, num mesmo domínio de aplicação. Os resultados obtidos podem ser observados na Figura 50.

No caso, o documento identificado por *E-commerce/entrega* não havia

nenhuma referência a RNF's (ambos os documentos pertencem a um mesmo domínio de aplicação). Este fato é parcialmente explicável, pois se trata de um documento extenso, que nos foi fornecido imediatamente após a primeira versão ser liberada. O experimento foi prejudicado, pois buscávamos justamente uma comparação que nos permitisse concluir pela adequação da abordagem comparativa. Resultados como estes também podem apontar para uma necessidade de revisão das expressões no dicionário de RNF's, pois talvez o resultado obtido seja na verdade um indicador de ausência de expressões nas categorias registradas.

COMPARANDO DOCUMENTOS (1) *e-commerce/vendas* e (2) *e-commerce/entrega*

	(1)	(2)
TOTAL DE REFERENCIAS A RNFS	138	0
	138	0
> externos	95	0
> externos > comunicação sistemas externos	1	0
> externos > custos	0	0
> externos > integração sistemas externos	0	0
> externos > interoperabilidade	93	0
> externos > legais	1	0
> processo	0	0
> processo > entrega	0	0
> processo > implementação	0	0
> processo > implementação > linguagens	0	0
> processo > padrões	0	0
> produto	43	0
> produto > ambiente operacional	1	0
> produto > confiabilidade	3	0
> produto > confiabilidade > integridade dos dados	0	0
> produto > confiabilidade > segurança	3	0
> produto > confiabilidade > tolerância a falhas	0	0
> produto > confidencialidade	39	0
> produto > confidencialidade > cadastrar usuários	0	0
> produto > confidencialidade > conectar/autenticar usuários	38	0
> produto > confidencialidade > desconectar	1	0
> produto > eficiência	0	0
> produto > eficiência > desempenho	0	0
> produto > eficiência > recursos	0	0
> produto > manutenibilidade	0	0
> produto > manutenibilidade > modifiabilidade	0	0
> produto > manutenibilidade > testabilidade	0	0
> produto > portabilidade	0	0
> produto > portabilidade > adaptabilidade	0	0
> produto > usabilidade	0	0
> produto > usabilidade > flexibilidade	0	0
> produto > usabilidade > operabilidade	0	0
> produto > usabilidade > web	0	0
> suporte	0	0
> suporte > documentação	0	0
> suporte > treinamento	0	0

Figura 50 - Comparando documentos através de RNF´s

5.4. Estudo de caso: documento Selic

Este estudo de caso utilizou documentos públicos relacionados ao sistema SELIC, do Banco Central do Brasil. Estes documentos compõem a documentação

dos usuários do sistema, e são estruturados em conjuntos de funcionalidades relacionadas. Utilizamos parte desses documentos, num total de quatro documentos e cinco mil, quatrocentos e sessenta e uma palavras. Os documentos estão disponíveis para acesso público no endereço http://www.bcb.gov.br/htms/spb/. Os documentos seguem uma estrutura padrão para o sistema SELIC; o total de palavras da porção de documentos avaliados é 5.461. A Tabela 26 apresenta resultados parciais obtidos após a aplicação do processo proposto.

Tabela 26- Símbolos do tipo sujeito e objeto para o sistema Selic

Sujeito	Padrão	Freq.	Objeto	Padrão	Freq.
cedente	N*ente	17	operação compromissada	N* PART*	35
clientes	N*entes	17	retorno de operação	N* PRP* N*	17
cessionário	N*ário	15	conta reservas	N* N*	16
correspondente	N*ente	10	pu de retorno	N* PRP* N*	16
instituição financeira	N* ADJ*	9	número de operação	N* PRP* N*	10
usuário	N*ário	9	reservas bancárias	N* N*	10
cedente da operação	N*ente CPR* N*	6	situação da operação	N* CPR* N*	9
cessionário da operação	N*ário CPR* N*	6	conta da instituição	N* CPR* N*	7
destinatário	N*ário	6	operação 1054/s	N* N*	7
emissor	N*or	6	pu da operação	N* CPR* N*	7
instituição liquidante	N*ão N*	6	endereço eletrônico	N* ADJ*	6

Contextos para o símbolo *conta da instituição*

- As pontas da operação (comprador e vendedor final) lançam a 1056/SEL1056 tipo 1 utilizando como contraparte a conta da instituição intermediária, e o sistema promove a atualização da operação.
- LIQ CED IFLiqdantCed Código da conta da instituição liquidante do cedente. Esse campo não deve ser preenchido se a liquidação for efetuada pelo próprio liquidante-padrão.
- LIQ CES IFLiqdantCes Código da conta da instituição liquidante do cessionário. Esse campo não deve ser preenchido se a liquidação for efetuada pelo próprio liquidante-padrão.

5.5. Avaliação dos resultados preliminares: construção do léxico

As taxas de *recall* e *precision* dos resultados do processo de extração de símbolos para o léxico estão sumarizadas na Tabela 27. As avaliações foram executadas por equipes independentes, uma delas composta por especialistas do Banco Central (estudo de caso com o documento Selic) e a outra por engenheiros

de software (estudo de caso com o documento Escalas).

Os resultados obtidos mostram a adequação da proposta para a construção do léxico para aplicações, e estamos trabalhando nos ajustes necessários para melhorar os resultados já obtidos. O algoritmo de extração de sintagmas nominais está em processo de revisão de forma a evitar que sintagmas que atendem a mais de um padrão sejam recuperados em duplicidade. Algumas modificações nos padrões utilizados estão sendo experimentadas, visando à obtenção de melhores taxas de *recall* e *precision* em relação aos objetos recuperados. Também observamos alguns problemas decorrentes da inserção de etiquetas incorretas, principalmente em relação a objetos, pois estes utilizam padrões mais gerais que aqueles utilizados para sujeitos.

Tabela 27 - Valores de recall e precision dos símbolos extraídos

	ec1: suj	**ec1: obj**	**ec2: suj**	**ec2: obj**
recall	92%	100%	78%	78%
precision	75%	78%	84%	56%

6
Conclusões

Nesta tese apresentamos uma abordagem para automação parcial de atividades associadas à verificação e validação de requisitos. Neste capítulo estruturamos as seções de forma a apresentar na seção 6.1 um resumo das principais características da abordagem proposta, contribuições desta tese e aspectos da ferramenta para suporte à abordagem. Na seção 6.2 comparamos nosso trabalho com outros de alguma forma relacionados, e discutidos limitações e restrições na seção 6.3, onde também relacionamos trabalhos futuros.

6.1.
Abordagem proposta e contribuições

A estratégia proposta trabalha com requisitos escritos em linguagem natural, dado que isto propicia maior envolvimento de clientes e usuários no processo de requisitos como um todo, e especificamente nas atividades de verificação e validação. As estratégias propostas consideram também aspectos próprios do desenvolvimento distribuído de software, com interessados desempenhando atividades em ambientes geograficamente separados.

A abordagem proposta está baseada em:

- uso de técnicas de tratamento da linguagem natural para automação parcial de atividades de verificação e validação
- uso de agentes de software encapsulando ferramentas utilizadas no tratamento da linguagem natural como *Web services*
- agentes de software sendo utilizados como agentes pessoais, agindo de forma autônoma e apoiando atividades necessárias a verificação e validação de requisitos

As contribuições desta tese são oriundas de duas vertentes de trabalho: (i) uso intensivo de técnicas de processamento da linguagem natural para avaliação de documentos gerados ou manipulados no processo de requisitos, e (ii) uso de

agentes de software e *web services* para possibilitar o reuso de ferramentas disponibilizadas pela comunidade de processamento da linguagem natural e também para possibilitar que algumas ações necessárias ao processo de V&V sejam realizadas de forma autônoma.

Com uso de técnicas de processamento da linguagem natural criamos procedimentos para geração de visões de requisitos, criação ou atualização do léxico da aplicação e detecção de erros, omissões e discrepâncias no conjunto de requisitos.

Com agentes de software desenvolvemos uma estratégia para uso de agentes pessoais dos participantes do processo de requisitos, permitindo que ajam de forma autônoma em nome do ator humano que representam. A estrutura criada possibilita também que mudanças em requisitos sejam detectadas pelos agentes participantes da plataforma, que podem então tomar decisões ou notificar o ator a quem estão associados.

6.1.1. Estratégias com uso de técnicas de processamento da linguagem natural

A **geração de visões de requisitos** (seção 3.1) está intimamente relacionada ao agrupamento de requisitos. Utilizando uma estratégia para extração de termos relevantes do próprio documento de requisitos, uma taxonomia em dois níveis é criada e utilizada para categorizar os requisitos, gerando agrupamentos onde os elementos possuem características comuns. A criação da taxonomia é semi-automática e está baseada em técnicas de extração de terminologia e na identificação de colocações.

Visões textuais e gráficas dos requisitos são geradas a partir dos agrupamentos, possibilitando aos participantes das atividades de V&V o trabalho sobre grupos menores e possivelmente menos complexos de requisitos. Para a geração de visões gráficas utilizamos mapas de tópicos, o que permite aos usuários a navegação e rápida percepção de relacionamentos entre requisitos.

Também possibilitamos que os usuários solicitem agrupamentos de requisitos simplesmente fornecendo um termo ou expressão relacionada ao seu trabalho. Desta forma ele pode criar agrupamentos de acordo com seu interesse ou perfil de atuação profissional e avaliar tais agrupamentos em relação a

características como completude, por exemplo.

O objetivo na criação dos agrupamentos é apoiar atividades de V&V, mas tais agrupamentos também podem ser utilizados para, por exemplo, definir alocação de requisitos a componentes de software ou mesmo atribuir agrupamentos a incrementos de software, no caso de se utilizar o desenvolvimento incremental de software.

A **criação ou atualização automática do léxico da aplicação** (seção 3.2) está baseada na identificação de termos não dicionarizados e na identificação de atores e recursos referidos nos documentos.

A identificação de termos não dicionarizados baseia-se na premissa que termos ausentes de dicionários usuais da língua fazem referência a termos próprios do domínio da aplicação ou da própria área de sistemas de informação. Tais termos, por possibilitarem diferentes interpretações, devem constar do léxico da aplicação. Esta estratégia pode ser utilizada também para apoiar a criação de um léxico para a organização, se aplicada a conjuntos de documentos relacionados aos negócios da organização.

A necessidade de léxicos abrangentes e contextualizados se faz mais presente em ambientes distribuídos de desenvolvimento, nos quais problemas de comunicação gerados por diferenças culturais, fusos horários e mesmo diferentes competências lingüísticas podem dificultar a realização de atividades do processo de requisitos. O léxico é uma ferramenta extremamente necessária para evitar ambigüidades dos termos próprios ao domínio da aplicação, considerando os problemas que poderiam decorrer de uma interpretação equivocada dos requisitos.

A identificação de atores e recursos para comporem o léxico da aplicação está fundamentada extração de sintagmas nominais do texto de documentos gerados ou manipulados pelo processo de requisitos. A estratégia proposta é baseada na utilização de um *POS tagger*, que identifica as classes gramaticais das palavras do texto, e na extração de sintagmas nominais que atendem a um conjunto de padrões pré-definidos.

Os padrões utilizados para a identificação de atores basearam-se na premissa que atores são entidades ativas, e que em documentos técnicos podem ser referidos por profissões ou papéis que desempenham. Os padrões foram definidos tendo por base um conjunto de terminações da língua portuguesa, terminações essas que referenciam funções ou profissões. Após um conjunto de experimentos,

identificamos um total de 82 padrões que remetem a substantivos (componentes de sintagmas nominais) com terminações específicas. Para a identificação de recursos, utilizamos um conjunto menor de padrões, apenas 9. O conjunto de padrões para recursos é menor, pois tais padrões são mais gerais que aqueles utilizados na identificação de atores.

A **detecção de erros, discrepâncias e omissões em requisitos** (seção 3.3) é realizada através da identificação de requisitos candidatos à duplicidade e de identificação de omissões em requisitos não funcionais.

Utilizamos medidas de similaridade entre documentos para identificar pares de requisitos candidatos à duplicidade. Utilizamos um índice único para a identificação da similaridade, dado pela média aritmética dos índices do coseno, Jaccard e Dice. Estas métricas são usuais na área de processamento de linguagem natural para identificar documentos de conteúdo similar. Se o índice obtido for maior que o limiar definido em 0,90, o par de requisitos é indicado como candidato à duplicidade e deverá ser analisado pelos participantes do processo de verificação. O limiar foi definido empiricamente após vários experimentos com documentos de requisitos.

Para a identificação de omissões restringimos nossa área de atuação aos requisitos não funcionais. Utilizando técnicas de análise de conteúdo, criamos um dicionário de categorias as quais correspondem diretamente a requisitos não funcionais extraídos de catálogos públicos. A cada uma dessas categorias é associado um conjunto de termos usualmente utilizados na organização para fazer referencias àquele requisito não funcional. O documento de requisitos é então analisado em relação ao dicionário de requisitos não funcionais e termos associados, sendo geradas tabelas de freqüência de termos. Essas tabelas são consolidadas nas categorias do dicionário de RNF's, possibilitando ao engenheiro de requisitos uma análise de omissões em relação aos RNF's que seriam desejáveis ou essenciais para a aplicação em pauta.

Para a análise de omissões também pode ser utilizada a comparação de dois documentos em relação ao dicionário de RNF's. Se a organização dispõe de um documento de requisitos que possa ser considerado padrão num determinado domínio de aplicação, então este documento pode ser utilizado como base para comparação da presença ou ausência de RNF's no documento sendo avaliado.

6.1.2. Estratégias com uso de agentes de software

Utilizamos agentes de software como agentes pessoais de representantes de participantes do processo de desenvolvimento, como observadores do ambiente de execução para detectar eventos relevantes, como comunicadores e como encapsuladores de serviços via ferramentas criadas em diferentes linguagens de programação.

Agentes pessoais são especializados e representam participantes com um perfil específico de atuação. Tais agentes são autônomos e, ao detectar ações relevantes por parte do humano que representam, executam ações de forma autônoma em nome deste humano. Esse é o caso do agente *Manager*, que representa o gerente do projeto e, uma vez definida a data para a verificação dos requisitos, modifica o estado do documento de requisitos, executa os procedimentos de pré-verificação e notifica os humanos envolvidos.

Como observador do ambiente de execução, o agente *Observador* identifica quando ocorrem eventos de modificação na base de requisitos e propaga essa informação para os demais agentes através do *blackboard*. Os agentes interessados em tal evento podem, então, tomar decisões baseados nessa informação. A propagação automática da ocorrência de eventos como este é fundamental para ambientes distribuídos de desenvolvimento, para garantir que toda a equipe seja informada de modificações em requisitos e desenvolva seu trabalho a partir de uma mesma versão.

No ambiente de execução, agentes e humanos colaboram e trocam informações. O agente *Comunicador* é responsável pelo envio de mensagens aos humanos, quando solicitado pelos demais agentes. Já a comunicação entre agentes se faz através do ambiente de execução, via *blackboard*.

Agentes também oferecem serviços a outros agentes ou humanos através de *Web services*, e no ambiente de execução o agente *Léxico* é um representante típico deste papel. Os serviços oferecidos por este agente incorporam ferramentas de processamento da linguagem natural, parte das quais disponibilizadas livremente pela comunidade da área. Quando solicitado por outros o agente *Léxico* executa o serviço solicitado. O uso de *Web services* possibilitou a incorporação rápida de várias ferramentas de processamento da linguagem natural

que, em outras situações, necessitariam ser desenvolvidas ou obrigariam a um trabalho de adaptação.

Na plataforma de agentes utilizada nesta tese encontramos as propriedades de autonomia, reatividade e colaboração. A plataforma utilizada oferece flexibilidade para gerenciamento de agentes, possibilitando dinamicamente a remoção ou inserção de agentes no ambiente de execução. Esta característica possibilita que agentes pessoais representem fielmente o conjunto de participantes no processo de desenvolvimento - os interessados. Agentes possuem continuidade temporal, monitorando e notificando de forma imediata a ocorrência de eventos relevantes aos interessados.

6.2. Comparação com trabalhos relacionados

Não encontramos, na literatura pesquisada, uma proposta para apoio às atividades de verificação e validação de requisitos que utilizasse de forma tão intensiva quanto a estratégia proposta neste trabalho técnicas de processamento da linguagem natural para apoio às atividades de verificação e validação de requisitos. O uso de agentes de software como apoio às atividades do próprio processo de desenvolvimento já foi abordado em diversos trabalhos. A comparação com trabalhos relacionados será feita de forma pontual, efetuando comparações em diferentes aspectos do nosso trabalho.

Agrupamentos de requisitos: na extensa pesquisa que fizemos encontramos poucos trabalhos abordando este tópico. A proposta descrita em [Hsia92] define incrementos apoiada na identificação de tipos de dados abstratos e numa técnica para agrupar requisitos que modifiquem ou utilizem um mesmo tipo abstrato de dados. O objetivo da abordagem é identificar requisitos que, agrupados, sejam alocados a um mesmo incremento; a técnica é proposta para sistemas com ênfase em manter a integridade dos dados, por exemplo um sistema para automação de bibliotecas. Esta técnica para agrupar requisitos exige que no momento do registro dos requisitos, sejam definidos os tipos abstratos de dados que irão ser utilizados, trazendo para a fase de requisitos uma definição que normalmente seria realizada numa etapa posterior do processo de desenvolvimento.

O trabalho pioneiro de Palmer e Liang [Palmer92] sobre indexação e agrupamento de requisitos defende a idéia que o agrupamento de requisitos agiliza a detecção de conflitos, incompletude, inconsistências e imprecisão internamente aos agrupamentos, e externamente entre os agrupamentos. Palmer propõe a classificação de requisitos em dois níveis de hierarquia: os requisitos são inicialmente agrupados num nível mais alto tendo por base os verbos indicativos de funcionalidades e um thesaurus de verbos construído previamente. Os grupos assim considerados são então subdivididos considerando-se uma medida de similaridade entre documentos, o coseno.

O trabalho mais recente descrito em [Chen05] apresenta uma abordagem baseada no agrupamento de requisitos para construção de modelos de *features.* Relações de dependência entre dois requisitos são identificadas se ambos acessam um mesmo recurso; são identificados 5 tipos básicos de relações de dependência, e a cada tipo é atribuído um peso. É construído um grafo no qual requisitos representam os nodos e relações de dependência são representadas por arcos. Os agrupamentos de requisitos são derivados desses grafos e gerados considerando os pesos atribuídos a cada relação de dependência. Nesta abordagem, requisitos podem estar presentes em mais de um agrupamento.

Nossa proposta para agrupamento de requisitos tem os mesmos objetivos da idéia de Palmer e Liang; também utilizamos uma taxonomia estruturada em dois níveis para o agrupamento dos requisitos. A taxonomia utilizada, no nosso caso, é construída a partir do próprio documento de requisitos. Os temas do primeiro nível são identificados através de medidas que tem por base a freqüência dos termos no documento, e os do segundo nível são obtidos através do uso de colocações, ou palavras que co-ocorrem no texto com uma freqüência maior que o esperado pelo acaso.

Geração de visões de requisitos: a literatura aponta poucas propostas com aplicação de técnicas de visualização a requisitos.

Na área de aspectos alguns trabalhos guardam semelhanças com o nosso: em [Baniassad04] é apresentada uma técnica para identificação e análise de aspectos em documentos de requisitos. Visões são construídas expondo comportamentos do sistema, através de análise léxica. A estratégia proposta por [Silva06] para integração de características transversais utiliza visualização para mostrar relacionamentos transversais entre requisitos.

Recentemente a comunidade de Engenharia de Requisitos mostrou interesse na aplicação de técnicas de visualização para requisitos: a 14th IEEE International Requirements Engineering Conference (2006) colocou, como um dos co-eventos, um workshop destinado especificamente a tratar do tema visualização em requisitos - o International Workshop on Requirements Engineering Visualization. Isto indica o grau de interesse atual da comunidade em trabalhos orientados à geração de visualizações em requisitos.

Um dos trabalhos apresentados [Ozkaya06] nesse evento trata do tema visualização de relacionamentos entre requisitos de forma geral. O autor registra que as ferramentas disponíveis para a fase de requisitos fazem pouco ou nenhum uso da visualização, e faz uma rápida avaliação de sua experiência no desenvolvimento de uma ferramenta para registro e visualização da rastreabilidade de requisitos. Infelizmente não são apresentados detalhes ou mesmo imagens geradas no decorrer do seu trabalho.

Criação do léxico da aplicação: parte da nossa estratégia para construção do léxico da aplicação utiliza sintagmas nominais, com vários trabalhos relacionados na literatura. Já para a estratégia que utilizamos para extrair termos próprios do domínio da aplicação, mesmo sendo uma estratégia simples baseada na comparação com dicionários da língua, não encontramos trabalhos relacionados.

No contexto de processamento de linguagem natural e recuperação de informação, o uso de sintagmas nominais para recuperação de informações é encontrado em muitos trabalhos. [Parreiras03] propõe seu uso em indexação de textos científicos e [Pérez03] os utiliza para a obtenção de conceitos que irão compor mapas conceituais. No contexto do processo de requisitos, trabalhos orientados à exploração de aspectos da linguagem natural ainda não são muito freqüentes.

Em [Harmain00] é descrito o CM-Builder, uma ferramenta *case* que avalia documentos de requisitos escritos em linguagem natural (língua inglesa) buscando relações semânticas nas sentenças de requisitos, visando à construção de um modelo inicial de classes em UML através da identificação de sintagmas nominais. O resultado inclui classes, atributos e relacionamentos, agrupados em modelos de classes e armazenados em arquivos cujo formato é adequado à manipulação posterior por ferramentas normalmente utilizadas nas etapas de

projeto do sistema. Os autores enfatizam que os resultados devem ser vistos como apoio ao trabalho do analista ou engenheiro de software, devendo ser refinados para efetivamente contribuir para um modelo inicial de classes.

A abordagem proposta por Boyd et al [Boyd05] utiliza um subconjunto controlado da linguagem natural para o registro de requisitos, com o objetivo de reduzir a ambigüidade. Esse trabalho investiga a expressividade sintática e semântica de um sub-conjunto da língua inglesa para uso em documentos de requisitos, focando especificamente nos verbos (ações). A identificação das entidades (sujeitos das ações) é realizada via sintagmas nominais.

Nosso trabalho aproxima-se do trabalho de [Harmain00], já referido, mas nossos objetivos são diferentes: visamos apoiar o processo de requisitos, enquanto eles buscam apoiar o processo de projeto do sistema. Utilizamos como fontes de informação não só documentos de requisitos, mas também outros documentos que sejam manipulados ou gerados no processo de requisitos – basta que tais documentos utilizem a linguagem natural.

Enquanto a abordagem descrita em [Boyd05] utiliza um sub-conjunto restrito da língua natural, nossa abordagem não restringe o uso da língua portuguesa nos documentos que manipula. Nosso objetivo é identificar atores/sujeitos e recursos/objetos relevantes, de forma a apoiar a construção ou atualização do léxico da aplicação. O processo criado também é útil na construção de um léxico para o domínio da organização, pois em organizações utilizando desenvolvimento distribuído de software dificuldades derivadas de diferentes capacidades lingüísticas, diferenças culturais e *delays* de comunicação tornam mais presente a necessidade de um léxico abrangente.

Nossa abordagem utiliza também uma *stop list* constituída de termos não relevantes para o domínio considerado, permitindo ajustes nos termos a serem extraídos e gerando resultados mais precisos na avaliação futura de documentos de um mesmo domínio. A incorporação de um lematizador também possibilitaria a extração de sinônimos. Assim como enfatizado por Harmain e Gaizauskas [Hairmain00], consideramos que os resultados obtidos não prescindem do processo de revisão e avaliação por especialistas do domínio; os sujeitos e objetos extraídos através do processo proposto constituem um subsídio importante para a construção ou atualização do léxico da aplicação.

Identificação de duplicidade em requisitos: a utilização de medidas de

similaridade para identificar duplicidade em requisitos é tema de ao menos dois trabalhos na área de requisitos: [Park00] e [Dag01] relatam resultados de trabalhos com uso de diferentes medidas de similaridade. Em [Park00] é utilizada a medida do coseno do ângulo formado pelos documentos, sendo que a matriz termo-documento é gerada utilizando-se a freqüência não dos termos, mas de expressões com afinidade léxica. Tais expressões são obtidas usando uma combinação de um parser para identificar relações entre palavras e uma técnica denominada de *sliding window*, que varre o texto considerando "janelas" de até cinco palavras para análise.

Os índices que utilizamos já foram objeto de trabalho investigativo da similaridade entre requisitos [Dag01]; nesse estudo foi avaliado um conjunto de requisitos escritos em linguagem natural (língua inglesa). Tais requisitos foram representados por matrizes termo-documento, sendo desconsiderados termos relacionados numa *stoplist* composta por artigos, pronomes, advérbios e outros. O estudo conclui que o uso desses indexadores, em particular Dice e coseno, resulta em alto grau de acerto na identificação de requisitos duplicados e baixa taxa de falsos positivos (requisitos indicados como duplicados, mas que na realidade não são).

Identificação de omissões em RNF's: utilizamos uma estratégia baseada na Análise de Conteúdo para identificar omissões em RNF's. Análise de conteúdo também foi utilizada em [Fantechi05] para identificar inconsistências em documentos de requisitos. Nessa abordagem o documento de requisitos é processado por um *parser* gramatical, e são extraídas triplas sujeito - ação - objeto (SAO). Esses elementos são inseridos num dicionário de categorias e a cada elemento é atribuído um peso, de acordo com a classe gramatical. A criação do dicionário segue a estrutura do documento: se requisitos funcionais são colocados separadamente de requisitos não funcionais, o dicionário estará estruturado em duas partições. O documento de requisitos é analisado, e a cada requisito é atribuído um valor que computa a presença das categorias de cada partição do dicionário no requisito. Esses valores são depois analisados, e são verificados os requisitos que apontem pesos relevantes para a partição indevida; por exemplo, um requisito funcional que apresente peso relevante para termos que constam no dicionário da partição de requisitos não funcionais.

Um problema apresentado por essa abordagem está na construção do

dicionário: se requisitos forem colocados numa seção incorreta, os componentes SAO que irão gerar as categorias da partição incluirão triplas SAO na partição incorreta, e a análise resultará inconsistente. Nossa abordagem trabalha com um dicionário construído previamente, abordando apenas requisitos não funcionais, e considerando expressões utilizadas na organização para registro dos RNF's.

6.3. Limitações e trabalhos futuros

Nosso trabalho apresenta limitações que relacionamos a seguir, juntamente com trabalhos futuros sugeridos para ampliar e consolidar as estratégias propostas.

Verificação de Requisitos: nossa estratégia incorpora de forma automática a busca por duplicidade em requisitos. Também apóia a identificação de omissões em requisitos não funcionais. Futuramente podemos investigar com uso de técnicas de processamento da linguagem natural a detecção automática de conflitos entre requisitos. Em relação aos resultados da verificação, uma possível ampliação envolveria a criação de um formulário para uso dos humanos na identificação de erros, discrepâncias e omissões em requisitos. Este formulário deveria ser pensado de forma a viabilizar de forma automática a consolidação, por um agente de software, dos relatórios gerados pelo grupo de verificadores. As tabelas para a análise de omissões de RNF's poderiam ser ampliadas de modo que o engenheiro de requisitos pudesse indicar as comparações que, executadas de forma automática, lhe subsidiassem de forma mais intensa a análise de omissões e de discrepâncias.

Controle das modificações em requisitos: nossa estratégia destaca especificamente um agente para controlar a ocorrência de modificações em requisitos, e propagar essa informação aos demais agentes da plataforma. Este aspecto seria mais bem atendido com a incorporação de um controlador de versões, de forma que cada um dos sites distribuídos pudesse manter um repositório próprio dos artefatos e documentos sendo manipulados. Isto exigiria um esforço maior para manter a consistência entre repositórios, o que poderia ser obtido através de um agente local ao site que ou executaria de forma autônoma operações de *check-in* após um determinado prazo ou notificaria o humano responsável pela modificação local da inconsistência detectada.

Rastreabilidade entre requisitos funcionais e requisitos não funcionais: uma vez definidas as expressões utilizadas no registro de RNF's, é possível criar automaticamente a matriz de rastreabilidade entre requisitos funcionais e não funcionais. Este é um trabalho futuro facilmente implementável.

Léxico da aplicação: extender a estratégia de construção ou atualização do léxico para símbolos do tipo verbo e estado, não abarcados nesta proposta, e realizar experimentos visando verificar a adequação da estratégia criada para construção de um léxico para a organização, utilizando diferentes tipos de documentos.

Validação de requisitos: o apoio oferecido às atividades de validação é limitado à geração de visões parciais de requisitos, permitindo ao participante do processo executar seu trabalho avaliando conjuntos menores de requisitos. As visões também podem ser geradas a partir da solicitação do validador, a partir de um tema de seu interesse. Isto poderia ser expandido para tratar também outros artefatos utilizados na validação, como *mock ups* e protótipos criados para esta fase.

7
Referências Bibliográficas

[Aires00] Aires, R. V. X. Implementação, adaptação, combinação e avaliação de etiquetadores para o português do Brasil. Dissertação de Mestrado, ICMC-USP, São Carlos, SP. 2000.

[Audy04] Audy, Jorge; Evaristo, Roberto; Watson-Manheim, Mary. "Distributed Analysis: the Last Frontier?" In: 37th Hawaii International Conference on System Sciences, 2004. Proceedings. pgs. 1-9.

[Audy04a] Audy, Jorge L. N. & Lopes, Leandro. "Towards a reference model for requirement engineering in distributed software development". In: CAiSE - 16th International Conference on Advanced Information Systems Engineering, Riga, Letonia. 2004. Proceedings.

[Baeza-Yates99] Baeza-Yates, R. & Ribeiro Neto, B. "Modern Information Retrieval". New York: ACM Press, 1999.

[Baniassad04] Baniassad, Elisa & Clarke, Sióbhan. "Theme: an Approach for Aspect-Oriented Analysis and Design". In: the International Conference on Software Engineering, 2004. Proceedings.

[Bardin77] Bardin, L. Análise de conteúdo. Lisboa, Ed. 70, 1977. 225 p.

[Bellifemine01] Bellifemine, F.; Poggi, A.; Rimassa, G. JADE: a FIPA2000 compliant agent development environment. In: Fifth international Conference on Autonomous Agents (AGENTS '01). Proceedings. ACM Press, New York, NY, 2001. pp. 216-217.

[Bianchi02] Bianchi, A.; Caivano, D.; Lanubile, F.; Rago, F. & Visaggio, G. "An Empirical Study of Distributed Software Maintenance". In: International Conference on Software Maintenance (ICSM.02). Proceedings.

[Blackburn01] BLACKBURN, M. R.; BUSSER, R.; NAUMAN, A. Removing Requirement Defects and Automating Test. Software Productivity Consortium NFP, 2001. Disponível em <http://www.software.org/ pub/taf/downloads/ RemovingRequirementDefects.pdf> Acesso em 26 nov 2002.

[Boehm76] Boehm, Barry. Software Engineering, IEEE Transactions on Computers, v. C-25, Dec 1976.

[Bohem01] Bohem, B. & Basili, V. Software Defect Reduction Top 10 List. In: IEEE Computer, vol. 34, nº 1, jan. 2001. pp. 135-137.

[Booch00] Booch, G. ; Rumbaugh, J. & Jacobson, I. "UML: guia do usuário". Rio de Janeiro: Campus, 2000. 472 p. ISBN 8535205624

[Boyd05] Boyd, S.; Zowghi, D. & Farroukh, A. "Measuring the expressiveness of a Constrained Natural Language: an Empirical Study". In: 13th IEEE

International Conference on Requirements Engineering (RE´05). Proceedings.

[Caldas01] Caldas Jr, J.; Imamura, C.Y.M.; Rezende, S.O. Avaliação de um Algoritmo de Stemming para o Língua Portuguesa. In: 2nd Congress of Logic Applied to Technology, 2001. Proceedings. Vol. 2. pp. 267–274.

[Carmel99] Carmel, E. "Global Software Teams". Prentice-Hall, 1999.

[Chang01] Chang, C.; Cai, L. "Agent based Requirements Evolution over the Internet". In: IEEE Workshop on Software Engineering on the Internet, The IEEE-CS/IPSJ 2001 Symposium on Applications and the Internet (SAINT 2001), Jan. 8-12, 2001. pp. 83-88.

[Chaves03] Chaves, M. S. Um estudo e apreciação sobre algoritmos de stemming para a língua portuguesa. In: IX Jornadas Iberoamericanas de Informática. Cartagena de Indias - Colômbia, 11-15 agosto de 2003.

[Chen05] Chen, K.; Zhang, W.; Zhao, H. & Mei, Hong. An Approach to Constructing Feature Models Based on Requirements Clustering. In: 13th IEEE International Conference on Requirements Engineering (RE´05). Proceedings. pp. 31-40.

[Cherry04] Cherry, S. & Robillard, P. "Communication Problems in Global Software Development: Spotlight on a New Field of Investigation". In: Third International Workshop on Global Software, May 24, 2004, Edinburgh, Scotland. Proceedings. http://gsd2004.uvic.ca/

[Cysneiros03] Cysneiros, Luiz Márcio; Yu, Eric. Requirements Engineering for Large-Scale Multi-Agent Systems. In: 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Portland, Oregon - USA, 2003. Proceedings. pp. 36-57.

[Dag01] Dag, J. N.; Regnell, B.; Carlshamre, P.; Andersson, M. & Karlsson, J. "Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development". In: 7th Int. Workshop on Requirements Engineering: Foundation for Software Quality, June 4-5 2001, Interlaken, Switzerland. Proceedings.

[Daile96] Daille, B. "Study and Implementation of Combined Techniques for Automatic Extraction of Terminology". In: Klavans, J., Resnik, P. The Balancing ACT- Combining Symbolic and Statistical Approaches to Language, The MIT Press, 1996. pp. 49-66.

[Damian03] Damian, D.; Eberlein, A.; Shaw, M. & Gaines, B. "Facilitation in Distributed Requirements Engineering". Requirements Engineering Journal, 8(1), 2003, pags 23-41.

[Damian03a] Damian, D. & Zowghi, D. "RE challenges in multi-site software development organizations". Requirements Engineering Journal, 8(3),2003, pgs. 149-160.

[Davis93] DAVIS, Alan et al. Identifying and Measuring Quality in Software Requirements Specifications**.** In: IEEE-CS INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, 1., 1993, Baltimore. **Proceedings**. Los Alamitos, California: IEEE Computer Society Press, may 1993, p. 141-152.

[Dellen96] Delle, B. & Maurer, F. "Integrating Planning and Execution in

Software Development Process". In: Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96). Proceedings, pp. 170–176.

[Dhavachelvan04] Dhavachelvan, P.; Uma, G.V. "Reliability Enhancement in Software Testing- An Agent-Based Approach for Complex Systems". Lecture Notes in Computer Science, vol. 3356,jan 2004. pp. 282 - 291.

[El-Emam00] El-Emam, K.& Birk, A. Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability. Journal of Systems and Software, 51(2). pp. 119-149.

[Fagan86] FAGAN, M. E. Advances in Software Inspections. **IEEE Transactions on Software Engineering**, vol. SE-12, nº 7, p. 744-751, july 1986.

[Faltings00] Faltings, B. Intelligente Agents: Software Technology for the new Millennium. In: Informatik 1/2000, Editorial. pags. 2-5. Disponível em <http://www.svifsi.ch/revue/pages/issues/n001/no001.html>. Acesso em 11.12.2003.

[Fantechi05] Fantechi, A.; Spinicci, E. **A Content Analysis Technique for Inconsistency Detection in Software Requirements Documents**. Anais do WER05 - Workshop em Engenharia de Requisitos, Porto, Portugal, Junho 13-14, 2005. pp 245-256.

[Ferber99] Ferber, Jacques. Multi-Agent Systems: an Introduction to Distributed Artificial Intelligence. Pearson Edicational Limited, 1999.

[Gaeta02] Gaeta, M. & Ritrovato, P. "Generalised Environment for Process Management in Cooperative Software Engineering". In: 26th Annual International Computer Software and Applications Conference (COMPSAC'02). Proceedings.

[Gershenson99] Gershenson, J. A. & Stauffer, L. A. "A taxonomy for design requirements from corporate customers". Research in Engineering Design, vol 11. pp 103-115.

[Gervasi02] Gervasi, V. & Nuseibeh,B. Lightweight validation of natural language requirements". In: Software Practice and Experience, vol. 32, no. 2, 2002. pp. 113-133

[Gonzalez05] Gonzalez, M.A.I. "Termos e Relacionamentos em Evidência na Recuperação de Informação". Tese de doutorado, Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS. 2005.

[Gorton96] Gorton, I. & Motwani, S. "Issues in Co-operative Software Engineering Using Globally Distributed Teams". In: Information and Software Technology Journal ,vol 38(10), 1996. págs. 647-655.

[Gruenbacher01] Gruenbacher, P.; Egyed, A. & Medvidovic, N. "Dimensions of Concerns in Requirements Negotiation and Architecture Modeling". In: International Conference on Software Engineering - ICSE 2001. Proceedings.

[Grundy05] Grundy, J.; Ding, G. & Hosking, J. "Deployed software component testing using dynamic validation agents". The Journal of Systems and

Software 74 (2005). pags 5-14.

[Haendchen05] Haendchen Filho, A. A Middleware Framework for Buiding Multi-Agent Systems in the Internet. Ph.D. Thesis, Pontifícia Universidade Católica do Rio de Janeiro PUC-Rio, Rio de Janeiro, RJ, Brasil, 2005.

[Haendchen07] Haendchen F., A.; Prado, H. A.; Lucena, C. J. P. A WSA-based architecture for building multi-agent systems. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007). Honolulu, Hawai'i. Proceedings. a ser publicado.

[Hammer96] HAMMER, T. et al. Requirements Metrics for Risk Identification. In: SOFTWARE ENGINEERING WORKSHOP, 21st, 1996, NASA-GSFC, MD. **Proceedings**. Disponível em <http://satc.gsfc.nasa.gov/support/SEW_DEC96/ sel.PDF>. Acesso em 21 out 2002.

[Hardy04] Hardy, C.; Harley, B. & Phillips, N. "Discourse Analysis and Content Analysis: Two Solitudes?". In: Qualitative Methods, vol. 2:1, Spring 2004. pp.19-22.

[Harmain00] Harmain, H. M. & Gaizauskas, R. "CM-Builder: An Automated NL-based CASE Tool". In: 15th IEEE International Conference on Automated Software Engineering (ASE'2000), 2000. Proceedings. pp. 45-53.

[Herbsleb03] Herbsleb, J. & Mockus, A. "An empirical study of speed and communication in globally distributed software development". IEEE Transactions on Software Engineering, vol 29(6), jun 2003. pgs. 481-494.

[Himmelspach03] Himmelspach, J.; Rohl, M. & Uhrmacher, A.M. "Simulation for testing software agents - an exploration based on James". In: Simulation Conference, 2003. Proceedings. pp. 799-807.

[Hoskinson05] Hoskinson, A. "Creating the Ultimate Research Assistant". IT Systems Perspective, IEEE Computer Society, november 05. pp. 97-99.

[Hsia92] Hsia, P. & Gupta, A. "Incremental Delivery Using Abstract Data Types and Requirements Clustering". In: Second International Conference on Systems Integration. Proceedings. IEEE Computer Society, June 1992. pp. 137-150.

[IEEE98] Institute of Electrical and Electronics Engineers. "System Requirements Specifications". Document Number: IEEE 830-1998, 1998.

[Jennings00] Jennings, N.R.. "On agent-based software engineering". In: Artificial Intelligence, vol. 117 (2000). pp. 277–296.

[Jennings01] Jennings, N.R., and Wooldridge, M.J.: Agent-Oriented Software Engineering. In: Bradshaw, J. (ed.): Handbook of Agent Technology. AAI/MIT Press (2001). Disponível em <http://www.ecs.soton.ac.uk/~nrj/download-files/agt-handbook.pdf>. Acesso em 12.01.04.

[Jones96] JONES, T. Capers. Applied Software Measurement: Assuring Productivity and Quality. New York, McGraw-Hill, 1996.

[Kotonya98] Kotonya, G. e Sommerville, I. Requirements Engineering: process and techniques. John Willey & Sons Ltd, 1998.

[Kowalski97] Kowalski, G. Information retrieval systems : theory and implementation. Boston : Kluwer Academic, 1997. 282 p.

[Leffingwell97] Leffingweel, D. Calculating the return on investment from more effective requirements management. American Programmer, vol. 10, nº5, abr 1997. pp.13-16.

[Leite90] - Leite, J.C.S.P.; Franco, A. P. "O uso de hipertexto na elicitação de linguagens de da aplicação". In: 4º Simpósio Brasileiro de Engenharia de Software, 1990. Anais. pp.124-133.

[Leite91] Leite, J.C.S.P.; Freeman, P.A. "Requirements Validation through Viewpoint Resolution". In: IEEE Transactions on Software Engineering: vol. 17, nº 12. pp. 1253-1269.

[Leite93] Leite, J.C.S.P. & Franco, A.P.M. "A Strategy for Conceptual Model Acquisition". In: First IEEE International Symposium on Requirements Engineering, San Diego, Ca, IEEE Computer Society Press, 1993. Proceedings. pp. 243-246.

[Leite94] Leite, J. C. S. P. Engenharia de Requisitos. PUC-Rio, Rio de Janeiro, 1994. Notas de aula.

[Leite95] Leite, J.C.S.P. & Oliveira, A.P. A Client Oriented Requirements Baseline – In: Second IEEE International Symposium on Requirements Engineering (RE'95), 1995. Proceedings. Los Alamitos: IEEE Computer Society Press, 1995. pp.108-115.

[Leite01] Leite, J. C. S. P. In: "Qualidade e Produtividade em Software", eds. Kival Chaves Weber et al. São Paulo, Ed. Makron Books, 2001.

[Li03] Li, Y.; Shen, W. & Ghenniwa, H. Collaborative and Proactive Data Agent for Distributed Design Environments. In: Journal of Integrated Design and Process Science, june 2003, Vol. 7, No. 2, pp. 71-85

[Li03a] Li, Y.; Shen, W. & Ghenniwa, H. Integrated description for Web service-oriented agents in e-Marketplaces. In: BASeWEB 2003 Workshop. Proceedings. Disponível em http://www.cs.unb.ca/baseweb/baseweb03/papers/li_ghenniwa_shen_revision-may12.pdf. Acesso em 12.06.2005.

[Liberato97] Liberato, Y. G. "A estrutura do SN em português: uma abordagem cognitiva". Tese de doutorado, 1997. UFMG, Departamento de Lingüística, Belo Horizonte.

[Lopes05] Lopes, L.; Prikladnicki, R.; Audy, J. & Majdenbaum, A. "Requirements Specification in Distributed Software Development - A Process Proposal". In: 38th Hawaii International Conference on System Sciences - HICSS. Hawaii, USA, 2005. Proceedings.

[Lowe04] Lowe, W. "Content analysis and its place in the (Methodological) scheme of things". In: Qualitative Methods, vol. 2:1, Spring 2004. pp.25-27.

[Luhn58] Luhn, H.P. The automatic creation of literature abstracts, IBM Journal of Research and Development, vol. 2, 1958. pp. 159-165.

[Manning99] Manning, Christopher D.; Schütze, Heinrich. Foundations of statistical natural language processing. Cambridge: MIT Press, c1999. 680

p. ISBN 0262133601 (enc.)

[Mason97] Mason, O. & Tufis, D. "Probabilistic Tagging in a Multi-lingual Environment: Making an English Tagger Understand Romanian". In: Third European TELRI Seminar, Montecatini, Italy, 1997. Proceedings.

[Matsubara03] Matsubara, Edson T.; Martins, Claudia A. & Monard, Maria C. PreTexT: uma ferramenta para pré-processamento de textos utilizando a abordagem bag-of-words. Relatório técnico nº 209, série Relatórios Técnicos do ICMC, São Carlos. 2003.

[Meadow00] Meadow, C. T.; Boyce, Bert & Kraft, Donald H. Text information retrieval systems. 2 ed. San Diego : Academic Press, 2000. 364 p.

[Message01] MESSAGE: Methodology for Engineering Systems of Software Agents - Final guidelines for the relevant problems areas where agent technology is appropriate. EURESCOM Participants in Project P907-GI (2001).

[Moraes99] Moraes, R. Análise de Conteúdo. In: Revista Educação, Porto Alegre, vol. 22 nº 37, março 1999. pp. 7-32.

[Orengo01] Orengo. V & Huyck, C. A stemming algorithm for the Portuguese language. In: Proceedings of the 8th International Symposium on String Processing and Information Retrieva l (SPIRE) 2001. pp 186–193.

[Overainder04] Overeinder, B. J.; Brazier, F. Scalable Middleware Environment for Agent-Based Internet Applications. Data Knowledge Engineering, 41(2-3):229-245, 2004. Elsevier Science Publishers, Amsterdam, The Netherlands, 2004.

[Ozkaya06] Ozkaya, I. Representing Requirements Relationships. In: Fisrt International Workshop on Requirements Engineering Visualization (REV´06). Proceedings.

[Palmer92] Palmer, J.D. & Liang, Y. "Indexing and Clustering of Software Requirements Specifications". In: Information and Decision Technologies vol. 18, nº 4. pp 283-299.

[Paré99] Paré, G. & Dubé, L. “Virtual Teams: An Exploratory Study of Key Challenges and Strategies”. In: 20th International Conference on Information Systems, dez 1999, Charlotte, North Carolina, United States. Proceedings. pp. 479-483.

[Park00] Park, S.; Kim, H.; Ko, Y. & Seo, J. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. In: Information and Software Technology 42 (2000). pp. 429–438.

[Parreiras03] Parreiras, F. "O uso de sintagmas nominais como fonte de descritores para textos de periódicos científicos". Escola de Ciência da Informação. Belo Horizonte, 2003. Disponível em http://www.fernando.parreiras.nom.br/publicacoes/sn.pdf.

[Pepper00] Pepper, S. "The TAO of Topic Maps, finding the way in the age of infoglut". In: XML Europe Conference, Paris, 2000. Proceedings.

[Pepper01] Pepper, Steve & Moore, Graham. "XML Topic Maps (XTM) 1.0". http://www.topicmaps.org/xtm/1.0/, Mar 2001. TopicMaps.Org Specification.

[Pérez03] Pérez, C. C. C.; Gasperin, C. & Vieira, R. "Extração semi-automática de conhecimento a partir de textos". In: IV Encontro Nacional de Inteligência Artificial (ENIA 2003), Campinas, 2003. Anasi da SBC, 2003, v. 7, pp.193-202.

[Perini98] Perini, M. A. "Gramática descritiva do português". 3ªed. - São Paulo: Ática, 1998. 380p. ISBN 8508055501

[Ponnurangam05] Ponnurangam, D. Uma, G. V. Fuzzy complexity assessment model for resource negotiation and allocation in agent-based software testing framework
Expert Systems with Applications, vol. 29,no 1, July 2005. pp. 105-11.

[Porter80] Porter. M. F. An algorithm for suffix stripping. Program, 14(3) pp 130−137. Disponível em http://www.tartarus.org/~martin/PorterStemmer/. Acesso em 06.04.2004.

[Porter95] PORTER, A. A. ; VOTTA JR, L. G.; BASILI, V. Comparing Detection Methods for Software Requirements Inspections: a replicated experiment. **IEEE Transactions on Software Engineering**, vol. 21, nº 6, p. 563-575, june 1995.

[Prikladnicki03] Prikladnicki, R., Audy, J.L.N., & Evaristo, R. "Global software development in practice: lessons learned". Software Process: Improvement and Practice, 8(4), 2003. pp. 267-281.

[Prikladnicki04] Prikladnicki, R. & Audy, J. "MuNDDoS - Um Modelo de Referência para Desenvolvimento Distribuído de Software". In: XVIII Simpósio Brasileiro de Engenharia de Software - 2004 - Brasília, DF, Brasil. Anais. pgs. 289-304.

[Rosenberg98] ROSENBERG, Linda. et al. Requirements, Testing, and Metrics. In: PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE, 15th, 1998, Utah. Proceedings. Disponível em <http://satc.gsfc.nasa.gov/support/ PNSCQ _OCT98/requirements_testing_and_metrics.html>. Acesso em 12 out 2002.

[Ross77] Ross, Douglas & Schoman, A. "Structured analysis for requirements definition". IEEE Transactions on Software Engineering. 1977. Vol. 3(1), pp. 6–15.

[Salton83] Salton, Gerard. Introduction to modern information retrieval. New York, NY : McGraw-Hill, c1983. 448 p.

[Salton88] Salton, G. & Buckley, C. Term-weighting approaches in automatic text retrieval. Information Processing and Management, vol. 24 (5), 1988. pp. 513–523.

[Sampaio05] Sampaio, A.; Chitchyan, R.; Rashid, A. & Rayson, P. "EA-Miner: a Tool for Automating Aspect-Oriented Requirements Identification". In: ASE 2005.

[Sardinha04] Sardinha, A. P. B. "Lingüística de Corpus". São Paulo: Ed. Manole. ISBN: 1676-4. 2004. 410 páginas.

[Sayão05] Sayão, M. & Leite, J. C. S. P. "Uso de Agentes no Processo de Requisitos em Ambientes Distribuídos de Desenvolvimento". In: Workshop de Engenharia de Requisitos, Lisboa, Portugal, 2005. Anais.

[Shull00] SHULL, F.; RUS, I.; BASILI, V. How Perspective-Based Reading can Improve Requirements Inspections. **IEEE Computer**, vol. 33, nº 7, p. 73-79, july 2000.

[Silva06] Silva, Lyrene F. Uma estratégia orientada a aspectos para modelagem de requisitos. 2006. 222 f.Tese (Doutorado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ.

[Sommerville04] Sommerville, I. Software Engineering. Pearson Educational Limited, seventh edition, 2004.

[Sommerville98] Sommerville, I.; Sawyer, P. & Viller, S. "Viewpoints for requirements elicitation: a practical approach". In: Third International Conference on Requirements Engineering, 1998. Proceedings. pp. 74 - 81

[Standish04] Standish Group. The CHAOS Report 2004.

[SWEBOK04] Guide to the Software Engineering Body of Knowledge. 2004.

[Teline03] Teline, M. F.; Almeida, G. M. B. & Aluísio, S. M. "Extração Manual e Automática de Terminologia: Comparando Abordagens e Critérios". In: 16th Brazilian Symposium on Computer Graphics and Image Processing - SIBGRAPI 2003. Proceedings.

[Terra04] Terra, E. "Lexical Affinities and Language Applications". PhD Thesis, University of Waterloo, Canada. 2004.

[Thrane80] THRANE, Torben. Referential-Semantic Analysis: Aspects of a Theory of Linguistic Reference. Londres: Cambridge University Press, 1980. 256p.

[Tufis98] Tufis, D. & Mason, O. Tagging Romanian Texts: a Case Study for QTAG, a Language Independent Probabilistic Tagger. In: First International Conference on Language Resources & Evaluation, Granada, Spain. Proceedings. pp. 589-596.

[Vieira01] Vieira, R. & Lima, V. L. S. (2001). "Lingüística Computacional: Princípios e Aplicações". In: As Tecnologias da Informação e a questão social: anais. Carlos Eduardo Ferreira (Ed.) Fortaleza, SBC. ISBN 85-88442-03-5 (v.2). pp 47-88.

[Wilson97] WILSON, W.M.; ROSENBERG, L.H.; HYATT, L. E. Automated Analysis of Requirement Specifications. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (IASTED), 1997, Boston, MA. Proceedings. Disponível em <http://satc.gsfc.nasa.gov/support/ICSE_MAY97/ arm/ICSE97-arm.htm>. Acesso em 13 set 2002.

[Witten00] Witten, I. H. & Frank, E. Data mining : practical machine learning tools and techniques with java implementations. San Francisco, CA : Morgan Kaufmann, 2000. 371 p.

[Wives04] Wives, Leandro. "Utilizando conceitos como descritores de textos para o processo de identificação de conglomerados (clustering) de documentos". Tese de doutorado, UFRGS. 2004.

[Wongthongtham06] Wongthongtham, P.; Chang, E.; Dillon, T. S. & Sommerville, I., Ontology-based multi-site software development. Journal of Systems Architecture (2006), doi:10.1016/j.sysarc.2006.06.008

[Wooldridge99a] Wooldridge, M. Intelligent Agents. In: Weiss, Gerhard (Ed.). Multiagent Systems - A Modern Approach. MIT Press, 1999. pp. 27-77.

[Wu06] Wu, S.; Ghenniwa, H.; Zhang, Y & Shen, W. Personal assistant agents for collaborative design environments. In: Computers in Industry (2006), doi:10.1016/j.compind.2006.04.010

[Yu95] Yu, E. Modelling Strategics Relationships for Process Reengineering. PhD Thesis, University of Toronto, 1995.

[Yu02] Yu, E., Agent-Oriented Modelling: Software Versus the World. In: Agent-Oriented Software Engineering AOSE-2001 Workshop Proceedings, Montreal, Canada - May 29th 2001. LNCS 2222.

[Zipf49] Zipf, G. Human Behaviour and the Principle of Least Effort. Addison-Wesley, 1949.

[Zowghi02] Zowghi, Didar. "Does Global Software Development need a different Requirements Engineering Process?" In: International Workshop on Global Software Development – ICSE 2002, Orlando, Florida, USA, 2002. Proceedings. pgs. 53-55.

Apêndice A
Padrões para extração de sujeitos/atores: doc. Escalas

<w POS="N">*ente <w POS="PRP">* <w POS="N">*
<w POS="N">*entes <w POS="PRP">* <w POS="N">*
<w POS="N">*enta <w POS="PRP">* <w POS="N">*
<w POS="N">*entas <w POS="PRP">* <w POS="N">*
<w POS="N">*or <w POS="PRP">* <w POS="N">*
<w POS="N">*ores <w POS="PRP">* <w POS="N">*
<w POS="N">*ora <w POS="PRP">* <w POS="N">*
<w POS="N">*oras <w POS="PRP">* <w POS="N">*
<w POS="N">*eiro <w POS="PRP">* <w POS="N">*
<w POS="N">*eiros <w POS="PRP">* <w POS="N">*
<w POS="N">*eira <w POS="PRP">* <w POS="N">*
<w POS="N">*eiras <w POS="PRP">* <w POS="N">*
<w POS="N">*ário <w POS="PRP">* <w POS="N">*
<w POS="N">*ários <w POS="PRP">* <w POS="N">*
<w POS="N">*ária <w POS="PRP">* <w POS="N">*
<w POS="N">*árias <w POS="PRP">* <w POS="N">*
<w POS="N">*es <w POS="PRP">* <w POS="N">*
<w POS="N">*eses <w POS="PRP">* <w POS="N">*
<w POS="N">*esa <w POS="PRP">* <w POS="N">*
<w POS="N">*esas <w POS="PRP">* <w POS="N">*
<w POS="N">*eus <w POS="PRP">* <w POS="N">*
<w POS="N">*euses <w POS="PRP">* <w POS="N">*
<w POS="N">*eusa <w POS="PRP">* <w POS="N">*
<w POS="N">*eusas <w POS="PRP">* <w POS="N">*
<w POS="N">*ia <w POS="PRP">* <w POS="N">*
<w POS="N">*ias <w POS="PRP">* <w POS="N">*
<w POS="N">*ente <w POS="N">*
<w POS="N">*entes <w POS="N">*
<w POS="N">*enta <w POS="N">*
<w POS="N">*entas <w POS="N">*
<w POS="N">*or <w POS="N">*
<w POS="N">*ores <w POS="N">*
<w POS="N">*ora <w POS="N">*
<w POS="N">*oras <w POS="N">*
<w POS="N">*eiro <w POS="N">*
<w POS="N">*eiros <w POS="N">*
<w POS="N">*eira <w POS="N">*
<w POS="N">*eiras <w POS="N">*
<w POS="N">*ário <w POS="N">*
<w POS="N">*ários <w POS="N">*
<w POS="N">*ária <w POS="N">*
<w POS="N">*árias <w POS="N">*
<w POS="N">*ês <w POS="N">*
<w POS="N">*êses <w POS="N">*
<w POS="N">*eses <w POS="N">*
<w POS="N">*esa <w POS="N">*
<w POS="N">*esas <w POS="N">*
<w POS="N">*eus <w POS="N">*
<w POS="N">*euses <w POS="N">*
<w POS="N">*eusa <w POS="N">*

<w POS="N">*eusas <w POS="N">*
<w POS="N">*ia <w POS="N">*
<w POS="N">*ias <w POS="N">*
<w POS="N">*ente
<w POS="N">*entes
<w POS="N">*enta
<w POS="N">*entas
<w POS="N">*or
<w POS="N">*ores
<w POS="N">*ora
<w POS="N">*oras
<w POS="N">*eiro
<w POS="N">*eiros
<w POS="N">*eira
<w POS="N">*eiras
<w POS="N">*ário
<w POS="N">*ários
<w POS="N">*ária
<w POS="N">*árias
<w POS="N">*ês
<w POS="N">*êses
<w POS="N">*eses
<w POS="N">*esa
<w POS="N">*esas
<w POS="N">*eus
<w POS="N">*euses
<w POS="N">*eusa
<w POS="N">*eusas
<w POS="N">*ia
<w POS="N">ias

Apêndice B
Padrões para extração de sujeitos/atores: doc. SELIC

<w POS="N">*ente <w POS="PRP">* <w POS="N">*
<w POS="N">*entes <w POS="PRP">* <w POS="N">*
<w POS="N">*enta <w POS="PRP">* <w POS="N">*
<w POS="N">*entas <w POS="PRP">* <w POS="N">*
<w POS="N">*or <w POS="PRP">* <w POS="N">*
<w POS="N">*ores <w POS="PRP">* <w POS="N">*
<w POS="N">*ora <w POS="PRP">* <w POS="N">*
<w POS="N">*oras <w POS="PRP">* <w POS="N">*
<w POS="N">*eiro <w POS="PRP">* <w POS="N">*
<w POS="N">*eiros <w POS="PRP">* <w POS="N">*
<w POS="N">*eira <w POS="PRP">* <w POS="N">*
<w POS="N">*eiras <w POS="PRP">* <w POS="N">*
<w POS="N">*ário <w POS="PRP">* <w POS="N">*
<w POS="N">*ários <w POS="PRP">* <w POS="N">*
<w POS="N">*ária <w POS="PRP">* <w POS="N">*
<w POS="N">*árias <w POS="PRP">* <w POS="N">*
<w POS="N">*ês <w POS="PRP">* <w POS="N">*
<w POS="N">*eses <w POS="PRP">* <w POS="N">*
<w POS="N">*esa <w POS="PRP">* <w POS="N">*
<w POS="N">*esas <w POS="PRP">* <w POS="N">*
<w POS="N">*eus <w POS="PRP">* <w POS="N">*
<w POS="N">*euses <w POS="PRP">* <w POS="N">*
<w POS="N">*eusa <w POS="PRP">* <w POS="N">*
<w POS="N">*eusas <w POS="PRP">* <w POS="N">*
<w POS="N">*ia <w POS="PRP">* <w POS="N">*
<w POS="N">*ias <w POS="PRP">* <w POS="N">*
<w POS="N">*ente <w POS="CPR">* <w POS="N">*
<w POS="N">*entes <w POS="CPR">* <w POS="N">*
<w POS="N">*enta <w POS="CPR">* <w POS="N">*
<w POS="N">*entas <w POS="CPR">* <w POS="N">*
<w POS="N">*or <w POS="CPR">* <w POS="N">*
<w POS="N">*ores <w POS="CPR">* <w POS="N">*
<w POS="N">*ora <w POS="CPR">* <w POS="N">*
<w POS="N">*oras <w POS="CPR">* <w POS="N">*
<w POS="N">*eiro <w POS="CPR">* <w POS="N">*
<w POS="N">*eiros <w POS="CPR">* <w POS="N">*
<w POS="N">*eira <w POS="CPR">* <w POS="N">*
<w POS="N">*eiras <w POS="CPR">* <w POS="N">*
<w POS="N">*ário <w POS="CPR">* <w POS="N">*
<w POS="N">*ários <w POS="CPR">* <w POS="N">*
<w POS="N">*ária <w POS="CPR">* <w POS="N">*
<w POS="N">*árias <w POS="CPR">* <w POS="N">*
<w POS="N">*ês <w POS="CPR">* <w POS="N">*
<w POS="N">*eses <w POS="CPR">* <w POS="N">*
<w POS="N">*esa <w POS="CPR">* <w POS="N">*
<w POS="N">*esas <w POS="CPR">* <w POS="N">*
<w POS="N">*eus <w POS="CPR">* <w POS="N">*
<w POS="N">*euses <w POS="CPR">* <w POS="N">*
<w POS="N">*eusa <w POS="CPR">* <w POS="N">*
<w POS="N">*eusas <w POS="CPR">* <w POS="N">*

<w POS="N">*ia <w POS="CPR">* <w POS="N">*
<w POS="N">*ias <w POS="PRP">* <w POS="N">*
<w POS="N">*ente <w POS="N">*
<w POS="N">*entes <w POS="N">*
<w POS="N">*enta <w POS="N">*
<w POS="N">*entas <w POS="N">*
<w POS="N">*or <w POS="N">*
<w POS="N">*ores <w POS="N">*
<w POS="N">*ora <w POS="N">*
<w POS="N">*oras <w POS="N">*
<w POS="N">*eiro <w POS="N">*
<w POS="N">*eiros <w POS="N">*
<w POS="N">*eira <w POS="N">*
<w POS="N">*eiras <w POS="N">*
<w POS="N">*ário <w POS="N">*
<w POS="N">*ários <w POS="N">*
<w POS="N">*ária <w POS="N">*
<w POS="N">*árias <w POS="N">*
<w POS="N">*ês <w POS="N">*
<w POS="N">*êses <w POS="N">*
<w POS="N">*eses <w POS="N">*
<w POS="N">*esa <w POS="N">*
<w POS="N">*esas <w POS="N">*
<w POS="N">*eus <w POS="N">*
<w POS="N">*euses <w POS="N">*
<w POS="N">*eusa <w POS="N">*
<w POS="N">*eusas <w POS="N">*
<w POS="N">*ia <w POS="N">*
<w POS="N">*ias <w POS="N">*
<w POS="N">*ão <w POS="N">*
<w POS="N">*ões <w POS="N">*
<w POS="N">*ente
<w POS="N">*entes
<w POS="N">*enta
<w POS="N">*entas
<w POS="N">*or
<w POS="N">*ores
<w POS="N">*ora
<w POS="N">*oras
<w POS="N">*eiro
<w POS="N">*eiros
<w POS="N">*eira
<w POS="N">*eiras
<w POS="N">*ário
<w POS="N">*ários
<w POS="N">*ária
<w POS="N">*árias
<w POS="N">*ês
<w POS="N">*êses
<w POS="N">*eses
<w POS="N">*esa
<w POS="N">*esas
<w POS="N">*eus
<w POS="N">*euses
<w POS="N">*eusa
<w POS="N">*eusas
<w POS="N">*ia
<w POS="N">*ias
<w POS="N">* <w POS="ADJ">* <w POS="ADV">*
<w POS="N">* <w POS="ADJ">*

Apêndice C
Dicionário de Requisitos Não-Funcionais

```
<?xml version="1.0" encoding="UTF-8"?>
<dictionary style="050805" patternengine="substring">
<cnode name="RNFS" desc="dicionário de requisitos não funcionais">
<cnode name="requisitos_nao_funcionais" desc="">
<cnode name="externos" desc="relacionados a aspectos externos ao
sistema">
<cnode name="comunicação sistemas externos" desc="outros sistemas
com os quais existirá troca de informações">
<pnode name="reply"/>
<pnode name="request"/>
</cnode>
<cnode name="custos" desc="restrições orçamentárias para o
sistema">
</cnode>
<cnode name="integração sistemas externos" desc="sistemas já
existentes com os quais haverá integração">
<pnode name="ace"/>
<pnode name="galileo"/>
<pnode name="sap"/>
</cnode>
<cnode name="interoperabilidade" desc="outros sistemas com os
quais haverá interação">
<pnode name="exportar"/>
<pnode name="exportação"/>
<pnode name="exportação de dados"/>
<pnode name="importar"/>
<pnode name="importar dados"/>
<pnode name="importação"/>
<pnode name="importação de dados"/>
<pnode name="integração"/>
<pnode name="migração de dados"/>
<pnode name="sistema externo"/>
<pnode name="transferência de dados"/>
</cnode>
<cnode name="legais" desc="leis específicas incidindo sobre
propriedades do sistema">
<pnode name="internacionalização"/>
</cnode>
</cnode>
<cnode name="processo" desc="características e aspectos que
restringem ou guiam o processo de desenvolvimento">
<cnode name="entrega" desc="prazos e outras características ou
propriedades necessários para a liberação do sistema">
</cnode>
<cnode name="implementação" desc="restrições ou padrões de
ambiente e tecnologia de desenvolvimento">
<cnode name="linguagens" desc="linguagens a utilizar no
desenvolvimento do projeto">
<pnode name="java"/>
<pnode name="xml"/>
</cnode>
```

```
</cnode>
<cnode name="padrões" desc="padrões de qualidade em uso na
organização">
</cnode>
</cnode>
<cnode name="produto" desc="características ou propriedades que o
sistema deverá presentar ou atender">
<cnode name="ambiente operacional" desc="ambiente operacional no
qual o software deverá executar">
<pnode name="internet"/>
<pnode name="rede local"/>
<pnode name="sistema operacional"/>
</cnode>
<cnode name="confiabilidade" desc="características relacionadas a
propriedades de execução mesmo em situações de falha">
<cnode name="integridade dos dados" desc="precisão e veracidade
das informações armazenadas">
<pnode name="transações atômicas"/>
</cnode>
<cnode name="segurança" desc="aspectos relacionados à segurança de
informações">
<pnode name="gerar log"/>
<pnode name="gerar um log"/>
<pnode name="log"/>
</cnode>
<cnode name="tolerância a falhas" desc="habilidade em manter a
operação em caso de falhas">
</cnode>
</cnode>
<cnode name="confidencialidade" desc="proteção no acesso ao
sistema e às informações manipuladas">
<cnode name="cadastrar usuários" desc="registro e manutenção de
usuários válidos">
<pnode name="activar utilizador"/>
<pnode name="ativar usuário"/>
<pnode name="controle de acesso"/>
<pnode name="manter usuário"/>
<pnode name="perfil de usuário"/>
<pnode name="perfis de usuário"/>
</cnode>
<cnode name="conectar e autenticar usuários" desc="validação de
usuários na conexão">
<pnode name="autenticação"/>
<pnode name="login"/>
<pnode name="logon"/>
<pnode name="password"/>
<pnode name="password * acesso"/>
<pnode name="senha"/>
<pnode name="senha de acesso"/>
<pnode name="validação de usuário"/>
</cnode>
<cnode name="desconectar" desc="encerramento de conexão">
<pnode name="desconectar"/>
<pnode name="encerrar sessão"/>
<pnode name="logoff"/>
<pnode name="logout"/>
</cnode>
</cnode>
<cnode name="eficiência" desc="aspectos relacionados ao grau de
recursos utilizados e tempo de processamento">
```

```
<cnode name="desempenho" desc="tempo de resposta e de
processamento">
<pnode name="tempo de execução"/>
<pnode name="tempo de resposta"/>
</cnode>
<cnode name="recursos" desc="recursos necessários para uso e
armazenamento de informações">
<pnode name="espaço em disco"/>
<pnode name="memória principal"/>
</cnode>
</cnode>
<cnode name="manutenibilidade" desc="características relacionadas
ao processo de manutenção">
<cnode name="modifiabilidade" desc="características relacionadas a
facilidade e ao  esforço necessários para modificações no
software">
</cnode>
<cnode name="testabilidade" desc="esforço necessário para a
verificação da correção">
</cnode>
</cnode>
<cnode name="portabilidade" desc="características que indicam o
grau de adaptação a diferentes plataformas">
<cnode name="adaptabilidade" desc="características relacionadas a
aspectos de utilização do software em diferentes ambientes
operacionais">
</cnode>
</cnode>
<cnode name="usabilidade" desc="características relacionadas à
aspectos de utilização do software">
<cnode name="flexibilidade" desc="características relacionadas à
adaptação da interface com usuário ">
<pnode name="adaptação ao usuário"/>
<pnode name="customização"/>
</cnode>
<cnode name="operabilidade" desc="esforço para operar e controlar
as operações">
<pnode name="facilidade de uso"/>
</cnode>
<cnode name="web" desc="características de usabilidade próprias
para sistemas web">
<pnode name="acesso via web"/>
<pnode name="navegador"/>
</cnode>
</cnode>
</cnode>
<cnode name="suporte" desc="relacionados a características de
apoio ao sistema">
<cnode name="documentação" desc="documentação necessária para a
instalação e utilização do software">
<pnode name="manual de suporte"/>
<pnode name="manual de usuário"/>
<pnode name="manual do usuário"/>
<pnode name="manual do utilizador"/>
</cnode>
<cnode name="treinamento" desc="capacitação dos usuários para a
correta operação do sistema">
</cnode>
</cnode>
</cnode>
</dictionary>
```

Apêndice D
Código dos agentes da aplicação

Agente Comunicador (communicator.java)

```
package requisitos.agents;

import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.mail.MessagingException;
import javax.mail.internet.AddressException;

import org.midas.as.AgentServer;
import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.midas.as.manager.execution.Logger;

import requisitos.business.entities.User;
import requisitos.business.entities.VerifyValidateProccess;
import requisitos.dao.RequirementDocumentDao;
import requisitos.web.email.Email;

public class Communicator extends Agent implements MessageListener
{
	private static String className = "[COMMUNICATOR AGENT]  ";
	private static SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy
HH:mm");

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Communicator",this);
	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{

	}

	public void boardChanged(Message msg)
	{
		String title = msg.getTitle();
```

```
                if ( title.equals("PreVerificationStage") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of PreVerificationStage
in "+documentName,true);
                        sendPreVerification(documentName);
                }
                if ( title.equals("VerificationStage") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of VerificationStage in
"+documentName,true);
                        sendVerification(documentName);
                }
                if ( title.equals("PreValidationStage") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of PreValidationStage
in "+documentName,true);
                        sendPreValidation(documentName);
                }
                if ( title.equals("ValidationStage") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of ValidationStage in
"+documentName,true);
                        sendValidation(documentName);
                }
                if ( title.equals("Finalization") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of Finalization in
"+documentName,true);
                        finalization(documentName);
                }
                if ( title.equals("SimilarityTest") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Notifying Users of SimilarityTests on
"+documentName,true);
                }
        }

        private void sendPreVerification(String documentName)
        {
                RequirementDocumentDao rdao =
(RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocume
```

```
ntDao");
                List<VerifyValidateProccess> vvproccessList =
rdao.getVerifyValidateProccessByDocumentName(documentName);

                VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-
1);
                Set<User> users = vvproccess.getUsers();

                for (User user : users)
                {
                        try
                        {
                                Email.EnviaEmail(user.getEmail(), "Rq.NET - Notificação
sobre o Documento "+vvproccess.getRequirementDocument().getName()+" - Pré-Verificação",
                                                "Caro
"+user.getUsername()+",\n\n" +
                                                "O documento
"+vvproccess.getRequirementDocument().getName()+" foi agendado para um processo de
verificação.\n\n"+
                                                "O dia marcado para o
início do processo é "+formatter.format(vvproccess.getPreVerification().getDeadline())+", até " +
                                                "lá o documento se
encontra no estágio de Pré-Verificação. Testes de similaridade serão rodados e enviados por "+
                                                "email assim que
concluídos.\n\n");
                        }
                        catch (AddressException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (MessagingException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        private void sendVerification(String documentName)
        {
                RequirementDocumentDao rdao =
(RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocume
ntDao");
                List<VerifyValidateProccess> vvproccessList =
rdao.getVerifyValidateProccessByDocumentName(documentName);

                VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-
1);
                Set<User> users = vvproccess.getUsers();

                for (User user : users)
                {
                        try
                        {
                                Email.EnviaEmail(user.getEmail(), "Rq.NET - Notificação
sobre o Documento "+vvproccess.getRequirementDocument().getName()+" - Verificação",
                                                "Caro
"+user.getUsername()+",\n\n" +
                                                "O documento
```

```
"+vvproccess.getRequirementDocument().getName()+" se encontra a partir de agora em processo de verificação.\n\n");
                        }
                        catch (AddressException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (MessagingException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        private void sendPreValidation(String documentName)
        {
                RequirementDocumentDao rdao = (RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocumentDao");
                List<VerifyValidateProccess> vvproccessList = rdao.getVerifyValidateProccessByDocumentName(documentName);

                VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-1);
                Set<User> users = vvproccess.getUsers();

                for (User user : users)
                {
                        try
                        {
                                Email.EnviaEmail(user.getEmail(), "Rq.NET - Notificação sobre o Documento "+vvproccess.getRequirementDocument().getName()+" - Pré-Validação",
                                                "Caro "+user.getUsername()+",\n\n" +
                                                "O documento "+vvproccess.getRequirementDocument().getName()+" teve seu processo de verificação concluído, e se "+
                                                "encontra no estágio de Pré-Validação. A etapa de validação deverá começar a partir do dia "+formatter.format(vvproccess.getPreValidation().getDeadline())+
                                                ".");

                        }
                        catch (AddressException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (MessagingException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        private void sendValidation(String documentName)
        {
```

```
		RequirementDocumentDao rdao = (RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocumentDao");
		List<VerifyValidateProccess> vvproccessList = rdao.getVerifyValidateProccessByDocumentName(documentName);

		VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-1);
		Set<User> users = vvproccess.getUsers();

		for (User user : users)
		{
			try
			{
				Email.EnviaEmail(user.getEmail(), "Rq.NET - Notificação sobre o Documento "+vvproccess.getRequirementDocument().getName()+" - Validação",
								"Caro "+user.getUsername()+",\n\n" +
								"O documento "+vvproccess.getRequirementDocument().getName()+" se encontra a partir de agora em processo de validação.\n\n");
			}
			catch (AddressException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (MessagingException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			} catch (Exception e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

	private void finalization(String documentName)
	{
		RequirementDocumentDao rdao = (RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocumentDao");
		List<VerifyValidateProccess> vvproccessList = rdao.getVerifyValidateProccessByDocumentName(documentName);

		VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-1);
		Set<User> users = vvproccess.getUsers();

		for (User user : users)
		{
			try
			{
				Email.EnviaEmail(user.getEmail(), "Rq.NET - Notificação sobre o Documento "+vvproccess.getRequirementDocument().getName()+" - Finalização",
								"Caro "+user.getUsername()+",\n\n" +
								"O documento "+vvproccess.getRequirementDocument().getName()+" teve seu processo de verificação ou validação finalizado. " +
								"Ele se encontra agora em aberto para modifiações e inserções.");
```

```
                        }
                        catch (AddressException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (MessagingException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }
}
```

Agente Analisador Léxico (lexical.java)

```
package requisitos.agents;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.regexp.RE;
import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.xml.sax.SAXException;

import requisitos.business.entities.Requirement;
import requisitos.business.entities.Theme;
import requisitos.business.entities.ThemeExcerpts;
import requisitos.stemmer.PortugueseStemmer;
import requisitos.stemmer.Stemmer;
import edu.harvard.wcfia.yoshikonverter.Converter;

@SuppressWarnings("unchecked")
public class Lexical extends Agent implements MessageListener
{
        //
        // Log
        //

        private static final Log LOGGER = LogFactory.getLog(Lexical.class);
```

```
	//
	// Constants
	//

	private static final String STOPWORD = "StOp_WoRd";

	//
	// Agent Interface
	//

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Lexical",this);
	}

	public void boardChanged(Message msg)
	{

	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{
		/*
		 * ============== Serviço - extractRequirements ===================
		 *
		 * Extrai os requisitos de um documento texto, para cada requisito
		 * cria um objeto do tipo Requirement, e retorna todos em uma lista
		 *
		 *
=====================================================================
		 */
		if (service.equals("extractRequirements"))
		{
			// Entrada
			File file = (File)in.get("file");

			// Execução
			List<Requirement> requirementList = extractRequirements(file);

			// Saída
			out.add(requirementList);
		}

		/*
		 *	 ==================	Serviço	-	concordance
=======================
		 *
		 * Dado um tema e um conjunto de requisitos, realiza a concordância
		 * selecionando os trechos em que o tema aparece, junto com as cinco
		 * palavras anteriores e posteriores, utiliza a stopList como filtro
		 * de identificação para palavras irrelevantes.
		 *
		 *
=====================================================================
		 */
		else if (service.equals("concordance"))
```

```
		{
			// Entrada
			Theme theme = (Theme) in.get("theme");
			List<String> stopList = (List<String>) in.get("stopList");
			List<Requirement> requirements = (List<Requirement>)
in.get("requirements");

			// Execução
			ThemeExcerpts excerpts = concordance(theme, stopList, requirements);

			// Retorno
			out.add(excerpts);
		}

		/*
		 *	================ Serviço - requirementsRegexMatcher
==================
		 *
		 * Dado um tema e uma lista de padrões, procura no texto dos requisitos
		 * quando o tema ocorre junto a cada um dos padrões, com uma distância
		 * de 4 ou menos palavras.
		 *
		 * Retorna um mapa em que as chaves são os padrões, e a entrada para
		 * cada padrão é uma lista com os códigos dos requisitos em que o
		 * padrão ocorre junto ao tema.
		 *
		 *
=====================================================================
		 */
		else if (service.equals("requirementsRegexMatcher"))
		{
			// Entrada
			String theme = (String)in.get("theme");
			List<String> patterns = (List<String>) in.get("patterns");
			List<Requirement> requirements = (List<Requirement>)
in.get("requirements");

			// Execução
			Map<String,Set<String>> matchs = requirementsRegexMatcher(theme,
patterns, requirements);

			// Retorno
			out.add(matchs);
		}

		/*
		 *	========================= Serviço - stemmer
==========================
		 *
		 * Dado uma palavra, ou um conjunto de palavras, este serviço retorna
		 * o(s) radicai(s) correspondentes.
		 *
		 *
=====================================================================
		 */
		else if (service.equals("stemmer"))
		{
			// Entrada
			String word = (String)in.get("word");
```

```
                    // Execução
                    Stemmer stem                = new PortugueseStemmer();

                    String  steemedWord = stem.stem(word);

                    // Retorno
                    out.add(steemedWord);
                }

                /*
                 *      ======================      Serviço     -     convertPdf
========================
                 *
                 * Lê o conteúdo de um arquivo pdf, e salva em um novo arquivo texto.
                 * Utiliza a biblioteca YoshikoderConverter
                 *
                 *
=============================================================================
                 */
                else if (service.equals("convertPdf"))
                {
                    // Entrada
                    File pdfFile = (File)in.get("pdfFile");
                    File txtFile = (File)in.get("txtFile");

                    try
                    {
                        // Execução
                        String text = Converter.inhalePdf(pdfFile);
                  dumpText(txtFile, text);
                    }
                    catch (IOException e)
                    {
                        throw new ServiceException("Unable to write text file -
"+txtFile.getAbsolutePath(),e);
                    }
                }

                /*
                 *      ======================      Serviço     -     convertDoc
========================
                 *
                 * Lê o conteúdo de um arquivo doc, e salva em um novo arquivo texto.
                 * Utiliza a biblioteca YoshikoderConverter
                 *
                 *
=============================================================================
                 */
                else if (service.equals("convertDoc"))
                {
                    // Entrada
                    File docFile = (File)in.get("docFile");
                    File txtFile = (File)in.get("txtFile");

                    try
                    {
                        // Execução
                        String text = Converter.inhaleDoc(docFile);
                  dumpText(txtFile, text);
                    }
```

```
                        catch (IOException e)
                        {
                                throw new ServiceException("Unable to write text file -
"+txtFile.getAbsolutePath(),e);
                        }
                }

                /*
                 *      ======================      Serviço     -     convertHtml
=========================
                 *
                 * Lê o conteúdo de um arquivo html, e salva em um novo arquivo texto.
                 * Utiliza a biblioteca YoshikoderConverter
                 *
                 *
=========================================================================
                 */
                else if (service.equals("convertHtml"))
                {
                        // Entrada
                        File htmlFile = (File)in.get("htmlFile");
                        File txtFile = (File)in.get("txtFile");

                        try
                        {
                                // Execução
                                String                  text                    =
Converter.inhaleHtml("file:\\"+htmlFile.getAbsolutePath());
                            dumpText(txtFile, text);
                        }
                        catch (SAXException e)
                        {
                                throw new ServiceException("Unable to read html file -
"+htmlFile.getAbsolutePath(),e);
                        }
                        catch (IOException e)
                        {
                                throw new ServiceException("Unable to write text file -
"+txtFile.getAbsolutePath(),e);
                        }
                }
        }

        //
        //  Methods
        //

        public static List<Requirement> extractRequirements(File fullFile)
        {
                LOGGER.info("Extracting Requirements");

                String line;

                try
                {
                        LOGGER.debug("Opening read and write buffers");

                        String          fullFilePath                            =
fullFile.getAbsolutePath().replaceAll(fullFile.getName(),"")+"//files//";
                        File   fullFilePathFile = new File(fullFilePath);
```

```
fullFilePathFile.mkdir();

BufferedReader br = new BufferedReader(new FileReader(fullFile));

Requirement requirement = null;
List<Requirement> requirementList = new ArrayList<Requirement>();

boolean firstLine = true;
boolean writing = false;

// Lê uma Linha
while ( (line = br.readLine()) != null)
{
	line = line.trim();

	// SE é vazia
	if (line.equals("") || firstLine)
	{
		// Pega o provável título do próximo arquivo
		String title;

		if (firstLine)
		{
			title = line;
			firstLine = false;
		}
		else
		{
			title = br.readLine();
		}


		// SE é um título
		if (!title.equals("") && (title.contains("RF") ||
title.contains("RNF")) )
		{
			// Remove qualquer lixo antes do título
			if (title.contains("RF"))
			{

				while (!title.startsWith("RF"))
					title = title.substring(1);

			}
			else
			{
				while (!title.startsWith("RNF"))
					title = title.substring(1);
			}


			// Troca barras por traços
			title.replaceAll("/","-");

			// Separa código do título
			String[] titleSplit = title.split(" ",2);

			// Cria atributos do requisito
			String reqCode = titleSplit[0];
```

```
                                        String reqTitle = titleSplit[1];

                                        writing = true;

                                        // Adiciona o requisito na lista de saída
                                        requirement = new
Requirement(reqCode,reqTitle);
                                        requirementList.add(requirement);

                                        LOGGER.debug("Requirement Extracted:
"+title);
                                }
                                // SENÃO
                                else
                                {
                                        // Não cria o arquivo
                                        writing = false;
                                }
                        }
                        // SENÃO
                        else
                        {
                                // Caso seja possível adiciona o texto ao arquivo
criado
                                if (writing)
                                {
                                        requirement.addText(line+"\n");
                                }
                        }
                }

                return requirementList;
        }
        catch(IOException e)
        {
                e.printStackTrace();
                return null;
        }
    }

    public static ThemeExcerpts concordance(Theme theme,List<String>
stopList,List<Requirement> requirements)
    {
        String themeName = theme.getName();
        ThemeExcerpts excerpts = new ThemeExcerpts(theme.getName());

        LOGGER.debug("Processando concordancia nos requisitos para o tema:
"+themeName);

        for (Requirement requirement : requirements)
        {
                LOGGER.debug("Requisito: "+requirement.getCode());

                String text = requirement.getText();
                String[] textParagraphs = StringUtils.split(text,"\n");

                for (String paragraph : textParagraphs)
                {
                        String[] textPhrases =
StringUtils.splitByWholeSeparator(paragraph,". ");
```

```
                    for (String phrase : textPhrases)
                    {
                        // Limpa a frase
                        String cleanText = cleanToken(phrase,stopList);

                        // Stemmiza a frase
                        Stemmer stemmer = new PortugueseStemmer();
                        String stemmizedText =
stemmer.stemString(cleanText);

                        // SE a linha contém a expressão OU o pós-buffer não
está vazio
                        int expressionCount =
StringUtils.countMatches(stemmizedText,themeName);

                        for (int k = 0; k < expressionCount; k++)

                        {
                            // Quebra o parágrafo pelo token
                            String[] paragraphPieceArray =
StringUtils.splitByWholeSeparator(stemmizedText,themeName);

                            StringBuilder prePiece = new
StringBuilder();
                            StringBuilder posPiece = new
StringBuilder();

                            for (int i = 0 ; i <= k; i++)
                            {
                                prePiece.append("
"+paragraphPieceArray[i].trim());

                                if (i < k)
                                    prePiece.append("
"+themeName);
                            }

                            for (int i = k+1 ; i <
paragraphPieceArray.length ; i++)
                            {
                                posPiece.append("
"+paragraphPieceArray[i].trim());

                                if (i < paragraphPieceArray.length-
1)
                                    posPiece.append("
"+themeName);
                            }

                            // Recupera o pedaço atual
                            String excerptPrePiece =
prePiece.toString().trim();
                            String[] excerptPrePieceSplit =
excerptPrePiece.split(" ");

                            // Recupera o próximo pedaço
                            String excerptPosPiece =
posPiece.toString().trim();
                            String[] excerptPosPieceSplit =
```

```
excerptPosPiece.split(" ");

                                                // Monta a extração
                                                StringBuilder excerptBuilder = new
StringBuilder();

                                                if (excerptPrePieceSplit.length>=5)
excerptBuilder.append(excerptPrePieceSplit[excerptPrePieceSplit.length-5]+" ");
                                                if (excerptPrePieceSplit.length>=4)
excerptBuilder.append(excerptPrePieceSplit[excerptPrePieceSplit.length-4]+" ");
                                                if (excerptPrePieceSplit.length>=3)
excerptBuilder.append(excerptPrePieceSplit[excerptPrePieceSplit.length-3]+" ");
                                                if (excerptPrePieceSplit.length>=2)
excerptBuilder.append(excerptPrePieceSplit[excerptPrePieceSplit.length-2]+" ");
                                                if (excerptPrePieceSplit.length>=1)
excerptBuilder.append(excerptPrePieceSplit[excerptPrePieceSplit.length-1]+" ");

                                                excerptBuilder.append(themeName+" ");

                                                if (excerptPosPieceSplit.length>=1)
excerptBuilder.append(excerptPosPieceSplit[0]+" ");
                                                if (excerptPosPieceSplit.length>=2)
excerptBuilder.append(excerptPosPieceSplit[1]+" ");
                                                if (excerptPosPieceSplit.length>=3)
excerptBuilder.append(excerptPosPieceSplit[2]+" ");
                                                if (excerptPosPieceSplit.length>=4)
excerptBuilder.append(excerptPosPieceSplit[3]+" ");
                                                if (excerptPosPieceSplit.length>=5)
excerptBuilder.append(excerptPosPieceSplit[4]+" ");

                                                String excerpt = excerptBuilder.toString();

                                                excerpts.addExcerpt(requirement,excerpt);


                                                LOGGER.debug("Extração: "+excerpt);
                                        }
                                }
                        }
                }

                return excerpts;
        }

        public static Map<String,Set<String>> requirementsRegexMatcher(String theme,
List<String> patterns, List<Requirement> requirements)
        {
                Map<String,String>          requirementTexts          =          new
HashMap<String,String>();
                Map<String,Set<String>> requirementAssociations = new HashMap<String,
Set<String>>();

                theme = new PortugueseStemmer().stem(theme);

                try
                {
                        LOGGER.info("Procurando associações entre os padrões e os
requisitos:");

                        // Recupera o texto de cada requisito
```

```
                        for (Requirement requirement : requirements)
                        {

        requirementTexts.put(requirement.getCode(),requirement.getText());
                        }

                        // Cruza cada padrão contra os textos recuperados
                        for (String pattern : patterns)
                        {
                                LOGGER.debug("Padrão: "+pattern);

                                RE regex1 = new
RE("("+theme+"[:alnum:]*){1}(([:blank:]+[:alnum:]+){1,4})?([:blank:]+)("+pattern+"[:alnum:]*)
{1}");
                                RE regex2 = new
RE("("+pattern+"[:alnum:]*){1}(([:blank:]+[:alnum:]+){1,4})?([:blank:]+)("+theme+"[:alnum:]*)
{1}");

                                Set<String> reList = new TreeSet<String>();

                                requirementAssociations.put(pattern,reList);

                                for (Requirement requirement : requirements)
                                {
                                        // Recupera o texto, joga todas as letras para
                                        // minúsculas e limpa acentos
                                        String text =
requirementTexts.get(requirement.getCode());
                                        text = uniformizeText(text);

                                        // Find all the matches.
                                        if(regex1.match(text))
                                        {
                                                int count = regex1.split(text).length-1;

                                                LOGGER.debug("Encontrado "+count+"
vezes em: "+requirement.getCode());
                                                reList.add(requirement.getCode());
                                        }
                                        else if(regex2.match(text))
                                        {
                                                int count = regex2.split(text).length-1;

                                                LOGGER.debug("Encontrado "+count+"
vezes em: "+requirement.getCode());
                                                reList.add(requirement.getCode());
                                        }

                                }
                        }

                        return requirementAssociations;
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                        return null;
                }
        }
```

```
		private static String uniformizeText(String text)
		{
			String fragment = StringUtils.lowerCase(text);

			fragment = fragment.replace('à','a');
			fragment = fragment.replace('á','a');
			fragment = fragment.replace('é','e');
			fragment = fragment.replace('í','i');
			fragment = fragment.replace('ó','o');
			fragment = fragment.replace('ú','u');
			fragment = fragment.replace('ã','a');
			fragment = fragment.replace('õ','o');
			fragment = fragment.replace('â','a');
			fragment = fragment.replace('ê','e');
			fragment = fragment.replace('ô','o');
			fragment = fragment.replace('ü','u');
			fragment = fragment.replace(',',' ');
			fragment = fragment.replace('.',' ');
			fragment = fragment.replace(';',' ');

			return fragment;
		}

		private static String cleanToken(String token, List<String> stopList)
		{
			String temp = token;

			temp = StringUtils.stripStart(temp," ");
			temp = StringUtils.stripEnd(temp," ");
			temp = StringUtils.remove(temp," ┌ ");
			temp = StringUtils.remove(temp,",");
			temp = StringUtils.remove(temp,".");
			temp = StringUtils.remove(temp,":");
			temp = StringUtils.remove(temp,";");
			temp = StringUtils.remove(temp,"[");
			temp = StringUtils.remove(temp,"]");
			temp = StringUtils.remove(temp,"{");
			temp = StringUtils.remove(temp,"}");
			temp = StringUtils.remove(temp,"(");
			temp = StringUtils.remove(temp,")");
			temp = StringUtils.remove(temp,"!");
			temp = StringUtils.remove(temp,"?");
			temp = StringUtils.remove(temp,"/");
			temp = StringUtils.remove(temp,"'");
			temp = StringUtils.remove(temp,"<");
			temp = StringUtils.remove(temp,">");
			temp = StringUtils.remove(temp,"*");
			temp = StringUtils.remove(temp,"\"");
			temp = StringUtils.remove(temp,"\\");
			temp = StringUtils.lowerCase(temp);

			for (String stopWord : stopList)
			{
				temp = StringUtils.replace(temp," "+stopWord+" "," "+STOPWORD+"
");
			}

			return temp;
		}
```

```
	private void dumpText(File txtFile, String text) throws IOException
	{
		BufferedWriter writer = new BufferedWriter(new FileWriter(txtFile));
		writer.write(text);
		writer.close();
	}
}
```

Agente Construtor do Léxico (lexicalConstructor.java)

```
package requisitos.agents;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;
import java.util.Map;

import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;

import corpus.FileTokeniser;

public class LexicalConstructor extends Agent implements MessageListener
{
	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("LexicalConstructor",this);
	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{
		if (service.equals("tagger"))
		{

		}
		/*
		 *	=====================	Serviço	-	tokenizer
=========================
		 *
		 *
		 *
		 *
=========================================================================
		 */
		else if (service.equals("tokenizer"))
		{
			// Entrada
			File srcFile = (File)in.get("srcFile");
			File txtFile = (File)in.get("txtFile");
```

```
                        try
                        {
                                FileTokeniser ft = new
FileTokeniser(srcFile.getAbsolutePath());
                                BufferedWriter writer = new BufferedWriter(new
FileWriter(txtFile));

                                while (ft.hasMoreTokens())
                                {
                                        writer.write(ft.getNextToken()+"\n");
                                }

                                writer.close();
                        }
                        catch (IOException e)
                        {
                                throw new ServiceException("Unable to write text file -
"+txtFile.getAbsolutePath(),e);
                        }
                }
        }

        public void boardChanged(Message msg)
        {

        }
}
```

Agente Manager (manager.java)

```
package requisitos.agents;

import java.util.List;
import java.util.Map;

import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.BoardException;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.midas.as.manager.execution.Logger;

public class Manager extends Agent implements MessageListener
{
        private static String className = "[MANAGER AGENT]  ";

        @Override
        protected void lifeCycle() throws LifeCycleException, InterruptedException
        {
                // Se registra no BlackBoard
                Board.addMessageListener("Manager",this);
        }

        @Override
        public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
        {
```

```
	}

	public void boardChanged(Message msg)
	{
		String title = msg.getTitle();

		if ( title.equals("PreVerificationStage") )
		{
			String documentName = msg.getContent();

			// Notifica os outros agentes
			Logger.addEntry(className+"Document "+msg.getContent()+" is
under PreVerificationStage, notifiyng other Agents...",true);

			try
			{

	Board.writeOnBoard(1,"Communicator","PreVerificationStage",documentName,this);

	Board.writeOnBoard(1,"Verifier","PreVerificationStage",documentName,this);
			}
			catch (BoardException e)
			{
				e.printStackTrace();
			}
		}
		if ( title.equals("VerificationStage") )
		{
			String documentName = msg.getContent();

			// Notifica os outros agentes
			Logger.addEntry(className+"Document "+msg.getContent()+" is
under VerificationStage, notifiyng other Agents...",true);

			try
			{

	Board.writeOnBoard(1,"Communicator","VerificationStage",documentName,this);

			}
			catch (BoardException e)
			{
				e.printStackTrace();
			}
		}
		if ( title.equals("PreValidationStage") )
		{
			String documentName = msg.getContent();

			// Notifica os outros agentes
			Logger.addEntry(className+"Document "+msg.getContent()+" is
under PreValidationStage, notifiyng other Agents...",true);

			try
			{

	Board.writeOnBoard(1,"Communicator","PreValidationStage",documentName,this);

			}
			catch (BoardException e)
```

```
                        {
                                e.printStackTrace();
                        }
                }
                if ( title.equals("ValidationStage") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Document     "+msg.getContent()+"     is
under ValidationStage, notifiyng other Agents...",true);

                        try
                        {

        Board.writeOnBoard(1,"Communicator","ValidationStage",documentName,this);

                        }
                        catch (BoardException e)
                        {
                                e.printStackTrace();
                        }
                }
                if ( title.equals("Finalization") )
                {
                        String documentName = msg.getContent();

                        // Notifica os outros agentes
                        Logger.addEntry(className+"Document    "+msg.getContent()+"    has
been released, notifiyng other Agents...",true);

                        try
                        {

        Board.writeOnBoard(1,"Communicator","Finalization",documentName,this);

                        }
                        catch (BoardException e)
                        {
                                e.printStackTrace();
                        }
                }
        }
}
```

Agente Observador (observer.java)

```
package requisitos.agents;

import java.util.List;
import java.util.Map;

import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.midas.as.manager.execution.Logger;

public class Observer extends Agent implements MessageListener
```

```
{
	private static String className = "[OBSERVER AGENT]  ";

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Observer",this);
	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{

	}

	public void boardChanged(Message msg)
	{
		String title = msg.getTitle();

		if ( title.equals("Document Created") ||
			 title.equals("Document Removed") ||
			 title.equals("Document Altered")  )
		{
			// Notifica os outros agentes
			Logger.addEntry(className+msg.getContent(),true);
		}

//		// Se o documento de requisitos foi alterado
//		if (message.equals("Document Altered"))
//		{
//			// Notifica os outros agentes
//			Board.writeForAll("Document Altered",this);
//		}
//
//		// Se um documento de requisitos foi criado
//		else if (message.equals("Document Created"))
//		{
//			// Notifica os outros agentes
//			Board.writeForAll("Document Created",this);
//		}
//
//		// Se o documento de requisitos foi removido
//		else if (message.equals("Document Removed"))
//		{
//			// Notifica os outros agentes
//			Board.writeForAll("Document Removed",this);
//		}
	}
}
```

Agente Estatístico (statistical.java)

```
package requisitos.agents;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

```
import java.util.Map.Entry;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;

import requisitos.business.entities.FrequencyMap;
import requisitos.business.entities.Requirement;
import requisitos.business.entities.ThemeExcerpts;
import requisitos.business.entities.ThemeMetrics;

@SuppressWarnings("unchecked")
public class Statistics extends Agent implements MessageListener
{
	//
	// Log
	//

	private static final Log LOGGER = LogFactory.getLog(Statistics.class);

	//
	// Constants
	//

	private static final String STOPWORD = "stop_word";

	//
	// Agent Interface
	//

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Statistics",this);
	}

	public void boardChanged(Message msg)
	{

	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{
		/*
		 *	==================	Serviço	-	frequencyMap
========================
		 *
		 * Dadas as extrações do Concordanciador e o texto completo com os
		 * requisitos, o serviço de frequencyMap calcula o Mapa de Frequência
		 * completo para o tema.
		 *
```

```
                *
================================================================
                */
                if (service.equals("frequencyMap"))
                {
                        // Entrada
                        ThemeExcerpts themeExcerpts = (ThemeExcerpts)
in.get("themeExcerpts");
                        List<Requirement> requirements = (List<Requirement>)
in.get("requirements");

                        // Execução
                        FrequencyMap frequencyMap = frequencyMap(themeExcerpts,
requirements);

                        // Saída
                        out.add(frequencyMap);
                }

                /*
                *       ====================== Serviço - metrics
==========================
                *
                * Dado o mapa de frequência, o serviço de metrics calcula os valores
                * de scoreT e informaçãoMútua para todos os termos que ocorreram
                * junto ao tema.
                *
                *
================================================================
                */
                else if (service.equals("metrics"))
                {
                        // Entrada
                        FrequencyMap frequencyMap = (FrequencyMap)
in.get("frequencyMap");

                        // Execução
                        ThemeMetrics themeMetrics = metrics(frequencyMap);

                        // Saída
                        out.add(themeMetrics);
                }
        }

        //
        // Methods
        //

        public static FrequencyMap frequencyMap(ThemeExcerpts
themeExcerpts,List<Requirement> requirements)
        {
                LOGGER.info("Calculando mapa de frequência");

                //
                // Saídas
                //

                int corpusCount = 0;
                                                        // N
                int tokenFrequency = 0;
```

```
                                                    // f(n)
                Map<String,Integer> nodesOcurrenceMap = new HashMap<String,Integer>();
        // f(c)[]
                Map<String,Integer>             nodesCoOcurrenceMap             =             new
HashMap<String,Integer>();          // f(n,c)[]
                Map<String,List<String>>          requirementOcurrenceMap          =          new
HashMap<String,List<String>>();

                //
                // Calcula f(n) e f(n,c)
                //

                tokenFrequency = themeExcerpts.getAllExcerpts().size();

                for (String code : themeExcerpts.getRequirementCodes())
                {
                        LOGGER.debug("Requisito: "+code);

                        // CO-OCURRENCE
                        Map<String,Integer>       reqNodesCoOcurrenceMap       =       new
HashMap<String,Integer>();          // f(n,c)[]

                        for (String excerpt : themeExcerpts.getExcerpts(code))
                        {
                                String[]                    tokens      = excerpt.split(" ");
                                Map<String,Boolean>          goneTokens          =          new
HashMap<String,Boolean>();

                                for (int i = 0; i < tokens.length; i++)
                                {
                                        String token = cleanToken(tokens[i]);

                                        if              (isTokenValid(token,true)              &&
!token.equals(themeExcerpts.getTheme()))
                                        {
                                                if
(reqNodesCoOcurrenceMap.containsKey(token))
                                                {

                                                        if
(!goneTokens.containsKey(token))
                                                        {

        reqNodesCoOcurrenceMap.put(token,reqNodesCoOcurrenceMap.get(token) + 1);

        goneTokens.put(token,true);
                                                        }
                                                }
                                                else
                                                {


        reqNodesCoOcurrenceMap.put(token,1);
                                                        goneTokens.put(token,true);
                                                }

                                                if
(requirementOcurrenceMap.containsKey(token))
                                                {
                                                        List<String>            reqs            =
```

```
requirementOcurrenceMap.get(token);
                                                        reqs.add(","+code);

                                                }

                                                else
                                                {
                                                        List<String>    reqs    =    new
ArrayList<String>();

        requirementOcurrenceMap.put(token,reqs);

                                                        reqs.add(code);
                                                }
                                        }
                                }
                        }

                        for             (Entry<String,Integer>          entry           :
reqNodesCoOcurrenceMap.entrySet())
                        {
                                LOGGER.debug(entry.getKey()+": "+entry.getValue());
                        }

                        for             (Entry<String,Integer>          entry           :
reqNodesCoOcurrenceMap.entrySet())
                        {
                                if (nodesCoOcurrenceMap.containsKey(entry.getKey()))
                                {
                                        Integer    newValue    =    entry.getValue()    +
nodesCoOcurrenceMap.get(entry.getKey());

        nodesCoOcurrenceMap.put(entry.getKey(),newValue);
                                }
                                else
                                {

        nodesCoOcurrenceMap.put(entry.getKey(),entry.getValue());

                                }
                        }
                }

                //
                // Calcula f(c) e N
                //

                LOGGER.debug("No texto todo: ");

                String text = getText(requirements);

                Set<String> nodesSet = nodesCoOcurrenceMap.keySet();
                String[] tokens = StringUtils.split(text," ");

                corpusCount = tokens.length;

                for (String node : nodesSet)
                {
                        int count = StringUtils.countMatches(text,node);
```

```
                    LOGGER.debug(node+": "+count);
                    nodesOcurrenceMap.put(node,count);

                }

                //
                // Adiciona na saida
                //

                FrequencyMap frequencyMap = new FrequencyMap();

                frequencyMap.setCorpusCount(corpusCount);
                frequencyMap.setTokenFrequency(tokenFrequency);
                frequencyMap.setNodesOcurrenceMap(nodesOcurrenceMap);
                frequencyMap.setNodesCoOcurrenceMap(nodesCoOcurrenceMap);

                return frequencyMap;
        }

        public static ThemeMetrics metrics (FrequencyMap frequencyMap)
        {
                //
                // Entradas
                //

                int corpusCount = frequencyMap.getCorpusCount();
                int tokenFrequency = frequencyMap.getTokenFrequency();
                Map<String,Integer>                nodesOcurrenceMap                =
frequencyMap.getNodesOcurrenceMap();
                Map<String,Integer>                nodesCoOcurrenceMap                =
frequencyMap.getNodesCoOcurrenceMap();

                Set<String> nodesOcurrenceSet = nodesOcurrenceMap.keySet();

                //
                // Saídas
                //

                Map<String,Double> tScoreMap = new HashMap<String,Double>();
// scoreT[]
                Map<String,Double>        mutuosInformationMap        =        new
HashMap<String,Double>(); // informacaoMutua[]

                for (String node: nodesOcurrenceSet)
                {
                        int nodeOcurrence = nodesOcurrenceMap.get(node);   // f(c)
                        int nodeCoOcurrence = nodesCoOcurrenceMap.get(node); // f(n,c)

                        //
                        // scoreT = ( f(n,c) – Y ) / X
                        //
                        // X = f(n,c)^1/2
                        // Y = ( f(n) f(c) / N )
                        //

                        // Calcula X = f(n,c)^1/2
                        double x = (double)Math.sqrt(nodeCoOcurrence);

                        // Calcula Y = ( f(n) f(c) / N )
                        double y = (double)tokenFrequency * nodeOcurrence / corpusCount;
```

```
                                        // Calcula scoreT = ( f(n,c) – Y ) / X
                                        double scoreT = (double)(nodeCoOcurrence-y)/x;

                                        //
                                        // mutuosInformation = log2 [ A / B ]
                                        //
                                        // A = (f(n,c) / N)
                                        // B = (f(n)/N) * (f(c)/ N)
                                        //

                                        // Calcula A = (f(n,c) / N)
                                        double a = (double)nodeCoOcurrence/corpusCount;

                                        // Calcula B = (f(n)/N) * (f(c)/ N)
                                        double b =
((double)tokenFrequency/corpusCount)*((double)nodeOcurrence/corpusCount);

                                        // Calcula mutuosInformation = log2 [ A / B ]
                                        double mutuosInformation = Math.log(a/b)/Math.log(2.0);

                                        //
                                        // Adiciona nos mapas
                                        //

                                        tScoreMap.put(node,scoreT);
                                        mutuosInformationMap.put(node,mutuosInformation);

                                }

                                ThemeMetrics metrics = new ThemeMetrics();

                                metrics.setTScoreMap(tScoreMap);
                                metrics.setMutuosInformationMap(mutuosInformationMap);

                                return metrics;
                        }

                        private static String getText(List<Requirement> requirements)
                        {
                                StringBuilder textBuilder = new StringBuilder();

                                for (Requirement requirement : requirements)
                                {
                                        String fragment = StringUtils.lowerCase(requirement.getText());

                                        fragment = fragment.replace('à','a');
                                        fragment = fragment.replace('á','a');
                                        fragment = fragment.replace('é','e');
                                        fragment = fragment.replace('í','i');
                                        fragment = fragment.replace('ó','o');
                                        fragment = fragment.replace('ú','u');
                                        fragment = fragment.replace('ã','a');
                                        fragment = fragment.replace('õ','o');
                                        fragment = fragment.replace('â','a');
                                        fragment = fragment.replace('ê','e');
                                        fragment = fragment.replace('ô','o');
                                        fragment = fragment.replace('ü','u');

                                        textBuilder.append(fragment+"\n");
```

```
		}

		return textBuilder.toString();
	}

	private static boolean isTokenValid(String token,boolean caseSensitive)
	{
		if (caseSensitive)
			token = StringUtils.lowerCase(token);

		if (token.equals(STOPWORD))
			return false;
		if (token.startsWith("rf"))
			return false;
		if (token.startsWith("rnf"))
			return false;
		if (token.equals("-"))
			return false;
		if (token.startsWith("&"))
			return false;
		if (token.equals(""))
			return false;
		if (token.equals(" "))
			return false;
		if (token.startsWith(" "))
			return false;
		if (token.startsWith("="))
			return false;
		if (token.endsWith(" "))
			return false;

		return true;
	}

	private static String cleanToken(String token)
	{
		String temp = token;

		temp = StringUtils.trim(temp);
		temp = StringUtils.remove(temp,",");
		temp = StringUtils.remove(temp,".");
		temp = StringUtils.remove(temp,":");
		temp = StringUtils.remove(temp,";");
		temp = StringUtils.remove(temp,"[");
		temp = StringUtils.remove(temp,"]");
		temp = StringUtils.remove(temp,"{");
		temp = StringUtils.remove(temp,"}");
		temp = StringUtils.remove(temp,"(");
		temp = StringUtils.remove(temp,")");
		temp = StringUtils.remove(temp,"!");
		temp = StringUtils.remove(temp,"?");
		temp = StringUtils.remove(temp,"/");
		temp = StringUtils.remove(temp,"'");
		temp = StringUtils.remove(temp,"<");
		temp = StringUtils.remove(temp,">");
		temp = StringUtils.remove(temp,"+");
		temp = StringUtils.remove(temp,"\"");
		temp = StringUtils.remove(temp,"\\");

		return temp;
```

```
	}
}
```

Agente Gerador de Visões (mapper.java)

```
package requisitos.agents;

import java.io.ByteArrayOutputStream;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.tmapi.core.Association;
import org.tmapi.core.Locator;
import org.tmapi.core.TMAPIException;
import org.tmapi.core.Topic;
import org.tmapi.core.TopicMap;
import org.tmapi.core.TopicMapSystem;
import org.tmapi.core.TopicMapSystemFactory;
import org.tmapi.index.core.TopicsIndex;
import org.tmapiutils.impexp.xtm.XTMSerializer;

import requisitos.business.entities.Requirement;
import requisitos.stemmer.PortugueseStemmer;

@SuppressWarnings("unchecked")
public class TopicMapper extends Agent implements MessageListener
{
	//
	// Log
	//

	private static final Log LOGGER = LogFactory.getLog(TopicMapper.class);

	//
	// Agent Interface
	//

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Lexical",this);
	}

	public void boardChanged(Message msg)
	{

	}

	@Override
```

```
	public void provide(String service, Map<String, Object> in, List<Object> out) throws ServiceException
	{
		/*
		 *	======================	Serviço	-	build	============================
		 *
		 * Este serviço cria um mapa de tópicos a partir de uma lista de
		 * requisitos.
		 *
		 *
		 ============================================================================
		 */
		if (service.equals("build"))
		{
			// Entrada
			List<Requirement>	requirements	=	(List<Requirement>) in.get("requirements");

			// Execução
			TopicMap topicMap = build(requirements);

			// Saída
			out.add(topicMap);
		}

		/*
		 *	==================	Serviço	-	setAssociations	======================
		 *
		 * Este serviço cria as associações entre os pares tema/termo e os
		 * requisitos, tem como entrada o tema, o mapa de tópicos com os
		 * requisitos, e o mapa de associações que indica para cada termo que
		 * ocorre junto ao tema a lista de requisitos relacionada.
		 *
		 *
		 ============================================================================
		 */
		else if (service.equals("setAssociations"))
		{
			// Entrada
			String theme = (String) in.get("theme");
			TopicMap topicMap = (TopicMap) in.get("topicMap");
			Map<String,Set<String>>	associationsMap	=	(Map<String, Set<String>>) in.get("associationsMap");

			// Execução
			TopicMap	newTopicMap	=	setAssociations(topicMap, associationsMap, theme);

			// Saída
			out.add(newTopicMap);
		}
	}


	//
	// Methods
	//
```

```
	public static TopicMap build(List<Requirement> requirements)
	{
		LOGGER.info("Mapping Requirements...");

		try
		{
			////////////////////////////////////////////////////////
			// 1. Cria Mapa Vazio
				//
			////////////////////////////////////////////////////////

			LOGGER.debug("Make empty topic map");

			TopicMapSystemFactory tSystemFactory =
TopicMapSystemFactory.newInstance();
			TopicMapSystem tSystem =
	tSystemFactory.newTopicMapSystem();
			TopicMap topicMap
	=	tSystem.createTopicMap("http://www.tmapi.org/examples/exampletm");

			////////////////////////////////////////////////////////
			// 2. Inicializa Tópicos Básicos
		//
			////////////////////////////////////////////////////////

			LOGGER.debug("Make basic topics");

			Topic baseRequirementTopic = topicMap.createTopic();

			Topic inverseRequirementTopic = topicMap.createTopic();
			Topic functionalRequirementTopic = topicMap.createTopic();
			Topic nonFunctionalRequirementTopic = topicMap.createTopic();

			baseRequirementTopic.createTopicName("Requisito",null);

	baseRequirementTopic.addSubjectIdentifier(topicMap.createLocator("locator:/baseRequi
rement"));

			inverseRequirementTopic.addType(baseRequirementTopic);
			inverseRequirementTopic.createTopicName("Requisito Inverso",null);

	inverseRequirementTopic.addSubjectIdentifier(topicMap.createLocator("locator:/baseRe
quirement/inverseRequirement"));

			functionalRequirementTopic.addType(baseRequirementTopic);
			functionalRequirementTopic.createTopicName("Requisito
Funcional",null);

	functionalRequirementTopic.addSubjectIdentifier(topicMap.createLocator("locator:/base
Requirement/functionalRequirement"));

			nonFunctionalRequirementTopic.addType(baseRequirementTopic);
			nonFunctionalRequirementTopic.createTopicName("Requisito Não
Funcional",null);

	nonFunctionalRequirementTopic.addSubjectIdentifier(topicMap.createLocator("locator:/
baseRequirement/nonfunctionalRequirement"));

			Topic textOccurrenceTopic = topicMap.createTopic();
			textOccurrenceTopic.createTopicName("Texto",null);
```

```
		textOccurrenceTopic.addSubjectIdentifier(topicMap.createLocator("locator:/baseRequire
ment/textOcurrence"));

				//////////////////////////////////////////////////////////
				// 3. Cria tópicos dos requisitos
			//
				//////////////////////////////////////////////////////////

				LOGGER.debug("Make extracted requirements topics");

				// PARA cada arquivo
				for (Requirement requirement : requirements)
				{
					String   requirementId  = requirement.getCode();
					String   requirementName  = requirement.getTitle();

					// Cria o tópico
					Topic classTopic;
					Topic requirementTopic = topicMap.createTopic();

					if	 (requirementId.startsWith("RF"))
						classTopic = functionalRequirementTopic;
					else if (requirementId.startsWith("RNF"))
						classTopic = nonFunctionalRequirementTopic;
					else if (requirementId.startsWith("RI"))
						classTopic = inverseRequirementTopic;
					else // TIPO NÃO IDENTIFICADO...
						continue;

					requirementTopic.addType(classTopic);
					requirementTopic.createTopicName(requirementId+"
"+requirementName,null);

		requirementTopic.addSubjectIdentifier(topicMap.createLocator("locator:/baseRequireme
nt#"+requirementId));

		requirementTopic.createOccurrence(requirement.getText(),textOccurrenceTopic,null);

					LOGGER.debug("Created topic - "+requirementId);
				}

				//////////////////////////////////////////////////////////
				// 5. Adiciona o mapa a lista de saida
			//
				//////////////////////////////////////////////////////////
				return topicMap;
			}
			catch(TMAPIException e)
			{
				e.printStackTrace();
				LOGGER.error("Internal error while creating topic map");
				return null;
			}
		}

		public static TopicMap setAssociations(TopicMap topicMap,Map<String,Set<String>>
associationsMap,String theme)
		{
			LOGGER.info("Mapping Requirements Associations...");
```

```
                // Stemmiza tema principal
                theme = new PortugueseStemmer().stem(theme);

                // Recupera subtemas
                Collection<String> subthemes = associationsMap.keySet();

                try
                {
                        ////////////////////////////////////////////////////////
                        // 1. Recupera Mapa        e Índice
                        //
                        ////////////////////////////////////////////////////////

                        TopicsIndex                topicMapIndex        =        (TopicsIndex)
topicMap.getHelperObject(org.tmapi.index.core.TopicsIndex.class);

                        if(topicMapIndex.getFlags().isAutoUpdated()==false)
                                topicMapIndex.reindex();

                        ////////////////////////////////////////////////////////
                        // 2. Inicializa Tópicos Básicos
                //
                        ////////////////////////////////////////////////////////
                        LOGGER.debug("Make basic topics");

                        // Tópico Tema
                        Topic baseThemeTopic = topicMap.createTopic();
                        baseThemeTopic.createTopicName("Tema",null);

                        // Tópico Subtema
                        Topic baseSubthemeTopic = topicMap.createTopic();
                        baseSubthemeTopic.createTopicName("Subtema",null);

                        Topic themeSubthemeAssociationTopic = topicMap.createTopic();

        themeSubthemeAssociationTopic.createTopicName("Relacionamento",null);

                        Topic associationTopic = topicMap.createTopic();
                        associationTopic.createTopicName("Relacionamento",null);

                        Topic associationRoleTopic = topicMap.createTopic();
                        associationRoleTopic.createTopicName("Relacionado A",null);

                        ////////////////////////////////////////////////////////
                        // 4. Cria tópico do tema
                        //
                        ////////////////////////////////////////////////////////

                        Topic themeTopic = topicMap.createTopic();
                        themeTopic.createTopicName(theme,null);
                        themeTopic.addType(baseThemeTopic);

                        ////////////////////////////////////////////////////////
                        // 4. Cria associações tema-subtema e subtema-requisito
                //
                        ////////////////////////////////////////////////////////

                        // PARA cada subtema
                        for (String subtheme : subthemes)
```

```
                {
                        LOGGER.debug("Mounting associations for subtheme: "+subtheme);

                        //
                        // 4.1. tema-subtema
                        //

                        Topic subthemeTopic = topicMap.createTopic();
                        subthemeTopic.createTopicName(theme+"_"+subtheme,null);
                        subthemeTopic.addType(themeTopic);

                        // Recupera requisitos associados
                        Collection<String> associatedRequirements = associationsMap.get(subtheme);

                        // Inicializa requisito
                        Topic requirementTopic = null;

                        // PARA cada requisito
                        for (String requirementId : associatedRequirements)
                        {
                                LOGGER.debug("- "+requirementId);

                                Locator subjectIdentifier = topicMap.createLocator("locator:/baseRequirement#"+requirementId.replaceAll(".txt",""));
                                requirementTopic = topicMapIndex.getTopicBySubjectIdentifier(subjectIdentifier);

                                Association subthemeRequirementAssociation = topicMap.createAssociation();


        subthemeRequirementAssociation.setType(associationTopic);

        subthemeRequirementAssociation.createAssociationRole(subthemeTopic,associationRoleTopic);

        subthemeRequirementAssociation.createAssociationRole(requirementTopic,associationRoleTopic);

                        }
                }

                ////////////////////////////////////////////////////////////
                // 5. Adiciona o mapa a lista de saida
        //
                ////////////////////////////////////////////////////////////
                return topicMap;
        }
        catch(TMAPIException e)
        {
                e.printStackTrace();
                return null;
        }
}

public static String serialize(TopicMap topicMap)
{
        // Cria stream de saída
```

```
                ByteArrayOutputStream xmlOutputStream = new ByteArrayOutputStream();

                try
                {
                        ////////////////////////////////////////////////////////
                        // 1. Serializa o mapa
                                //
                        ////////////////////////////////////////////////////////

                        XTMSerializer xtmS = new XTMSerializer();
                        xtmS.serialize(xmlOutputStream, topicMap);

                        ////////////////////////////////////////////////////////
                        // 2. Adiciona o mapa a lista de saida
                //
                        ////////////////////////////////////////////////////////
                        return xmlOutputStream.toString();
                }
                catch(TMAPIException e)
                {
                        e.printStackTrace();
                        return null;
                }
        }
}
```

Agente Rastreador (tracker.java)

```
package requisitos.agents;

import java.util.List;
import java.util.Map;

import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;

public class Tracker extends Agent implements MessageListener
{
        @Override
        protected void lifeCycle() throws LifeCycleException, InterruptedException
        {
                // Se registra no BlackBoard
                Board.addMessageListener("Tracker",this);
        }

        @Override
        public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
        {

        }

        public void boardChanged(Message msg)
        {

        }
}
```

Agente Validador (validator.java)

```
package requisitos.agents;

import java.util.List;
import java.util.Map;

import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;

public class Validator extends Agent implements MessageListener
{
	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Validator",this);
	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{

	}

	public void boardChanged(Message msg)
	{

	}
}
```

Agente Verificador (verifier.java)

```
package requisitos.agents;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.DecimalFormat;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.ResourceBundle;
import java.util.Set;

import javax.mail.MessagingException;
import javax.mail.internet.AddressException;

import org.midas.as.AgentServer;
import org.midas.as.agent.board.Board;
import org.midas.as.agent.board.Message;
import org.midas.as.agent.board.MessageListener;
```

```
import org.midas.as.agent.templates.Agent;
import org.midas.as.agent.templates.LifeCycleException;
import org.midas.as.agent.templates.ServiceException;
import org.midas.as.manager.execution.Logger;

import requisitos.business.entities.Requirement;
import requisitos.business.entities.RequirementDocument;
import requisitos.business.entities.User;
import requisitos.business.entities.VerifyValidateProccess;
import requisitos.dao.RequirementDocumentDao;
import requisitos.util.CalculateSimilarity;
import requisitos.util.ValuedAttribute;
import requisitos.web.email.Email;

public class Verifier extends Agent implements MessageListener
{
	private static String className = "[VERIFIER AGENT]  ";

	ResourceBundle resourceBundle = ResourceBundle.getBundle("core");

	@Override
	protected void lifeCycle() throws LifeCycleException, InterruptedException
	{
		// Se registra no BlackBoard
		Board.addMessageListener("Verifier",this);
	}

	@Override
	public void provide(String service, Map<String, Object> in, List<Object> out) throws
ServiceException
	{

	}

	public void boardChanged(Message msg)
	{
		String title = msg.getTitle();

		if ( title.equals("PreVerificationStage") )
		{
			String documentName = msg.getContent();

			// Notifica os outros agentes
			Logger.addEntry(className+"Running similarity tests on
"+documentName,true);
			try
			{
				calculateSimilarity(documentName);
				Logger.addEntry(className+"Similarity tests on
"+documentName+" completed, notifying Communicator Agent",true);

	Board.writeOnBoard(1,"Communicator","SimilarityTest",documentName,this);
			}
			catch (Exception e)
			{
				Logger.addEntry(className+"Error while running similarity
tests on "+documentName,false);
				e.printStackTrace();
			}
```

```
		}
	}

	private void calculateSimilarity(String documentName) throws Exception
	{
		RequirementDocumentDao requirementDao = (RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocumentDao");

		// 1. Recupera o diretório do perl e do pretex
		String perlPath = resourceBundle.getString("perl.path")+"\\bin";
		String pretexPath = resourceBundle.getString("pretex.path");

		// 2. Limpa os diretórios docs,stembase e discover
		cleanDirectories(pretexPath);
		System.out.println("path pretex "+pretexPath);

		// 3. Cria os arquivos de requisito
		RequirementDocument document = requirementDao.getRequirementDocumentByName(documentName);
		Set<Requirement> requirements = document.getRequirements();
		createRequirementFiles(pretexPath,requirements);

		// 4. Invoca o processo "stem.pl"
		invokeStem(pretexPath);

		// 5. Invoca o processo "report.pl"
		invokeReport(pretexPath);

		// 6. Calcula as similaridades
		List<ValuedAttribute> similarityList = calculateSimilarity();

		// 7. Monta e envia o relatório
		sendEmail(documentName,similarityList);
	}

	private void sendEmail(String documentName, List<ValuedAttribute> similarityList)
	{
		String report = generateReport(similarityList);

		RequirementDocumentDao rdao = (RequirementDocumentDao)AgentServer.getApplicationContext().getBean("requirementDocumentDao");
		List<VerifyValidateProccess> vvproccessList = rdao.getVerifyValidateProccessByDocumentName(documentName);

		VerifyValidateProccess vvproccess = vvproccessList.get(vvproccessList.size()-1);
		Set<User> users = vvproccess.getUsers();

		for (User user : users)
		{
			try
			{
				Email.EnviaEmail(user.getEmail(), "Rq.NET - Relatório de Requisitos Similares em "+vvproccess.getRequirementDocument().getName(),
					"Caro(a) "+user.getUsername()+",\n\n" +
					"Os testes de similaridade em  "+vvproccess.getRequirementDocument().getName()+" foram concluídos.\n\n"+
```

```
                                                                    "O sistema encontrou
"+similarityList.size()+" pares de requisitos similares, listados abaixo, do par mais "+
                                                                    "similar, para o menos
similar: \n\n"+ report);
                        }
                        catch (AddressException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (MessagingException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }

        }

        private String generateReport(List<ValuedAttribute> similarityList)
        {
                System.out.println(className+" Gerando Relatório");

                DecimalFormat formatter = new DecimalFormat("0.00");

                String report = "";

                report += "Similaridade\tRequisitos\n";

                Collections.sort(similarityList);
                Collections.reverse(similarityList);

                for (ValuedAttribute<String> va : similarityList)
                {
                        Double factor = va.getAttributeValue();
                        String reqs = va.getAttribute();

                        report += formatter.format(factor)+"\t\t"+reqs+"\n";
                }

                return report;
        }

        private List<ValuedAttribute> calculateSimilarity()
        {
                System.out.println(className+" Calculando Similaridades");
                return new CalculateSimilarity().calculateSimilarity();
        }

        private void invokeReport(String pretexPath) throws IOException,InterruptedException
        {
                System.out.println(className+" Rodando report.pl");

                //Roda "report.pl"
                Runtime runtime = Runtime.getRuntime();
                Process proc = runtime.exec("cmd.exe /c report.bat C:\\Perl\\bin\\ 2> nul: 1>
nul:",new String[]{},new File(pretexPath));

                InputStream inputstream = proc.getInputStream();
                InputStreamReader inputstreamreader = new InputStreamReader(inputstream);
```

```
                BufferedReader    bufferedreader    = new BufferedReader(inputstreamreader);

                String line;

                while ((line = bufferedreader.readLine()) != null)
                {
//                      System.out.println(line);
                }

                proc.waitFor();
        }

        private void invokeStem(String pretexPath) throws IOException,InterruptedException
        {
                System.out.println(className+" Rodando stem.pl");

                // Roda "stem.pl"
                Runtime runtime = Runtime.getRuntime();
                Process proc = runtime.exec("cmd.exe /c stem.bat C:\\Perl\\bin\\ 2> nul: 1>
nul:",new String[]{},new File(pretexPath));

                InputStream          inputstream          = proc.getInputStream();
                InputStreamReader inputstreamreader = new InputStreamReader(inputstream);
                BufferedReader    bufferedreader    = new BufferedReader(inputstreamreader);

                String line;

                while ((line = bufferedreader.readLine()) != null)
                {
//                      System.out.println(line);
                }

                proc.waitFor();
        }

        private void createRequirementFiles(String pretexPath,Set<Requirement> requirements)
throws IOException
        {
                System.out.println(className+" Criando arquivos de requisitos");

                for (Requirement requirement : requirements)
                {
                        File                reqFile                =                new
File(pretexPath+"//docs//"+requirement.getCode()+".txt");
                        BufferedWriter writer  = new BufferedWriter(new FileWriter(reqFile));

                        writer.append(requirement.getCode()+"
"+requirement.getTitle()+"\n"+requirement.getText());
                        writer.close();
                }
        }

        private void cleanDirectories(String pretexPath)
        {
                System.out.println(className+" Limpando Diretórios");

                File docsDir = new File(pretexPath+"//docs");
                File[] docsFiles = docsDir.listFiles();

                for (File file : docsFiles)
```

```
                {
                        file.delete();
                }

                File stembaseDir = new File(pretexPath+"//stembase");
                File[] stembaseFiles = docsDir.listFiles();

                for (File file : docsFiles)
                {
                        file.delete();
                }

                File discoverDir = new File(pretexPath+"//discover");
                File[] discoverFiles = docsDir.listFiles();
                System.out.println(className+"discoverDir "+discoverDir);
                for (File file : docsFiles)
                {
                        file.delete();
                }
        }
}
```

Anexo A
Stopwords utilizadas

a	à	abaixo	acaso	acerca
acima	acola	acolá	ademais	adentro
adiante	afinal	afora	agora	agorinha
ai	aí	ainda	alem	além
algo	alguem	alguém	algum	alguma
algumas	alguns	ali	alias	aliás
amiude	amiúde	ante	antes	ao
aonde	aos	apenas	apesar	apos
após	apud	aquela	àquela	aquelas
àquelas	aquele	àquele	aqueles	àqueles
aqui	aquilo	àquilo	as	às
assim	ate	até	atras	atrás
atraves	através	basicamente	bastante	bastantes
bem	bom	ca	cá	cada
cade	cadê	cadern	caso	cem
certa	certamente	certas	certeiramente	certo
certos	chez	cinco	cinquenta	com
comigo	como	comumente	conforme	confronte
conosco	conquanto	consequentemente	conseqüentemente	consigo
consoante	contanto	contigo	contra	contudo
convosco	cuja	cujas	cujo	cujos
da	dai	daí	dali	dantes
daquela	daquelas	daquele	daqueles	daqui
daquilo	das	de	debaixo	defronte
dela	delas	dele	deles	demais
dentre	dentro	depois	desde	dessa
dessas	desse	desses	desta	destas
deste	destes	detras	detrás	deveras
dez	dezenove	dezesseis	dezessete	dezoito
diante	disso	disto	diversos	do
dois	donde	doravante	dos	doze
duas	dum	duma	dumas	duns
durante	duzentos	e	é	eis
ela	elas	ele	eles	em
embaixo	embora	enfim	enquanto	entanto
entao	então	entre	entretanto	essa
essas	esse	esses	esta	estas
este	estes	eu	exatamente	exceto
exceto	felizmente	fora	frequentemente	freqüentemente
graças	hoje	ibidem	idem	in
inclusive	inda	infelizmente	inicialmente	isso
isto	ja	já	jamais	la
lá	largamente	lha	lhas	lhe
lhes	lho	lhos	logo	mais
mal	malgrado	mas	me	mediante
melhor	menos	meramente	mesma	mesmas
mesmo	mesmos	meu	meus	mil
milhao	mim	minha	minhas	mui

muita	muitas	muitissimo	muitíssimo	muito
muitos	mutuamente	na	nada	nadinha
nalgum	nalguma	nalgumas	nalguns	nao
não	naquela	naquelas	naquele	naqueles
naquilo	nas	nela	nelas	nele
neles	nem	nenhum	nenhuma	nessa
nessas	nesse	nesses	nesta	nestas
neste	nestes	ninguem	ninguém	nisso
nisto	no	nos	nós	nossa
nossas	nosso	nossos	noutra	noutras
noutro	noutros	novamente	nove	novecentos
noventa	num	numa	numas	nunca
nunquinha	nuns	o	oitenta	oito
oitocentos	onde	ontem	onze	ora
os	ou	outra	outras	outrem
outro	outrora	outros	outrossim	para
pela	pelas	pelo	pelos	per
perante	pero	pois	por	porem
porém	porquanto	porque	porquê	portanto
porventura	possivelmente	posteriormente	posto	pouca
poucas	pouco	poucos	pra	praquela
praquelas	praquele	praqueles	praquilo	pras
praticamente	prela	prelas	prele	preles
preste	prestes	previamente	primeiramente	principalmente
priori	pro	pró	pronto	propria
própria	proprias	próprias	proprio	próprio
proprios	próprios	pros	prós	proximo
próximo	quais	quaisquer	qual	qualquer
quando	quanta	quantas	quanto	quantos
quao	quão	quarenta	quase	quatorze
quatro	quatrocentos	que	quê	quem
quer	quiça	quiçá	quinhentos	quinze
raramente	realmente	recentemente	salvante	salvo
se	segundo	seguramente	seis	seiscentos
seja	sem	sempre	senao	senão
sequer	sessenta	sete	setecentos	setenta
seu	seus	sim	simplesmente	so
só	sob	sobre	sobremaneira	sobremodo
sobretudo	somente	sua	suas	tais
tal	talvez	tambem	também	tampouco
tanta	tantas	tanto	tantos	tao
tão	tao-so	tão-só	tao-somente	tão-somente
te	teu	teus	ti	tirante
toda	todas	todavia	todo	todos
tras	trás	tres	treze	trezentos
trilhao	trinta	tu	tua	tuas
tudo	um	um	uma	umas
uns	varias	várias	varios	vários
versus	vezes	via	vice-versa	vinte
visto	voce	você	voces	vocês
vos	vós	vossa	vossos	vulgo

Livros Grátis

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

Milhares de Livros para Download:

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