FENIX - SISTEMA DE FILTRAGEM PERSONALIZADA DE INFORMAÇÕES PARA A WEB

Flávia Coimbra Delicato

Dissertação de Mestrado
Profa. Luci Pirmez, Dra.
Prof. Luiz Fernando Rust da Costa Carmo, Dr.UPS

Instituto de Matemática / Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Junho de 2000

FENIX - SISTEMA DE FILTRAGEM PERSONALIZADA DE INFORMAÇÕES PARA A WEB

Flávia Coimbra Delicato

Dissertação submetida ao corpo docente do Núcleo de Computação Eletrônica/Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências em Informática.

Aprovada por:

Profa._____ - Presidente
Luci Pirmez, Dra.Sc.

Prof. _____- Co-orientador
Luiz Fernando Rust da Costa Carmo, Dr. UPS

Prof. _____
Júlio Salek Aude, Dr.Sc.

Prof. _____
Vitório Bruno Mazzola, Dr.UPS

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2000

DELICATO, FLÁVIA COIMBRA.

FENIX – Sistema de Filtragem Personalizada de Informações/ Flávia Coimbra Delicato. Rio de Janeiro: UFRJ/IM/NCE, 2000.

xii, 139p. il.

Dissertação (Mestrado) - Universidade Federal do Rio de Janeiro, IM/NCE, 2000.

1. Filtragem Personalizada de Informações. 2. Agentes. 3. Tese (Mestr. - UFRJ/IM/NCE). I. Título.

AGRADECIMENTOS

Aos meus pais, por terem me dado a Vida.

Ao Paulo, por ter me trazido amor e alegria quando eu mais precisava.

Ao Renato, pelo carinho e companheirismo durante os primeiros anos do Mestrado.

Aos professores Arnaldo, Maria Luiza e Salek, por ajudarem a viabilizar minha vinda para o NCE.

Ao Rust, pela paciência com que atendeu aos meus pedidos de orientação e pela cuidadosa revisão do meu texto.

À Rosane, pela produtiva troca de informações (e também de lamentações).

E especialmente à Luci, sem a qual esse Mestrado não seria possível, por ter sido mais do que professora, ajudando-me, aconselhando-me e compreendendo todos os momentos difíceis pelos quais passei durante esse tempo.

RESUMO

DELICATO, Flávia Coimbra. **Fenix - Sistema de Filtragem Personalizada de Informações para a Web**. Orientadores: Luci Pirmez e Luiz Fernando Rust da Costa Carmo. Rio de Janeiro. UFRJ/IM/NCE, 2000. Diss.

Atualmente, a Internet disponibiliza uma extensa quantidade de informações, para uma vasta gama de usuários, tornando-se difícil de manipular. Este trabalho propõe o uso de Agentes Inteligentes para a filtragem personalizada de páginas WWW. O sistema desenvolvido é formado por um conjunto de agentes autônomos, fixos e adaptativos. Seu objetivo é auxiliar o usuário em sua tarefa de pesquisa na rede, liberando informações consideradas relevantes e descartando as irrelevantes, a partir de um perfil de preferências pessoais. Através de uma realimentação do usuário, os agentes aprendem e gradativamente conseguem produzir resultados melhores ao longo do tempo. Para a representação das informações adota-se o modelo vetor espacial. O uso de algoritmos genéticos é proposto como um mecanismo auxiliar de aprendizado, visando otimizar os parâmetros usados pelos agentes. Os testes realizados em ambiente simulado tiveram resultados bastante promissores, reduzindo sensivelmente a quantidade de informações com que o usuário precisa lidar.

ABSTRACT

DELICATO, Flávia Coimbra. **Fenix - Personalized Information Filtering System for WEB Pages**. Advisors: Luci Pirmez and Luiz Fernando Rust da Costa Carmo. Rio de Janeiro. UFRJ/IM/NCE, 2000. Diss.

With the current growth of the information available in Internet, users are facing an information overload. This work proposes a multiagent system for Web pages personalized filtering. The system is composed by a set of autonomous and adaptive agents that automatically provide relevant documents to the user according to a preferences profile. The agents learn with the user feedback and attempt to produce better results over time. The vector space model was adopted to represent the information and the agents' learning mechanisms was relevance feedback and genetic algorithms The initial results of tests performed in a simulated environment were quite promising. The proposed system proved to be a useful tool to reduce the amount of information the user has to deal with.

LISTA DE SIGLAS E ABREVIATURAS

AG	Algoritmos Genéticos
IA	Inteligência Artificial
IF	*Information Filtering*
IR	*Information Retrieval*
JVM	*Java Virtual Machine*
ML	*Machine Learning*
MVC	Modelo Vista Controle
NDPM	*Normalized Distance-Based Performance Measure*
P	Precisão
R	*Recall*
WWW	*World Wide Web*

LISTA DE ILUSTRAÇÕES

FIGURAS

TABELAS

SUMÁRIO

1 INTRODUÇÃO

MOTIVAÇÃO

Atualmente, a informação pode ser obtida sob várias formas: jornais, dados financeiros, pesquisas científicas, etc, e representa um dos mais importantes recursos do mundo moderno. As principais atividades políticas, econômicas e intelectuais dependem do fluxo rápido e fácil de informações através de uma variedade de meios, como livros, rádios, televisão e redes eletrônicas.

A computação moderna e a tecnologia de redes tornam possível hoje organizar, armazenar e transportar grandes volumes de dados para qualquer parte do mundo. A expectativa quanto aos ambientes de processamento de informações do futuro é de que sejam constituídos por diversas redes heterogêneas interligadas, distribuídas ao redor do planeta, com novas unidades de processamento podendo ser adicionadas e unidades existentes removidas a qualquer momento. A Internet é um exemplo irrefutável dessa tendência (SICHMAN, DEMAZEAU, 1996).

O crescimento exponencial da quantidade de informações disponíveis na Internet vem provocando um impacto significativo na comunidade de usuários. Estima-se que a taxa de aumento de informações publicadas em qualquer meio tem dobrado a cada 20 (vinte) meses nos últimos anos (PIATETSKY-SHAPIRO, FRAWLEY, 1991). O maior aumento observado foi no âmbito da rede mundial Internet.

A introdução da "*World Wide Web*" (WWW) foi a principal responsável pelo crescimento explosivo nas publicações da Internet, tornando a rede acessível a uma gama muito grande de usuários leigos. A todo momento, surgem novos servidores de informação oferecendo os mais diversos tipos de dados. Pesquisadores tentam encontrar meios confiáveis para o pagamento eletrônico, viabilizando o uso da rede como um importante "mercado virtual".

Esse grande aumento das informações disponibilizadas pela Internet, embora favoreça a disseminação do conhecimento e a obtenção de produtos e serviços, também torna a busca de material relevante um verdadeiro desafio. Muitos indivíduos e

organizações têm percebido que o problema real não é mais obter informação suficiente, mas sim separar o que é útil da imensa quantidade de material inútil. Em vez do usuário precisar “correr atrás” da informação desejada, o ideal seria ter a informação fluindo seletivamente para ele. Para isso, seriam necessárias ferramentas que capturassem “perfis” das necessidades de informações do usuário e encontrassem os ítens de informação relevantes a essas necessidades.

Trabalhos recentes, que surgem da interseção das áreas de recuperação de informações e de *softwares* agentes, oferecem algumas soluções inovadoras para esse problema.

Recuperação de informações é um campo bem estabelecido da Ciência de Informação que lida com problemas de recuperação a partir de grandes coleções de documentos em resposta a consultas de usuários. Uma consulta é uma expressão textual que descreve a necessidade de informação do usuário. Um documento é a unidade organizacional da coleção de informações. A coleção consiste em um grande número de documentos. Um documento relevante é todo aquele que contém informação relacionada à consulta. O objetivo de um sistema de recuperação de informações é comparar a consulta com a coleção e retornar um conjunto de documentos para o usuário, frequentemente classificados de acordo com sua presumida relevância.

O problema da recuperação de informações baseada em consulta pode ser dividido em duas categorias: busca *ad hoc* e roteamento. Nas buscas *ad hoc*, a recuperação baseia-se apenas na consulta inicial fornecida pelo usuário e a coleção consultada em geral é estática. No roteamento de informações, existe um conjunto de documentos previamente classificados pelo usuário quanto a sua relevância que pode ser usado para complementar a consulta inicial. Sistemas de roteamento tentam, então, categorizar os documentos não examinados para encontrar um sub-conjunto provavelmente relevante para o usuário. Em geral, as coleções usadas por sistemas de roteamento são dinâmicas e assume-se que o usuário tenha uma ou mais consultas fixas que definem áreas de interesse contínuo. A resposta ótima para uma consulta seria o sistema encontrar todos os documentos relevantes e não retornar nada que fosse irrelevante.

Atualmente, assume-se que os sistemas de recuperação de informações (IR) são particularmente voltados para busca *ad hoc*, e aqueles voltados para a tarefa de roteamento são conhecidos como sistemas de filtragem de informações (IF), considerados como uma especialização da área de IR.

Em comparação com os estudos de IR e IF, a pesquisa sobre agentes é um campo relativamente novo, que surgiu da área de inteligência artificial. Agentes podem ser definidos como *softwares* cujo objetivo é realizar tarefas em benefício de seus usuários, geralmente de forma autônoma, desempenhando o papel de assistentes pessoais. Atualmente, agentes têm sido aplicados em diversas áreas, tais como no comércio eletrônico, no gerenciamento de redes e na recuperação e filtragem de informações.

Abordagens recentes procuram unir a área de IR e a tecnologia agentes com o propósito de construir sistemas para assistir o usuário em suas tarefas de busca de informações, particularmente no domínio da Internet. O uso popular de agentes como substitutos do usuário em suas pesquisas na rede depende da confiança do usuário na capacidade do agente de inferir seus hábitos e preferências pessoais; só assim um usuário pode sentir-se seguro para delegar ao agente a responsabilidade por tais tarefas. Este problema pode ser resolvido com uma abordagem de aprendizado automático, onde o agente aprende os hábitos de seu usuário através de interações ao longo do tempo (MAES, 1994). Um agente aprendiz pode adquirir sua competência gradualmente através da observação e imitação do usuário, através de instruções explícitas ou realimentação por parte do usuário ou através de recomendações recebidas de outros agentes.

A partir do aprendizado das preferências individuais de seus usuários, os agentes de filtragem tornam-se capazes de reduzir efetivamente a quantidade de informações com que os usuários têm que lidar.

OBJETIVOS GERAIS

Este trabalho propõe o uso da tecnologia de agentes para a filtragem personalizada de informações. Para isso serão analisados detalhadamente os principais conceitos relacionados à tecnologia de agentes, bem como à busca automática de informações. Será elaborado um sistema de filtragem e classificação de documentos composto por um conjunto de agentes cujo objetivo principal é satisfazer as necessidades de informação dos seus usuários. Esses agentes deverão ser capazes de receber realimentação do usuário quanto a relevância das informações recuperadas (técnica de aprendizado baseada em realimentação por relevância – FRAKES, BAEZA-YATES (1992)) e assim refinar suas buscas, obtendo resultados melhores ao longo do tempo. O uso de algoritmos genéticos (GOLDBERG, 1989, GOLDBERG, 1994) também será alvo de estudos no intuito de compor um mecanismo de aprendizado complementar.

O espaço de busca a partir do qual as informações serão recuperadas será a *Web*, onde uma página WWW é considerada um item de informação para o sistema. A escolha do espaço de busca da aplicação baseia-se na constatação de que grande parte dos sistemas de filtragem e recomendações de informações atualmente existentes são voltados para o domínio de artigos de *usenet* ou correio eletrônico, sendo que poucos são destinados a filtragem de páginas WWW. Serão analisados os principais modelos de representação de informações usados em sistemas de recuperação de informação e, em particular, o modelo vetor espacial (SALTON, 1989), verificando a sua adaptação para o uso com páginas *Web.*

REVISÃO BIBLIOGRÁFICA

O estudo da filtragem e classificação de documentos situa-se na interseção entre Máquinas de Aprendizado ("*Machine Learning*" ou ML) e Recuperação de Informações ("Information Retrieval" ou IR), havendo, portanto, uma vasta literatura a respeito desse assunto.

Na comunidade de IR, variações da técnica de realimentação por relevância têm sido estudadas no contexto da tarefa de roteamento de informações, conforme descrito nas conferências TREC (*TExt Retrieval Conferences*) (HARMAN, 1994, BUCKLEY *et al.*, 1994, ALLAN, 1995). Alguns trabalhos de filtragem de informações que utilizam a

realimentação por relevância como mecanismo de aprendizado são o de SHETH e MAES (1993) e o de FOLTZ e DUMAIS (1992). Há, também, várias trabalhos de comparações entre essa técnica e técnicas de ML não-incrementais (SCHUTZE *et al.*, 1995, LANG, 1995, PAZZANI *et al*, 1996). Uma das desvantagens das técnicas não-incrementais é o grande número de exemplos necessários antes do algoritmo de aprendizado poder ser aplicado (BALABANOVIC, 1997). O sistema Newst (SHETH, 1994) usa realimentação por relevância e algoritmos genéticos (AG) para fornecer filtragem personalizada de artigos da *usenet*.

Outros sistemas usam diversas técnicas para tentar detectar padrões no comportamento do usuário. Por exemplo, o InfoScope (FISCHER, STEVENS, 1991) aprende usando um sistema baseados em regras que registra os tópicos interessantes vistos pelo usuário no passado. As recomendações de novos tópicos para o usuário são feitas com base em quão recentes, frequentes e espaçados são os tópicos vistos anteriormente. A desvantagem dessa abordagem é que ela é restrita a fazer recomendações de tópicos que estavam dentro do domínio de interesses passados do usuário. Da mesma forma, sistemas de navegação assistida (ARMSTRONG *et al.*, 1995), que também visam a recomendação de páginas WWW, são restritos às seções da *Web* visitadas pelo usuário, recomendando, a partir delas, *links* apropriados a serem acessados.

YANG e KORFHAGE (1993) usam algoritmos genéticos para evoluir uma população de consultas individuais a fim de se obter uma consulta ótima. Eles assumem que os interesses do usuário são fixos, não havendo aprendizado da população durante seu ciclo de vida.

BALABANOVIC (1998) propôs uma arquitetura híbrida que usa a abordagem de filtragem de texto baseada em conteúdo em conjunto com filtragem colaborativa na construção de um sistema de recomendação de páginas *Web*. Esse trabalho adota o modelo vetor espacial (SALTON, 1968), o método de aprendizado baseado em realimentação por relevância e sugere o uso de algoritmos genéticos (GOLDBERG, 1989) como possível solução para alguns dos problemas encontrados na filtragem baseada em texto.

ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em sete capítulos. Os capítulos 2 e 3 apresentam uma descrição dos conceitos e técnicas utilizados ao longo do desenvolvimento do trabalho. O capítulo 2 fornece uma introdução à tecnologia de agentes e uma visão geral sobre inteligência artificial, enfatizando os algoritmos genéticos. No capítulo 3 são abordados as técnicas de busca e filtragem automática de informações e o modelo de representação de informações adotado no trabalho. O capítulo 4 apresenta a descrição do sistema Fenix, abordando a metodologia de desenvolvimento e detalhando os componentes de sua arquitetura. O ambiente de desenvolvimento e os detalhes de implementação do sistema são tratados no capítulo 5. O capítulo 6 traz uma análise dos resultados obtidos e, finalmente, as conclusões e sugestões de trabalhos futuros são relatadas no capítulo 7.

2 CONCEITOS GERAIS SOBRE AGENTES E INTELIGÊNCIA ARTIFICIAL

O presente trabalho propõe o desenvolvimento de um sistema de filtragem de informações personalizado baseado na tecnologia de agentes e adotando como mecanismos de aprendizagem técnicas derivadas da área de recuperação de informações e da área de inteligência artificial. O sistema proposto apresenta características como autonomia e capacidade de adaptação, que o enquadram na maioria das definições para agentes encontradas na literatura (GRAESSER, FRANKLIN, 1996, SHOHAM, 1993, CAGLAYAN, HARRISON, 1997). A principal técnica utilizada pelo sistema para aprender as preferências do usuário e adaptar-se a elas é a realimentação por relevância (*relevance feedback*), tradicionalmente empregado em sistemas de recuperação de informações e abordado no capítulo 3. Além disso, o sistema foi projetado para explorar um mecanismo de aprendizado auxiliar, que visa principalmente a introdução de diversidade nos resultados obtidos com a realimentação por relevância. Esse mecanismo consiste no uso de algoritmos genéticos, que é uma técnica de inteligência artificial baseada nos princípios da evolução natural.

Este capítulo tem por objetivo dar uma visão geral do mundo dos agentes e a seguir fornecer uma breve introdução sobre a área de inteligência artificial, detendo-se mais detalhadamente na técnica de algoritmos genéticos. Na seção 2.1 são vistas algumas definições da palavra "agente" e são analisados os atributos e as várias classificações de agentes. A seção 2.2 aborda os agentes inteligentes e na seção 2.3 são vistas as principais aplicações dos agentes na Internet. Na seção 2.4 é feita uma introdução à inteligência artificial, descrevendo-se seu histórico e suas principais linhas de pesquisa. Na seção 2.5 são apresentados os principais conceitos sobre algoritmos genéticos e é analisada a aplicação desta técnica em sistemas de filtragem e recuperação de informações.

AGENTES

A definição de agente segundo o Webster's New World Dictionary (GURALMILE, 1970) é: "uma pessoa ou coisa que age ou é capaz de agir ou é autorizado a agir, por outra." No entanto, o significado da palavra pode variar de acordo com o contexto em

que se insira. A sua inserção no campo da informática tem se tornado cada vez mais popular nos últimos anos, embora não se encontre uma definição consensual para agentes nesse contexto. Tal fato provavelmente decorre do estudo de agentes cobrir diversas áreas de investigação e desenvolvimento. A seguir são analisadas algumas das definições encontradas na literatura.

A definição dada por NWANA (1996) enfatiza a delegação de tarefas, assemelhando-se bastante à definição comum da palavra "agente". Para ele, um agente é definido como “um componente de *hardware* e/ou *software* que é capaz de realizar tarefas em benefício de seu usuário”.

KETCHPEL e GENESERETH (1994) restringem a sua definição aos agentes de *software*. Para esses autores, “agentes são componentes de *software* que se comunicam entre si através da troca de mensagens em uma linguagem de comunicação de agentes”.

Já para GRAESSER e FRANKLIN (1996), a autonomia é a característica mais importante dos agentes. Segundo eles, um agente autônomo é um sistema, situado em um ambiente, que pode perceber as mudanças que ocorrem nesse ambiente e reagir a elas. Além disso, os agentes também podem atuar por conta própria a fim de alcançar seus objetivos, sem a necessidade de receber estímulos do ambiente. Os autores destacam ainda a capacidade dos agentes de estarem continuamente ativos.

Para o propósito deste trabalho será adotada a definição dada por CAGLAYAN e HARRISON (1997), segundo a qual um agente é "uma entidade computacional que realiza tarefas delegadas pelo usuário de forma autônoma”. A seguir são analisados os principais atributos e as várias classificações de agentes.

Atributos e Classificações de Agentes

Cada uma das definições apresentadas na seção 2.1 propõe características ou atributos que são essenciais para o conceito de agente, como, por exemplo, a autonomia, a reatividade, a capacidade social e a continuidade temporal. Além dessas, existem ainda outras características que costumam ser associadas aos agentes, como a mobilidade e a inteligência. A seguir são analisadas as principais características consideradas cruciais para compreender o mundo dos agentes.

- **Inteligência** - é o grau de raciocínio e comportamento aprendido, a habilidade do agente de aceitar a sentença de objetivos (metas) do usuário e desempenhar a sua tarefa.

- **Autonomia** - é a capacidade do agente de ter controle sobre suas próprias ações (GRAESSER, FRANKLIN, 1996). Um agente pode ser considerado autônomo em relação ao ambiente ou em relação a outros agentes. Um agente autônomo tem a capacidade de decidir como satisfazer o pedido que recebe do usuário, controlando de forma dinâmica as ações que irá tomar e decidindo o momento mais apropriado para agir. WOOLDRIDGE e JENNINGS (1994) acrescentam que o agente, além de possuir controle sobre seu comportamento, deve também possuir controle sobre seu estado interno. NISSEN *et al*. (1995) definem a autonomia dos agentes no contexto da Internet como a sua capacidade de estarem ativos mesmo quando o usuário está desconectado da rede.

- **Pro-atividade** – é a capacidade do agente de iniciar ações por conta própria para atingir os seus objetivos, não se limitando a responder a estímulos do ambiente.

- **Continuidade temporal** - a continuidade temporal impõe que o agente esteja continuamente ativo no ambiente. Grande parte dos *softwares* existentes não tem esta característica, já que executam uma ou mais tarefas e terminam.

- **Persistência** - segundo BELGRAVE (1995), persistência é a capacidade apresentada pelo agente de manter um estado interno consistente através do tempo, sem alterá-lo ao acaso.

- **Reatividade** - conforme WOOLDRIDGE e JENNINGS (1994), reatividade é a propriedade que permite aos agentes perceberem seus ambientes e responderem adequadamente às mudanças neles ocorridas. O ambiente de percepção do agente pode ser caracterizado como: o mundo real, um usuário interagindo através de uma interface gráfica, uma coleção de outros agentes, a Internet, ou uma combinação dos ítens anteriores.

- **Capacidade social** – é a capacidade do agente de comunicar-se com outros agentes e com seus usuários, geralmente através de uma linguagem de comunicação entre agentes. Essa comunicação pode resultar em uma cooperação, diminuindo o esforço individual de cada agente na realização de sua tarefa.

- **Capacidade de adaptação** – é a capacidade do agente de alterar o seu comportamento com base na experiência. Essa característica é considerada também como a capacidade de aprendizagem dos agentes.
- **Mobilidade** – é a capacidade do agente de se mover no ambiente. Um agente móvel é aquele que é capaz de se transportar de uma máquina para outra durante a sua execução.

Os agentes podem ou não possuir uma ou mais das capacidades supracitadas a fim de efetuar as tarefas para as quais são projetados. Em função dessas capacidades, eles podem receber classificações, tais como: agentes fixos, agentes móveis, agentes inteligentes, agentes móveis inteligentes.

Os agentes podem também ser classificados de acordo com o tipo de tarefa que executam. Atualmente, algumas das principais áreas em que podem ser empregados agentes são: recuperação de informações, filtragem de informações, gerenciamento de mensagens, computação móvel, interfaces adaptativas, gerência de redes, comércio eletrônico, etc.

Dentre os tipos de agentes, aqueles que possuem o atributo de inteligência são classificados como agentes inteligentes e podem ser empregados em várias das tarefas descritas acima. Na seção a seguir os agentes inteligentes são analisados mais detalhadamente por serem relevantes ao escopo do presente trabalho.

AGENTES INTELIGENTES

A inteligência pode ser vista como a capacidade do agente de compreender a sentença de objetivos do usuário e executar as tarefas a ele delegadas. Outra definição para inteligência é a medida das capacidades de raciocínio e aprendizagem dos agentes.

Das definições para agentes inteligentes encontradas na literatura, pode-se concluir que esses agentes devem ter um comportamento orientado a metas e devem possuir alguma forma de compreender as necessidades do usuário e adaptar-se a elas. As capacidades de raciocínio e aprendizado dos agentes geralmente são obtidas com a utilização de técnicas do domínio da Inteligência Computacional.

Segundo MAES (1994), um número cada vez maior de usuários leigos farão uso de computadores e estações de trabalho em um futuro próximo. Consequentemente, torna-se imperiosa a mudança do paradigma de interação usuário-computador. No paradigma atual, denominado gerenciamento direto, o usuário inicializa e monitora a totalidade dos eventos. O paradigma emergente, denominado gerenciamento indireto, é baseado na tecnologia de agentes. Nele, o usuário é engajado em um processo onde homem e agentes computacionais iniciam a comunicação, monitoram eventos e executam tarefas. No gerenciamento indireto, o agente é visto como um assistente pessoal do usuário, sendo a ele atribuídas quatro funções gerais:

- desempenhar tarefas em benefício do usuário;
- treinar ou ensinar o usuário;
- ajudar na interação entre usuários colaboradores;
- monitorar eventos e processos.

Alguns exemplos de aplicações específicas em que os agentes inteligentes têm sido usados desempenhando o papel de assistentes pessoais são a computação móvel, as interfaces gráficas adaptativas e a filtragem de informações. Particularmente no âmbito da Internet tornam-se cada vez mais necessárias ferramentas que auxiliem o usuário a lidar com a crescente quantidade de informações disponíveis.

APLICAÇÕES DE AGENTES NA INTERNET

Os agentes para a Internet servem como “corretores” de informações entre os fornecedores (servidores Internet) e os consumidores, geralmente usuários que interagem através de navegadores. Nesse papel de intermediários eletrônicos, os agentes para Internet comparam as necessidades de informação dos usuários da rede com os atributos dos fornecedores, o tipo e o conteúdo das informações fornecidas. Eles podem estar localizados no próprio servidor onde a informação é armazenada, acessando-a diretamente e realizando tarefas para seus usuários, com ou sem interação direta com os mesmos. As aplicações mais relevantes para o propósito deste trabalho são as que dizem respeito aos agentes para recuperação e filtragem de informações, detalhadas a seguir.

Agentes Para a Recuperação de Informações

A principal motivação para o desenvolvimento de agentes para a recuperação de informações é o crescimento da informação disponível na Internet. O resultado disso é que a informação relevante existente é em quantidade superior àquela que os usuários conseguem analisar. Se for possível delegar a pesquisa de informação a um ou mais agentes, os usuários gastarão muito menos tempo com essa tarefa, além dos custos associados ao tempo de conexão na rede diminuírem.

Se os agentes tiverem capacidade de comunicação entre si, existe ainda a possibilidade de colaboração entre eles, minimizando o esforço individual de cada um e reduzindo ainda mais o tempo gasto na pesquisa. Além disso, os agentes podem ser construídos de forma a obterem informação gerada dinamicamente, mantendo uma base de dados sempre atualizada.

Em paralelo com a aplicação de agentes na tarefa de recuperação de informações, existem os agentes usados para filtrar fluxos de informações disponíveis, liberando para o usuário apenas o que for potencialmente relevante e descartando o que for irrelevante, com base em um conhecimento prévio de seus interesses pessoais. Esses são os agentes para a filtragem de informações, detalhados na próxima seção.

Agentes de Filtragem de Informações

Os agentes de filtragem reúnem dados recentes sobre uma necessidade de informação de um usuário a partir de várias fontes, filtram o conteúdo desses dados baseando-se nas preferências pessoais do usuário e apresentam os dados potencialmente mais relevantes (tipicamente como uma página HTML ou através de correio eletrônico). Os agentes de filtragem em geral usam um número fixo de servidores de informações e capturam as preferências do usuário em um perfil pessoal (CAGLAYAN e HARRISON, 1997).

Atualmente há vários agentes de filtragem de informações disponíveis na *Web*. As principais aplicações encontradas são para a filtragem de correio eletrônico, artigos de *usenet*, páginas WWW e para a criação de “jornais personalizados", como o NewsHound, da San Jose Mercury News e o NewsPage Direct, da Individual Inc. (CAGLAYAN e HARRISON, 1997). O principal benefício desses agentes é lidar com a sobrecarga de informações recebidas pelo usuário e liberar apenas o que for potencialmente relevante tendo como base um perfil de preferências pessoais.

A figura 1 mostra a arquitetura típica de um agente de filtragem de artigos da *usenet.* O mecanismo de indexação da figura associa palavras-chaves a cada artigo. Essas palavras podem ter um grau de relevância ou peso associado. Uma medida de relevância comum é a frequência de uso das palavras. Essa medida baseia-se na assunção de que escritores têm uma tendência natural de repetir certas palavras a fim de enfatizar o ponto principal. Essa hipótese forma a premissa principal do processamento automático de texto, representada na figura 2. Dessa forma, a maioria dos sistemas de filtragem e recuperação de informações modela documentos como conjuntos de termos e a contagem da frequência desses termos.

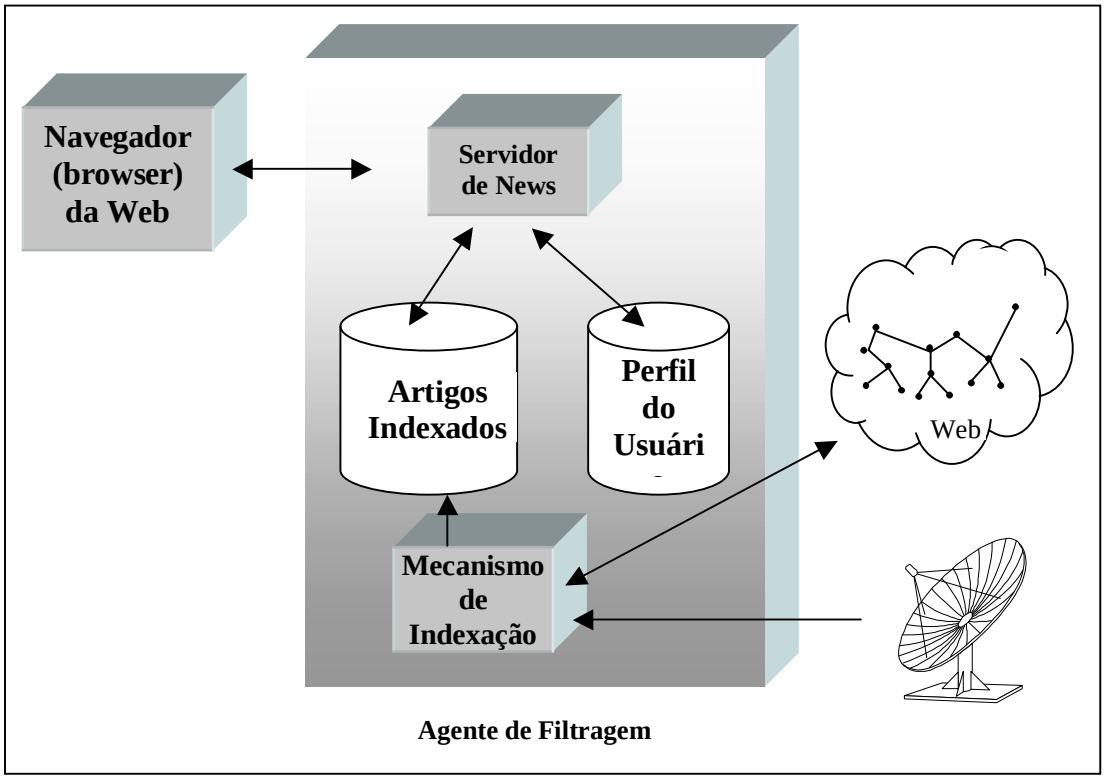


Figura 1: Arquitetura de um agente de filtragem de *news.*

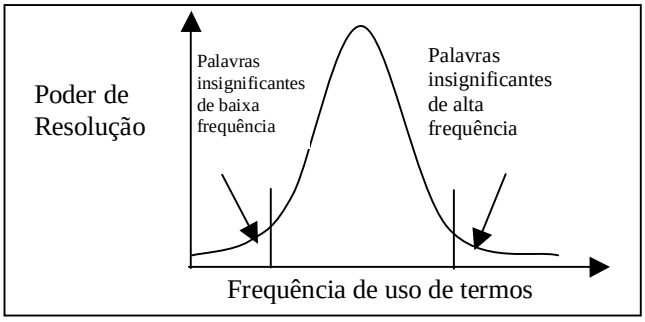


Figura 2: Filtragem baseada na frequência de uso das palavras

As seções acima abordaram os principais conceitos sobre o estudo dos agentes. Nas seções seguintes é apresentada uma visão geral sobre a área de inteligência artificial em geral e sobre a técnica de algoritmos genéticos em particular, analisando sua utilização em sistemas de filtragem e recuperação de informações.

UMA VISÃO GERAL SOBRE INTELIGÊNCIA ARTIFICIAL (IA)

Oficialmente, o termo "inteligência artificial" (IA) nasceu em 1956 em uma conferência de verão em Dartmouth College, NH, USA (MCCORDUCK, 1979). As diferentes correntes de pensamento em IA têm estudado formas de estabelecer comportamentos "inteligentes" nas máquinas. Embora a área de IA seja estudada academicamente desde os anos 50, só recentemente tem gerado um interesse crescente devido ao surgimento de aplicações comerciais práticas. Um fator decisivo para o sucesso dessa transição do ambiente acadêmico para a indústria são os enormes avanços tecnológicos dos equipamentos computacionais ocorridos nas últimas duas décadas. Pode-se afirmar que o campo de IA tem como objetivo o contínuo aumento da "inteligência" do computador, pesquisando, para isto, também os fenômenos da inteligência natural.

Existem duas linhas principais de pesquisa para a construção de sistemas inteligentes: a linha conexionista e a linha simbólica. A linha conexionista visa a modelagem da inteligência humana através da simulação dos componentes do cérebro, isto é, de seus neurônios, e de suas interligações. Esta proposta foi formalizada inicialmente em 1943, quando McCulloch e Pitts propuseram um primeiro modelo matemático para um neurônio. Durante um longo período essa linha de pesquisa não foi muito ativa, mas o advento dos microprocessadores, pequenos e baratos, tornou viável a implementação de máquinas de conexão compostas de milhares de microprocessadores, o que, aliado à solução de alguns problemas teóricos importantes, deu um novo impulso às pesquisas na área. O modelo conexionista deu origem à área de redes neuronais artificiais.

Os princípios da linha simbólica são apresentados no artigo *Physical symbol systems* de NEWELL (1980). O sucesso dos sistemas especialistas (do inglês "*expert*

system"), a partir da década de setenta, estabeleceu a manipulação simbólica de um grande número de fatos especializados sobre um domínio restrito como o paradigma corrente para a construção de sistemas inteligentes do tipo simbólico. As principais áreas de pesquisa em IA simbólica são atualmente: sistemas especialistas, aprendizagem, representação e aquisição de conhecimento, tratamento de informação imperfeita, visão computacional, robótica, controle inteligente, inteligência artificial distribuída, modelagem cognitiva, arquiteturas para sistemas inteligentes, linguagem natural e interfaces inteligentes.

Além das linhas conexionista e simbólica, observa-se hoje o crescimento de uma nova linha de pesquisa em IA, baseada na observação de mecanismos evolutivos encontrados na natureza, tais como a auto-organização e o comportamento adaptativo. Nessa linha, os modelos mais conhecidos são os autômatos celulares e os algoritmos genéticos (BARTO, 1975, FARMER *et al.*, 1983, GOLDBERG, 1989, HOLLAND, 1975, HOLLAND, 1986).

ALGORITMOS GENÉTICOS

Algoritmos genéticos são uma técnica de IA baseada nos princípios da evolução natural. Eles requerem 5 componentes principais:

- uma forma de codificar soluções para um problema em cromossomos;
- uma função de avaliação que retorna um valor para cada cromossomo;
- uma forma de inicializar a população de cromossomos;
- operadores que podem ser aplicados aos "pais" durante a reprodução para alterar sua composição genética, como, por exemplo, mutação, *crossover* e operadores específicos ao domínio do problema;
- parâmetros para o algoritmo e para os operadores.

O princípio básico de um algoritmo genético (AG) consiste em considerar uma população de indivíduos onde cada um representa uma solução potencial para um problema. O sucesso relativo de cada indivíduo no problema (medido pela função de avaliação) corresponde a sua aptidão, e é usado para reproduzir seletivamente os indivíduos mais aptos, que irão produzir descendentes similares, porém não idênticos, para a próxima geração. Alguns membros da população são modificados por operadores genéticos para formar novas soluções. Nas transformações unárias (do tipo mutação),

novos indivíduos são criados por uma pequena mudança em um único indivíduo. Nas transformações de ordem mais alta (do tipo *crossover*), novos indivíduos são criados através da combinação de partes de vários indivíduos. Parâmetros de controle para os operadores, indicando a probabilidade de ocorrência de *crossover* e mutação na população, devem ser cuidadosamente selecionados para se obter o melhor desempenho.

Após certo número de gerações, onde o espaço de indivíduos é explorado de forma eficiente, o algoritmo converge, ou seja, não apresenta mais melhoras no valor de aptidão de seus indivíduos. O melhor indivíduo (mais apto) provavelmente representa a solução ótima para o problema em questão.

Algoritmos Genéticos para Recuperação de Informações

Existem na literatura vários exemplos de implementações de algoritmos genéticos para aplicações de IR. GORDON (1988) apresentou uma abordagem baseada em algoritmos genéticos para a indexação de documentos. Nessa abordagem, conjuntos de palavras-chaves são associados a um documento e alterados ao longo do tempo usando-se os operadores de mutação e *crossover*. Nesse trabalho, uma palavra-chave representa um gene (bit), uma lista de palavras (correspondendo a um documento) representa um indivíduo (uma cadeia de bits), e uma coleção de documentos inicialmente julgados relevantes por um usuário representa a população inicial. Baseado em uma medida de aptidão, a população inicial evolui através de gerações e eventualmente converge para uma população ótima, ou seja, um conjunto de palavras-chave que melhor descreve os documentos.

YANG e KORFHAGE (1993) desenvolveram métodos de recuperação adaptativos baseados em algoritmos genéticos usando o modelo vetor-espacial e realimentação por relevância. Eles analisaram o efeito de adotar algoritmos genéticos em bancos de dados extensos, o impacto de operadores genéticos, e a capacidade de busca paralela do AG, obtendo resultados favoráveis em relação a outras técnicas de otimização.

Quando se utilizam algoritmos genéticos em sistemas de recuperação de informação, os conceitos (palavras-chaves) são seriamente restringidos pela cadeia de bits disponível na população inicial. O sistema poderá usar somente as palavras-chaves

dos documentos fornecidos pelo usuário para encontrar outros documentos relevantes. Uma palavra-chave considerada relevante, mas que não se encontra no documento inicial, não será considerada na otimização do Algoritmo Genético. Para solucionar este problema, CHEN e KIM (1993) propuseram, em seu sistema GANNET, a introdução de uma rede neural de termos relacionados, que age constantemente fornecendo genes novos e úteis para a evolução de um Algoritmo Genético. Os genes mais aptos provenientes de uma rodada do Algoritmo Genético podem chamar o módulo de rede neural a fim de produzir outros genes similares. Estes novos genes podem ser usados na próxima rodada do Algoritmo Genético. Através do uso de uma rede de termos, o algoritmo genético pode mudar dinamicamente sua representação fundamental. Espera-se que isto venha a amenizar o problema de representação de tamanho fixo em Algoritmos Genéticos e, consequentemente, melhorar o desempenho de otimização (KOZA, 1992).

CONSIDERAÇÕES FINAIS

Este capítulo abordou alguns conceitos básicos sobre a teoria de agentes, apresentando definições, classificações e áreas de aplicações nas quais eles podem ser empregados. A seguir foi feita uma introdução sobre a a área de inteligência artificial, enfatizando-se a técnica de algoritmos genéticos por ser abordada como mecanismo de aprendizado complementar no presente trabalho. O capítulo seguinte trata da filtragem e recuperação automática de informações, detendo-se principalmente nas técnicas de filtragem de texto, as quais fazem parte do escopo deste trabalho.

3 BUSCA AUTOMÁTICA DE INFORMAÇÕES

Nos últimos anos, vários sistemas têm sido desenvolvidos com o objetivo de assistir o usuário em sua busca por informações na *Web*. A maioria desses sistemas pertence a uma das três categorias: mecanismos de busca, sistemas de navegação assistida e sistemas de filtragem personalizada de informações. O sistema proposto neste trabalho visa fornecer aos seus usuários um serviço de filtragem personalizada de páginas WWW, pertencendo portanto à última categoria citada.

Este capítulo tem por objetivo introduzir os principais conceitos relacionados à busca automática de informações. Sistemas de filtragem e mecanismos de busca possuem diversas características similares e algumas diferenças básicas, as quais são analisadas nas seções 3.2 e 3.3. A seção 3.3.2 trata da tecnologia de filtragem de texto, uma sub-área dos sistemas de filtragem de informações, e descreve o processo de indexação automática de texto. A seção 3.3.2.3 aborda detalhadamente o modelo de representação de informações adotado no presente trabalho. Na seção 3.4 é descrita a técnica de realimentação por relevância, usada como principal mecanismo de aprendizado do sistema. Algumas considerações finais são feitas na seção 3.5.

CONCEITOS INICIAIS

O termo "busca" de informação pode ser usado como um termo genérico para descrever qualquer processo através do qual usuários buscam obter informação de sistemas de informações automatizados. A filtragem de texto é um tipo de busca de informações. No processo de filtragem, assume-se que o usuário esteja buscando informações que tratam de um interesse específico a longo prazo. Cabe aqui estabelecer uma distinção entre "processo" e "sistema". No contexto deste trabalho, um "processo" é uma atividade conduzida por pessoas, podendo ou não ter o auxílio de uma máquina. Já o termo "sistema" implica em um sistema automatizado (isto é, uma máquina), projetado para suportar pessoas engajadas naquele processo. Um sistema de filtragem de informações é, então, um sistema desenvolvido com a finalidade de suportar um processo de filtragem de informações.

Sistemas de filtragem de informações são tipicamente projetados para classificar grandes volumes de informações geradas dinamicamente e apresentar ao usuário as

fontes de informações provavelmente relevantes às suas necessidades. "Fontes de informações" referem-se a entidades que contêm informações em uma forma compreensível pelo usuário.

Um processo de filtragem de informações envolve geralmente três sub-tarefas: coletar as fontes de informações, selecioná-las e apresentá-las (OARD, MARCHIONINI, 1996). Estas mesmas tarefas são também fundamentais para os processos de recuperação de informações. A constatação das semelhanças entre filtragem e recuperação de informações levou BELKIN e CROFT (1992) a sugerir que o processo de filtragem seria uma aplicação útil para as técnicas já desenvolvidas para os sistemas de recuperação.

Qualquer processo de filtragem ou recuperação de informações começa com o objetivo (meta) do usuário. Nos sistemas desenvolvidos para a filtragem de informações, assume-se que as necessidades do usuário são relativamente específicas e mudam pouco em relação à taxa na qual as fontes de informação tornam-se disponíveis. Em contraste, sistemas de recuperação, embora também visem atender a necessidades específicas, em geral são projetados para acessar fontes de informações relativamente estáveis. Um sistema de recuperação de informações tradicional pode ser usado para realizar um processo de filtragem, acumulando documentos por um certo período, emitindo uma consulta sobre esses documentos, e então descartando os documentos não selecionados.

A seguir é feita uma descrição mais detalhada das características dos sistemas de busca e de filtragem de informações, a fim de se obter um melhor entendimento das diferenças entre eles.

MECANISMOS DE BUSCA DE INFORMAÇÕES

Os mecanismos de busca existentes na *Web* são a forma mais usual para auxiliar as pesquisas na Internet. Estes mecanismos são, certamente, uma boa forma para encontrar documentos que contenham determinadas palavras, designadas por palavras-chaves. No entanto, eles possuem diversos problemas:

- o usuário que procura a informação deve saber escolher o melhor conjunto de palavras-chave para a obtenção dos resultados mais relevantes, o que nem sempre ocorre;
- a informação é pesquisada em um contexto genérico, surgindo como respostas várias informações irrelevantes;
- o mapeamento da informação é feito reunindo-se meta-informações sobre os documentos disponíveis na Internet; este método consome muito tempo e gera muito tráfego de dados na rede;
- um servidor poderoso é necessário para efetuar a indexação dos muitos documentos existentes, guardar os índices e suportar a pesquisa sobre esses índices para muitos usuários;
- a informação na Internet é muito dinâmica; frequentemente, mecanismos de busca fazem referência a informações que foram transferidas para outro local;
- os mecanismos atuais não aprendem através de suas buscas e nem se ajustam a seus usuários;
- o servidor onde reside o serviço de busca pode estar fora do ar ou demasiadamente ocupado para permitir uma conexão;
- o usuário deve manter uma ligação à Internet durante todo o tempo em que a pesquisa é efetuada.

A tecnologia de agentes aplicada ao processo de busca de informações pode oferecer algumas vantagens quando comparada com os mecanismos de busca tradicionais, tais como:

- agentes podem buscar informações baseados em contextos semânticos, utilizando ferramentas como "*tesaurus*" (BAEZA-YATES RIBEIRO-NETO, 1999, SALTON, 1989);
- agentes podem criar suas próprias bases de dados sobre as informações disponíveis na Internet, atualizando-as e expandindo-as após cada busca. Além disso, agentes podem se comunicar e cooperar com outros agentes, realizando suas buscas de modo mais rápido e eficiente e reduzindo o tráfego na rede. Eles também são capazes de realizar a busca diretamente na fonte, levando a uma queda ainda maior no uso da banda passante;

- agentes podem estar sempre disponíveis, quando residem na própria máquina do usuário.

Essas vantagens, entre outras, têm motivado o desenvolvimento de agentes voltados para aplicações de busca e recuperação de informações.

SISTEMAS DE FILTRAGEM DE INFORMAÇÕES

Filtragem de informações é também conhecida na literatura como roteamento (do inglês "*routing*") de informações. Um sistema de filtragem assiste ao usuário filtrando o fluxo de dados e liberando apenas as informações julgadas relevantes. As principais diferenças que podem ser apontadas entre sistemas de filtragem e mecanismos de busca são:

- sistemas de filtragem de informações geralmente envolvem repetidas interações ao longo de múltiplas sessões; os usuários são mais passivos e possuem objetivos a longo prazo, enquanto que nos mecanismos de busca de informações os usuários tipicamente têm uma necessidade de informação a curto prazo, satisfeita em uma única sessão.
- mecanismos de busca representam as necessidades de informação do usuário através de consultas, enquanto que os sistemas de filtragem utilizam perfis para representar os interesses específicos dos usuários;
- mecanismos de busca estão interessados na seleção de informações a partir de uma base de dados relativamente estática, enquanto que os sistemas de filtragem estão principalmente interessados na seleção ou eliminação de informações a partir de um fluxo de dados dinâmico.

Os sistemas de filtragem de informações podem ser classificados baseando-se na forma através da qual os documentos são filtrados. Sistemas que selecionam documentos baseados nas características de seus conteúdos são chamados de cognitivos, enquanto que sistemas baseados em recomendações e anotações feitas por outros usuários são chamados de sistemas sociais ou colaborativos (KJERSTI, 1997). Se a informação estiver sendo pesquisada com o intuito de manter uma comunidade de usuários atualizada em determinado assunto, a filtragem social é a mais indicada. Entretanto, se a informação estiver sendo pesquisada baseada no tópico (assunto), de forma independente de outros usuários, os sistemas cognitivos são os mais apropriados.

Qualquer que seja a abordagem adotada, para a filtragem ser bem sucedida, o sistema deve ser capaz de representar e atender os interesses e preferências dos seus usuários. Como tais preferências variam muito de acordo com o usuário, sistemas de filtragem de informações devem ser altamente personalizados.

Sistemas de Filtragem de Informações Personalizados

Sistemas de filtragem personalizados devem liberar para o usuário informações relevantes de forma consistente e em tempo hábil. Devem também ser capazes de suportar mudanças nas necessidades do usuário, adaptando-se a elas. Um exemplo desses sistemas é a criação de jornais personalizados, já existentes na *Web*. Por exemplo, o NewsHound[1] analisa diversos jornais para construir um jornal personalizado, de acordo com as preferências indicadas pelos seus usuários. Esse jornal é depois enviado através de correio eletrônico.

Um sistema de filtragem personalizado deve satisfazer 3 requisitos:

- **especialização** - uma vez que a filtragem envolve interações repetidas com o usuário, o sistema deve ser capaz de identificar padrões no seu comportamento; o sistema deve inferir seus hábitos e especializar-se a eles, recomendando o máximo de assuntos relevantes e o mínimo de irrelevantes;
- **adaptação** - os interesses do usuário não devem ser considerados constantes; o sistema deve ser capaz de perceber as mudanças de interesses e adaptar-se a elas;
- **exploração** - um sistema de filtragem deve ser capaz de explorar novos domínios de informações para encontrar assuntos de interesse potencial do usuário.

Em um sistema de filtragem de informações personalizado, a interação entre o usuário e o sistema é feita através de ciclos. Quando o usuário acessa o sistema pela primeira vez, ele define seus interesses. O sistema formaliza essa informação e constrói um perfil para o usuário. O sistema pesquisa, então, um conjunto pré-definido de fontes de informação na *Web* e indexa todos os documentos recuperados. Cada vez que o usuário acessa o sistema, são executados os seguintes passos:

[1] Disponível em: http://www.hound.com/help/about.htm

- todos os documentos ainda não lidos pelo usuário são avaliados pelo sistema, que gera uma pontuação (*score*) baseada no perfil do usuário;
- de acordo com as pontuações e um limite estabelecido (*threshold*), o sistema decide que documentos liberar para o usuário;
- o usuário recebe uma lista de títulos de documentos potencialmente interessantes juntamente com suas pontuações, e seleciona alguns para ler;
- para cada documento lido, o usuário pode fornecer realimentação positiva ou negativa, de acordo com sua relevância;
- a realimentação é registrada e usada para melhorar o desempenho do sistema, modificando suas posteriores pontuações de documentos.

Uma importante especialização dos sistemas de filtragem de informações é a filtragem de texto, onde o domínio de atuação do sistema é puramente textual, não incluindo outras mídias, tais como vídeo, áudio ou elementos gráficos.

Tecnologia de Filtragem de Texto

A filtragem de texto é uma especialização dos sistemas de filtragem de informações, onde o domínio das informações é puramente textual. Um sistema típico de filtragem de texto requer sempre alguma forma de representar os documentos que estão sendo filtrados, uma forma de representar a necessidade de informação do usuário, um modo de comparar documentos com as necessidades do usuário e um modo de usar os resultados dessa comparação. Em sistemas de recuperação, as necessidades de informação do usuário são expressas através de uma expressão de consulta, geralmente composta por um conjunto de palavras-chave. Em sistemas de filtragem, essas necessidades são expressas em geral através de um perfil individual de preferências, que pode ser construído manualmente pelo usuário, ou inferido pelo sistema ao longo de diversas sessões de interação com o usuário. Tanto os perfis quanto os documentos processados consistem em ítens de informação para o sistema.

O objetivo final de um sistema de filtragem de texto é automatizar o processo de examinar os documentos, fazendo comparações entre as representações dos documentos e os perfis. Esse processo automatizado é bem sucedido quando produz resultados similares aos produzidos por seres humanos.

A princípio, o cálculo de semelhança entre perfis e documentos pode envolver uma comparação direta de palavras ou sentenças usadas nos mesmos. Na realidade, porém, os vocabulários usados mostram uma variedade substancial e o número de palavras ou sentenças pode ser tão grande, que uma comparação completa de texto entre diferentes ítens de informação torna-se impossível. O conteúdo de documentos e perfis pode então ser caracterizado associando-se descrições especiais, ou índices, aos ítens de informação e, assim, representando o conteúdo do texto através desses índices. Os índices podem ser usados como substitutos de documentos ou perfis durante as operações de busca e recuperação. O processo de construir substitutos de documentos associando identificadores a ítens de informação é conhecido como indexação.

2.1.1.1 Indexação de Texto

No passado, operações de indexação eram normalmente realizadas por especialistas treinados em associar descrições a conteúdos. Atualmente, o texto dos documentos é usado como base para indexação e a análise do texto é feita por procedimentos automáticos.

A eficiência de qualquer sistema de indexação é controlada por dois parâmetros principais: a exaustividade da indexação e a especificidade de termo (SALTON, 1989). O primeiro reflete o grau no qual todos os aspectos de um assunto são realmente reconhecidos no produto da indexação. O segundo refere-se ao grau de generalização ou especialização dos termos. Quando termos genéricos são usados para a indexação, muitos ítens de informação úteis são provavelmente recuperados, junto com uma proporção substancial de ítens irrelevantes.

Ao julgar os efeitos de exaustividade e especificidade, é conveniente fazer referência aos parâmetros de eficácia usados em recuperação de informações: lembrança (ou *recall*) e precisão. *Recall* (R) é a proporção de material relevante que é recuperado, enquanto precisão (P) é a proporção de material recuperado que é relevante. Ou seja:

$$R = \frac{\text{no de ítens relevantes recuperados}}{\text{no total de ítens relevantes na coleção de documentos}} \qquad (1)$$

$$P = \frac{\text{no de ítens relevantes recuperados}}{\text{no total de ítens recuperados}} \qquad (2)$$

Na prática, é difícil obter simultaneamente bons valores de *recall* e precisão, por isso deve-se buscar um compromisso entre os dois. Quando o vocabulário de indexação é estreito e específico, a precisão da recuperação é favorecida, às custas do *recall*. Por outro lado, quando o vocabulário é amplo e não específico, o *recall* é favorecido às custas da precisão. Em geral, deve-se optar por um ponto médio no espectro do desempenho, onde nem *recall* e nem precisão sejam excessivamente baixos (SALTON, 1989).

Outro aspecto relativo à teoria de indexação relaciona-se ao uso de métodos de indexação de termos únicos em contraste com os métodos que usam termos em contexto, conhecidos como multi-termos ou baseados em frases. No primeiro caso, os índices associados aos ítens de informação consistem de termos isolados, cada um pertinente a algum aspecto do conteúdo do texto, não havendo indicadores de relacionamentos entre termos. No segundo caso, são construídas associações entre os termos usados para a indexação.

Métodos baseados em termos únicos possuem limitações devido à duas razões principais. Em primeiro lugar, termos individuais usados fora de contexto frequentemente possuem significado ambíguo e, por outro lado, o mesmo conceito pode ser descrito por muitos termos diferentes. Em segundo lugar, muitos termos são, ou específicos demais ou vagos demais para serem úteis na indexação. Métodos que consideram relações entre termos não possuem tais limitações e podem representar vários aspectos semânticos das informações. São, entretanto, muito mais difíceis de utilizar.

Este trabalho adota um método de indexação de termo individual e por essa razão as teorias sobre as quais esse tipo de método se baseia são detalhadas a seguir.

2.1.1.2 Teorias de Indexação de Termo Individual

A associação de termos índices a documentos e consultas é feita com o intuito de distinguir material relevante de irrelevante. Entretanto, na ausência de exemplos específicos de consultas e avaliações de relevância do usuário, pouco se sabe a priori sobre o que faz um item ser relevante ou não. Uma alternativa para a indexação

consiste, então, em se considerar as propriedades de ocorrência dos termos em coleções de documentos, em vez de analisar a relevância de documentos individuais.

Na composição de textos escritos, palavras com funções gramaticais como “e”, “de”, “ou” e “mas” exibem frequências de ocorrências aproximadamente iguais em todos os documentos de uma coleção. Além disso, a maioria dessas palavras caracterizam-se por altas frequências de ocorrência em todos os textos. Por outro lado, palavras não funcionais que podem estar relacionadas ao conteúdo de documentos tendem a ocorrer com frequências altamente variáveis nos diferentes textos de uma coleção. Portanto, a frequência de ocorrência de palavras não funcionais pode ser usada para indicar a importância do termo para a representação do conteúdo.

Vários trabalhos de indexação foram baseados em cálculos de frequência de termos (tf). Entretanto, essa medida satisfaz apenas um do objetivos básicos da recuperação, o *recall*. Não se obtém uma alta precisão quando se associam termos de alta frequência a textos de documentos porque alta precisão implica na habilidade de distinguir documentos individuais uns dos outros a fim de se evitar a recuperação indesejada de ítens irrelevantes. Um termo de alta frequência é, portanto, aceitável para fins de indexação, somente se sua frequência de ocorrência não é igualmente alta em todos os documentos de uma coleção. A precisão é, de fato, melhor atendida por termos que ocorrem raramente em coleções de documentos, pois tais termos são certamente capazes de distinguir os poucos documentos nos quais eles ocorrem dos muitos nos quais estão ausentes. Se a frequência de documento df_j é definida como o número de documentos em uma coleção de N documentos nos quais o termo T_j ocorre, então uma indicação apropriada de valor de termo como um discriminador de documento pode ser dada usando uma função inversa da frequência de documento do termo. Um típico fator de frequência inversa de documento (idf) é dada por (SPARCK JONES, 1972):

$$\text{Idf} = \log\left(\frac{N}{\text{df}_j}\right) \qquad (3)$$

As considerações anteriores sobre frequência de termo e frequência de documento podem ser combinadas em um único modelo de indexação baseado em frequência, postulando que os melhores termos para indexação, ou seja, aqueles que satisfazem ambas as funções de *recall* e precisão, são os que ocorrem frequentemente

em documentos individuais, mas raramente no restante da coleção. Um típico indicador da importância de termos desse tipo é o produto (tf x idf), onde a importância, ou peso, $w_{i\,j}$ de um termo T_j em um documento D_i é definido como a frequência do termo (tf) multiplicada pelo inverso da frequência de documento (idf) (SALTON, YANG, 1973, SALTON *et al*., 1975):

$$w_{ij} = tf_{ij} \times \log(\frac{N}{df_j}) \qquad (4)$$

O cálculo dos pesos dos termos de um documento consiste em uma das etapas da construção de índices para esse documento.

Um esquema geral para a indexação automática de texto amplamente utilizado em trabalhos de filtragem e recuperação de informações pode ser descrito através das etapas descritas a seguir (SALTON, 1989).

1. Identificar as palavras (termos) individuais que ocorrem nos documentos de uma coleção.
2. Eliminar palavras funcionais comuns consultando um dicionário especial ou "*stop list*", que contém uma lista de palavras funcionais de alta frequência.
3. Usar uma rotina automática de extração de sufixos e prefixos ("*stemming*") para reduzir cada palavra restante à forma de radical, reduzindo a uma forma comum todas as palavras que exibem a mesma raiz.
4. Para cada radical de palavra T_j em um documento D_i, calcular um fator de peso $w_{i\,j}$ composto pela frequência de termo e pelo inverso da frequência de documento, como proposto na expressão (4) acima. SALTON (1989) sugere que seja escolhida uma frequência limite tf_{min}, e sejam considerados apenas os termos com tf maior do que tf_{min} para representar o documento.
5. Representar cada documento D_i pelo conjunto de radicais de palavras T_j e seus respectivos pesos, isto é: $D_i = (T_1, W_{i\,1} ; T_2, W_{i\,2} ; ....)$.

Esse esquema pode ser aplicado a textos de consultas ou de documentos, assumindo-se que sentenças de consultas são originalmente disponíveis na forma de linguagem natural.

Vários modelos matemáticos foram propostos com o objetivo de representar documentos e consultas no contexto de IR. Entre eles, destacam-se:

- **modelo Booleano** (HARMAN *et al*., 1992) - compara sentenças de consultas booleanas com os conjuntos de termos usados para identificar o conteúdo de documentos;
- **modelo probabilístico** (ROBERTSON, SPARCK-JONES, 1976) - baseado no cálculo de probabilidades de relevância para os documentos de uma coleção; e
- **modelo vetor-espacial** (SALTON, 1968) - representa consultas e documentos por conjuntos de termos semelhantes aos que foram descritos acima, e calcula similaridades globais entre documentos e consultas.

Desses, o modelo vetor-espacial é o mais simples, em muitos casos o mais produtivo, e é o que será adotado neste trabalho.

2.1.1.3 Modelo Vetor-Espacial

O modelo vetor-espacial assume que um conjunto de termos disponíveis é usado para identificar os documentos armazenados e as necessidades de informações do usuário (SALTON, 1968). Consultas e documentos, podem, então, ser representados como vetores de termos:

$$D_i = (a_{i1}, a_{i2} , ... a_{it}) \qquad (5)$$

e

$$Q_j = (q_{j1}, q_{j2} , ... q_{jt}) \qquad (6)$$

onde os coeficientes a_{ik} e q_{jk} representam os valores do termo k no documento D_i ou na consulta Q_j, respectivamente. a_{ik} (ou q_{jk}) é tipicamente configurado para 1 (um) quando o termo k aparece e 0 (zero) quando está ausente do vetor. Uma alternativa a esse esquema de representação binário é fazer os coeficientes do vetor assumirem valores numéricos que reflitam a importância do termo no respectivo documento ou consulta.

Seja uma situação na qual t termos distintos estão disponíveis para caracterizar o conteúdo de um documento. Cada um dos t termos pode ser identificado por um vetor termo T e um espaço vetorial pode ser definido onde os vetores T são linearmente independentes. Em tal espaço, qualquer vetor pode ser representado como uma

combinação linear dos t vetores termos. Portanto, o r-ésimo documento D_r pode ser escrito como (SALTON, 1989):

$$D_r = \sum_{i=1}^{t} a_{ri} \times T_i \qquad (7)$$

onde os a_{ri}'s são interpretados como os componentes de D_r ao longo do vetor T_i.

Em um espaço vetorial, a similaridade entre dois vetores x e y pode ser medida pelo produto $x \cdot y = |x||y| \cos \alpha$, onde $|x|$ é o comprimento de x e α é o ângulo entre os dois vetores. Portanto, dado um documento D_r e uma consulta Q_j representados na forma da expressão (7), a similaridade entre consulta e documento pode ser calculada como:

$$D_r \cdot Q_s = \sum_{i,j=1}^{t} a_{ri} q_{sj} T_i \cdot T_j \qquad (8)$$

Esse cálculo depende da especificação dos componentes da consulta e do documento, e do conhecimento das correlações de termos $T_i \cdot T_j$ para todos os pares de termos. Os componentes vetoriais são usualmente gerados por uma operação de indexação do tipo descrito na seção 3.3.2.2, a partir da qual se obtém uma matriz de termos para a coleção de documentos.

As correlações dos termos não são geralmente disponíveis a priori, e não são fáceis de gerar. Esse problema costuma ser contornado assumindo-se que os termos são não correlacionados, o que faz com que os vetores de termos sejam ortogonais ($T_i \cdot T_j$ = 0, exceto para i = j e $T_i \cdot T_j$ = 1). Com isso, a expressão de cálculo da similaridade é reduzida para a forma de simples soma de produtos:

$$\text{Sim}(D_r, Q_s) = \sum_{i,j=1}^{t} a_{ri} q_{sj} \qquad (9)$$

Essa fórmula pode ser usada com vetores de comprimento variável, exibindo um número variável de termos. Entretanto, na maioria dos trabalhos de recuperação de

informações, os vetores são geralmente normalizados, o que permite a substituição da fórmula de similaridade por expressões normalizadas, tais como (SALTON, 1989):

$$\text{Sim}(D_r, Q_s) = \frac{\sum_{i=1}^{t} a_{ri} q_{si}}{\sqrt{\sum_{i=1}^{t} (a_{ri})^2 \times \sum_{i=1}^{t} (q_{si})^2}} \qquad (10)$$

Há três principais razões pelas quais é vantajoso gerar coeficientes de similaridade entre consultas e documentos em ambientes de IR:

- os documentos podem ser organizados em ordem decrescente de similaridade com a consulta, possibilitando ao usuário selecionar que ítens avaliar primeiro.
- o tamanho do conjunto recuperado pode ser adaptado aos requerimentos do usuário, recuperando apenas os poucos ítens de maior classificação para usuários casuais e fornecendo um grupo de ítens mais exaustivo para usuários especializados que podem requerer alto *recall*.
- ítens previamente recuperados em uma busca podem ajudar a melhorar as formulações de consultas através de realimentação por relevância.

A principal desvantagem do modelo vetorial relaciona-se à ortogonalidade assumida, e, portanto, à independência entre os termos. Independência entre termos significa que conhecer o peso $w_{i,j}$ associado ao par termo-documento (k_i, d_j) não dá qualquer informação sobre o peso $w_{i+1,\,j}$ associado ao par ($k_{\,i+1}$, d_j). Tal assunção é claramente uma simplificação, porque as ocorrências de termos índices em um documento não podem ser consideradas como não relacionadas. Seja o exemplo em que os termos “computador” e “rede” são usados para indexar um documentos que envolve a área de redes de computadores. Frequentemente, nesse documento, o aparecimento de uma dessas duas palavras atrai o aparecimento da outra. Então essas palavras são correlacionadas e seus pesos poderiam refletir essa correlação.

Embora a independência mútua seja uma grande simplificação, ela facilita muito a tarefa de calcular pesos de termos e permite uma rápida classificação dos resultados. Explorar as correlações entre termos para melhorar a classificação final de documentos não é uma tarefa fácil e não há exemplos na literatura de abordagens que tenham

demonstrado claramente que correlações de termos são de fato vantajosas para fins de classificação de coleções genéricas (BAEZA-YATES, RIBEIRO-NETO, 1999).

Outra desvantagem do modelo vetor-espacial é a falta de justificativa teórica para algumas das operações de manipulação dos vetores. Por exemplo, a escolha de uma medida particular de similaridade para uma certa aplicação não é prescrita por quaisquer considerações teóricas, e é deixada a cargo do usuário.

A despeito de sua simplicidade e das desvantagens citadas, o modelo vetor-espacial é uma estratégia de classificação bastante flexível para uso com coleções genéricas. Métodos de classificação alternativos, como o modelo probabilístico (ROBERTSON, SPARCK-JONES, 1976) ou o booleano (HARMAN *et al*., 1992) têm sido comparados com o modelo vetor-espacial e o consenso parece ser que, em geral, o modelo vetor-espacial é superior ou tão bom quanto as alternativas conhecidas. Além disso, ele é simples e rápido. Por essas razões, é um modelo bastante popular atualmente (BAEZA-YATES, RIBEIRO-NETO, 1999).

2.1.1.4 Considerações Especiais Para a Indexação de Páginas *Web*

Uma das dificuldades de aplicar o modelo vetor-espacial para um corpo grande e dinâmico de dados tal como a *Web* é que é impossível medir acuradamente as frequências de documentos. Em vez disso, elas são estimadas usando uma amostra de páginas obtidas aleatoriamente. Adota-se então um dicionário padrão construído a partir dessas páginas. Caso um termo de um documento não conste no dicionário, assume-se que sua frequência de documento é 1 (um).

É importante mencionar outras dificuldades específicas ao se trabalhar com páginas *Web*. Tais dificuldades não são comumente descritas na literatura de recuperação de informações, que lida geralmente com domínios mais uniformes como artigos de jornais ou *usenet*. Em geral, os problemas relacionados a aplicações voltadas para a *Web* podem ser divididos em duas categorias: problemas com os dados em si e problemas que dizem respeito ao usuário e sua interação com o sistema (BAEZA-YATES, RIBEIRO-NETO, 1999). Os principais problemas relacionados aos dados são descritos a seguir.

- Dados distribuídos - os dados na *Web* encontram-se espalhados por muitos computadores e plataformas. Esses computadores são interconectados sem uma topologia pré-definida e com as conexões da rede variando grandemente em confiabilidade e comprimento de banda disponíveis.
- Alta porcentagem de dados voláteis - devido à dinâmica da Internet, novos computadores e dados podem ser adicionados ou removidos facilmente (estima-se que 40% da *Web* mude a cada mês (KAHLE, 1997). Há também grande flutuação de *links* e problemas de realocação quando domínios ou nomes de arquivos mudam ou desaparecem.
- Grande volume - o crescimento exponencial da *Web* possui problemas de "escalabilidade" que são difíceis de resolver.
- Dados redundantes e desestruturados - as páginas HTML não são bem estruturadas e é comum que muitos dados da *Web* sejam repetidos ("espelhados" ou "copiados") ou bastante similares.
- Qualidade dos dados - a *Web* pode ser considerada como um novo meio de publicação. Entretanto não há, na grande maioria dos casos, nenhum processo editorial. Os dados podem, então, ser falsos, inválidos (por exemplo, por serem muito antigos), mal escritos ou podem conter muitos erros de diferentes fontes (gramaticais, na sintaxe HTML, de digitação, de OCR, etc). É preciso ter o cuidado de projetar algoritmos robustos o suficiente para suportarem tamanhos ou formatos inesperados de dados.
- Dados heterogêneos - além de ter que lidar com múltiplos tipos de meios e, portanto, múltiplos formatos, também há o problema de diferentes idiomas e diferentes alfabetos.
- Dados não textuais - há uma tendência crescente em diminuir a proporção de texto ASCII facilmente analisável nas páginas WWW devido ao advento de ferramentas que permitem projeto e formatos mais flexíveis (Java, Acrobat PDF, etc). Além disso, páginas *Web* projetadas profissionalmente em geral contêm grande número de imagens.

A maioria desses problemas (tais como a variedade de tipos de dados e a sua pobre qualidade) não são solucionáveis simplesmente por melhorias de *software*. De

fato, muitos deles não mudarão (como no caso da diversidade de idiomas) porque são características intrínsecas à natureza humana.

A segunda classe de problemas são aqueles enfrentados pelo usuário durante a interação com o sistema de recuperação. Há basicamente dois problemas: como especificar uma consulta e como interpretar a resposta fornecida pelo sistema. Sem levar em conta o conteúdo semântico de um documento, não é fácil especificar precisamente uma consulta, a menos que ela seja muito simples. Além disso, mesmo se o usuário for capaz de elaborar bem a consulta, a resposta podem ser milhares de páginas WWW. Como manipular uma resposta grande? Como classificar os documentos? Como selecionar os documentos que realmente são de interesse do usuário?

Então, o desafio global, a despeito dos problemas intrínsecos da *Web*, consiste em submeter uma boa consulta ao sistema de busca e obter uma resposta gerenciável e relevante (BAEZA-YATES, RIBEIRO-NETO, 1999). Na prática, deve-se tentar obter essa última meta mesmo para consultas pobremente formuladas.

MODIFICAÇÕES DE CONSULTAS E REALIMENTAÇÃO POR RELEVÂNCIA

Uma das operações mais importantes e difíceis em recuperação de informações é gerar consultas que possam identificar sucintamente documentos relevantes e rejeitar os irrelevantes. Os usuários frequentemente submetem consultas contendo termos diferentes dos termos usados para indexar uma grande parte dos documentos relevantes. VAN RIJSBERGEN (1986) falou dos limites de fornecer bons resultados baseado apenas na consulta inicial, e indicou a necessidade de modificar aquela consulta para aumentar o desempenho da recuperação.

Uma vez que se reconhece a dificuldade de realizar uma pesquisa com sucesso na primeira tentativa, costumam-se conduzir buscas iterativamente, e reformular as sentenças da consulta baseando-se na avaliação dos documentos previamente recuperados. Um método para gerar automaticamente formulações de consultas melhoradas é a realimentação por relevância (IDE, 1971, SALTON, 1989, ROCCHIO, 1971). Em um ciclo de realimentação por relevância, uma lista de documentos

recuperados é apresentada ao usuário que, após examiná-la, marca aqueles que são relevantes. Uma consulta inicial pode, então, ser melhorada iterativamente tomando-se um vetor (de termos) de consulta disponível e adicionando termos a partir de documentos relevantes, enquanto se subtraem termos a partir de documentos irrelevantes. Um único ciclo de realimentação frequentemente produz melhoras de 40 a 60% na precisão da busca (SALTON, 1989).

A principal consideração por trás deste método é que documentos relevantes a uma consulta podem ser considerados semelhantes, no sentido de que eles são representados por vetores razoavelmente similares. Consequentemente, se um documento recuperado foi identificado como relevante para uma dada consulta, a formulação da consulta pode ser melhorada aumentando sua similaridade com esse documento. Resta, então, encontrar uma forma eficiente de "mover" uma dada consulta em direção aos ítens relevantes e "afastá-la" dos irrelevantes. Os passos em realimentação por relevância podem ser conduzidos iterativamente de tal modo que a consulta aproxime-se da consulta ótima gradualmente, conforme a relevância de mais documentos se torna conhecida.

Há três maneiras clássicas de reformular uma consulta e gerar uma consulta modificada. Maiores explicações, bem como as fórmulas usadas, podem ser vistas em (BAEZA-YATES, RIBEIRO-NETO, 1999).

As principais vantagens do método de realimentação por relevância são a simplicidade e os bons resultados fornecidos. A simplicidade é obtida pelo fato dos pesos modificados serem calculados diretamente a partir do conjunto de documentos recuperados. Os bons resultados são observados experimentalmente e devem-se à característica dos vetores de consultas modificados realmente refletirem uma parte da semântica da consulta desejada. Além dessas, este método apresenta ainda as seguintes vantagens em relação a outras estratégias de reformulação de consultas (BAEZA-YATES, RIBEIRO-NETO, 1999):

- protege o usuário de detalhes do processo de reformulação, já que ele apenas tem que fornecer um julgamento de relevância sobre os documentos;
- quebra a tarefa completa de busca em uma sequência de pequenos passos mais fáceis de manipular; e

- fornece um processo controlado para enfatizar os termos relevantes dos documentos, enquanto diminui a ênfase dos irrelevantes.

No modelo vetor-espacial, o processo de realimentação por relevância pode ser usado para melhorar a consulta original de duas formas:

- termos presentes em documentos previamente recuperados que tenham sido identificados como relevantes para a consulta do usuário são adicionados aos vetores representando as consultas originais (esse processo é conhecido como expansão da consulta (FRAKES, BAEZA-YATES, 1992));
- os pesos dos termos já presentes na consulta original são alterados conforme as características de ocorrência dos termos nos documentos previamente recuperados (processo conhecido como repesagem dos termos (FRAKES, BAEZA-YATES, 1992)).

CONSIDERAÇÕES FINAIS

Este capítulo tratou dos principais conceitos relacionados à busca automática de informações, detendo-se particularmente na tecnologia de filtragem de texto e apresentando o modelo de indexação de documentos utilizado no presente trabalho. O modelo apresentado foi o vetor-espacial que, apesar de algumas desvantagens, é bastante popular devido principalmente a sua simplicidade de implementação e sua eficiência comprovada em diversos trabalhos (BAEZA-YATES, RIBEIRO-NETO, 1999). Foi também abordado o método de modificação de consulta baseado em realimentação por relevância, que é o principal mecanismo usado neste trabalho através do qual um agente aprende as preferências dos usuários e adapta-se a elas.

O próximo capítulo apresenta a descrição geral do Sistema Fenix, sua metodologia de desenvolvimento e os componentes de sua arquitetura.

4 SISTEMA FENIX

Este trabalho propõe o uso da tecnologia de agentes para a construção de um sistema de filtragem personalizada de informações, denominado de Fenix. O sistema Fenix é formado por um conjunto de agentes autônomos, adaptativos e fixos, cuja função é satisfazer as necessidades de informações de seus usuários. Os agentes realizam buscas por documentos na Internet, recebem realimentação do seu usuário sobre a relevância dos ítens recuperados e constróem um perfil de interesses, que vai se especializando ao longo de múltiplas sessões de interação. Os resultados das buscas vão sendo gradualmente melhorados conforme o perfil se ajusta aos interesses específicos de seu usuário, com um número crescente de documentos relevantes sendo recuperados e irrelevantes sendo descartados. Os mecanismos de aprendizagem adotados pelos agentes são realimentação por relevância (*relevance feedback*) e algoritmos genéticos. Os perfis usados para filtrar as informações baseiam-se em termos que são comparados com os conteúdos dos documentos pesquisados. O modelo adotado para a representação de perfis e documentos é o vetor-espacial, descrito no capítulo 3. O domínio de atuação dos agentes é a *Web*, sendo uma página WWW considerada como um item de informação para o sistema.

No sistema proposto, cada agente é modelado como uma coleção de perfis individuais. Em conjunto, todos os perfis de uma coleção tentam atender aos interesses de um usuário e adaptar-se a eles. Um usuário pode ter vários agentes, cada um atendendo às suas necessidades de informação em um determinado assunto.

O agente é o responsável por disparar a execução de tarefas de busca e filtragem, uma para cada perfil. Essas tarefas são autônomas, já que, uma vez disparadas, comunicam-se com diferentes mecanismos de busca e classificam os documentos obtidos nas buscas sem necessidade de interação com o usuário, inferindo as preferências do mesmo a partir de seu perfil. O caráter autônomo das tarefas de busca e filtragem permite que elas sejam consideradas como sub-agentes no sistema Fenix. Cada um dos sub-agentes, utilizando mecanismos de busca diferentes, percorre as páginas da *Web* em busca de documentos contendo as palavras-chaves fornecidas pelo usuário. O conjunto de documentos obtidos é submetido ao processo de filtragem, de

acordo com o modelo adotado. Os documentos selecionados são aqueles com maior grau de similaridade com o perfil correspondente. Esses documentos são, então, fornecidos para o agente responsável, que é encarregado de reunir os resultados de todos os sub-agentes, classificá-los de acordo com sua relevância potencial para o assunto específico e apresentá-los para o usuário. O usuário pode fornecer realimentação positiva ou negativa sobre um documento. A realimentação do usuário tem o efeito de modificar o perfil usado para recuperar aquele documento, baseado na sua relevância. A coleção de perfis está continuamente evoluindo em resposta a mudanças dinâmicas nas preferências do usuário. Essas mudanças podem ser introduzidas através de alterações nos termos usados para as consultas ou quando for aplicado o módulo de algoritmo genético.

Este capítulo apresenta a descrição do sistema Fenix. A seção 4.1 aborda a metodologia usada no seu desenvolvimento e a seção 4.2 descreve os componentes funcionais que fazem parte de sua arquitetura.

METODOLOGIA DE DESENVOLVIMENTO

A adoção de uma metodologia de desenvolvimento é fundamental para se garantir a qualidade e a facilidade de manutenção do sistema sendo projetado, especialmente para sistemas com um certo grau de complexidade. Uma boa metodologia deve especificar as atividades a serem executadas durante o processo de desenvolvimento, bem como a sequência de execução, identificando, para cada uma, seus pré-requisitos e os produtos a serem gerados. Além disso, deve possuir modelos e alguma forma de notação gráfica para representar os diferentes aspectos do sistema projetado.

As quatro atividades básicas do desenvolvimento de um sistema de informações são: o levantamento dos requisitos, a análise, o projeto e a implementação. Uma fase posterior de manutenção do sistema costuma também ser incluída. Cada metodologia propõem diferentes abordagens e modelos a serem seguidos em cada atividade.

O Sistema Fenix foi desenvolvido segundo a metodologia de orientação a objetos. A escolha dessa abordagem baseou-se no seu uso cada vez mais frequente no projeto de sistemas de informações e nas suas inúmeras vantagens, entre as quais destacam-se:

- a reutilização de componentes é oferecida;
- a interoperabilidade é facilitada;
- a compreensão do sistema é mais fácil, pois o *gap* semântico entre a realidade e o modelo é pequeno;
- as modificações no modelo tendem a ser locais, já que elas frequentemente resultam de um item individual representado por um único objeto. Como resultado, é obtido um sistema facilmente manutenível e extensível.

Assim, optou-se por adotar a metodologia de orientação a objetos em todas as fases do desenvolvimento do sistema, desde a análise até a sua implementação com uma linguagem de programação orientada a objetos.

Na abordagem orientada a objetos não há uma distinção clara entre as atividades realizadas durante as fases de análise e projeto. O mesmo esquema notacional é adotado em ambas as fases. Na análise são definidos os objetos e as classes que irão compor o sistema. Essa fase inclui a identificação de atributos e operações para as classes. No projeto, as classes e objetos definidos são refinados, acrescentando-lhes detalhes de projeto adicionais (PRESSMAN, 1995). Além disso, define-se como os objetos são derivados de cada classe e como esses objetos se comunicam e se inter-relacionam. A estrutura global de relacionamento entre os objetos consiste na arquitetura do sistema.

Após a análise e o projeto, passa-se para a especificação dos componentes de programa (módulos) que são combinados para formar um programa completo. Embora um componente de programa seja uma abstração da fase de projeto, ele deve poder ser representado no contexto da linguagem de programação com a qual o sistema será implementado.

Modelo-Vista-Controle

Para a identificação dos objetos que compõem o sistema Fenix foi adotada a infra-estrutura MVC (modelo-vista-controle), um modelo clássico de arquitetura usado em sistemas orientados a objetos (BURBECK, 1987). Aplicações MVC são divididas em várias tríades, cada uma compreendendo um relacionamento entre um objeto modelo, um objeto vista e um objeto controlador. Esses objetos interagem através de mensagens e notificações, conforme a figura 3.

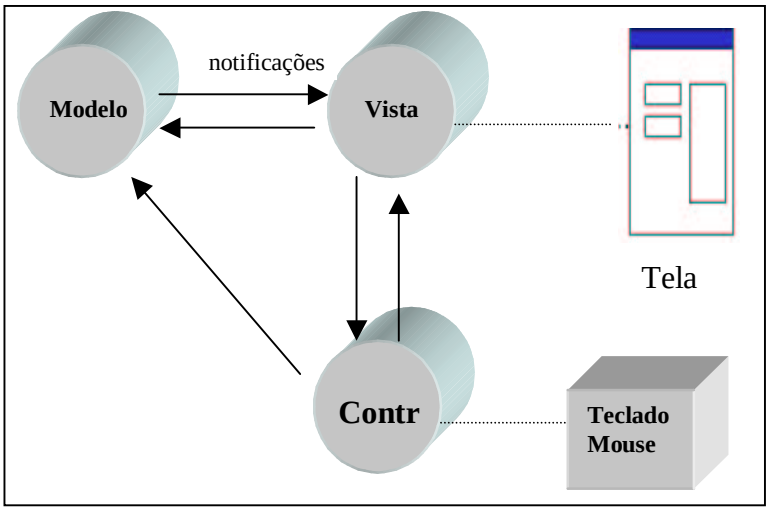


Figura 3: Infra-estrutura Modelo Vista e Controle

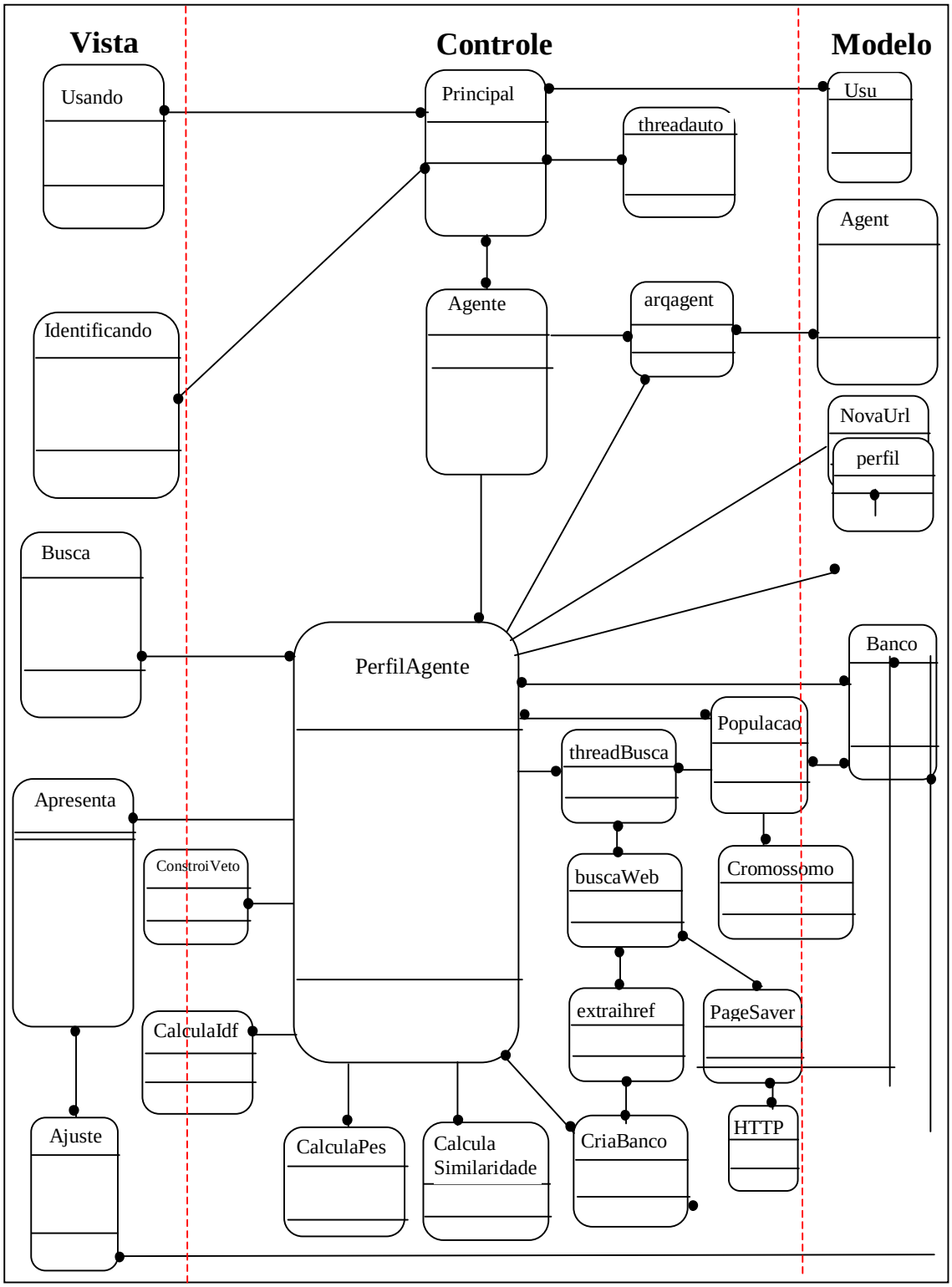
Os componentes do sistema são, então, explicitamente separados e manipulados por esses três tipos de objetos, cada um especializado em sua tarefa:

- **objetos Modelo** - gerenciam os dados e o comportamento do domínio da aplicação. Representam uma abstração do mundo externo, dividido em componentes que são os objetos do sistema. Objetos da categoria modelo respondem a requisições de informações sobre seu estado (geralmente recebidas de objetos vista), e respondem a instruções para alterar seu estado (geralmente recebidas de objetos controladores);
- **objetos Vista** - gerenciam a interface gráfica da aplicação. São objetos especializados em apresentar o modelo, representando uma abstração de uma janela ou painel. Eles recebem e enviam mensagens para objetos controladores, e enviam mensagens para objetos modelo;
- **objetos Controladores** - interpretam as entradas de mouse e teclado do usuário e comandam os objetos modelo e vista para responderem apropriadamente. São objetos especializados em manipular pedidos e operações para os demais objetos do sistema.

Esses três componentes comunicam-se entre si e com outras vistas e controles ativos no sistema.

Os objetos identificados segundo o modelo MVC durante a fase de análise do sistema Fenix foram organizados como módulos funcionais na fase de projeto. Esses módulos compõem a arquitetura do sistema. Dentro de cada módulo, foram definidas ligações entre as classes componentes. Essas ligações representam conexões de mensagens trocadas entre os objetos, conexões de ocorrência, ou relações de herança.

As principais classes definidas para o sistema Fenix, categorizadas segundo o paradigma MVC, podem ser vistas no diagrama de classes da figura 4, bem como as relações entre elas. A notação adotada baseou-se em (RUMBAUGH *et al*., 1991), onde as linhas representam conexões de ocorrência entre os objetos (ou seja, relações específicas entre as instâncias de objetos) com suas respectivas cardinalidades.



ARQUITETURA

O sistema Fenix é composto de vários módulos, cada um responsável pela execução de um conjunto de funções essenciais para o objetivo do sistema. Esses módulos são implementados como conjuntos de classes, as quais, por sua vez, pertencem a uma das

categorias propostas pelo modelo MVC ("modelo-vista-controle"), descritas na seção 4.1.1 e revistas a seguir.

- **Vista** - classes responsáveis pela implementação da interface gráfica com o usuário, sendo em geral derivadas de objetos gráficos da biblioteca padrão Java, tais como painéis e janelas.
- **Controle** - classes controladoras, responsáveis por instanciar as demais classes e controlar a execução e as chamadas de seus métodos, de acordo com os pedidos do usuário.
- **Modelo** - incorporam os objetos persistentes do sistema, ou seja, os dados do domínio da aplicação a serem armazenados em arquivos de diferentes tipos.

Esta seção dá uma visão geral do funcionamento sistema Fenix. A seguir sua arquitetura é detalhada, descrevendo-se cada um dos módulos componentes.

Descrição Geral do Sistema

O sistema Fenix foi projetado como um conjunto de módulos inter-relacionados compostos por várias classes. Os módulos trocam mensagens e parâmetros entre si e são controlados por um módulo central, que exerce a coordenação sobre todos os demais.

A interação do usuário com o sistema é feita através de uma interface gráfica, a qual possibilita a comunicação entre um usuário e seus respectivos agentes. Por meio dessa interface, um usuário pode criar ou carregar agentes, iniciar sessões de busca e filtragem, ler e avaliar os documentos apresentados pelos agentes. O componente do sistema responsável pela interface gráfica é o módulo de interface com o usuário, o qual incorpora todas as classes necessárias para a obtenção das entradas do usuário e para a apresentação das saídas do sistema.

A busca por novos documentos é realizada através de mecanismos de busca disponíveis na *Web*, e o módulo de busca do sistema se encarrega de fornecer as entradas para esses mecanismos e tratar os resultados fornecidos por eles. Os resultados das buscas são armazenados localmente em estruturas de arquivos disponíveis na biblioteca padrão da linguagem Java. Esses arquivos, juntamente com outros que contêm informações sobre os agentes e os usuários, compõem o banco de dados do sistema.

Os documentos resultantes das buscas e armazenados no banco de dados local são filtrados pelo módulo de filtragem do sistema e utilizados para criar e atualizar perfis individuais de interesses dos usuários. A atualização dos perfis é feita através dos dois mecanismos de aprendizado adotados. O primeiro baseia-se na avaliação dos documentos feita pelo usuário e é controlado pelo sub-módulo de realimentação por relevância. O segundo mecanismo consiste na técnica de algoritmos genéticos, especificado como o sub-módulo de algoritmos genéticos, não implementado na presente versão do sistema. Os dois sub-módulos compõem o módulo de aprendizado.

A seguir é descrita a arquitetura completa do sistema, com o detalhamento de cada um dos seus módulos componentes.

Descrição dos Módulos

A arquitetura do sistema Fenix, conforme ilustrado na figura 5, é composta pelos seguintes módulos funcionais:

- **Módulo de Interface com o Usuário** - permite que o usuário crie seus agentes, carregue agentes já existentes, leia documentos recuperados e forneça realimentação para documentos lidos. É composto por todas as classes da categoria Vista.
- **Módulo de Aprendizado** - é responsável por garantir que o conjunto de perfis reflita os interesses do usuário, adaptando-se a eles ao longo do tempo. Os métodos de aprendizagem adotados são a realimentação por relevância e algoritmos genéticos e foram especificados como sub-módulos independentes. O sub-módulo de realimentação de relevância tem o objetivo de alterar os perfis com base na relevância dos documentos para o usuário. O sub-módulo do AG consiste na execução de todas as etapas de um algoritmo genético.
- **Módulo de Filtragem de Informações** - é responsável por encontrar, nas páginas WWW recuperadas pelo módulo de busca, documentos semelhantes aos perfis do usuário. O processo de filtragem consiste em transformar documentos em suas representações vetoriais, calcular valores de similaridade entre documentos e perfis, ordenar os documentos de acordo com sua similaridade, e apresentar ao usuário apenas aqueles com valor de similaridade maior que um limite pré-fixado.

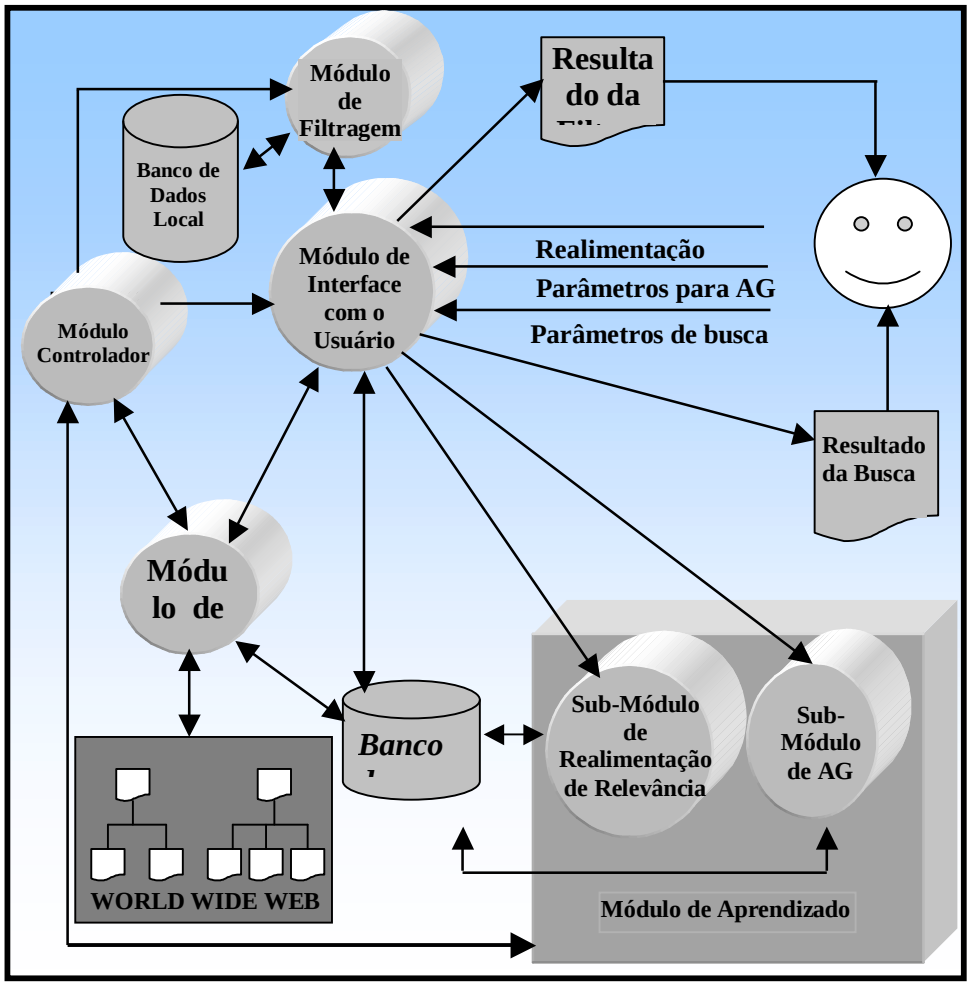


Figura 5: Arquitetura do Sistema

- **Módulo de Busca de Informações** - é responsável por invocar mecanismos de busca existentes na *Web* passando-lhes as palavras-chaves do usuário, obter seus resultados e armazená-los em um banco de dados local;
- **Banco de Dados Local** - conjunto de arquivos nos formatos ASCII e binário, contendo os dados dos usuários, os seus respectivos perfis e as páginas capturadas na rede WWW. É composto por todas as classes da categoria Modelo.
- **Módulo Controlador** - é composto pelas classes controladoras, a partir das quais todas as demais são criadas e seus métodos invocados a fim de atender às funções de cada módulo descrito.

A seguir apresenta-se a descrição detalhada de cada um dos módulos, bem como das suas classes componentes. Os diagramas de classes separados por módulo podem ser vistos no anexo 1.

MÓDULO DE INTERFACE COM O USUÁRIO

Este módulo tem a funcionalidade de apresentar uma interface gráfica através da qual é feita a interação com o usuário. Ele é composto por todas as classes do modo Vista.

A interação do usuário com o sistema Fenix se faz inicialmente através de seu cadastramento, onde ele deve informar seus dados pessoais e escolher um *login* e uma senha. O usuário cadastrado, após se identificar, pode escolher entre três opções: criar um novo agente, carregar um agente já existente ou selecionar o modo de execução autônoma do agente.

Ao criar um novo agente, o usuário deve escolher um nome, uma cor de fundo para a tela do agente e fornecer os parâmetros de busca. Inicialmente, havia sido prevista a implementação completa de um módulo de busca em Java. Dessa forma, o usuário deveria fornecer o número máximo de páginas a ser recuperado por documento, a profundidade de links por página e o número máximo de documentos a ser recuperado por sessão de busca. Entretanto, optou-se por utilizar mecanismos de busca já existentes, necessitando assim apenas implementar sua integração aos demais módulos. Com essa opção, o usuário precisa entrar somente com o número máximo de documentos exibidos por sessão (o padrão é 30), além da expressão de consulta.

Uma consulta no sistema Fenix é uma combinação de palavras-chaves (tecnicamente chamadas de termos), separadas por espaços em branco, não sendo permitido o uso de conectivos lógicos. É assumida automaticamente a existência do conectivo AND entre cada termo. Como resultado da busca inicial, é apresentada uma lista com os nomes dos documentos recuperados. Após a leitura dos documentos desejados, o usuário pode fornecer realimentação (*feedback)* positiva ou negativa conforme sua relevância para aquele assunto específico.

O usuário pode visualizar as páginas dos documentos através do botão "Exibe". Esse botão ativa o navegador local, como, por exemplo, o Netscape ou o Internet Explorer, que deve estar configurado no sistema.

O usuário pode alterar a URL de um documento, caso encontre um *link* mais interessante a partir da página inicial, através do botão "Altera". Pode, ainda, incluir manualmente uma URL de interesse, que não tenha sido recuperada pelo agente.

Ao terminar uma sessão, o usuário pode ou não salvar o agente recém-criado. Caso opte por salvar o agente, são armazenadas as referências para os documentos que receberam realimentação positiva (suas URLs). Os conteúdos desses documentos são usados para criar os vetores de termos e pesos que irão compor os perfis iniciais referentes àquele assunto e ao usuário específico. Cada documento tem, associado a ele, um campo de "*status*", que indica se o mesmo ainda não foi lido ("N"), se já foi avaliado pelo agente ("S") ou se é um documento que faz parte do perfil inicial do agente ("P"). Além disso, possui os campos "*feedback*", que armazena o valor da realimentação fornecida pelo usuário, e "*score*", que armazena o valor de similaridade com relação ao perfil, calculado pelo agente.

O usuário, ao se identificar e escolher a opção de carregar um agente existente, seleciona um dentre uma lista de agentes pertencentes a ele. A seguir, pode escolher uma das seguintes ações:

- ler algum documento recuperado;
- fornecer realimentação sobre algum documento lido;
- iniciar uma nova busca por documentos, referente ao assunto específico;
- iniciar o módulo do algoritmo genético para aquele agente.

Além das opções de criar ou carregar agentes, o usuário pode selecionar o “Modo Autônomo”, no qual são realizadas sessões de busca e filtragem sem interação com o usuário. Essa opção é totalmente controlada pela classe *threadauto*, e funciona da seguinte forma:

- o sistema verifica todos os agentes pertencentes ao usuário;
- para cada agente, é verificado o *status* de todos os seus documentos;

- caso haja documentos ainda não lidos, o usuário recebe a mensagem informando que há documentos a serem lidos e o nome do agente correspondente;
- caso todos os documentos de todos os agentes tenham sido lidos, são iniciadas novas buscas para cada um dos agentes do usuário; após as buscas, é exibida a mensagem informando que o usuário deve escolher a opção de "Visualizar" para os respectivos agentes.

Todas as classes que compõem o módulo de interface são derivadas da classe Panel do pacote AWT ("*Abstract Windows Toolkit*") de Java. Elas exibem painéis para a entrada e apresentação de dados. São elas: classe *Usando*, classe *Identificando*, classe *Apresenta* e classe *Busca* (anexo 1).

A classe *Usando* exibe um painel para a entrada dos dados de novos usuários. A classe *Identificando* permite a identificação de usuários já cadastrados, a fim de controlar o acesso ao sistema. A classe *Apresenta* exibe um painel com todas as informações referentes aos documentos pertencentes a um agente ou aos documentos recuperados em uma sessão de busca (figura 6). A classe *Busca* exibe um painel para a configuração dos parâmetros de busca.

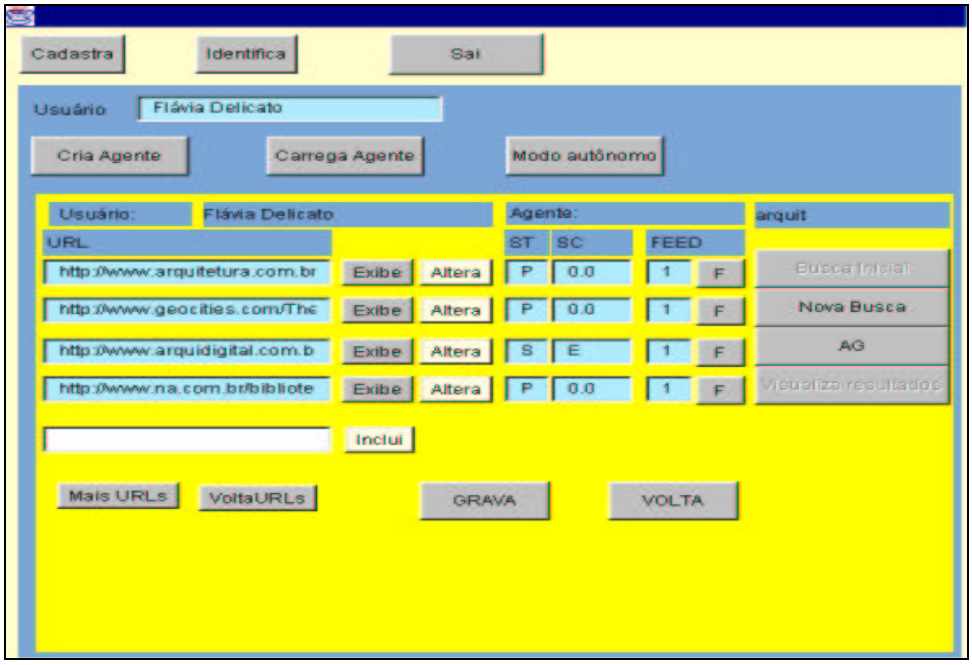


Figura 6: Painel exibido pela classe *Apresenta* para a apresentação dos documentos referentes a um agente.

Além dessas classes principais, também fazem parte do módulo de interface os painéis de criação e seleção de agentes, controlados pela classe *Agente*. Esses painéis permitem que o usuário entre com os parâmetros para a criação de um novo agente ou selecione um agente já criado para a visualização de seu conteúdo. Todas as telas do sistema podem ser vistas no anexo 2.

As classes *Identificando* e *Usando* são controladas pela classe *Principal*, enquanto as classes *Apresenta* e *Busca* são controladas pela classe *PerfilAgente*.

MÓDULO DE BUSCA DE INFORMAÇÕES

O módulo de busca é responsável por reunir as informações referentes ao assunto desejado que foram coletadas na *Web* pelos diferentes mecanismos de busca e salvá-las em um banco de dados local. Esse módulo foi inicialmente projetado para funcionar de forma semelhante aos mecanismos de busca baseados em palavras-chaves. Através da utilização de classes pré-definidas da linguagem Java, páginas WWW seriam percorridas em busca das palavras-chaves indicadas. Os *links* das páginas seriam também recuperados recursivamente até um limite fixado pelo usuário. Entretanto, ao longo do desenvolvimento do trabalho, optou-se por usar mecanismos de busca já existentes. Esse módulo, então, passou a ser responsável apenas pela interface com esses mecanismos, passando-lhes as palavras-chaves, recuperando o resultado da pesquisa e convertendo as páginas encontradas para a forma de *strings*.

Para ativar os mecanismos de busca, foi criada uma classe chamada *threadBusca* (figura 4), que é derivada da classe Thread de Java e representa uma tarefa do sistema operacional. Essa é a classe controladora do módulo de busca e é instanciada diretamente a partir da classe *PerfilAgente*, com quem troca mensagens. Como cada agente é composto por vários perfis, são criadas instâncias da classe *threadBusca* para cada perfil. Essa classe, por sua vez, comunica-se com a classe *BuscaWeb*, que faz a interação com os mecanismos de busca. Cada tarefa aciona a chamada de um mecanismo diferente e a identificação do mecanismo escolhido é um parâmetro passado para a classe *BuscaWeb* e salvo no correspondente arquivo de perfil. Foram usados os seguintes sistemas, que encontram-se disponíveis nos endereços citados no anexo 4: Altavista, Lycos, Cadê e Miner.

Os resultados das buscas vêm na forma de uma página HTML com a relação dos dez documentos mais relevantes encontrados pelo respectivo mecanismo de busca. Essa página é manipulada pelas classes *HTTP* e *PageSaver* (figura 4), que convertem o fluxo de dados fornecido para a forma de uma *string* única. Essa *string* é passada para a classe *extraihref*, a qual, por sua vez, extrai cada um dos *links* para os documentos relevantes. As URLs desses *links* são, então, passadas para a classe controladora do agente (*PerfilAgente*). Essas URLs são salvas em arquivos locais, a fim de que os processos posteriores de filtragem e seleção dos documentos possam ser realizados sem necessidade de conexão com a Internet.

As tarefas de busca são sub-agentes do sistema Fenix, pois desempenham uma função fundamental ao sistema e possuem certo grau de autonomia. Cada agente delega a esses sub-agentes o papel de obter os documentos que serão filtrados e passados para a classe *Apresenta* do módulo de interface para apresentação ao usuário. As classes desse módulo são todas controladas pelas classes *PerfilAgente e threadBusca*.

MÓDULO DE FILTRAGEM

O processo de filtragem consiste em transformar documentos em suas representações vetor-espaciais, encontrar documentos similares aos perfis e selecionar aqueles de maior pontuação para apresentar ao usuário. Várias classes foram criadas com o propósito de gerar essas representações vetor-espaciais e calcular a similaridade entre perfis e documentos, obtendo uma pontuação (*score*) para cada documento.

Após uma busca inicial, baseada nas palavras-chaves fornecidas pelo usuário, o sistema apresenta uma lista de documentos para serem lidos e avaliados pelo usuário. Os ítens que recebem realimentação positiva são convertidos para a sua representação vetorial e vão compor os perfis do usuário. Cada documento com realimentação positiva dá origem a um arquivo de perfil, que vai conter, inicialmente, a URL desse documento, seu *status* (“P”), o valor da realimentação recebida, bem como a representação vetorial do documento (termos e pesos).

Nas buscas posteriores, cada novo documento recuperado é convertido para sua representação vetorial, sua similaridade com relação ao respectivo perfil é calculada e

uma pontuação lhe é atribuída. Apenas documentos com pontuação maior do que um limite fixado são apresentados.

Representação de perfis e documentos

A representação para perfis e documentos adotada neste trabalho é baseada no modelo vetor-espacial descrito no capítulo 3. Nele, documentos e consultas são representados como vetores em um espaço vetorial. Uma distância métrica, que mede a proximidade entre os vetores, é definida sobre o espaço. Cada perfil de usuário é convertido para sua representação vetorial, bem como os documentos recuperados ao longo das sessões. Os documentos cujas representações possuem maior grau de proximidade com o vetor perfil são os resultados da filtragem. Assim, cada perfil é usado para pesquisar as páginas WWW em busca de documentos similares a ele.

Documentos

O processo de representação dos documentos foi baseado nas etapas descritas no capítulo 3. O texto completo de cada documento é analisado a fim de se extraírem as palavras ou termos individuais (termos índices), eliminar palavras funcionais a partir de uma lista de exclusão e eliminar palavras identificadas como “*tags*” HTML.

Na etapa de remoção de “*tags*” para a indexação dos documentos, deve-se ter o cuidado de identificar “*meta-tags*”, que em geral são importantes para o propósito de representação do conteúdo de suas páginas. As “*tags*” “meta”, por exemplo, são colocadas no cabeçalho de documentos e possuem diversos atributos, os quais fornecem informações sobre, por exemplo, a autoria, a localização, datas de criação e alteração, tamanho e formato do documento HTML. Os principais atributos para fins de indexação de documentos são "*description*" e "*keywords*". O atributo "*description*" retorna uma descrição resumida da página e o atributo "*keywords*" fornece um conjunto de palavras-chave a serem associadas ao conteúdo da página. “*Tags*” “meta” podem ser bastante úteis para fornecer palavras-chave e descrições sobre páginas que, por várias razões, perdem seu conteúdo textual, como é o caso, por exemplo, de páginas que usam "*frames*".

No sistema proposto, ao ser identificada uma “*tag*” “meta”, o conteúdo de seus atributos "*keywords*" e "*description*" é usado para compor o vetor de termos do documento, já que contém palavras significativas para a representação do texto.

Após a identificação dos termos que serão usados para indexar o documento, aplica-se um algoritmo para extração de sufixos e prefixos (operação conhecida como “*stemming*”). A aplicação de algoritmos desse tipo reduz as palavras a seus radicais, eliminando redundâncias e diminuindo o tamanho do vetor de termos gerado. Neste trabalho foi implementado o algoritmo de Porter (PORTER, 1980), amplamente utilizado em sistemas de filtragem para a extração de radicais de palavras da língua inglesa. A opção de executar ou não o algoritmo é deixada a cargo do usuário, e só deve ser empregada quando os documentos forem escritos em inglês. Documentos em outros idiomas têm seus termos armazenados integralmente nos vetores, sem redução à forma de radicais.

Uma vez que os termos não são todos igualmente importantes para a representação do conteúdo do texto do documento, pesos são associados aos termos, na proporção de sua presumida importância para fins de identificação do conteúdo. O texto de um documento é então representado como um vetor de termos e seu respectivo vetor de pesos Ti = { Wij }, onde Wij representa o peso do termo tj no texto Ti.

Um documento WWW pode conter vários campos em seu cabeçalho, como URL, autor, local, tipo de gerador HTML, etc, seguidos pelo texto que compõe o corpo da página. Neste trabalho, é considerado relevante apenas o texto em si, aqui denominado de campo palavra-chave e que consiste na área delimitada pelas “*tags*” </BODY> da página.

O vetor de pesos para o campo palavra-chave é obtido através de uma análise completa do texto do documento. O peso de um termo do campo palavra-chave é o produto de sua frequência de termo (tf) pelo seu inverso da frequência de documento (idf), conforme descrito no capítulo 3. A frequência de termo normalmente reflete a importância do termo no documento. O inverso da frequência do documento (idf) é um fator que realça os termos que aparecem em poucos documentos enquanto desvaloriza os termos que ocorrem em muitos documentos. O efeito resultante é que as feições

específicas do documento ficam destacadas enquanto as feições distribuídas pelo conjunto de documentos têm menor importância.

Neste trabalho adotou-se para o cálculo do peso dos termos a fórmula normalizada proposta por SALTON e BUCKLEY (1988):

$$W_{iq} = (0.5 + 0.5 \frac{tf_{iq}}{\max tf_q}) \times idf_q, \qquad (11)$$

onde tf_{iq} é o número de ocorrências do termo q no documento i, max tf_i é a frequência máxima de qualquer termo no documento i e idf_q é o inverso da frequência de documento do termo q na coleção de documentos.

Para o cálculo de idf foi adotada a fórmula sugerida por CROFT e HARPER (1979), onde o valor de idf é normalizado para compensar o efeito de diferentes tamanhos
de documentos:

$$idf_q = \log_2 (\frac{N - n_i}{n_i}), \qquad (12)$$

onde N é o número total de documentos na coleção, dos quais n_i contêm um termo q.

Neste trabalho, uma coleção de documentos é composta de todos os documentos recuperados por um perfil em uma sessão de busca. O tamanho da coleção é um parâmetro que pode ser escolhido pelo usuário (o padrão é 30) e é mantido constante para o perfil. Cada perfil aciona sempre o mesmo mecanismo de busca. Os documentos recuperados nas diferentes sessões irão, portanto, variar pouco, já que a introdução de novos documentos nas bases consultadas pelos mecanismos de busca não ocorre com frequências muito altas. Esse fator é importante para a validade de um perfil pois os pesos dos termos no perfil correspondem a medidas relativas a um determinado conjunto de documentos. Se a coleção mudar, os pesos devem ser recalculados, a fim de que o perfil possa ser considerado representativo para avaliar os documentos daquela coleção.

O algoritmo de indexação de texto foi implementado como um processo de 3 estágios. Enquanto as frequências de termos (tf) em um documento podem ser calculadas para um documento individual, os inversos das frequências de documentos (idf) não podem ser calculados até a coleção completa de documentos ter sido analisada.

No primeiro estágio, os documentos são analisados e os termos individuais são extraídos a fim de compor o vetor de termos. No segundo estágio, os valores de tf são calculados e armazenados. Esse processo é repetido para cada um dos N documentos e é realizado pelos métodos da classe *ConstroiVetor* (figura 4). No terceiro estágio, os idf's são calculados. Uma segunda análise sobre os dados armazenados fornece as contagens de documentos para todos os termos, isto é, o número de documentos em que cada termo aparece. Esses cálculos são feitos pelos métodos da classe *CalculaIdf* (figura 4). A seguir, calculam-se os pesos dos termos através do produto de tf e idf, conforme descrito na equação (11), e o vetor de pesos para cada documento pode ser gerado. Esse cálculo é feito pelos métodos da classe *CalculaPeso* (figura 4).

Perfis

Um perfil consiste de um conjunto de informações sobre os documentos recuperados e avaliados. Para cada documento são armazenados dados como sua localização na rede (URL), sua realimentação e a representação vetorial do documento caso este tenha recebido realimentação positiva do usuário. A representação vetorial consiste em vetores de termos e pesos semelhantes aos descritos anteriormente para documentos.

Para fins de ilustração, a figura 7 apresenta um exemplo em texto do conteúdo de um arquivo de perfil para um agente referente ao assunto "Equipamentos Fotográficos".

```
Proteqpf.0
Eqpfot.bin
http://www.mark-e-marca.pt/agf/agf_mhk.htm 1 P 0.0
http://saudetotal.com/saude/derma/fotogr02.htm 1 F 0.0
Type 2.8323 content 1.7830 description 2.8323 content 0.0 Continuação 2.8323
Fotografia 1.0066 Dermatológica: 2.8323 aspectos 1.4747 discute 4.1398 técnica
```

```
2.8323 dermatológica 3.1847 keywords 2.47058 content 2.8323 Profissional
3.4348 Artigo 3.1847 doc 2.6646 Burjato 2.8323 Jr 2.1971 update 1.27
01/09/1997 2.8323 Dr 1.54 Dacio 1.79 Junior 3.4348  dermatologia 1.783 tornou
2.0049 acessório 0.9844 obrigatório 2.1971 imagem 2.8323 suporte 3.1847
adaptado 2.8323 observação 2.8323 médica 2.8323 seguimento 2.8323 quadro
1.7800 clínico 2.8323 Equipamento 2.8323...
```

Figura 7: Exemplo do conteúdo de um arquivo de perfil para o assunto "Equipamentos Fotográficos".

A primeira linha contém o nome do arquivo. A segunda contém o nome do arquivo de busca. As duas linhas seguintes contém os nomes, *feedback*, *status* e valor de similaridade das URLs que compõem o perfil. As demais linhas contêm os termos do perfil, com seus respectivos pesos. Na realidade, os perfis não são armazenados em arquivos de texto, mas em arquivos de objetos e os dados das URLs são representados através de referência a objetos da classe *NovaUrl* (figura 4).

O perfil representa a consulta do usuário. No sistema proposto não há distinção entre as representações do perfil e dos documento, pois o perfil é formado a partir de documentos que receberam realimentação positiva. Os passos a serem seguidos para a construção do perfil são, então, idênticos aos realizados para extrair a representação vetorial dos documentos. Os documentos a serem convertidos em perfis iniciais são aqueles que receberam realimentação positiva do usuário na primeira sessão de busca do agente correspondente. A partir daí, o perfil cresce e é modificado em buscas posteriores.

Avaliação e Seleção dos Documentos

Na clássica representação vetor-espacial, documentos são recuperados encontrando-se vetores na proximidade do vetor consulta. Uma série de exemplos de medidas de similaridade comumente usadas em sistemas de recuperação podem ser vistos em (VAN RIJSBERGEN, 1979). Na aplicação proposta, foram testadas as fórmulas de Jaccard e do cosseno, tendo sido obtidos valores de similaridade bastante próximos em ambos os casos. Na versão final foi adotada a fórmula do cosseno (SALTON, BUCKLEY, 1988), conforme descrita abaixo:

$$S(Ti^{d}, Ti^{p}) = \frac{\sum_{j=1}^{t} w_{ij}^{d} w_{ij}^{p}}{\sqrt{\sum_{j=1}^{t} (w_{ij}^{d})^{2} \times \sum_{j=1}^{t} (w_{ij}^{p})^{2}}} \qquad (13)$$

onde “d” indica que o campo pertence a um documento e o “p” indica que o campo pertence ao perfil.

Após a busca inicial, os documentos recuperados nas sessões de busca posteriores são convertidos para suas representações vetoriais a fim de serem avaliados pelo agente. A pontuação dos documentos é calculada usando os produtos escalares dos vetores de pesos do perfil e do documento, conforme a equação (13). Esse cálculo é feito pela classe *CalculaSimilaridade* (figura 4), em um processo que compreende as etapas descritas a seguir.

Inicialmente, o vetor de pesos do perfil é ordenado, de forma a deixar os termos mais relevantes (os de maior peso) nas suas posições inicias e os menos relevantes nas posições finais. Os elementos no vetor de termos devem ser alocados de modo a acompanhar as mudanças do vetor de pesos, a fim de se manter a correspondência posicional termo-peso. Como os tamanhos dos vetores do perfil e do documento sendo comparado são diferentes, a comparação é feita considerando o vetor de menor tamanho. Ou seja, se o vetor de termos do perfil for menor, ele irá controlar o número de termos a serem comparados; caso contrário a comparação será feita considerando um número de termos igual ao tamanho do vetor de documentos. A ordenação do perfil garante que os termos de maior relevância serão preferencialmente usados no cálculo.

Para cada termo do vetor de termos do perfil até a posição considerada, faz-se uma pesquisa para verificar se o mesmo existe no vetor de termos do documento. Caso exista, o produto do peso do termo no perfil pelo peso do mesmo termo no documento é armazenado em um vetor auxiliar, a ser usado para o cálculo da similaridade. Caso o termo não exista no documento, será armazenado o valor zero na posição do vetor auxiliar correspondente àquele termo. Aplica-se a fórmula (14) para todos os elementos do vetor auxiliar, obtendo-se assim o valor S de similaridade ou seja, a pontuação para o documento.

Os documentos recuperados através da tarefa de busca, disparada pelo seu respectivo perfil, têm suas similaridades calculadas com relação a esse perfil. A classe controladora de cada agente é responsável por reunir os documentos gerados por todos os perfis, juntamente com seus valores de pontuação, classificá-los de acordo com essa pontuação, eliminar as repetições e passá-los para a classe de interface *Apresenta*, responsável por apresentá-los ao usuário. A fim de possibilitar a comparação entre os valores de pontuação dos documentos recuperados por diferentes perfis, esses valores devem ser normalizados de modo a ficarem restritos ao intervalo fechado [-1,1], implicando que:

$$S(Ti^{p}, Ti^{p}) = 1 \qquad (14)$$

Ou seja, o maior valor de similaridade possível é 1, e só ocorre quando as representações do perfil e do documento são idênticas.

Os valores de similaridade calculados pelo sistema são convertidos para uma escala de classificação a fim de serem apresentados ao usuário. Essa escala proporciona uma indicação melhor da relevância do documento para o usuário, do que números fracionários indicando seu grau de semelhança com o perfil. Foi adotada uma escala de 5 pontos com os adjetivos:

- TERRÍVEL: para pontuações iguais a 0.2
- REGULAR: para pontuações entre 0.2 e 0.3
- NEUTRO: para pontuações entre 0.3 e 0.5
- BOM: para pontuações entre 0.5 e 0.8
- EXCELENTE: para pontuações maiores do que 0.8

Neste trabalho adotou-se o limite de 0.2 como o valor de pontuação mínima que um documento deve ter para ser apresentado ao usuário.

Os documentos não são armazenados no arquivo de perfil, apenas seus endereços (URLs), porém podem ser recuperados e apresentados mediante a requisição do usuário.

Durante o processo de seleção dos documentos pelos diferentes perfis, deve-se evitar apresentar o mesmo documento mais de uma vez. Para isso, os agentes mantêm uma estrutura de dados (um objeto Hashtable do pacote util de Java) com todos os documentos recuperados pelos perfis, sem repetição. É a partir dessa estrutura que os documentos são selecionados para apresentação.

Realimentação

Cada perfil de um agente mantém a lista dos documentos recuperados e informações sobre seu *status*, indicando se cada um já recebeu ou não realimentação (*feedback*) do usuário e o valor dessa realimentação.

A realimentação fornecida pelo usuário tem dois efeitos: a modificação da representação do perfil e a modificação do valor de aptidão do perfil, no caso do algoritmo genético ser implementado. O perfil tem que incorporar as mudanças geradas pela realimentação do usuário antes que novos documentos possam ser avaliados. Uma vez que o vetor de termos para um documento avaliado encontra-se disponível, o perfil é modificado conforme a equação (15) descrita no módulo de aprendizado. Os termos já existentes são repesados e termos novos são incluídos.

Todas a classes do módulo de filtragem são controladas pela classe *PerfilAgente*, e através dela passam seus resultados para a classe de interface *Apresenta*. Além disso, os dados que compõem os arquivos de perfil do usuário são passados para as classes que representam o banco de dados do sistema.

MÓDULO DE APRENDIZADO

Este módulo é responsável por garantir que o conjunto de perfis reflita os interesses do usuário da melhor forma, recuperando o máximo de documentos relevantes e descartando os irrelevantes. Os métodos de aprendizagem adotados pelo sistema são realimentação por relevância e algoritmos genéticos. Ambos os métodos foram projetados de forma independente, como dois sub-módulos separados. Nas seções seguintes é feita a descrição destes sub-módulos.

Sub-módulo de Realimentação por Relevância

No método de realimentação por relevância, um vetor de consulta original (representado pelo perfil) é alterado com base na realimentação dada pelo usuário para os documentos recuperados pelo perfil.

Para representações vetor-espaciais, o método para reformulação de uma consulta em resposta à realimentação do usuário é ajuste vetorial. Uma vez que consultas e documentos são ambos vetores, o vetor de consulta é movido espacialmente para mais perto de vetores de documentos que receberam realimentação positiva e para mais longe dos vetores de documentos que receberam realimentação negativa.

Seja um perfil P, que contribuiu com um documento D para apresentação ao usuário. O usuário fornece realimentação, que é um inteiro positivo ou negativo, para o documento D. Cada termo no perfil P presente no documento D é modificado na proporção da realimentação recebida, isto é, o peso Wik de cada termo t_k no perfil é alterado proporcionalmente a uma taxa de aprendizado e à realimentação recebida. α é a taxa de aprendizado, que indica a sensibilidade do perfil à realimentação do usuário, e em geral assume valores entre 0.5 e 1 (SHETH, 1994). Assim:

$$\forall i, k: \; Wik^{p} = Wik^{p} + \alpha * f * Wik^{d} , \qquad (15)$$

onde o índice *d* refere-se a termos no vetor do documento e o índice *p* refere-se a termos no perfil. A fórmula acima corresponde a uma variação do algoritmo de ROCCHIO (1971), sugerida por SHETH (1994).

O efeito resultante é que, para aqueles termos já presentes no perfil, os pesos dos termos são modificados em proporção à realimentação. Esse é um processo conhecido como repesagem de termos, em sistemas de recuperação de informações. Os termos que já não estejam presentes no perfil devem ser adicionados a ele, em um processo conhecido como expansão de consulta.

Um problema relacionado à implementação é que o tamanho do vetor de termos pode crescer indefinidamente quando novos termos são introduzidos no perfil através de realimentação. O tamanho pode ser controlado atribuindo-se um limite máximo para o número de termos no perfil. SALTON e BUCKLEY (1990) tratam desse problema

quando abordam a questão da expansão da consulta. Segundo eles, resultados de experimentos sugerem que a expansão da consulta usando todos os termos dos documentos relevantes recuperados é apenas ligeiramente melhor do que fazendo-se uma seleção desses termos. Eles recomendam, então, que apenas um número limitado de termos seja adicionado (principalmente com a finalidade de aumentar o tempo de resposta). No presente sistema foram testados perfis com tamanhos máximos do vetor de termos variando de 20 a 300 e adotou-se o valor de 300.

A classe responsável pela implementação da modificação dos perfis em resposta à realimentação do usuário é a classe *Ajuste* (figura 4). Ela é instanciada a partir da classe de interface *Apresenta*, que passa os dados necessários para o ajuste. A partir desses dados, a classe *Ajuste* comunica-se com o banco de dados do sistema a fim de ler e atualizar os arquivos de perfil.

Sub-módulo de Algoritmo Genético

No sistema proposto, um agente é modelado como uma população de perfis, onde cada perfil busca documentos semelhantes a ele, os quais serão de interesse potencial para o usuário. Vários trabalhos na área de Inteligência Computacional demonstraram que, se uma população aprende durante seu ciclo de vida, a evolução em direção a indivíduos mais aptos ocorre muito mais rápido. Essa característica é conhecida como o efeito "Baldwin" (ACKLEY, LITTMAN, 1992, HINTON, NOWLAN, 1987). Baseado nisso, os perfis dos agentes são ajustados de acordo com a realimentação do usuário, a fim de que eles possam "aprender" durante seu tempo de vida, diminuindo o tempo necessário para se chegar a uma solução ótima (tempo de convergência).

A definição formal de uma população P em uma iteração t é dada na equação abaixo. Ela é definida como um conjunto, onde cada elemento é um par de perfil e sua aptidão:

$$P(t) = \{(p, f(p))\} \qquad (16),$$

onde p representa um perfil e f (p) sua aptidão. Cada perfil é convertido para uma representação binária, e corresponde a um indivíduo ou cromossomo da população. Os operadores genéticos de *crossover* e mutação atualizam a população a cada geração,

introduzindo novos membros e aproveitando aqueles com maior valor de aptidão. O objetivo final é evoluir a população em direção a uma otimização global.

O módulo do algoritmo genético compreende as seguintes etapas:

- representação e inicialização da população;
- avaliação da aptidão média da população;
- reprodução (seleção);
- recombinação (*crossover* e mutação) e geração da nova população.

2.1.1.5 Inicialização da População

Para inicializar uma população, é necessário primeiro decidir o número de genes para cada indivíduo (ou cromossomo) e o número total de cromossomos na população inicial, ou seja, o tamanho da população. Quando se adota AG em recuperação de informações (IR), cada gene (bit) no cromossomo (cadeia de bits) representa uma certa palavra-chave ou termo. O *loci* (localização de certo gene) decide a existência (1) ou não existência (0) de um termo. Um cromossomo, portanto, representa um documento ou perfil que consistem de múltiplos termos. A população inicial contém um conjunto de documentos que foram julgados relevantes por um usuário através de realimentação. O objetivo do AG é encontrar um conjunto ótimo de termos que melhor correspondam às necessidades do usuário.

O tamanho da população necessário para o sistema Fenix é menor do que em aplicações de AG típicas por várias razões. Primeiro, cada indivíduo na população é capaz de aprender a partir de realimentação durante seu tempo de vida entre duas gerações. A falta de diversidade devido a uma população menor é compensada pelo mecanismo de aprendizagem, que tem efeitos também sobre o tempo de convergência do AG. Considerando o domínio da aplicação, ou seja, sistemas de filtragem de informações, há também muito menos mudanças ao longo das gerações do que em aplicações típicas de AG (SHETH, 1994). Uma proporção significativa dos interesses do usuário provavelmente permanecerá estável, ou, no máximo, mudará gradualmente.

O principal objetivo do AG para o sistema proposto é introduzir diversidade na população de perfis, através das operações de recombinação genética. Observa-se que,

após um certo número de sessões de busca e aplicação de realimentação por relevância, os perfis vão se tornando extremamente especializados, a ponto de não obterem mais bons resultados na busca. Ou seja, os termos componentes do perfil tornam-se tão específicos que dificilmente serão obtidos documentos com altos graus de similaridade, principalmente considerando-se que o espaço de busca (as páginas pesquisadas pelos mecanismos de busca) não muda com muita frequência. Assim, novos termos não podem mais ser introduzidos no perfil, e ocorre uma estagnação. Nesse caso, algumas rodadas do AG poderiam recombinar os termos de diferentes perfis e revitalizar a população, permitindo a saída dessa estagnação.

2.1.1.6 Representação dos perfis em cromossomos

Os passos para converter um perfil de um agente em sua representação como cromossomo são:

- toma-se o vetor de termos de cada um dos arquivos de perfil do agente em questão e constrói-se um vetor de termos único para a população;
- para cada arquivo de perfil, constrói-se um cromossomo (vetor binário) baseado na existência (1) ou não (0) de cada termo do vetor da população no vetor de termos do perfil. O número de genes (bits) do cromossomo é, então, o número total de termos, sem repetição, de todos os vetores de termos dos perfis da população.

2.1.1.7 Avaliação da Função de Aptidão

A função de aptidão adotada é baseada na própria medida de similaridade entre o perfil (cromossomo) e os documentos recuperados em uma busca. A aptidão do perfil é a média aritmética dos valores de similaridade de todos os documentos recuperados em uma busca realizada por aquele perfil. Após calcular a aptidão dos perfis, calcula-se a aptidão média para a população, que é a média das aptidões de todos os perfis. O objetivo final a ser alcançado pelo AG é aumentar a aptidão média da população. Assim, a cada iteração do AG calcula-se a aptidão média da população a fim de decidir se novas iterações serão realizadas ou não. Caso os valores de aptidão diminuam ou estabilizem, considera-se que o AG convergiu.

2.1.1.8 Reprodução e Recombinação

Devido ao pequeno tamanho da população, todos os cromossomos são usados para a reprodução, não sendo necessária nenhuma função de seleção dos indivíduos. O usuário deve estar ciente de que, para aplicar o módulo de AG, o agente deverá ter um número mínimo de perfis, e esse número deverá ser par, a fim de permitir as operações de recombinação. Assumiu-se como 10 esse número mínimo.

Inicialmente, os cromossomos da população são ordenados por sua aptidão média, a fim de se fazer a recombinação entre pares de indivíduos com aptidões mais semelhantes. Cada par é, então, submetido aos operadores genéticos de *crossover* e, eventualmente, de mutação.

Sugere-se a adoção de um *crossover* de dois pontos. Esses pontos indicam localizações dos genes nos cromossomos (seu loci) e correspondem aos índices do vetor de bits que representa o cromossomo. Os pontos são gerados aleatoriamente com base no número de genes do cromossomo. Todos os genes (bits) localizados entre os dois pontos são trocados entre os dois cromossomos pais para gerar dois novos descendentes.

O operador de mutação é definido a fim de contribuir para o caráter exploratório da população de perfis, buscando adicionar novos termos não presentes no perfil original. Esse operador é unário e age na base de bit-a-bit. Cada bit em todos os cromossomos da população tem a mesma chance de sofrer mutação, ou seja, mudar de 0 para 1 e vice-versa.

Sendo **pm** o valor de probabilidade de mutação e **m** o número total de genes de um cromossomo, o número esperado de bits que sofrerão mutação é dado pelo produto de **pm** multiplicado por **m** e pelo tamanho da população. Valores típicos adotados para **pm** variam entre 0.01 e 0.03 (GORDON, 1988). Para cada cromossomo que sofreu *crossover* e para cada bit dentro do cromossomo é gerado um número aleatório r, a partir do intervalo de (0 ... 1); se r > pm, o bit é selecionado para sofrer mutação.

Uma vez tendo sido produzidos os novos cromossomos (perfis), deve-se realizar uma busca para cada perfil, a fim de se avaliar a aptidão média da nova população. Para isso, os cromossomos são reconvertidos em suas representações como vetores de

termos, tarefas de busca são iniciadas e os valores de similaridades dos documentos resultantes são calculados com base nos novos vetores de termos. Caso a aptidão média do perfil seja maior do que a anterior (antes da reprodução), os vetores de termos dos perfis são modificados definitivamente. Caso contrário, mantêm-se os vetores originais e as alterações produzidas pelos operadores genéticos são descartadas. Essa operação, em termos de AG, corresponde a aproveitar os indivíduos mais aptos e eliminar os menos aptos.

Após a reprodução, *crossover*, e mutação, a nova população está pronta para sua próxima geração. O restante das evoluções são simplesmente repetições cíclicas dos passos acima até o sistema alcançar um número pré-determinado de gerações (fixado pelo usuário) ou convergir (isto é, não apresentar melhora na aptidão média da população).

Embora não tenham sido implementadas, as classes do sub-módulo de algoritmo genético foram especificadas e seus principais campos e métodos foram definidos. A principal classe do sub-módulo é a classe *População* (figura 4), que representa a população de perfis de um agente. Essa classe instancia indivíduos como objetos da classe *Cromossomo*, e possui métodos para realizar as operações genéticas e gerar descendentes a partir dos indivíduos pais. A classe *Cromossomo* representa um indivíduo da população, ou seja, um vetor binário indicando a presença ou ausência de termos do vetor de termos da população no vetor de termos do perfil e a respectiva aptidão desse indivíduo na população.

BANCO DE DADOS

O banco de dados do sistema Fenix é composto por todas as informações dos usuário, dos seus agentes e dos respectivos perfis, bem como pelos documentos recuperados nas buscas.

Esse banco é implementado através de um conjunto de classes que incorporam a persistência do sistema. Em orientação a objetos entende-se por persistência como a propriedade dos objetos de preservar os valores dos seus atributos além do tempo de execução de um programa. Dessa forma, cria-se a necessidade de manter estes atributos em algum tipo de repositório para uso posterior. Uma das formas possíveis para

armazenar dados persistentes é utilizar os gerenciadores de banco de dados. Outras formas são os sistemas de arquivos ou até mesmo dispositivos especiais de *hardware*.

Uma vez identificadas as informações persistentes do sistema, as quais devem ser mantidas em um banco de dados, é conveniente isolá-las em objetos persistentes. Deste modo, forma-se uma camada de objetos persistentes no modelo. Essa camada compõe a categoria Modelo da infra-estrutura MVC (“modelo-vista-controle”) e representa uma abstração do mundo externo, ou seja, as informações do domínio da aplicação.

No presente trabalho, diferentes estruturas são usados para armazenar os objetos persistentes, sendo aproveitadas as classes de manipulação de arquivos já existentes na biblioteca padrão de Java. A biblioteca Java disponibiliza arquivos de vários tipos, permitindo desde a criação de estruturas de armazenamento flexíveis e voltadas para o domínio da aplicação, como os arquivos de objetos, até arquivos de texto puro ou de acesso aleatório. Todos os tipos de arquivos são criados instanciando-se classes da biblioteca padrão e são manipulados através dos métodos pré-definidos para essas classes.

A classe *Usu* (figura 4) é responsável pelos dados dos usuários e é armazenada em um arquivo de acesso aleatório chamado Usuários.dat. A classe de Java que manipula a entrada e saída para arquivos desse tipo é RandomAccessFile. Os dados dos agentes são gerenciados pelas classes *arqagente* e *Agent* (figura 4), e são também armazenados em um arquivo de acesso aleatório, chamado “Agentes.dat”. Cada agente possui diversos arquivos de perfis. Os perfis são instâncias da classe *perfil*(figura 4), a qual por sua vez, é composta de objetos da classe Vector de Java para representar os vetores de termos e pesos e objetos da classe *NovaUrl*, a qual contém todos os atributos de cada documento do perfil. Esses atributos são o nome da URL, seu *status*, realimentação (*feedback)* e a pontuação (*score*) atribuída. A classe *perfil* é derivada da classe Object de Java e implementa a interface Serializable. Essa interface permite a criação de arquivos de objetos. Os arquivos contendo os perfis são manipulados através das classes ObjectInputStream e ObjectOutputStream. Essas classes fazem a codificação e decodificação de classes, armazenando em arquivos de objeto o nome da classe, sua assinatura e os valores de seus campos, bem como referências a outros objetos

relacionados. Elas possuem métodos para a leitura (readObject( )) e para a escrita (writeObject) de objetos em arquivos.

A classe *Banco* representa as páginas recuperadas, que são salvas localmente a fim de serem manipuladas sem necessidade de conexão à rede. Ela é declarada como uma classe derivada de Object, e também implementa a interface Serializable, a fim de possibilitar a gravação de arquivos de objetos. Foi criada uma classe chamada cria_*banco* para fazer a interface entre a classe controladora *PerfilAgente* e as operações de acesso aos dados da classe *Banco*. Esta classe, através dos métodos das classes ObjectInputStream e ObjectOutputStream de Java, lê e armazena os dados das páginas.

MÓDULO CONTROLADOR

As classes pertencentes ao módulo controlador são da categoria Controle do modelo MVC. Elas são responsáveis por criar as instâncias das demais classes e invocar seus métodos, controlando a comunicação entre os diferentes objetos componentes do sistema. São elas:

- ***Principal*** - controla o sistema como um todo, exibindo a tela principal, a partir da qual as demais classes são instanciadas de acordo com as operações solicitadas pelo usuário;
- ***Agente*** - controla o comportamento de todos os agentes de um usuário, permitindo a criação de novos agentes ou o carregamento de agentes já existentes;
- ***PerfilAgente*** - controla o comportamento de um agente específico de um usuário, instanciando tarefas de busca, recebendo os resultados e controlando os documentos a serem passados para a classe de apresentação desses resultados ao usuário;
- ***threadauto*** - controla o modo autônomo, sendo responsável pela execução de novas buscas sem a interação com o usuário. Verifica se existem agentes para o usuário específico com documentos ainda não lidos e informa ao usuário, ou inicia novas tarefas de busca para agentes cujos documentos já foram todos avaliados pelo usuário;
- ***threadBusca*** - controla o funcionamento de uma tarefa de busca, obtendo os resultados gerados por um mecanismo de busca e convertendo-os para uma forma que possa ser manipulada pelas demais classes.

As três primeiras classes, apesar de terem papéis de controle, são derivadas da classe Panel de Java, possuindo, então uma camada de apresentação. As demais classes de interface são adicionadas a esses painéis principais. As classes *threadauto* e *threadBusca* são derivadas da classe Thread de Java e representam tarefas do sistema operacional.

CONSIDERAÇÕES FINAIS

O sistema Fenix foi desenvolvido segundo a abordagem de Orientação a Objetos e o projeto das classes que o compõem baseou-se na infra-estrutura MVC. Sua arquitetura consiste em um conjunto de classes agrupadas em módulos funcionais. Cada módulo é responsável por uma função essencial do sistema, e comunica-se com os demais a fim de atingir o objetivo para o qual o sistema foi desenvolvido. O próximo capítulo aborda o ambiente de desenvolvimento e a implementação do sistema, descrevendo as etapas da implementação, relatando problemas encontrados no decorrer do trabalho e analisando as possíveis soluções.

5 AMBIENTE DE DESENVOLVIMENTO E IMPLEMENTAÇÃO DO SISTEMA

O sistema Fenix foi desenvolvido segundo o paradigma de Orientação a Objetos desde a fase da análise até a sua implementação com uma linguagem de programação orientada a objetos. Existem diversas linguagens orientadas a objetos disponíveis atualmente, cada uma com vários ambientes de desenvolvimento. A linguagem utilizada para a implementação do sistema Fenix foi Java, da Sun Microsystems Inc. (ANUFF, 1996) e o ambiente de desenvolvimento foi o Jbuilder Standard 3.0, da Borland Corporation, que utiliza o JDK ("*Java Development Kit"*) versão 1.1.

Este capítulo tem por objetivo apresentar o ambiente de desenvolvimento do sistema e descrever sua implementação. A seção 5.1 explica e justifica a escolha da linguagem e do ambiente usados e descreve algumas soluções e considerações adotadas na construção do código. Os detalhes de implementação são abordados na seção 5.2.

AMBIENTE DE DESENVOLVIMENTO

A escolha da linguagem Java como plataforma de desenvolvimento de agentes teve como motivação algumas particularidades dessa linguagem: é simples, robusta, segura, multiplataforma, totalmente orientada a objetos, e já traz pacotes prontos de métodos e classes para trabalho com os protocolos utilizados pela Internet, como, por exemplo, http e ftp.

Há várias ferramenta de desenvolvimento em Java, as quais ajudam a implementar programas de forma mais rápida e eficiente. Dentre as várias ferramentas desse tipo disponíveis no mercado, produzidas por diferentes fabricantes, pode-se citar: Java Studio e Java Workshop[2], ambos da Sun Microsystems; JavaPlan[3], da Lighthouse Design Limited, o Visual J++[4] da Microsoft System, o Visual Café for Java[5], da

[2] Disponíveis em: http://www.sun.com

[3] Disponível em: http://www.lighthouse.com/Product.html

[4] Disponível em: http://www.microsoft.com/visualj

[5] Disponível em: http://café.symantec.com

Symantec, VisualAge for Java[6], da IBM, e a família Jbuilder[7], da Borland Corporation. A família Jbuilder disponibiliza três diferentes ambientes de desenvolvimento, todos visuais, orientados a componentes, de alto grau de escalabilidade e compatibilidade com outras ferramentas. Sua versão Standard oferece a funcionalidade necessária para a criação de todos os tipos de programa em Java, para sistemas de pequeno e médio porte. A versão Profissional é voltada para aplicações de maior porte, geralmente para empresas, e a versão Cliente Servidor incorpora funções de conectividade com bancos de dados SQL, suportando JDBC e ODBC, além de fornecer suporte a objetos distribuídos através de conectividade com CORBA. Jbuilder roda em Windows NT, Windows 98 e Windows 95.

Adotou-se para o desenvolvimento do presente trabalho o Jbuilder Standard 3.0, que utiliza o JDK (*"Java Development Kit"*) versão 1.1. A escolha foi motivada pelo fato dessa ferramenta ser uma das únicas que usa a linguagem Java padrão, incorporando o kit de desenvolvimento fornecido pela Sun, e pelo seu conhecimento prévio por parte da autora deste trabalho.

Várias classes da biblioteca padrão Java foram utilizadas para a implementação do sistema, como por exemplo, classes gráficas para a interface com o usuário, classes que representam arquivos de dados para os objetos persistentes, classes que representam tarefas do sistema operacional, e classes específicas para aplicações voltadas para ambientes da Internet. A utilização de APIs Java já existentes otimizou bastante o trabalho de implementação.

A linguagem Java permite a implementação de dois tipos de programas: *applets* e aplicativos. Os aplicativos rodam em qualquer plataforma que tenha a máquina virtual Java ("*Java Virtual Machine*" ou JVM), também conhecida como interpretador Java. A JVM é um computador "abstrato" que roda aplicativos Java. É um computador abstrato no sentido de que é baseado em *software* e roda sobre várias plataformas de *hardware*. Essa abordagem de interpretador desempenha uma papel importante na portabilidade da

[6] Disponível em: http://www.software.ibm.com/ad/vajava/

[7] Disponível em: http://www.borland.com/jbuilder

linguagem. Atualmente Java roda em Windows95, WindowsNT, OS/2, Solaris, Linux e algumas versões do Unix.

Já as *applets* são programas especiais criados para serem executados a partir de páginas HTML. Um usuário, ao carregar em sua máquina uma página HTML que contém uma *applet*, provoca o envio de uma cópia do código executável da *applet* para esta máquina, onde ela irá rodar localmente.

Os navegadores atuais impõem uma série de restrições de segurança à execução de *applets*. A menos que ela seja gravada na máquina local, os navegadores só permitem que uma *applet* leia e grave dados, ou conecte-se a soquetes, no computador (*host*) de origem. Para entender o raciocínio, é importante visualizar os três computadores envolvidos:

- computador de origem: o computador que entrega aos usuários a página HTML contendo a *applet*;
- computador local: o computador do usuário, que roda a *applet*;
- computador contendo depósitos de dados de terceiros, que a *applet* gostaria de acessar.

A regra de segurança dos navegadores diz que uma *applet* só pode ler e gravar dados no computador de origem. Já aplicativos Java não sofrem esse tipo de restrição, podendo ler ou gravar arquivos locais.

Um dos requerimentos do sistema Fenix é manter arquivos com as informações sobre os agentes e os usuários. Caso se optasse pelo desenvolvimento de uma *applet*, seria preciso implementar uma solução para a criação e manipulação desses arquivos. Existem pelo menos três alternativas:

- criar os arquivos no servidor onde reside a *applet* e fazer a comunicação com a máquina do usuário via CGI (*Common Gateway Interface*). Nessa alternativa, a *applet* invoca um *script* CGI que transmite as informações ao gerenciador de banco de dados localizado no servidor. O *script* CGI poderia ser um aplicativo Java rodando na máquina do servidor;
- criar os arquivos no servidor e fazer a *applet* conectar-se a um programa em JavaScript, que abre uma conexão com um servidor de banco de dados. A interação

com o banco pode ser feita através de APIs nativas de Java, ou usando JDBC (*Java Database Connectivity*). JDBC é uma interface desenvolvida pela Sun que permite a comunicação segura e conveniente entre bancos de dados e *applets;*

- criar os arquivos no servidor e fazer a comunicação com a máquina do usuário via *servlets*. *Servlets* são uma extensão padrão para a plataforma Java usadas para criar aplicações que requeiram conectividade com bancos de dados, autenticação de usuários, geração dinâmica de HTMLs, entre outras funções. *Servlets* podem ser vistas como *applets* que rodam no lado servidor, e oferecem vantagens, como portabilidade, flexibilidade e eficiência, quando comparadas com as tradicionais CGIs. *Servlets* podem ser implementadas com o *Java Servlet Development Kit* (JSDK), da Sun Microsystems.

Qualquer dessas soluções gera tráfego na rede, já que os dados dos agentes são constantemente consultados em todas as operações do sistema. Além disso, com um número elevado de usuários, cada um possuindo vários agentes, o espaço ocupado no disco do servidor cresceria rapidamente.

Quando se adota um sistema de filtragem colaborativa, no qual os perfis dos diferentes usuários devem ser compartilhados, é imprescindível o acesso aos arquivos dos diferentes usuários, sendo, portanto, a sua gravação no servidor a solução mais indicada. Entretanto, com o método de filtragem adotado no presente trabalho, os agentes de diferentes usuários não cooperam entre si e os arquivos de cada usuário só são acessados pelos seus próprios agentes, não sendo necessário o compartilhamento dos dados. Nesse caso, a solução mais indicada é a gravação dos arquivos na máquina do próprio usuário.

Tendo em vista o que foi exposto, e acrescentando o fato de que o uso de qualquer alternativa para a implementação de *applets* demandaria conhecimento adicional de programação de CGIs ou da utilização das APIs para JDBC e *Servlets*, optou-se nesta versão pelo desenvolvimento de um aplicativo Java, com os dados sendo gravados na máquina do usuário, em estruturas de arquivos criadas a partir de classes da própria linguagem.

IMPLEMENTAÇÃO

O objetivo da fase de implementação é a construção de um protótipo que incorpore toda a funcionalidade necessária para analisar a viabilidade do sistema proposto. O protótipo foi usado em uma série de sessões simuladas de interação com o usuário a fim de avaliar o funcionamento e a eficiência do sistema.

Java é uma linguagem totalmente orientada a objetos. O desenvolvimento do protótipo consistiu na implementação de todas as classes modeladas durante as fases de análise e projeto, com seus respectivos atributos e métodos. A documentação das classes implementadas, com a descrição de seus objetivos e principais métodos, pode ser vista no anexo 3.

A implementação do protótipo foi feita de maneira incremental, ao longo de cinco etapas. Em cada uma delas, novos módulos eram adicionados, e novas funções acrescentadas aos módulos já existentes. O protótipo intermediário gerado a cada etapa era testado e os erros corrigidos.

Esta seção apresenta inicialmente as atividades realizadas a cada etapa da implementação. A seguir são analisadas algumas características do protótipo implementado. Na sub-seção 5.2.3 são descritos alguns problemas e dificuldades encontrados e são investigadas suas possíveis soluções. A sub-seção 5.2.4 trata de assuntos de eficiência do sistema.

Atividades Realizadas

A implementação do protótipo do sistema Fenix foi realizada ao longo de cinco etapas. A seguir são listadas as principais atividades desenvolvidas em cada uma dessas etapas.

Etapa 1

- Implementação das classes componentes do módulo de interface, omitindo detalhes como o tratamento das entradas do usuário, a emissão de mensagens de confirmação e erro, e a criação de painéis com formatos muito elaborados.

- Implementação das classes componentes do módulo de busca. Nessa etapa, apenas os métodos responsáveis pela busca inicial foram implementados. Com a finalidade de simplificação estabeleceu-se a integração do módulo com um único mecanismo de busca e a execução de uma única tarefa de busca para cada agente.
- Implementação das classes componentes do banco de dados local e dos métodos responsáveis pela criação do arquivo inicial de perfil a partir dos documentos recuperados na busca que receberam realimentação positiva do usuário.
- Implementação dos métodos do módulo controlador que gerenciam a comunicação entre as classes dos diferentes módulos do sistema.

Etapa 2

- Implementação das classes componentes do sub-módulo de realimentação por relevância.
- Inclusão, no módulo de busca, dos métodos responsáveis pela realização de tarefas de busca para um perfil já existente. Esses métodos realizam novas buscas e submetem os documentos obtidos aos métodos do módulo de filtragem. Esses, por sua vez, realizam o processo de filtragem e passam o resultado para o módulo de interface que faz a apresentação para o usuário. A coordenação entre os módulos é realizada pelo módulo controlador. Nesta etapa ainda se adota a integração com um único mecanismo de busca ao longo de uma única tarefa de busca.
- Realização de testes com diferentes fórmulas para o cálculo de similaridade entre perfis e documentos.

Etapa 3

- Implementação da classe responsável pela modificação dos perfis a partir dos novos documentos avaliados pelo usuário.
- Realização de testes de diferentes fórmulas para o cálculo de peso de termos e para o ajuste vetorial.

Etapa 4

- Implementação de várias tarefas de busca para cada agente, cada uma ativando diferentes mecanismos de busca na *Web*. Teste de diferentes números de tarefas para cada agente, avaliando o impacto, no tempo gasto em uma sessão de busca, da execução simultânea de muitas tarefas.
- Implementação do controle da integração dos resultados das várias tarefas de busca (sub-agentes), feito pelo módulo controlador do sistema.
- Implementação de estruturas de dados para armazenar os resultados das várias buscas evitando a ocorrência de redundância das páginas recuperadas.

Etapa 5

- Refinamento do módulo de interface com o usuário, incluindo a conversão dos valores de pontuação dos documentos para uma escala de classificação e a elaboração de painéis com formatos de apresentação mais incrementados.
- Implementação do modo de operação autônoma do sistema.

Ao longo de cada etapa, os componentes do sistema foram sendo testados e os erros de sintaxe e lógica foram sendo detectados e corrigidos. Após o término das cinco etapas, iniciou-se a fase de avaliação do sistema. Nessa fase adotou-se uma medida de avaliação e foi analisada a viabilidade da proposta e a eficiência alcançada pelo sistema.

Características do Protótipo

Antes de discutir os problemas encontrados durante a fase de implementação, é importante resumir as limitações assumidas na construção do protótipo. O tratamento de algumas dessas limitações em trabalhos futuros poderá tornar o sistema mais robusto e flexível. Outras características assumidas decorrem da própria abordagem de filtragem adotada para o sistema, a qual baseia-se no conteúdo puramente textual dos documentos. No protótipo implementado:

- a filtragem baseia-se apenas nas palavras contidas nas páginas. Elementos como o tamanho, a estrutura ou o número de *links* das páginas foram ignorados;
- são analisadas apenas páginas textuais, o que elimina muitos recursos disponíveis na Web, já que atualmente há uma quantidade cada vez maior de imagens nas páginas;

- foram ignorados possíveis problemas associados com o caráter transitório das páginas *Web*. Algumas páginas são relativamente estáticas, outras têm uma vida limitada e outras ainda são geradas dinamicamente de acordo com a requisição do usuário. No protótipo corrente, uma página nunca é apresentada duas vezes para um usuário, e é mantida no banco local apenas sua versão mais atual, o que é uma solução apenas parcial para o problema de atualização das páginas fornecidas;
- não há boa escalabilidade, já que não se usa nenhum tipo de estrutura índice para o armazenamento de dados dos agentes. Caso o número de usuários e perfis do sistema cresça, é necessária a adoção de um sistema de indexação dos perfis e a implementação de algoritmos para agilizar o processamento desses perfis a fim de tornar o sistema viável em termos de velocidade de resposta. Em sua implementação atual, o sistema funciona bem com uma quantidade pequena de usuários (no máximo vinte), cada um mantendo um conjunto de no máximo 10 agentes com no máximo 10 perfis cada um. Cabe ressaltar que cada perfil inicia uma tarefa do sistema operacional, então muitos perfis podem gerar uma sobrecarga no sistema.

Problemas de Implementação

Alguns problemas foram encontrados ao longo da construção do protótipo do sistema, e sua descrição pode ser de grande importância para futuros trabalhos na área.

- Uso de Molduras (*Frames)* nas páginas consultadas

Uma tendência crescente no projeto de páginas HTML atuais é o uso de molduras (*frames),* as quais permitem a visualização e navegação pelo conteúdo das páginas de forma bastante amigável para o usuário. Nesse tipo de página o documento recuperado pelos agentes contém apenas as "*tags"* HTML de definição das molduras, e não o conteúdo de texto significativo das páginas. Ou seja, aquilo que o usuário vê no navegador e identifica como relevante não é realmente o texto que será salvo para compor os vetores de termos usados no modelo de representação adotado. O texto salvo corresponde na verdade apenas às "*tags*" que compõem a página de definição de molduras. Uma possível solução para esse problema seria o usuário olhar o código fonte do documento HTML, uma opção que o próprio navegador fornece, obter o nome da página que está sendo exibida na moldura visualizada, e que contém realmente o conteúdo de texto relevante e depois acrescentar manualmente o nome da página

desejada no campo URL do agente. Essa operação certamente só pode ser esperada por parte de usuários com algum conhecimento da estrutura de páginas HTML. Para usuários leigos, não aptos a realizar esse tipo de operação, o efeito resultante de selecionar páginas com molduras é que os dados que efetivamente irão fazer parte do perfil não refletem o texto que o usuário viu e julgou relevante.

Uma segunda opção seria programar o agente para buscar automaticamente todas as páginas (*links*) referenciadas pelo parâmetro "SRC" na página de definição de frames, e compor o vetor de termos a partir do conjunto dessas páginas. Entretanto, essa busca por links pode não parar apenas em um primeiro nível de profundidade, pois pode haver uma página contendo somente um índice dos assuntos e só então se chegar ao conteúdo de texto efetivamente relevante. Como não é possível estabelecer uma condição de parada significativa para a obtenção das páginas referenciadas pelos *links*, optou-se por não se implementar esse tipo de tratamento.

Em páginas que usam molduras o papel das "*meta-tags*" é de importância fundamental, sendo os atributos dessas "*tags*" em geral o único conteúdo de texto relevante para o propósito de indexação da página. No protótipo implementado adotou-se um esquema de utilização de "*meta-tags*" para a construção do vetor de termos de documentos.

- Uso de Tabelas nas páginas consultadas

Outro recurso comumente usado em páginas HTML que também dificulta a construção de vetores de termos significativos a partir do conteúdo textual das páginas é o uso de tabelas. Quando o texto relevante da página está inserido em tabelas, ele encontra-se delimitado por "*tags*" de definição de tabelas e é completamente removido na rotina de remoção de código HTML. A solução para esse problema seria incluir todas as *tags* relativas a criação de tabelas em uma lista e, ao encontrar qualquer *tag* HTML na página, compará-la com a lista. Caso seja identificada uma "*tag*" referente à inclusão e formatação de tabelas, o conteúdo do texto delimitado por ela deveria ser analisado a fim de remover apenas o código HTML, preservando o texto em si. Essa solução não foi implementada no protótipo, sendo sugerida como um aperfeiçoamento a ser feito em uma versão mais completa do sistema.

- Páginas com muitas figuras e pouco conteúdo textual

Uma tendência atual no projeto de páginas HTML é utilização de recursos não textuais para enriquecer a apresentação e atrair o interesse do usuário, além de facilitar a absorção das idéias que se desejam transmitir. Páginas com muito conteúdo gráfico não são representativas para a utilização de sistemas de filtragem baseados em conteúdo. O uso das "*meta-tags*" pode ser uma forma de tornar páginas desse tipo mais adequadas para a filtragem.

- Redirecionamento de páginas

Um problema que apareceu frequentemente nos testes iniciais do sistema Fenix foi o redirecionamento dinâmico das páginas feito pelos navegadores de modo transparente para os usuários. Muitas vezes, a página que o usuário vê no navegador e que é efetivamente salva pelo agente caso seja considerada relevante, não contêm na verdade o conteúdo visualizado, mas sim, uma rotina de direcionamento, como por exemplo um código em javascript. Para tratar desse problema, seria necessário que o agente analisasse o código HTML e detectasse essa rotina, obtendo nela a URL que seria o destino final, e só então armazenando o seu conteúdo. Como não há um padrão que possa ser identificado como um código de direcionamento no arquivo fonte HTML, esse tipo de análise torna-se extremamente difícil, e não foi implementada no sistema proposto.

Problemas de Eficiência

Como ocorre com o desenvolvimento de qualquer *software*, ao longo da implementação do protótipo do sistema Fenix várias soluções de compromisso tiveram que ser adotadas com relação ao desempenho do sistema. Embora não seja objetivo deste trabalho fazer qualquer tipo de avaliação de desempenho em termos computacionais, uma descrição de alguns problemas encontrados e das abordagens de soluções adotadas podem ser úteis para o desenvolvimento de sistemas similares no futuro.

A mais importante solução de compromisso no estágio de implementação foi o clássico compromisso espaço-tempo. Recalcular informação consome tempo mas

economiza em espaço de armazenamento. Armazenar informação economiza tempo de computação, mas requer memória e espaço em disco. O componente do sistema Fenix que mais consome tempo é sem dúvida o cálculo dos pesos dos termos para a indexação dos documentos. A princípio, os vetores de termos e pesos de cada documento não são armazenados, mas sim, gerados no momento da filtragem. . Diferentes perfis podem obter as mesmas páginas como resultado de suas buscas. Se cada perfil calcula os pesos dos termos de forma independente, é ocasionada uma redundância de processamento, que consome tempo e recursos computacionais. Em vez disso, se todos os termos índices e os respectivos pesos calculados fossem armazenados, a indexação e a comparação entre perfis e documentos gastariam muito menos tempo e recursos. Além disso, quando há um número elevado de usuários, pode-se obter uma economia em larga escala pré-processando todos os documentos disponíveis no banco de dados local.

Há duas sugestões específicas a respeito de pré-processamento que poderiam trazer grandes benefícios potenciais. A primeira é calcular as frequências dos termos (tf) uma vez e armazená-las durante todo o tempo de vida da página no sistema. Uma vez que a contagem de termos pode ser calculada para documentos individuais, ela pode ser realizada apenas uma vez, quando a página for introduzida por um perfil no banco de dados local. A outra sugestão é a respeito do cálculo da frequência de documento (idf). Adotando-se um esquema de cálculo de idf baseado em um dicionário fixo de palavras obtido a partir de um conjunto de páginas coletadas aleatoriamente na *Web*, o valor de idf para cada termo também se mantém constante e pode ser armazenado. Com isso aumenta-se a velocidade de processamento global do sistema, já que os pesos dos termos para os documentos armazenados não precisam ser recalculados por diferentes perfis acessando os mesmos documentos. Entretanto, o espaço requerido para o banco de dados local do sistema seria maior.

Nessa primeira versão do protótipo, onde o número de usuários e perfis foi controlado e mantido pequeno, a diferença no espaço de armazenamento ou no tempo de processamento envolvidos nas duas opções de implementação não é significativa. Optou-se por manter apenas as referências para os documentos armazenadas, com os vetores de termos e pesos sendo gerados sempre que necessário. Entretanto, foi utilizada uma estrutura de dados do tipo tabela *hash* para armazenar os documentos que estão sendo processados por todos os perfis de um agente, evitando a repetição e o

processamento redundante quando um mesmo documento é usado por mais de um perfil.

Outra solução de compromisso adotada diz respeito ao número de perfis de cada agente. Cada perfil de um agente é responsável por disparar a execução de uma tarefa (*thread*) do sistema operacional que será usada para ativar um mecanismo de busca na *Web* e realizar a filtragem das páginas obtidas. Um maior número de perfis faz o agente recuperar maior número de páginas, aumentando a possibilidade de encontrar material relevante. Entretanto, a execução simultânea de muitas tarefas compromete o desempenho do sistema, diminuindo o tempo de resposta. Diferentes números de tarefas foram testados a fim de analisar o impacto desse número no tempo gasto pelo sistema para apresentar os resultados de uma sessão de busca. Optou-se por fixar como 10 (dez) o número máximo de tarefas permitidas. Portanto, um agente pode ter no máximo dez perfis.

CONSIDERAÇÕES FINAIS

O sistema Fenix foi desenvolvido segundo a abordagem de orientação a objetos. A linguagem de programação adotada foi Java, da Sun Microsystems Inc. e o ambiente de desenvolvimento foi o Jbuilder, da Borland Corporation. A implementação do protótipo do sistema foi feita de modo incremental através de várias etapas e alguns problemas surgiram ao longo dessas etapas, sendo analisadas diferentes soluções para os mesmos.

Este capítulo tratou da descrição do ambiente de desenvolvimento do sistema e das atividades desenvolvidas ao longo da fase de implementação do protótipo. Foram descritas as limitações do protótipo implementado e os principais problemas encontrados, analisando-se as possíveis soluções. No próximo capítulo é feita a avaliação dos resultados obtidos em várias séries de sessões simuladas de execução do protótipo.

6 ANÁLISE DOS RESULTADOS

O objetivo deste capítulo é analisar os resultados obtidos ao longo de uma série de testes realizados com o sistema Fenix. Durante a fase de implementação do sistema foi construído um protótipo que incorporava toda a funcionalidade prevista para o sistema e cujo objetivo era avaliar a viabilidade da proposta apresentada. Após o término da implementação foram realizados vários testes do protótipo a fim de verificar se ele satisfazia os requerimentos do sistema projetado e se o objetivo final do trabalho fora atingido.

A seção 6.1 apresenta os conceitos necessários ao entendimento das medidas de avaliação adotadas em sistemas de filtragem e recuperação e a seguir explica e justifica a preferência pelas medidas que foram usadas para avaliar o sistema Fenix. A seção 6.2 analisa alguns aspectos característicos da filtragem baseada em texto puro. Na seção 6.3 são descritos os testes realizados com o protótipo e os resultados obtidos.

AVALIAÇÃO DA RECUPERAÇÃO

O tipo de avaliação a ser adotado em um sistema de IR depende dos objetivos do sistema a ser avaliado. Certamente, qualquer *software* tem que fornecer a funcionalidade para a qual ele foi concebido. Então, o primeiro tipo de avaliação a ser considerado é uma análise funcional na qual as funções especificadas para o sistema são testadas uma a uma. Nesta análise deve ser incluída uma fase de análise de erro na qual um usuário comporta-se de forma errática tentando fazer o sistema falhar. Esse é um procedimento simples que pode ser muito útil para detectar erros de programação. Considerando que o sistema tenha passado a fase da análise funcional, segue-se com uma avaliação do desempenho do sistema.

As medidas mais comuns de desempenho de sistemas são tempo e espaço. Quanto mais curto é o tempo de resposta e quanto menor é o espaço de armazenamento usado, melhor é considerado o sistema. Há uma solução de compromisso inerente entre a complexidade no tempo e no espaço que frequentemente tem que ser resolvida.

Em um sistema projetado para fornecer recuperação de dados, como por exemplo, um sistema de consulta a bancos de dados, o tempo de resposta e o espaço

requerido são usualmente as métricas de maior interesse e as normalmente adotadas para avaliar o sistema. Em um sistema de recuperação de informações outras métricas, além de tempo e espaço, são em geral mais importantes na avaliação do sistema. Em sistemas de IR, a requisição de consulta do usuário é inerentemente vaga, de modo que os documentos recuperados não são respostas exatas e devem ser classificados de acordo com sua relevância para a consulta. Tal classificação da relevância introduz um componente inexistente em sistemas de recuperação de dados e que desempenha um papel central em recuperação de informações. Sistemas de recuperação de informações requerem, portanto, a avaliação do quão preciso é o conjunto de respostas fornecido ao usuário. Esse tipo de avaliação é referido como avaliação do desempenho de recuperação. As principais medidas de desempenho de recuperação usadas em sistemas de recuperação de informações usam conceitos como relevância, precisão e *recall*, os quais são examinados a seguir.

Necessidade de Informação e Relevância

O conceito de "necessidade de informação" de um usuário é a base para a representação desse usuário dentro dos sistemas de filtragem e recuperação de informações. INGWERSEN (1992) sugere uma divisão das necessidades de informação em: necessidade verificativa ou de localização (o usuário quer verificar ou localizar ítens conhecidos), necessidade tópica consciente (o usuário quer esclarecer, revisar ou explorar aspectos de assuntos conhecidos) e necessidade tópica confusa (o usuário quer explorar novos conceitos fora do escopo de seus assuntos conhecidos).

A relevância tópica de um documento é definida como sua relevância para uma necessidade tópica consciente, determinada por um juiz humano. Essa é a definição de relevância usada para avaliar sistemas nas conferências TREC (*Text Retrieval Conferences*) (HARMAN, 1994, HARMAN, 1996) e em geral corresponde ao uso comum da palavra "relevância" na comunidade de recuperação de informação.

Neste trabalho, em vez de modelar um usuário explicitamente, o sistema modela apenas os documentos de interesse do usuário, através de um perfil construído no mesmo espaço vetorial dos documentos. A relevância tópica é o único fator capturado por esse tipo de representação, e será usada para avaliar os resultados do sistema.

Dado um vetor **m** representando o perfil de um usuário e um vetor **d** representando um documento, a relevância tópica desse documento é medida tomando-se o produto interno entre **m** e **d**, após normalizar cada vetor para ter comprimento unitário:

relevância tópica de **d** para **m** = SIM (**m**,**d**) (17)

Diz-se que uma relação de preferência $\succ$ entre d e d' é fracamente linear (BALABANOVIC, 1998) se existir um perfil de usuário **m** tal que para todo d,d' pertencentes a um conjunto de documentos D:

$$d \succ d' \Rightarrow SIM(m,d) > SIM(m,d') \quad (18)$$

Os conceitos de relevância tópica e de relação de preferência foram utilizados na avaliação dos resultados do sistema Fenix, conforme descrito nas seções a seguir.

Avaliação Baseada em Relevância

Um conceito de relevância baseado em valores binários é o princípio mais usado na avaliação de sistemas de recuperação de informações e serve de base para comparações entre vários trabalhos de IR, tais como as realizadas nas conferências TREC (HARMAN, 1994, HARMAN, 1996). A principal assunção é que, dada uma consulta e uma coleção de documentos, é possível dividir a coleção em porções relevantes e irrelevantes com relação à consulta. As medidas mais populares em recuperação de informações, precisão e *recall*, dependem desse tipo de julgamento de relevância. Essas medidas são resumidas na tabela 1, que categoriza os documentos em quatro tipos após uma recuperação resultante de uma consulta. Na prática, a maioria dos sistemas retorna uma lista classificada de documentos em vez de apenas categorizá-los em um conjunto "recuperado" (apresentado ao usuário) e um conjunto "não recuperado" (descartado). Pode-se considerar isso como uma extensão vertical da tabela. Da mesma forma, a maioria dos usuários pode classificar os documentos retornados em mais de duas categorias, levando a uma extensão horizontal da tabela. A seção seguinte mostra como estender a tabela em ambas as dimensões.

	Relevantes	Irrelevantes

Recuperados	a	b
Não Recuperados	c	d

Precisão (proporção de documentos recuperados que são relevantes)= a / (a + b)

Recall (proporção de documentos relevantes que são recuperados)= a / (a + c)

Tabela 1: Medidas considerando relevância em valores binários, onde o número total de documentos na coleção |D| = a + b + c + d.
Fonte:(BALABANOVIC,1998).

A noção de relevância usada em recuperação de informações é problemática em geral e particularmente para o domínio estudado no presente trabalho. Estudos sobre o comportamento de usuários mostram que os mesmos têm dificuldade em fazer julgamentos de relevância consistentes por um período longo de tempo, quando são solicitados para avaliar documentos em uma escala de relevância absoluta (LESK, SALTON, 1971). Há, também, uma discordância considerável entre os julgamentos de diferentes usuários (SARACEVIC, 1995). A maneira usual de contornar esse problema é testar sistemas de IR com coleções padronizadas de documentos e consultas, e, dessa forma, tornar os valores de precisão e *recall* comparáveis.

Uma vez que o domínio do presente trabalho é a *Web*, não se pode adotar o uso de uma coleção padronizada. Além disso, torna-se praticamente inviável medir ou mesmo estimar o valor de *recall*, quando a coleção de documentos é a *Web* inteira.

Por outro lado, LESK (1971) mostrou que juízes humanos são bons quando fazem julgamentos relativos de uma coleção de documentos e esses julgamentos são consistentes ao longo do tempo e quando comparados entre os diferentes juízes. Dessa forma, optou-se, neste trabalho, por adotar uma medida de desempenho que requer apenas julgamentos relativos. A medida ndpm ("*normalized distance-based performance measure*"), proposta por YAO (1995), é uma distância entre a classificação dada pelo usuário a um conjunto de documentos e a classificação dada pelo sistema aos mesmos documentos, normalizada para estar entre 0 e 1. Essa medida relativa é mais apropriada do que as de precisão e *recall* para o domínio em questão. Além disso, a comparação direta entre a classificação do usuário e a do sistema fornece uma medida monovalorada, que é mais simples de interpretar do que gráficos de precisão e *recall*. Na seção seguinte são descritas as medidas de avaliação baseadas em comparação entre classificações.

Avaliação Usando Distância entre Classificações

A medida de desempenho baseada em distância normalizada (ndpm) (YAO, 1995) é definida com relação a pares de documentos classificados. A idéia é medir a distância entre a classificação prevista por um sistema de recomendação ou filtragem e a classificação real dada pelo usuário.

Se $\succ_p$ é a classificação prevista pelo sistema e $\succ_d$ é a classificação desejada pelo usuário, então a distância entre dois documentos d.d' $\in$ D com relação a essas classificações é definida como (BALABANOVIC, 1998):

$$\delta_{\succ_p,\succ_d}(d.d') = \begin{cases} 2 \text{ se } (d \succ_p d' \text{ e } d' \succ_d d) \text{ ou } (d \succ_d d' \text{ e } d' \succ_p d) \\ 1 \text{ se } (d \succ_d d' \text{ ou } d' \succ_d d) \text{ e } d \sim_p d' \\ 0 \text{ caso contrário} \end{cases} \quad (19)$$

Em outras palavras, se o sistema de filtragem prediz que um documento d é melhor que d' e o usuário diz que d' é melhor do que d, então a distância entre as duas classificações em relação a d e d' é 2. Se o usuário diz que d' é melhor do que d ou d é melhor que d', mas o sistema coloca d e d' na mesma categoria de classificação, então a distância entre as duas classificações em relação a d e d' é 1. Caso contrário, a distância é 0 (zero).

A medida ndpm completa para uma coleção D pode ser definida adicionando-se δ de todos os pares não ordenados de documentos em D, e normalizando o resultado dividindo-se por duas vezes o número de pares na classificação desejada $\succ_d$:

$$\text{ndpm}(\succ_p,\succ_d) = \frac{\sum_{d,d' \in D} \delta_{\succ_p,\succ_d}(d.d')}{2|\succ_d|} \quad (20)$$

Essa definição garante que o valor de ndpm situa-se no intervalo entre [0,1], com um valor de 0.5 representando a "linha de base" do desempenho.

Uma vez que essa é uma distância métrica entre as classificações do sistema e as do usuário para um conjunto de documentos, ela lida com ambas as extensões vertical e horizontal da tabela 1. O sistema Fab foi um dos primeiros a usar essa medida na prática (BALABANOVIC, SHOHAM, 1997). Outros sistemas que também a adotaram foram os descritos em ASNICARD e TASSO (1997) e em (BALABANOVIC, 1998). No sistema Fenix, essa medida foi usada conforme o esquema descrito na seção 6.3.

~~6.2~~LIMITAÇÕES DA ABORDAGEM BASEADA EM TEXTO PURO

Um sistema de filtragem baseado em conteúdo puramente textual apresenta algumas deficiências que devem ser mencionadas antes de passar para a descrição e análise dos testes realizados.

Em geral, sistemas desse tipo fornecem uma análise superficial e aplicam-se a apenas alguns tipos de páginas WWW. Em alguns domínios de informações, como cinema, música, lazer, etc, a relevância dos ítens de informação não é facilmente identificada por qualquer método de extração de características disponível atualmente. Para esses domínios, seria necessário um modelo que analisasse não apenas os termos individuais que compõem um documento, mas o conteúdo semântico do mesmo, e o contexto no qual ele está inserido (BALABANOVIC, 1998).

Sistemas baseados em texto também são deficientes na análise de páginas que possuem grande parte de seu conteúdo sob a forma de imagens, o que é cada vez mais comum atualmente. Se não houver um texto associado a imagem, a avaliação das páginas pelo sistema será muito pouco significativa em termos de sua verdadeira relevância para o usuário.

Outro problema, que tem sido estudado extensivamente nas áreas de IF e IR, é a super-especialização ("*over-fitting*"). Quando um perfil de preferências de um usuário torna-se muito especializado a um pequeno conjunto de documentos, o sistema não consegue obter novos documentos para apresentação ao usuário, já que os valores de similaridade serão muito baixos exceto para ítens rigorosamente similares aos já classificados. Frequentemente isso é tratado injetando-se um fator de aleatoriedade

como, por exemplo, as operações de *crossover* e mutação (componentes da técnica de algoritmos genéticos) conforme proposto em (SHETH, MAES, 1993).

Finalmente, há um problema comum com a maioria dos sistemas de recomendações: a obtenção da realimentação do usuário. Classificar documentos é uma tarefa onerosa para os usuários, portanto, quanto menos classificação for requerida melhor a aceitação e a utilização do sistema. Com a abordagem baseada em texto puro, as classificações do próprio usuário são o único fator influenciando o desempenho futuro do sistema, e parece não haver um meio de reduzir a quantidade de classificações sem também reduzir o desempenho da recuperação (BALABANOVIC, 1998). Uma maneira de contornar esse problema é adotar uma abordagem colaborativa, na qual o sistema usa a classificação de vários usuários para a construção e adaptação dos perfis.

DESCRIÇÃO DOS TESTES SIMULADOS

Um dos principais propósitos deste trabalho é analisar a aplicabilidade do método de filtragem adotado, geralmente encontrado na literatura em aplicações de filtragem de correio eletrônico e artigos da *usenet*, ao domínio da *Web*.

A implementação do protótipo do sistema foi feita de modo incremental ao longo de cinco etapas. Os teste funcionais do sistema consistiram em executar o protótipo a cada etapa, depurar e eliminar os erros de compilação e lógica que por ventura surgissem. Ao término da implementação, foram realizados testes de erro, onde dados absurdos eram introduzidos no sistema e as falhas no comportamento previsto da execução eram corrigidas. Na análise dos erros, deu-se ênfase à verificação do comportamento dos métodos relacionados à construção dos vetores de termos e pesos à partir das páginas recuperadas. É importante que o sistema seja suficientemente robusto de modo a suportar a presença de dados espúrios nas páginas ou a queda da conexão de rede durante a extração do conteúdo da página, já que tais fatores podem levar à introdução de elementos estranhos nos arquivos de busca. O protótipo foi depurado até apresentar boa resposta a esse tipo de problema, ou seja, até que todos os tipos de erros e exceções decorrentes da presença nas páginas de caracteres inválidos ou pertencentes a esquemas notacionais diferentes dos padrões Unicode ou ASCII tenham sido tratados no código do protótipo. Após os testes funcionais, passou-se para a avaliação do desempenho da recuperação do sistema.

A fim de se avaliar o funcionamento da aplicação, foram adotadas duas abordagens. Na primeira, criou-se um ambiente de simulação, com um banco de dados local contendo páginas referentes a assuntos selecionados, e que seriam usadas pelos agentes para a filtragem e a recuperação de informações. Na segunda abordagem, o protótipo foi colocado para rodar no ambiente real da Internet. Em ambos os casos, os testes foram realizados com usuários fictícios, não sendo feito testes com usuários reais.

A seção 6.3.1 descreve o ambiente simulado criado para os testes do protótipo. A seção 6.3.2 apresenta o esquema adotado para a avaliação dos resultados. Na seção 6.3.3, são analisados os resultados obtidos nos testes realizados.

Ambiente de Simulação para Testes

Para otimizar o tempo gasto nos testes do protótipo do sistema, foi criado um banco de documentos armazenados localmente obtidos a partir das páginas WWW. A criação desse banco local diminui o tempo de rodada do protótipo, já que não é necessário contabilizar o tráfego na rede durante a fase de busca das páginas. Os documentos foram recuperados através de mecanismos de busca disponíveis na *Web* e pertenciam a variadas categorias de assuntos, alguns considerados relevantes para usuários simulados e outros aleatórios. O protótipo foi modificado para fazer a leitura dos documentos a partir do banco local em vez de disparar tarefas de busca na *Web*. As demais etapas de processamento, como a filtragem, a criação e o ajuste dos perfis, foram mantidas inalteradas.

Esquema de Avaliação Baseada na Acurácia do Perfil do Usuário

Um dos métodos adotados para avaliar o desempenho da recuperação de informações baseado na distância ndpm é o teste de acurácia do perfil (BALABANOVIC, 1998), descrito a seguir.

Dado um conjunto D de documentos, é possível inferir a classificação desejada $\succ_d$ para cada um desses documentos. Em um experimento real, $\succ_d$ é derivada a partir da realimentação do usuário e em uma sessão simulada, $\succ_d$ tem que ser gerada de acordo com os interesses do usuário simulado. Para calcular a classificação prevista $\succ_p$ sendo

dados um perfil **m** e um conjunto de documentos **D**, os documentos são classificados de acordo com o valor SIM (**m**, $\mathbf{d_i}$).

Obtendo-se as duas classificações resultantes $\succ_p$ e $\succ_d$, a equação 20 fornece o valor de ndpm, ou seja, a distância entre as classificações. Quando essa distância é medida a intervalos regulares, uma diminuição progressiva indica que o sistema está se adaptando cada vez mais aos interesses do usuário. O teste de acurácia do perfil consiste, então, em medir o valor ndpm em pontos sucessivos no tempo. As curvas de aprendizado geradas a partir da variação dos valores de ndpm em relação ao tempo são especialmente apropriadas para sistemas de filtragem e de recomendação, uma vez que a velocidade de adaptação pode ser um fator decisivo na opção do usuário em usar o sistema.

A formulação original de YAO (1995) assume que a coleção inteira de documentos é classificada tanto pelo usuário quanto pelo sistema. Essa classificação é claramente impraticável quando a coleção é a *Web*. Uma alternativa é medir as classificações previstas e reais somente para os conjuntos de recomendações fornecidos para os usuários. Entretanto, esses são documentos cuidadosamente selecionados para satisfazer o usuário e, provavelmente, encontram-se próximos uns dos outros em termos de preferências do usuário, tornando-se, assim, inadequados para uso com a medida descrita.

Além disso, um usuário real, quando usa um sistema de filtragem ou de recomendação de informações, descobre rapidamente que o sistema se adapta baseado em sua realimentação. Sua realimentação deixa, então, de refletir seus interesses reais e torna-se uma ferramenta para o usuário manipular o sistema.

Neste trabalho, adotou-se o esquema proposto por BALABANOVIC (1998), onde tenta-se evitar que a classificação do usuário seja tendenciosa. A intervalos regulares é fornecida aos usuários uma lista especial de documentos selecionados aleatoriamente a fim de que os usuários os classifiquem de acordo com seus interesses. A classificação consiste em cinco categorias: Excelente - Bom - Neutro - Pobre - Terrível. Essas classificações dos usuários formam a classificação desejada $\succ_d$ e não são usadas para a adaptação do sistema, apenas para testar seu desempenho.

O sistema de filtragem calcula a similaridade dos documentos avaliados pelo usuário em relação ao perfil daquele usuário. As pontuações calculadas pelo sistema são convertidas para cada categoria de classificação de acordo com uma faixa de valores (tabela 2). As classificações obtidas formam as classificações previstas do sistema $\succ_p$. Tendo-se os valores $\succ_d$ e $\succ_p$, pode-se calcular a distância ndpm entre elas e construir a curva de aprendizado do sistema.

PONTUAÇÃO (*SCORE*) (S)	**CATEGORIA**
S < = 0.1	TERRÍVEL
S > 0.1 e S < 0.2	REGULAR
S > = 0.2 e S < 0.3	NEUTRO
S > = 0.3 e S < 0.5	BOM
S > = 0.5	EXCELENTE

Tabela 2: Conversão das pontuações (scores) de similaridade em categorias de classificação.

No esquema proposto, o conjunto total D de documentos é dividido em um conjunto de treinamento e um conjunto de teste. Na avaliação do sistema Fenix, cada iteração de treinamento foi seguida de uma iteração de teste. Uma iteração de treinamento é uma iteração regular de simulação do sistema. Em cada iteração, para cada usuário, um conjunto de documentos é selecionado a partir da porção não vista do conjunto de treinamento, usando o perfil corrente do usuário. A classificação do usuário para esses documentos é simulada de acordo com suas preferências conhecidas. O algoritmo de aprendizado usa essa classificação e as representações dos documentos para atualizar o perfil do usuário.

Em uma iteração de teste, para cada usuário, o conjunto de teste inteiro é classificado de acordo com o perfil do usuário, formando a classificação prevista $\succ_{p.}$ A classificação desejada $\succ_d$ é conhecida a partir das preferências do usuário. A distância ndpm entre essas classificações é calculada e registrada. O resultado desejado nos testes é que a distância ndpm entre as classificações dadas pelo usuário e as do sistema diminua gradualmente com o tempo, conforme o perfil do usuário vai sendo ajustado.

Foram realizadas várias sessões de testes simulados de interação entre o sistema e “usuários”, a fim de se avaliar o desempenho dos agentes. Os testes consistiram na escolha de diversos assuntos de interesse e na execução de várias sessões de busca, filtragem e avaliação dos resultados pelo “usuário”. Adotou-se o mesmo esquema nos testes realizados em ambiente local simulado e nos testes realizados no ambiente real da Internet. Na apresentação e na análise dos resultados não será feita distinção entre os dois ambientes.

Primeiramente, era realizada uma busca inicial baseada em palavras-chaves. Um número determinado de documentos resultantes da busca era apresentado para o usuário, e este selecionava alguns considerados relevantes. Os documentos relevantes selecionados eram usados para a construção do perfil do usuário. O conjunto total de documentos resultantes da busca formavam o corpo D de documentos, e este corpo D era dividido em um conjunto de teste e um conjunto de treinamento. O usuário classificava, segundo as categorias acima, cada um dos documentos do corpo D.

A busca inicial era considerada como a iteração 0 do agente. A partir daí, seguiam-se novas iterações de teste e de treinamento, onde a realimentação fornecida pelo usuário a alguns documentos em cada iteração era usado para treinar o sistema. A cada iteração de teste, os valores de classificação do sistema e do usuário eram usados para calcular a distância ndpm. Os valores de ndpm calculados eram usados para construir um gráfico a fim de observar a variação de ndpm ao longo das iterações. Esse gráfico representa a adaptação do sistema ao longo do tempo aos interesses de seu usuário.

Foram criados agentes para os mais variados assuntos, como mergulho, arquitetura, algoritmos, redes, corba, delphi, inteligência artificial, viagens e outros, conforme pode ser visto na tabela 3. Foram testados diferentes tamanhos de coleções e diferentes números de sessões. As palavras-chaves escolhidas variaram desde consultas com termos genéricos até bastante específicos. O tamanho dos vetores de termos também foi um parâmetro analisado, escolhendo-se diferentes números de termos e comparando-se as curvas de ndpm obtidas. Os resultados dos testes são analisados a seguir.

Nome do Agente	Palavras-chaves	Tamanho da coleção	Número de sessões	Tamanho do vetor de termos
IntArt	Inteligência Artificial	11	5, 6, 7	300
RedesComp	Redes de Computadores	11, 22	5, 6	300
Redes	Redes	22	5	20, 50, 100, 300
RedesNeu	Redes Neurais	13	5	300
Algorit	Algoritmos	25	5	300
AlgGen	Algoritmos Genéticos	19	5	20, 50, 100, 300
AlgComp	Algoritmos Computacionais	12	5	300
IF	Information Filtering	19	5, 6	20, 50, 100, 300
BDDist	Distributed Database	18	5	20, 50, 100, 300
Mergulho	Mergulho	19, 38	5	300
Fotografia	Fotografia	10, 21	5	300
CurFot	Cursos de Fotografia	21	5	300
EqFot	Equipamento Fotográfico	21	5	20, 50, 100, 300
AgInt	Intelligent Agents	11	5	300
Arquit	Arquitetura	8, 12	5	20, 50, 100, 300
AgMov	Mobile Agents	25	5, 6	300
Agente	Agentes	13	5	300
Corba	Corba	18, 7, 10	5	20, 50, 100, 300
Delphi	Delphi	6	3	20, 50, 100, 300
Linux	Linux	12	5, 7	300
Java	Java	7	3, 5, 6	300
LingOO	Object Oriented Language	14	5	300
Viagem	Viagem	25	5	300
Medita	Meditação	10	5	300

Tabela 3: relação de agentes criados para avaliar o sistema.

Resultados

Foi realizado um total de 155 sessões simuladas. Para cada assunto considerado foi criado um agente. Cada agente tinha no máximo 10 perfis. Cabe aqui ressaltar que cada perfil inicia uma tarefa do sistema operacional e um número excessivo de perfis sendo processados simultaneamente pode gerar uma sobrecarga no sistema. A seguir são descritas as principais conclusões em relação aos parâmetros estudados, obtidas a partir da avaliação dos testes realizados. Os gráficos que apresentaram as variações de parâmetros mais significativas são mostrados para exemplificar as conclusões descritas.

- **Conclusões Quanto à Especificidade da Consulta**

A partir dos testes realizados observou-se que o grau de especificidade dos termos usados
na consulta reflete diretamente no desempenho de recuperação do sistema. Usuários com interesses pouco definidos em geral submetem consultas com expressões genéricas, e as primeiras rodadas do agente servem apenas para especificar melhor o sub-conjunto de interesses do usuário dentre os vários assuntos presentes nos documentos recuperados. Através dos primeiros valores de realimentação fornecidos, o usuário direciona, então, o agente, na descoberta de sua linha de interesse específico dentro do assunto em questão. Tal procedimento acarreta uma maior demora na adaptação do agente, refletida por valores mais altos de ndpm nas primeiras rodadas.

Se a consulta inicial fornecer termos mais precisos, nas primeiras rodadas o agente já estará trabalhando em um sub-conjunto menor de documentos, e poderá especializar-se melhor e mais rapidamente às necessidades específicas do usuário dentro da gama de interesses bem definidos. Esse tipo de consulta gera curvas de ndpm com valores mais baixos logo no início. Como exemplo desse comportamento pode-se observar a curva da figura 8, onde os valores de ndpm nas sessões iniciais são mais baixos para o agente criado com as palavras-chaves “Redes Neurais” (termo mais específico) e mais altos para o agente criado com as palavra-chave “Redes”(termo mais genérico). O mesmo comportamento foi observado para o conjunto de agentes criados para “Algoritmos”, “Algoritmos Computacionais” e “Algoritmos Genéticos” e para o conjunto relacionado a “Agentes”, “Agentes Inteligentes” e “Agentes Móveis”.

Sugere-se, então, nos casos em que o usuário não esteja bem certo quanto ao tipo de assunto desejado, que ele use a busca inicial do agente apenas para descobrir a categoria de seu interesse dentro do assunto pesquisado. Nesse caso, recomenda-se que os resultados da busca inicial não sejam usados para gerar o perfil do agente, mas sim, para auxiliar na formulação de uma expressão de consulta mais específica. A partir da melhor definição de suas necessidades, o usuário faz novamente a busca inicial, e dessa vez salva os resultados, criando o perfil. O uso do próprio agente para o auxílio na construção de uma expressão de consulta mais adequada, que será depois refinada ao longo das sucessivas rodadas, irá melhorar o desempenho global do sistema.

Outro fato observado é que, pelas características do modelo adotado (baseado no conteúdo textual dos documentos), agentes para assuntos como viagens, interesses comercias, lazer, e esportes tendem a ter desempenho mais pobre, pois esse tipo de assunto em geral envolve muitas imagens e pouco texto. Os melhores resultados foram encontrados em sessões de assuntos científicos. Os valores mais baixos de ndpm final foram obtidos para os agentes criados para "Linguagem Orientada a Objetos"(ndpm=0.09), para "Algoritmos Genéticos" (ndpm=0.07) e para "Agentes Móveis"(ndpm =0.09), três assuntos altamente específicos. Os valores de ndpm finais mais elevados foram para o assunto "Meditação"(ndpm =0.27) e "Viagem"(ndpm=0,29).

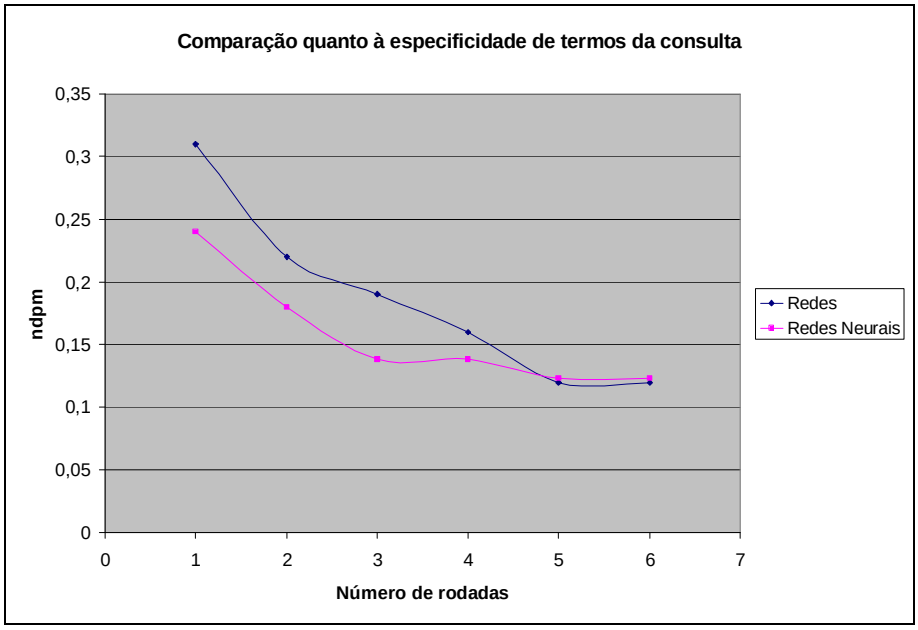


Figura 8: ndpm ao longo de seis sessões para os agentes criados para "Redes" e "Redes Neurais".

- **Conclusões Quanto ao tamanho da coleção**

Para fins de teste do protótipo, o tamanho da coleção corresponde ao conjunto total de documentos (D) obtidos na busca inicial do agente e que é separado em um sub-conjunto de teste e outro de treinamento. O tamanho da coleção no sistema Fenix é um parâmetro que pode ser configurado pelo usuário a cada sessão de busca. Valores diferentes podem ser configurados a cada sessão e caso o usuário não informe nada, o

sistema usa o valor padrão de 30 documentos como o número máximo a ser recuperado em uma busca. Eventualmente, em sessões de interação reais, o número de documentos efetivamente apresentados ao usuário pode ser bem menor do que o número máximo considerado pois muitos documentos da coleção recuperada na busca são descartados no processo de filtragem.

O tamanho máximo padrão adotado para a coleção no sistema Fenix foi escolhido após os testes do protótipo, onde foram experimentados tamanhos variando de 6 a 38 documentos.

Os resultados dos testes apontaram para o fato de que com coleções muito pequenas (em geral menos de 13 documentos) o perfil de preferências do usuário acaba se tornando muito restrito e em um pequeno número de iterações tende a aparecer o problema da super-especialização ("*over-fitting*"), frequentemente relatado em trabalhos de máquinas de aprendizado (BALABANOVIC, 1998). Esse problema se agrava quanto maior for a semelhança entre os documentos que compõem a coleção. Nesses casos, o perfil torna-se tão especializado àquele conjunto particular de documentos, que fica extremamente difícil encontrar novos documentos no espaço de busca que sejam similares ao perfil. A estagnação resultante pode ser observada pelos valores constantes de ndpm após algumas rodadas, indicando que as avaliações feitas pelo sistema não estão mais melhorando, provavelmente porque os pesos dos termos no perfil estão constantes e novos termos não estão sendo adicionados.

Os agentes criados para os assuntos "Delphi" e "Java" exemplificam essa estagnação, pois apresentaram pouca ou nenhuma redução nos valores de ndpm após a terceira sessão, e os valores finais ainda eram relativamente altos (figuras 9 e 10).

Por outro lado, coleções muito grandes consomem mais recursos de processamento e memória, e não parecem influenciar de forma muito significativa no desempenho do sistema, conforme se constatou ao testarem-se diferentes tamanhos de coleções para o mesmo assunto. O agente para "Mergulho", por exemplo, teve o tamanho de coleção configurado para 19 documentos e, a seguir, para 38, que corresponde ao dobro da coleção inicial, e os valores de ndpm foram praticamente idênticos, conforme pode ser visto na figura 11. O mesmo comportamento foi

apresentado pelos agentes para "Redes de Computadores", "Fotografia" e "Corba", indicando que, a partir de um tamanho de coleção mínimo que evite a super-especialização muito rápida, não há necessidade de aumentar demais o conjunto de documentos utilizados.

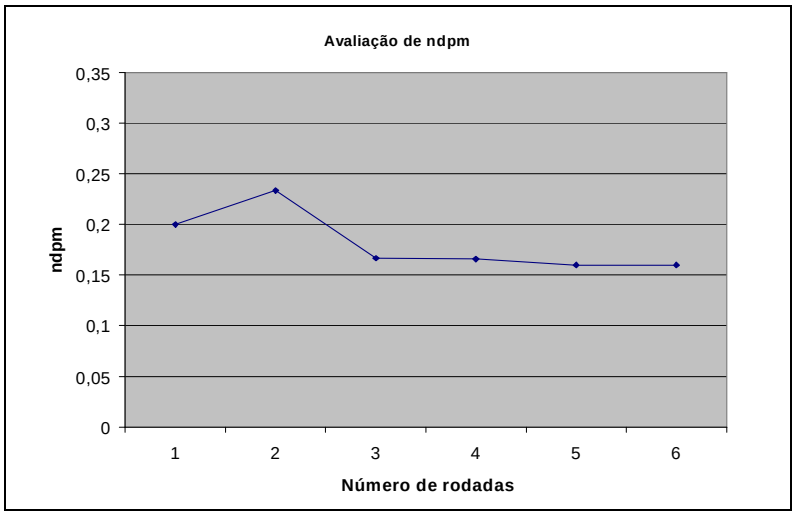


Figura 9: ndpm ao longo de seis sessões para o agente “Delphi”.

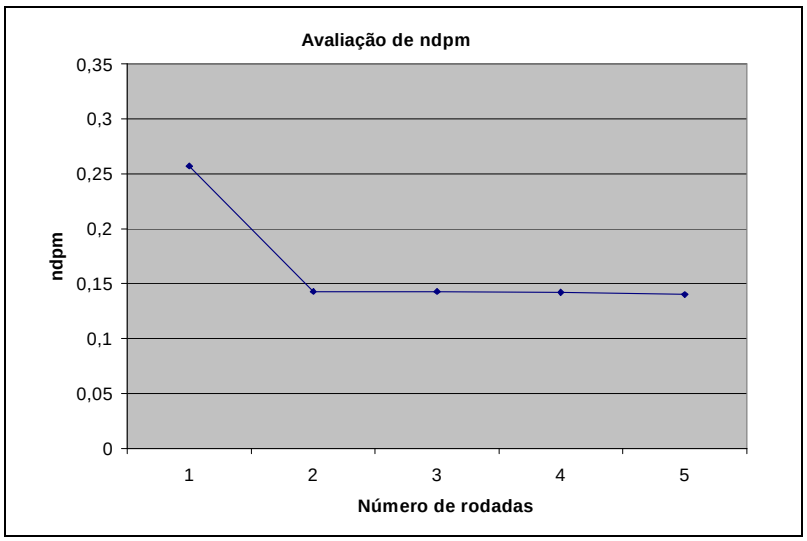


Figura 10: ndpm ao longo de cinco sessões para o agente “Java”.

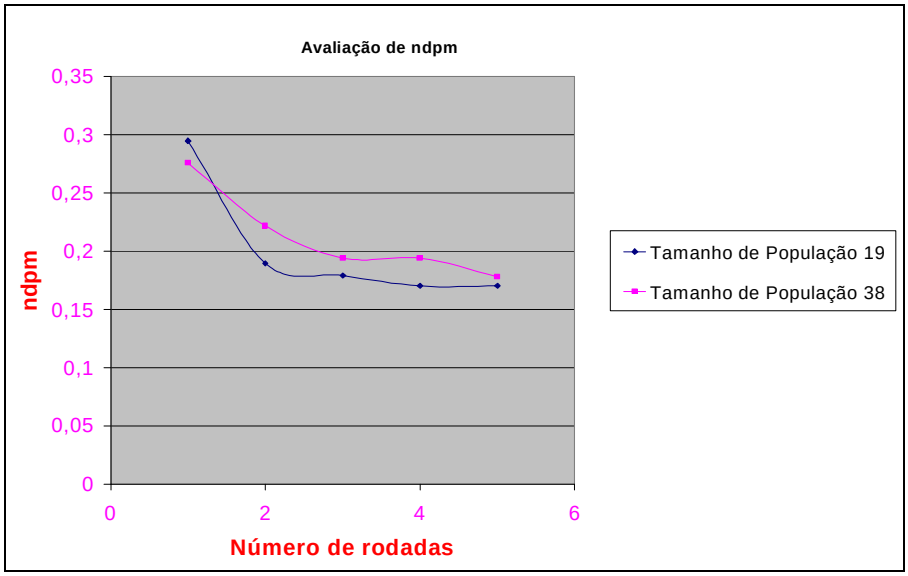


Figura 11: ndpm para o agente “Mergulho” com tamanhos de população de 19 e 38.

- **Conclusões Quanto ao número de sessões**

Nos testes realizados com o protótipo, o conjunto D de documentos pertencentes à coleção foi mantido constante ao longo das sessões, já que o interesse residia nos diferentes valores de avaliação dados pelos agentes aos documentos, conforme o perfil ia sendo ajustado. Dessa forma, as alterações do perfil decorriam do fato de, a cada sessão, o usuário fornecer realimentação para um certo número de documentos. Os valores de realimentação possíveis foram estipulados como +1, 0 e -1.

No ambiente de utilização real do sistema, novas buscas são realizadas com realimentação ao longo das sessões, tanto de forma autônoma quanto iniciadas pelo usuário. Assim, novos documentos são introduzidos na coleção conquanto tenham um valor de similaridade mínimo em relação ao perfil para não serem descartados na filtragem. As novas buscas contribuem para a diversificação dos termos presentes no perfil. Entretanto, é importante ressaltar que cada perfil está associado a um mecanismo de busca na *Web* específico. Se o intervalo de tempo decorrente entre as buscas for pequeno, provavelmente haverá pouca diferença nos documentos recuperados, já que a frequência de inclusão de novas páginas nos repositórios utilizados pelos mecanismos de busca não é muito alta.

Experimentos descritos em (BALABANOVIC, 1998) apontam para o fato dos usuários reais poderem ser extremamente restritivos, atribuindo pontuações altas apenas para páginas exatamente sobre o assunto desejado e pontuações muito baixo para as outras. Essa característica pode levar a uma ausência de exemplos positivos para o aprendizado e subsequentemente a um comportamento altamente aleatório. Segundo Balabanovic, um sistema de filtragem trabalha melhor quando o usuário o “guia”, porém começando com um tópico grosseiro (genérico) e vagarosamente especializando esse tópico. Uma forma do sistema evitar essa ausência de exemplos positivos é permitir que o usuário forneça suas próprias páginas de exemplo.

Nos testes com o protótipo do Fenix, procurou-se simular esse comportamento dos usuários da melhor forma possível. A cada sessão de interação com o sistema um número fixo de documentos era escolhido aleatoriamente para o usuário fornecer realimentação. Somente documentos exatamente pertinentes ao assunto desejado receberam as classificações "EXCELENTE" e "BOM" do usuário simulado e apenas esses documentos receberam realimentação positiva nas iterações de treinamento do sistema. Documentos classificados como "NEUTROS" receberam realimentação igual a zero e não contribuíram para a composição do perfil. Documentos classificados como "REGULAR" e "TERRÍVEL" receberam realimentação negativa.

O caráter restritivo observado no julgamento dos usuários pode fazer com que páginas que tenham um grau de pertinência razoável ao assunto sejam classificadas como neutras e não venham a contribuir para a construção de perfis um pouco mais genéricos. Perfis muito especializados favorecem o aparecimento da super-especialização, que pode levar a estagnação do agente.

Os resultados dos testes mostraram que, em geral, não ocorre uma melhora significativa nos valores de ndpm após a quinta sessão de execução do protótipo, ou seja, após 5 iterações de teste/treinamento. Pode-se observar essa tendência na curva da figura 13, que apresenta os valores de ndpm médios para todos os agentes. Esses resultados indicam a possível ocorrência da super-especialização. Provavelmente o que ocorre é que, pela tendência simulada para o usuário de selecionar apenas as páginas rigorosamente relevantes para o assunto desejado as quais são em geral mais semelhantes entre si, em poucas sessões o perfil torna-se extremamente especializado

àquele sub-conjunto específico de documentos. Dessa forma, os valores de classificação fornecidos pelo sistema para outros documentos é cada vez menor, exceto para documentos muito similares ao perfil. Como consequência, as avaliações de ndpm não melhoram e eventualmente podem até piorar, quando por exemplo, a classificação de um documento com um grau de relevância médio passa de "NEUTRO" para "REGULAR" à medida em que o perfil torna-se mais restritivo.

O fornecimento de páginas de exemplo pelo próprio usuário pode diversificar um pouco a composição dos perfis, evitando a estagnação do sistema. Essa alternativa foi implementada no protótipo de duas formas: através dos botões "Altera" e "Inclui". Com o botão "Altera" o usuário pode alterar manualmente a URL de um documento recuperado pelo agente caso ele descubra um *link* mais interessante descendo um ou mais níveis a partir do documento originalmente recuperado. Com essa opção, o documento original fornecido pelo agente continua fazendo parte da coleção e o novo documento é incluído com realimentação positiva para fins de treinamento do agente. O botão "Inclui" permite que o usuário entre com a URL de um página totalmente nova para o sistema, a qual será usada no treinamento do agente mas não será considerada para fins de avaliação do desempenho.

Uma outra solução para o problema da estagnação do agente seria a introdução de novos termos no perfil através da implementação do módulo de algoritmo genético. Os novos termos poderiam ser obtidos a partir da utilização de operadores genéticos que promovem a recombinação de termos entre diferentes perfis ou a substituição aleatória de termos dentro de um perfil individual (operadores de *crossover* e mutação). A diversidade que pode ser introduzida no sistema através dessas operações é o principal objetivo da utilização de algoritmos genéticos para o sistema proposto.

Apesar de não ter se observado mudança significativa nos valores de ndpm após a quinta sessão, os valores médios obtidos nesse ponto já eram bastante baixos, levando a resultados de filtragem considerados satisfatórios. Considera-se um resultado como satisfatório quando a maioria das classificações de documentos feitas pelos agentes são semelhantes às do usuário, o que indica que os documentos recomendados pelo sistema são de fato relevantes àquele assunto.

- **Conclusões Quanto ao tamanho do vetor de termos**

A escolha do número de termos que devem ser usados na construção do vetor de termos de um documento é um assunto que tem sido alvo de estudos no âmbito dos sistemas de filtragem de informações baseados em conteúdo. Usar todas as palavras de um documento para indexá-lo tende a conduzir ao problema da super-especialização. Além disso, aumenta o espaço de memória e a quantidade de processamento requeridos. Experimentos recentes (PAZZANI *et al.*, 1996) mostraram que usar um número excessivo de palavras leva a uma diminuição do desempenho em sistemas de classificação de páginas WWW baseados em métodos de aprendizado supervisionado, como é o caso da realimentação por relevância.

Na avaliação do protótipo do Fenix foram feitos vários testes onde o número máximo de termos usados para indexar cada documento variou de 20 a 300. A figura 12 mostra os valores de ndpm média para os agentes usados nesses testes, ao final de 6 iterações. Pelo gráfico observa-se que com tamanhos de vetores de 20 e 50 o sistema não obteve o máximo desempenho possível, indicando que esses tamanhos são sub-ótimos para fins de indexação. Metade dos agentes usados nos testes não apresentou melhora significativa quando se elevou o tamanho de 100 para 300. A outra metade apresentou alguma melhora, como reflete o gráfico da figura 12.

O trabalho de PAZZANI *et al*. (1996) obteve valores ótimos de desempenho para vetores com tamanho igual a 96 termos. BALABANOVIC (1998) relata em seu trabalho que não houve aumento de desempenho para vetores com mais de 100 termos.

Como na média dos testes realizados com o Fenix verificou-se uma pequena melhora para vetores de 300 termos, optou-se por adotar esse número como o tamanho máximo na construção dos vetores de termos. Cabe ressaltar que muitos documentos, após passarem por todas as etapas de extração de termos acabam sendo indexados por um número de termos bastante inferior a 300.

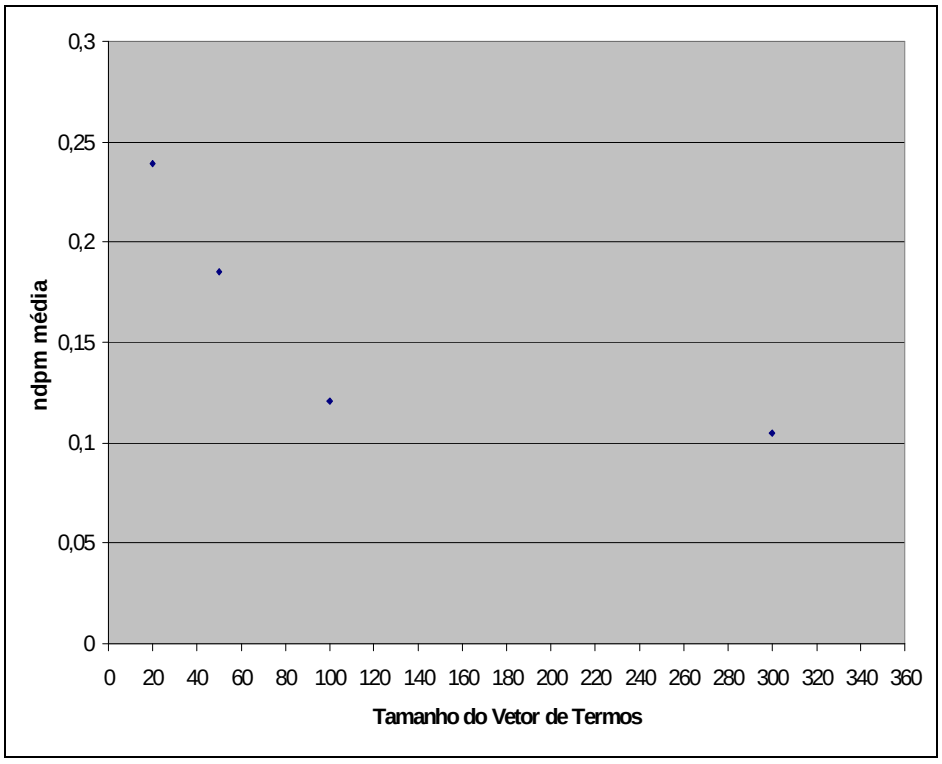


Figura 12: Comparação dos valores médios de ndpm para diferentes tamanhos do vetor de termos.

- **Escalabilidade do Sistema**

Nos testes realizados foi criado um agente para cada assunto de interesse dos usuários simulados. Um mesmo "usuário" poderia ter vários assuntos de interesse. Estipulou-se que o número máximo de assuntos (agentes) por usuário era de 10 (dez). Além disso, cada agente possui um conjunto de perfis individuais. O número de perfis de um agente é controlado pelo número de documentos que recebem realimentação positiva após a busca inicial: cada documento gera um arquivo de perfil. Como um perfil dispara uma tarefa do sistema operacional e todos os perfis de um agente são processados simultaneamente, muitos perfis podem gerar sobrecarga no sistema. Por outro lado, um número muito pequeno de perfis restringe o desempenho de recuperação, já que a filtragem de novos documentos leva em conta os termos de todos do perfis. Adotou-se

como sendo 4 (quatro) o número mínimo e 10 (dez) o máximo para a quantidade de perfis de um agente.

Considerando-se que no modo autônomo pode-se ter o processamento simultâneo de todos os perfis de todos os agentes de um usuário, o sistema pode ter o número máximo de 100 tarefas sendo executadas concomitantemente. Esse número não foi atingido nos testes, procurando-se usar um número menor de agentes por usuário. Com a faixa de valores testada (em média 6 perfis por agente e 7 agentes por usuário) o sistema apresentou resultados satisfatórios, não havendo degradação no tempo de resposta. Entretanto, o protótipo implementado não possui boa escalabilidade, já que não adota nenhum tipo de estrutura índice para o armazenamento dos dados dos agentes. Caso o número de agentes e perfis cresça, o tempo gasto no processamento dos vetores de termos e pesos de perfis e documentos poderá elevar-se muito, comprometendo a velocidade de resposta do sistema.

Para que o sistema completo seja viável em termos de tempo de resposta é necessária a utilização de um esquema de indexação dos arquivos de perfis e a implementação de algoritmos que agilizem o processamento desses arquivos.

KJERSTI (1997) propôs um método de indexação de perfis para o modelo vetor-espacial que foi analisado e adaptado para este trabalho e sua implementação é apresentada no capítulo 7 como sugestão para trabalhos futuros.

Considerações Finais

A figura 13 mostra a distância ndpm média calculada a cada ponto de avaliação para todos os 22 assuntos. Para todos os assuntos, observou-se uma diminuição progressiva dessa distância ao longo das sessões. Essa diminuição indica que os agentes foram se adaptando às preferências do usuário, aumentando, assim, a probabilidade de recuperar um maior número de documentos relevantes e de descartar os irrelevantes.

Os resultados obtidos são semelhantes aos descritos em (BALABANOVIC, 1998), onde um sistema multiagentes híbrido foi implementado para a recomendação de páginas WWW. Balabanovic propôs uma arquitetura em que combinava filtragem baseada em conteúdo com filtragem colaborativa. O desempenho de seu sistema também foi avaliado com a medida ndpm e a curva obtida teve comportamento bastante

similar à obtida nos testes do sistema Fenix, ilustrada na figura 13. Em seu trabalho foram realizadas 25 sessões de avaliação, o valor médio inicial de ndpm foi de 0.4 e o final foi 0.001. Esse resultado comprova a eficácia de um sistema de filtragem baseado no conteúdo de documentos e usando realimentação por relevância como mecanismo de aprendizado.

De uma maneira geral pode-se dizer que o sistema atinge o melhor desempenho de recuperação quando os termos da consulta são mais específicos, os agentes possuem entre 4 e 10 perfis e os vetores de termos que representam os documentos têm tamanho máximo igual a 300. Métodos baseados em conteúdo são fortemente dependentes de uma boa escolha dos documentos que compõem a coleção. Quando o domínio estudado é a *Web*, surgem problemas decorrentes de páginas com molduras, tabelas ou conteúdos não textuais. Como a escolha fica a cargo do usuário, não há como melhorar o desempenho sem contar com valores adequados de realimentação, que garantam que o perfil aprenderá um modelo de preferências significativo.

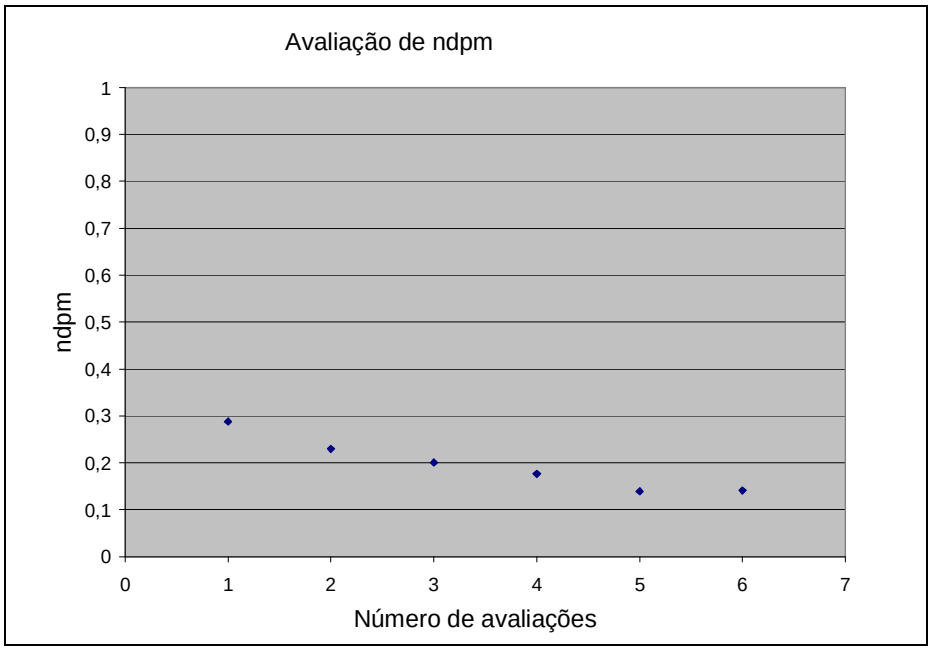


Figura 13: ndpm média ao longo das sessões de avaliação.

Uma das vantagens da representação vetor-espacial é que o conteúdo dos perfis adaptados pode ser inspecionado, a fim de se verificar se a maioria dos termos

armazenados realmente consiste de termos pertinentes ao assunto representado pelo perfil.

Por exemplo, em um perfil relativo a “fotografia”, pode-se verificar no próprio arquivo que dos 66 termos de maior peso, 46 são efetivamente referentes ao assunto em questão (figura 14).

```
Proteqpf.0
Eqpfot.bin
http://www.mark-e-marca.pt/agf/agf_mhk.htm 1 P 0.0
http://saudetotal.com/saude/derma/fotogr02.htm 1 F 0.0
Type 2.8323 Fotografia 1.0066 Dermatológica: 2.8323 aspectos 1.4747
técnica 2.8323 Profissional 3.4348 Artigo 3.1847 Burjato 2.8323 Jr
2.1971 Junior 3.4348 acessório 0.9844 obrigatório 2.1971 imagem 2.8323
suporte 3.1847 adaptado 2.8323 observação 2.8323 médica 2.8323
seguimento 2.8323 Equipamento 2.8323 Fotográfico 2.47 Iluminação
2.8323 Macrofotografia 2.5405 Microfotografia 2.8323 Arquivamento
2.8323 conservação 1.4283 uso 3.042 dermatológico 2.8323 câmara 2.8323
filme 2.8323 35mm 2.47 bitola 1.78 filmes 2.1971 reversíveis 2.8323
processado 2.8323 montados 1.6582 molduras 2.8323 5x5cm 2.8323
fotográfica 3.4348 composta 2.8323 Corpo 2.8323 caixa 2.8323
componentes 3.2114 luz 3.2114 objetiva 2.1971 momento 2.8323 foto
2.9380 Visor 3.4348 objeto 1.5405 fotografado 2.1971 fotógrafo 3.1847
estudo 1.5405 ângulo 1.4747 sistema 3.1847 reflex 1.7830 visão 1.7830
erro 3.6288 paralax 3.4348 Diafragma 2.0049 obturador 2.1971 tempo
2.8323 incidência 2.8323 hora 2.8323 disparo 2.8323 sincronismo 2.8323
ótimo 1.4283 programado 2.8323 forma 2.8323 câmaras 2.8323 automáticas
2.8323 Elemento 2.1971 aparelho 2.8800 objetivas 2.8323 grande 1.27
angulares 2.4737 deformação 1.78 fotografar
```

Figura 14: conteúdo de um arquivo de perfil para o agente "Fotografia" ao final das avaliações.

7 CONCLUSÕES E TRABALHOS FUTUROS

O sistema Fenix é um Sistema de Filtragem de Informação capaz de especializar-se conforme os interesses do usuário, adaptar-se às mudanças e explorar o domínio de informações potencialmente relevantes. Os resultados iniciais obtidos a partir da avaliação do protótipo do sistema foram bastante promissores. O sistema proposto demonstrou ser uma ferramenta bastante útil na filtragem de informações obtidas da *Web*. O método de realimentação por relevância adotado provou ser um mecanismo eficiente para a reformulação de consultas, sendo responsável por uma melhora progressiva nos resultados da filtragem. Entretanto, há ainda algum trabalho adicional a ser feito a fim de tornar o sistema implementado um programa mais completo e amigável.

Neste capítulo são feitas algumas considerações sobre os resultados obtidos e são sugeridos trabalhos futuros que venham a complementar e melhorar o sistema proposto. A seção 7.1 descreve algumas alternativas de implementação como sugestão para melhorar o desempenho e a escalabilidade do sistema em trabalhos futuros. Na seção 7.2 são feitas algumas considerações finais sobre os resultados obtidos e as principais contribuições do presente trabalho.

TRABALHOS FUTUROS

O protótipo descrito foi muito útil para demonstrar a viabilidade do sistema proposto, porém possui algumas limitações que devem ser contornadas a fim de se construir um programa mais completo e eficiente e que possibilite a realização de experimentos maiores.

Um dos problemas apontados é a necessidade de um sistema que comporte uma maior escalabilidade. O número de usuários e, portanto, de perfis manipulados por um sistema de filtragem disponibilizado para uso em ambiente real cresce potencialmente, e o sistema tem que processar o fluxo de documentos que entram em um tempo adequado. A eficiência do processo de filtragem é, então, uma questão crucial no desenvolvimento de um sistema de filtragem de informações. Algumas técnicas para aumentar a eficiência do processamento das informações para um número crescente de usuários

devem ser avaliadas a fim de se comportar a escalabilidade do sistema. Nesta seção são analisadas estruturas de dados e algoritmos que podem ser usados para realizar filtragem de informações em larga escala de forma eficiente com o modelo vetor-espacial.

Uma possível solução para a questão da escalabilidade é a implementação de um sistema com múltiplos agentes de busca coletando páginas WWW em vários meios Os agentes de busca depositariam as páginas obtidas em um repositório central onde agentes pessoais de usuários individuais recuperariam aquelas que melhor satisfizessem os perfis de seus usuários. A realimentação fornecida iria tanto para o agente pessoal (que atualizaria o perfil do usuário) quanto para os agentes de busca, que agora estariam servindo a grupos de usuários em vez de usuários individuais e tirando proveito dos interesses compartilhados. Um sistema desse tipo pode ser empregado para investigar problemas de escalabilidade, particularmente para descobrir como o desempenho degrada quando o número de usuários aumenta, para um conjunto fixo de agentes de busca.

Utilização dos agentes por diferentes usuários

A princípio, o sistema Fenix foi projetado de forma que cada agente visa especializar-se aos interesses de um único usuário. Entretanto, os resultados das buscas poderiam ser compartilhados por diferentes usuários, além do proprietário do agente. Uma forma de obter essa "colaboração" entre os usuários é comparando-se cada documento obtido em uma busca com todos os perfis de usuários armazenados, e não apenas com o perfil responsável pela busca. Entretanto, como os perfis relacionam-se aos mais variados assuntos e não há nenhum tipo de mecanismo de agrupamento de perfis de assuntos semelhantes, isso causaria uma enorme sobrecarga de processamento no sistema, principalmente com o número crescente de perfis e usuários. O esquema ideal seria descobrir inicialmente que perfis teriam maior possibilidade de representar um interesse potencial com relação a um dado documento. Uma forma de implementar esse esquema é gerar listas com os termos presentes em todos os perfis, onde cada termo faz referência a todos os perfis em que aparece. Assim, em vez de se comparar um documento com todos os perfis arquivados, compara-se inicialmente com a lista e, a partir dela, se obtêm os perfis mais prováveis de se ajustarem ao documento, ou seja, aqueles que contém maior número de termos em comum com o documento.

Para reduzir o número de perfis que devem ser examinados, uma possibilidade é construir um índice invertido de perfis. Para cada termo x, são coletados todos os perfis que o contêm para formar sua lista invertida. Cada elemento da lista é composto pelo identificador de um perfil contendo x e o peso de x nesse perfil. Então um perfil com p termos será encontrado em p posições, cada uma em uma lista diferente. Ao processar um documento D, é necessário apenas examinar os perfis nas listas invertidas dos termos que estão em D.

Para comparar um documento com esses perfis são necessários dois vetores: um para armazenar valores pré-estabelecidos de limites de relevância (vetor *THRESHOLD*) e outro para manter as pontuações dos perfis (vetor *SCORE*). O número de entradas em cada vetor é igual ao número de perfis que o sistema manipula. Cada perfil tem uma entrada em cada vetor.

Quando um documento D chega, o vetor *SCORE* é inicializado atribuindo-se zero a todas as suas posições. Para cada termo x com peso w no documento a lista invertida de x deve ser recuperada e cada perfil P na lista deve ser processado. Se o peso de x em P é u, o elemento *SCORE* [P] é incrementado de w x u. Após todos os termos do documento serem processados, um perfil cuja entrada em SCORE seja maior que a entrada em *THRESHOLD* combina com o documento.

Esse método consome mais espaço de memória do que quando não se usa qualquer tipo de indexação de perfis, mas é mais eficiente em termos de processamento, especialmente quando o número de usuários e, portanto, de perfis, torna-se muito grande.

Local de filtragem das informações

Um sistema de filtragem rodando no ambiente de utilização real da Internet possui um conjunto de usuários e acessa um certo número de fontes de informações. Um fator investigado em sistemas de filtragem para a *Web* é o local onde realizar a filtragem dos documentos. Existem basicamente três opções: a filtragem pode ser feita nas próprias fontes de informações, na máquina do usuário ou em um servidor de filtragem. Realizar a filtragem na máquina do usuário é uma alternativa de alto custo e que ocasiona tráfego intenso na rede, já que a banda passante é gasta para transmitir informações irrelevantes

e é gerado muito desperdício de processamento local. Realizar a filtragem nas próprias fontes de informações também implica em alto custo, uma vez que os usuários precisam ter seus perfis replicados em todas as fontes possíveis. O servidor de filtragem de informações é um bom compromisso de solução entre essas duas alternativas (SHIVAKUMAR, GARCIA-MOLINA, 1998). Ele coleta informação de um conjunto de fontes e roteia para os usuários interessados. Pode haver múltiplos servidores de filtragem na rede, cada um atendendo um conjunto diferente (talvez sobrepostos) de usuários e fontes de informações.

Um servidor de filtragem de informações deve ser capaz de disponibilizar informação relevante para os usuários em tempo hábil para o serviço ser útil. Levando em conta o número potencialmente crescente de usuários e a alta taxa de geração de novos documentos nas diferentes fontes, o servidor deve empregar algoritmos e estruturas de dados para acelerar o processo de filtragem, do tipo descrito na seção 7.3.1.

CONSIDERAÇÕES FINAIS

O trabalho descrito adota conceitos de várias áreas de pesquisas acadêmicas, incluindo recuperação de informações, máquina de aprendizado, modelagem do usuário e agentes, com a finalidade de propor um sistema de filtragem de informações para a *Web*. O principal objetivo deste trabalho foi aplicar a tecnologia de agentes no desenvolvimento de um sistema de filtragem personalizado de páginas WWW.

Os resultados dos testes simulados realizados com o protótipo do sistema Fenix em ambientes controlados foram bastante satisfatórios. Eles confirmaram que um sistema de filtragem para a *Web* empregando o modelo vetor espacial com mecanismo de aprendizado baseado em realimentação por relevância pode recomendar com sucesso documentos de interesse para usuários com preferências bem definidas. Como trabalho futuro, a implementação do Algoritmo Genético seria uma promissora solução para introduzir diversidade no sistema a fim de evitar o problema da super-especialização e para modelar a adaptação de mudanças de interesse do usuário a longo prazo.

Os valores de desempenho obtidos em testes simulados baseados na distância ndpm foram similares aos encontrados em outros trabalhos de filtragem de informações. BALABANOVIC (1998) propôs uma abordagem para a recomendação de páginas WWW que combinava o método de filtragem baseado em conteúdo usando realimentação por relevância, com o método de filtragem colaborativa, no qual as recomendações de outros usuários afetam a resposta do sistema. Os resultados descritos em seu trabalho foram bastante semelhantes aos obtidos com o Fenix, onde foi adotada uma abordagem bem mais simples de implementar.

O impacto de sistemas como Fenix em uma empresa pode ser altamente significativo. Muitas das tarefas de busca de informações podem ser automatizadas, pois o sistema é capaz de filtrar e priorizar diversos assuntos interessantes, reduzindo, assim, o consumo de tempo gasto nessas atividades.

Agentes de Filtragem de Informação são uma grande promessa para o gerenciamento das inúmeras informações disponíveis.

ANEXO 1 - DIAGRAMAS DE CLASSES DOS MÓDULOS COMPONENTES DO SISTEMA FENIX

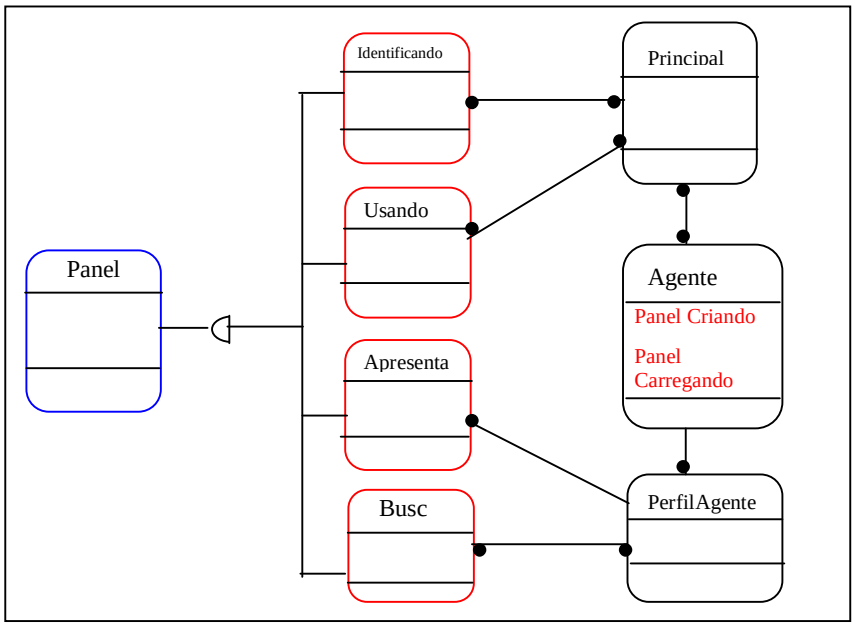


Diagrama de Classes do Módulo de Interface. Os diagramas em vermelho indicam as classes componentes do módulo e os diagramas em azul indicam classes da biblioteca padrão de Java.

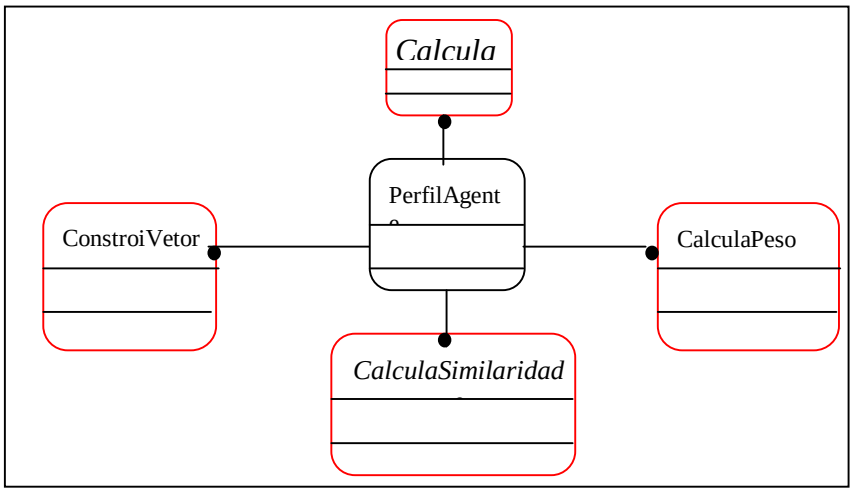


Diagrama de Classes do Módulo de Filtragem.

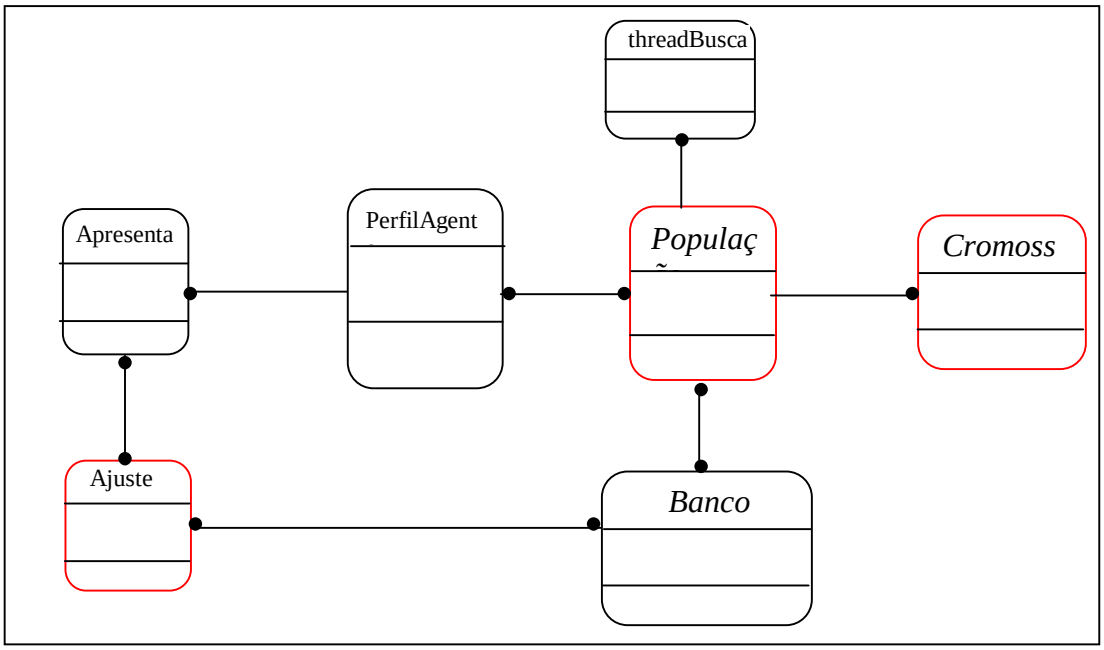


Diagrama de Classes do Módulo de Aprendizado, incluindo os sub-módulos de realimentação por relevância e de algoritmos genéticos.

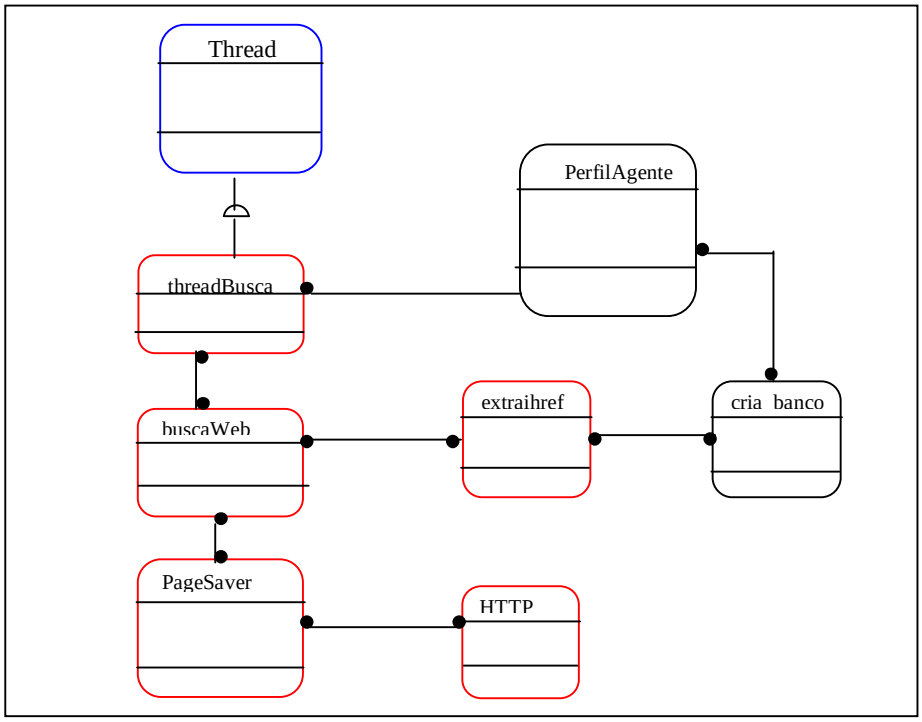


Diagrama de Classes do Módulo de Busca

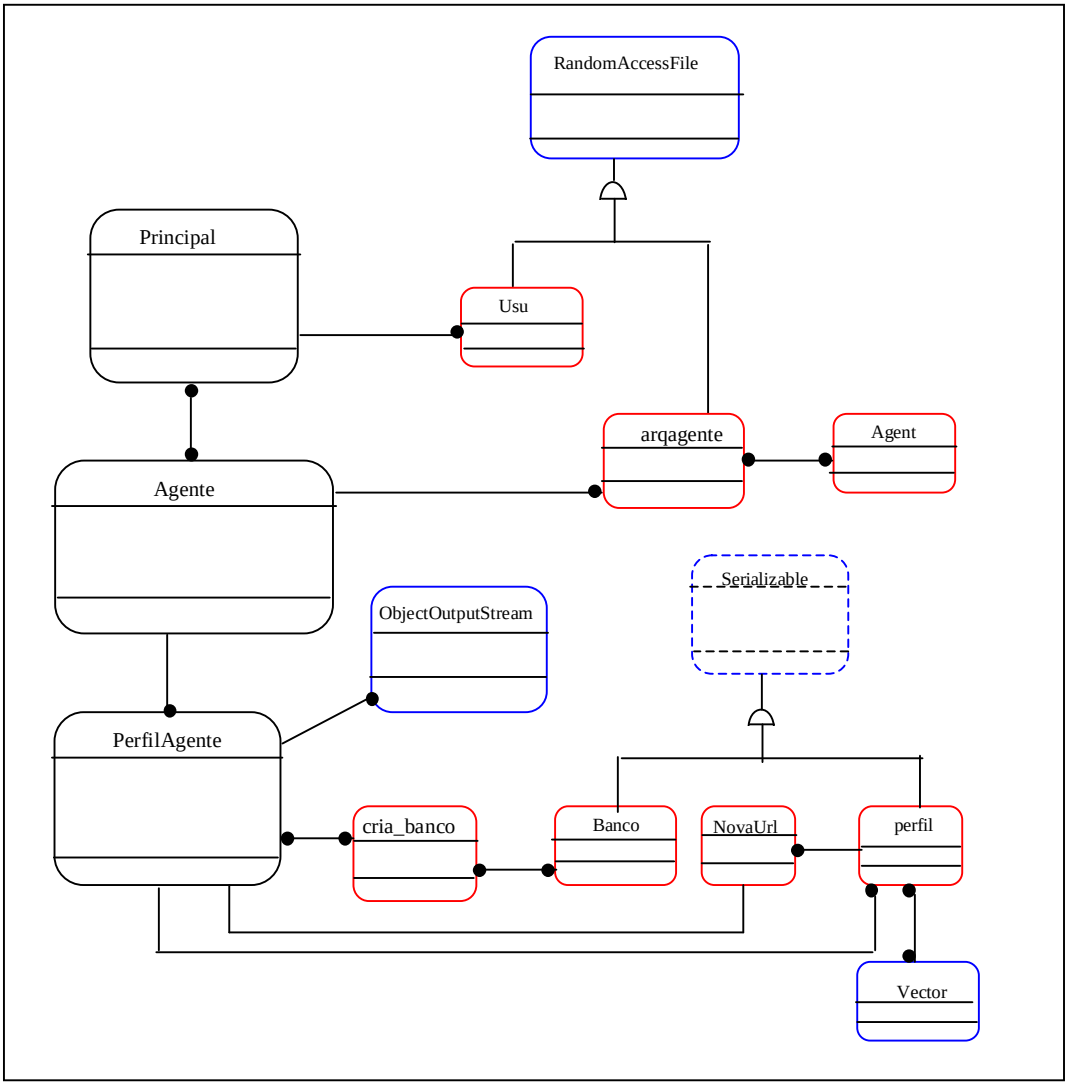


Diagrama de Classes do Módulo de Banco de Dados.

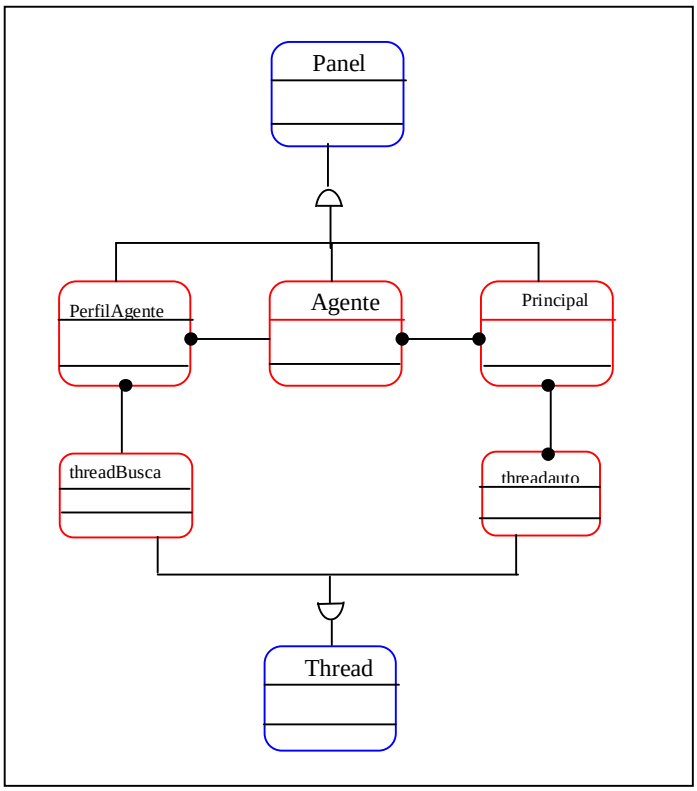
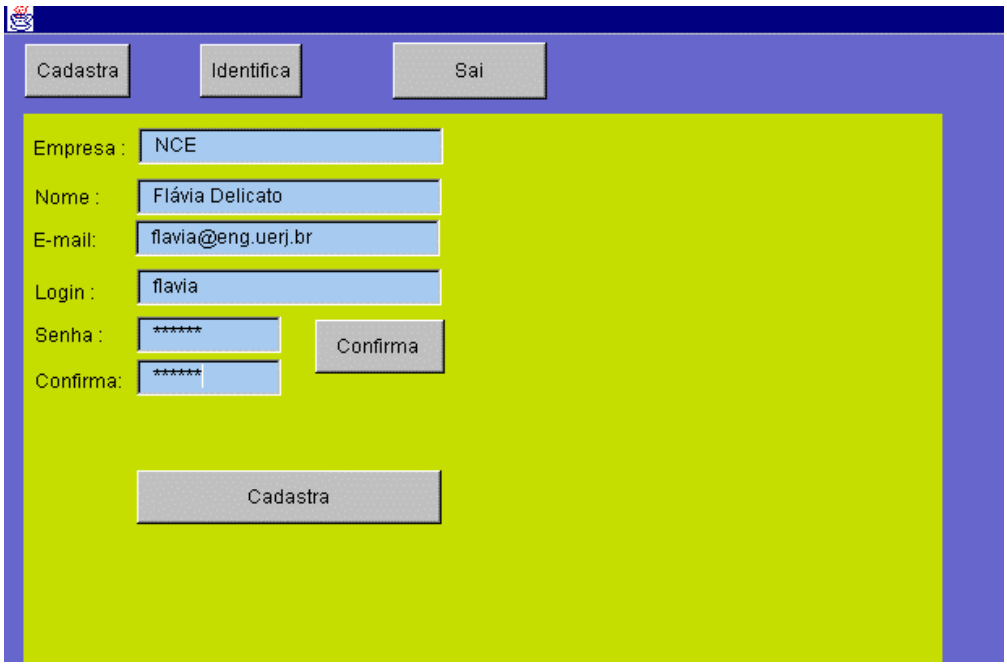
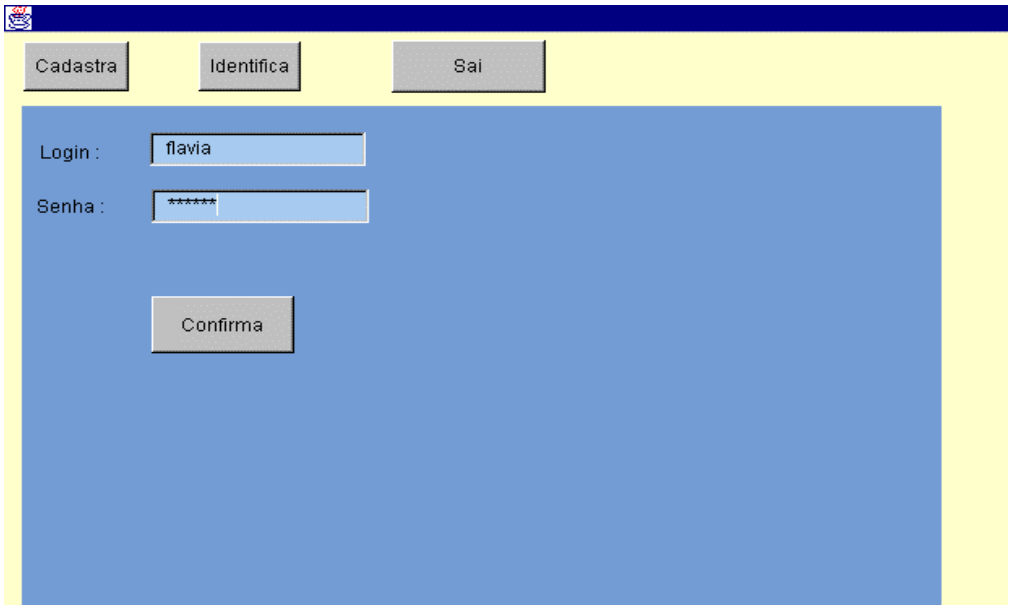


Diagrama de Classes do Módulo Controlador.

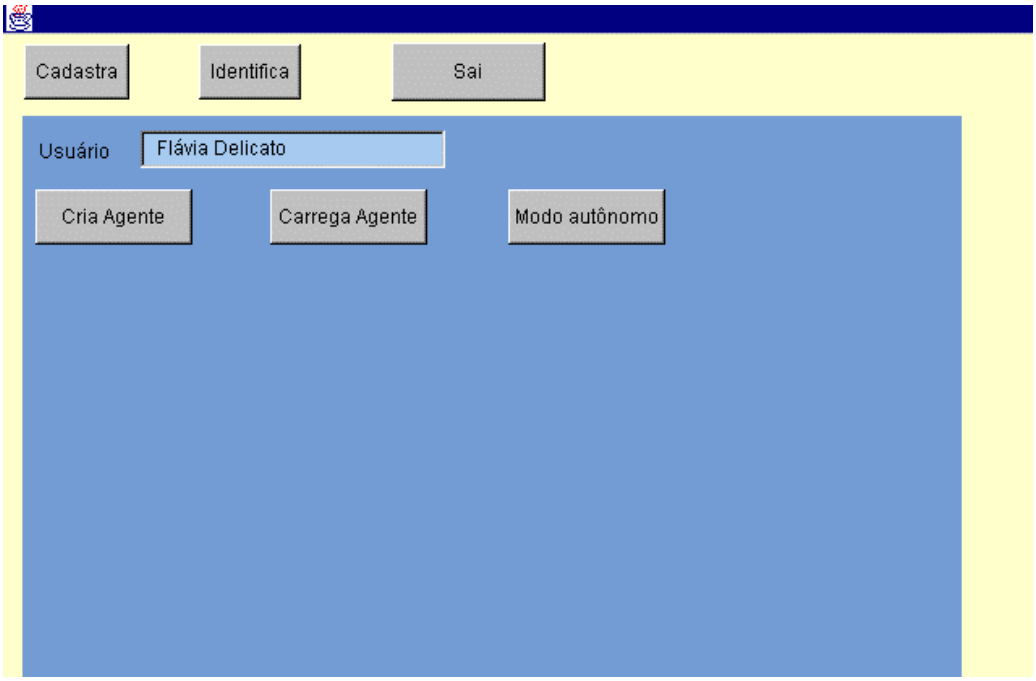
ANEXO 2 - TELAS DO SISTEMA



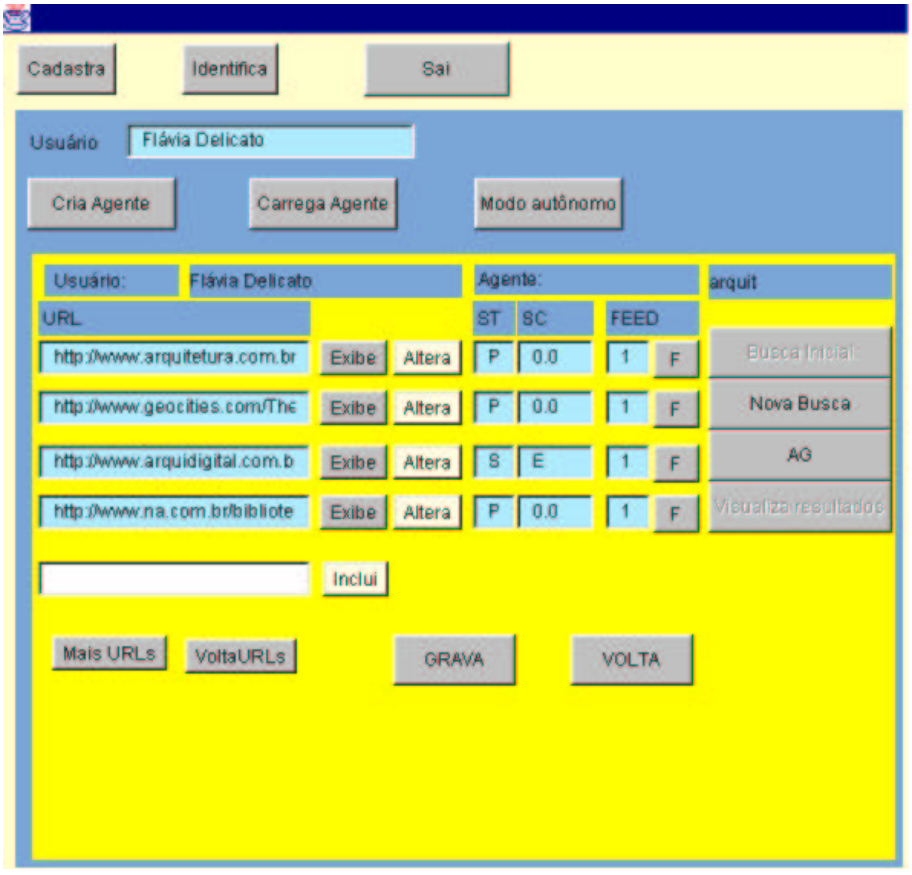
Tela exibida pela classe *Usando* para o cadastramento de novos usuários.



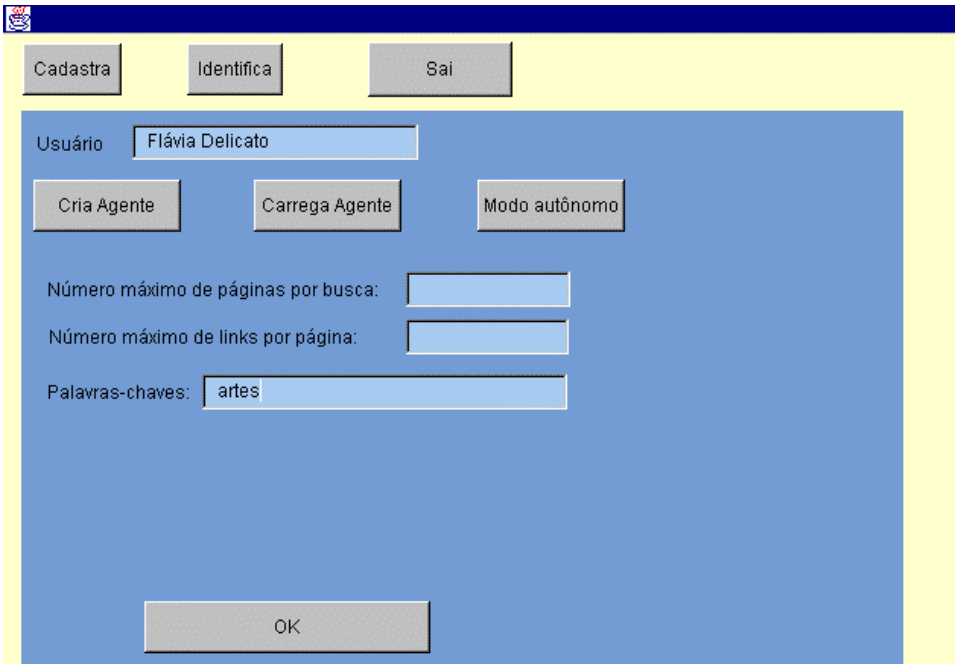
Tela exibida pela classe *Identificando* para o acesso ao sistema de usuários cadastrados.



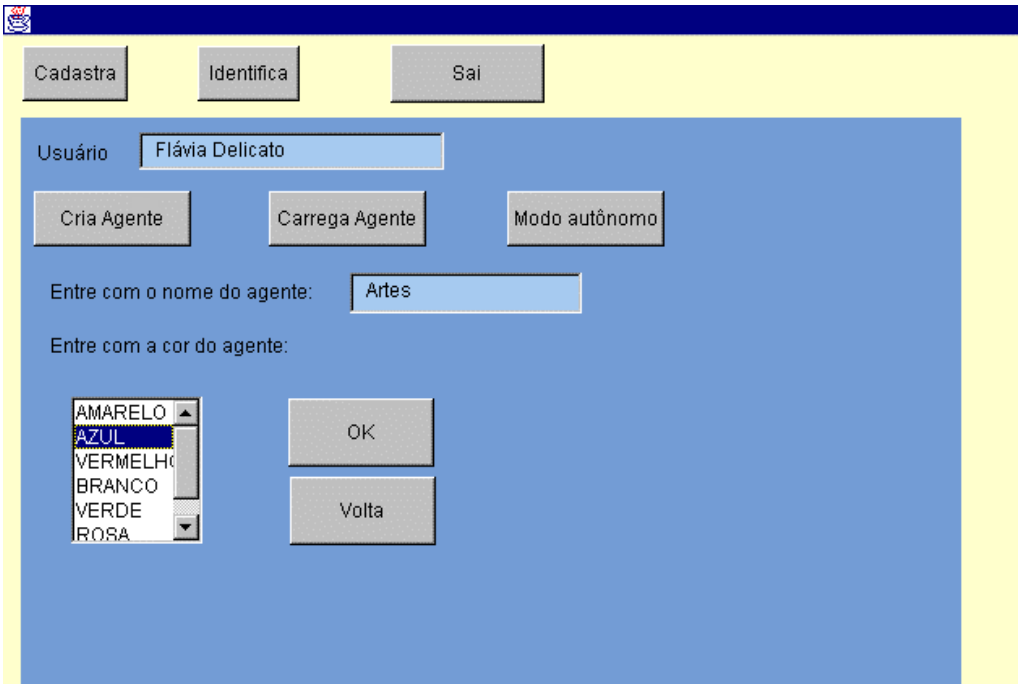
Tela a partir da qual o usuário pode criar, carregar um agente ou ativar o modo autônomo do sistema.



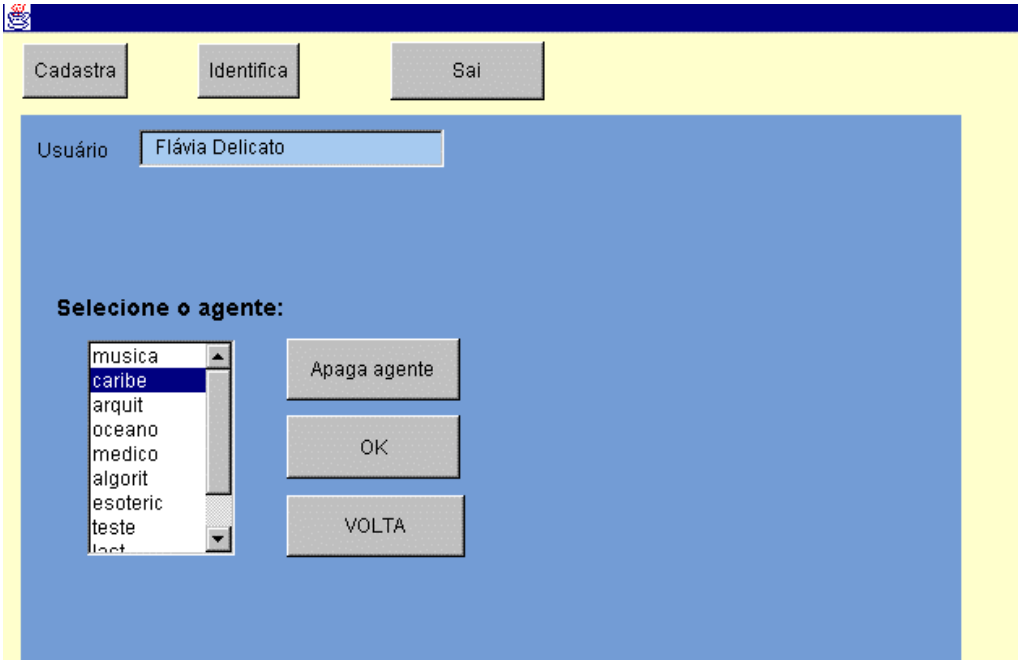
Tela exibida pela classe *Apresenta* para a apresentação dos documentos referentes a um agente.



Tela exibida pela classe *Busca* para a configuração dos parâmetros de busca.



Tela exibida pela classe *Agente* para a criação de um novo agente.



Tela exibida pela classe *Agente* para carregar um agente já existente.

ANEXO 3 - DESCRIÇÃO DAS CLASSES COMPONENTES DO SISTEMA FENIX

I. Classe Identificando

I.1. Objetivos

A classe Identificando tem a função de exibir um painel para a entrada dos dados de identificação de um usuário já cadastrado. Esses dados são seu *login* e sua senha, os quais são validados através de consulta ao arquivo de usuários do sistema. Caso um usuário não cadastrado tente acessar o sistema, será emitida uma mensagem de erro, e o acesso não será permitido.

I.2. Principais Métodos

- Identificando: método construtor, sem parâmetros, que cria o painel e adiciona todos os componentes gráficos.
- getLogin: retorna a *string* de texto do campo *login*.
- getSenha: retorna a *string* de texto do campo senha.
- termina: esconde a classe, tornando o painel invisível.
- publica: exibe a classe, tornando o painel visível.

II. Classe Usando

II.1. Objetivos

A classe Usando tem a função de exibir um painel para a entrada dos dados de um novo usuário. Usuários que desejem acessar ao sistema devem se cadastrar, informando seu nome, instituição, endereço eletrônico, e selecionando um *login* e uma senha, que serão usados para sua identificação e acessos futuros.

II.2. Principais Métodos

- getEmail: retorna o valor da *string* do campo *e-mail*.
- getLogin: retorna o valor da *string* do campo *login*.
- getNome: retorna o valor da *string* do campo nome.
- getSenha: retorna o valor da *string* do campo senha.
- getConfirma: retorna o valor da *string* do campo de confirmação da senha.

- getEmpresa: retorna o valor da *string* do campo empresa.
- limpa: limpa todos os campos de dados.
- publica: exibe a classe, tornando o painel visível.
- termina: esconde a classe, tornando o painel invisível.
- usuOk: exibe mensagem de que o usuário foi cadastrado, confirmando a operação de entrada dos dados.
- usuErro: exibe mensagem de que já existe um usuário cadastrado com os mesmos dados de identificação.

III. Classe Apresenta

III.1. Objetivos

A classe Apresenta tem a função de exibir o painel com todos os dados referentes aos documentos de um agente. Esses documentos podem ser os que foram recuperados em uma sessão de busca ou os que estão gravados nos arquivos de perfil do agente. Para cada documento são apresentados campos com o nome de sua URL, seu *status*, valor de realimentação e pontuação atribuídos. Através do painel dessa classe, o usuário pode solicitar a visualização do documento no navegador local, fornecer realimentação, e disparar tarefas de busca.

III.2. Principais Métodos

- Apresenta: método construtor que cria os componentes gráficos da classe e os adiciona ao painel de apresentação.
- configurlfs: configura os valores dos campos de texto com o nome da Url, realimentação, pontuação e *status*.
- setLogin: configura o *login* do usuário ativo.
- setPalavras: configura os palavras-chaves usadas pelo agente no módulo de busca.
- setFlag: configura uma variável lógica que indica se uma sessão corrente de busca é a busca inicial ou não para um agente.
- mostra: exibe o painel da classe.
- esconde: esconde o painel da classe.
- setNomeUsu: configura o nome do usuário ativo.
- processa_url: constrói os vetores de termos e pesos de um documentos, através da criação de instâncias das classes responsáveis.

- cria_arq_perfil: responsável pela criação dos arquivos de perfil iniciais de um agente. Chama o método processa_url ( ) para cada um dos documentos recuperados na busca inicial, e grava os vetores de termos e pesos gerados no arquivo de perfil, juntamente com a identificação de cada documento,
- ordena: ordena todas as URLs de um agente pelo nome do arquivo de perfil correspondente, a fim de serem passadas para a classe Ajuste.
- ajuste: prepara os dados para o ajuste de um perfil. Passa todos os documentos obtidos por um arquivo de perfil, com seus respectivos valores de pontuação, *status* e realimentação, para a classe Ajuste.

IV. Classe Agente

IV.1. Objetivos

Classe controladora de todos os agentes de um usuário. Permite que o usuário crie novos agentes ou carregue agentes já existentes, através de painéis para a criação e carregamento. Oferece, ainda, a opção de ativar o modo de funcionamento autônomo do sistema.

IV.2. Principais Métodos

- Agente: método construtor da classe. Cria os painéis, adicionando todos os componentes gráficos, e exibe o painel apresenta, mantendo os demais invisíveis.
- setCor: configura a variável cor, que representa a cor de fundo de um agente.
- setAuto: chama o método setAuto da classe PerfilAgente, indicando que foi acionado o modo de execução autônomo.
- setLogin: configura a variável *login*.
- getNomeAgente: retorna o nome de um novo agente.
- getNomeListAgente: retorna o nome de um agente selecionado para carregamento.
- setUsu: configura o nome do agente, exibindo-o no campo usutxt.
- carrega_agente: obtêm a lista de todos os agentes de um usuário, e exibe-os para seleção no painel carregando.
- exibe_perfil: apresenta todos os dados de um agente selecionado, através da chamada de métodos da classe PerfilAgente.
- le: abre e lê o conteúdo de um arquivo de perfil de um agente, carregando um vetor com os nomes das urls de seus documentos.

- cria_agente: exibe o painel para criação de agente.
- busca_inicial: chama o método da classe PerfilAgente, para exibir o painel de configuração dos parâmetros de busca.
- publica: torna os painéis da classe visíveis.
- esconde: torna os painéis da classe invisíveis.

V. Classe Principal

V.1. Objetivos

Principal classe controladora. Controla o sistema como um todo. Exibe os painéis para cadastramento e acesso ao sistema.

V.2. Principais Métodos

- Principal: método construtor da classe. Cria todas as variáveis e exibe os botões e o painel de apresentação do sistema.
- modo_auto: dispara a execução de uma tarefa autônoma do sistema.
- cadastra_usu: exibe a classe para cadastramento de um novo usuário.
- confirma_usu: efetiva o cadastramento dos dados de um novo usuário.
- efetiva: confirma os dados de um usuário cadastrado, permitindo o seu acesso ao sistema.
- main: método principal, a partir do qual é iniciada a execução do aplicativo que corresponde ao sistema Fenix.

VI. Classe PerfilAgente

VI.1. Objetivos

Classe controladora de um agente específico de um usuário. Permite a apresentação de seus dados e a ativação de sessões de busca e filtragem de documentos.

VI.2. Principais Métodos

- setLogin: configura o *login* do usuário ativo.
- setNome: configura o nome do usuário ativo.
- setCor: configura o valor da cor de fundo do agente ativo.
- setAuto: configura o modo de execução para autônomo.
- inclui_url: permite a inclusão a mão pelo usuário de uma nova url na lista dos documentos recuperados em uma busca.

- esconde: esconde a classe, tornando invisível seu painel principal.
- mostra: exibe a classe, tornando visível seu painel principal.
- busca_ini: exibe a classe Busca, para a entrada dos parâmetros de busca. Configura a variável flag_busca_ini para true.
- busca_nova: obtêm todos os nomes dos arquivos de perfil do agentes ativo, e as palavras-chaves do agente. Exibe o painel da classe Busca, com o campo de palavras-chaves já preenchido, e configura a variável flag_busca_ini para false.
- ok_busca: verifica o valor corrente da variável flag_busca_ini. Caso seja *true*, inicia uma nova busca; caso contrário, inicia uma tarefa de busca para cada arquivo de perfil do agente, e calcula a similaridade dos documentos recuperados em relação ao respectivo perfil.
- configura_exibe: configura os dados dos documentos obtidos nas buscas que serão passados para a classe Apresenta, a fim de serem exibidos para o usuário.
- exibe_urls: chama os métodos da classe Apresenta para limpar a tela e apresentar os documentos.
- exibe_browser: chama o navegador configurado no sistema para exibir a página de um documento.

VII. Classe Ajuste

VII.1. Objetivos

Essa classe é responsável pela implementação do método de aprendizado baseado na realimentação por relevância. Ela atualiza o perfil de um agente com base na realimentação fornecida pelo usuário, aplicando a fórmula de ajuste vetorial adotada no trabalho.

VII.2. Principais Métodos

- Ajuste: método construtor da classe. Recebe uma referência para a classe Apresenta.
- setArqPerf: configura os nomes dos arquivos de perfil e de busca.
- setDados: configura os dados das urls provenientes da busca.
- le_arq: lê um arquivo de perfil.
- processa_url: processa cada url de um perfil, obtendo seus vetores de termos e pesos.
- ajuste: faz o ajuste vetorial do perfil, modificando pesos e adicionando termos, com base nos vetores d termos e pesos das urls obtidas na busca.

VIII. Classe Busca

VIII.1. Objetivos

Exibe um painel para a apresentação e configuração dos parâmetros de busca.

VIII.2. Principais Métodos

- Busca: método construtor, que cria o painel e adiciona os componentes gráficos.
- mostra: exibe a classe, tornando o painel visível.
- esconde: esconde a classe, tornando o painel invisível.
- getPalavras: retorna as palavras-chaves do campo palavratxt.
- setPalavras: configura as palavras-chaves do campo palavratxt.

IX. Classe Banco

IX.1. Objetivos

Representa o banco de dados local das páginas recuperadas.

IX.2. Principais Métodos

- Banco: método construtor, que recebe os parâmetros para configurar todos os campos de dados da classe.

X. Classe threadBusca

X.1. Objetivos

Classe derivada da classe Thread do pacote java.lang. Representa uma tarefa do sistema operacional. Responsável por realizar uma sessão de busca na *Web*, através da interação com o mecanismo de busca configurado, e retornar os documentos obtidos.

X.2. Principais Métodos

- threadBusca: método construtor da classe. Recebe uma referência ao objeto PerfilAgente que o invocou.
- run: método que sobregrava o método run ( ) da classe Thread de Java. Dispara a execução da tarefa.

XI. Classe perfil

XI.1. Objetivos

Representa um objeto perfil de um agente. Implementa a interface Serializable de Java, que permite a manipulação de arquivos de objetos. Cada instância dessa classe será armazenada em um arquivo de objetos.

XI.2. Principais Métodos

- perfil: método construtor que recebe como parâmetros valores para os termos, pesos e urls do perfil.
- setNome: configura os nomes dos arquivos de perfil e busca.
- getNome: retorna os nomes dos arquivos de perfil e busca.
- setUrl: configura uma url do vetor de urls do perfil.
- setTermos: configura os termos do vetor de termos do perfil.
- setPesos: configura os pesos do vetor de pesos do perfil.

XII. Classe NovaUrl

XII.1. Objetivos

Representa um objeto url do sistema, ou seja, todas as informações sobre um documento recuperado e/ou armazenado. Implementa a interface Serializable de Java.

XII.2. Principais Métodos

- NovaUrl: método construtor sem parâmetros.
- NovaUrl: método construtor que recebe como parâmetros valores para todos os campos de dados da classe.

XIII. Classe arqagente

XIII.1. Objetivos

Faz a interface com o arquivo de agentes, lendo e manipulando seus dados.

XIII.2. Principais Métodos

- arqagente: método construtor sem parâmetros.
- arqagente: método construtor que recebe como parâmetros o *login* do usuário, o nome do agente, sua cor de fundo e as palavras-chaves.
- getArquivo: retorna um vetor com os nomes de todos os arquivos de perfil de um agente.

- getAgent: retorna um vetor com os nomes de todos os agentes de um usuário.
- consulta: lê o arquivo de agentes e procura todos os nomes de agentes e de arquivos de perfil de um dado usuário.
- apaga: apaga os dados relativos a um agente.
- consulta2: lê o arquivo de agentes e procura todos os nomes de arquivos de perfil, sendo dados um determinado agente e um usuário.
- getCor: retorna a cor do agente, obtida no arquivo de agentes.
- getPalavras: retorna as palavras-chaves do agente, obtidas no arquivo de agentes.

XIV. Classe buscaWeb

XIV.1. Objetivos

Essa classe é criada pela classe threadBusca, e é responsável pela interação com os mecanismos de busca. Ela chama o mecanismo selecionado passando as palavras-chaves do usuário, recebe o fluxo de dados retornado pelo mecanismo, e envia esse fluxo para ser tratado pela classe extraihref.

XIV.2. Principais Métodos

- buscaWeb: método construtor da classe, que recebe como parâmetros as palavras-chaves, o nome do arquivo de busca a ser criado, e uma *string* que indica qual o mecanismo de busca a ser chamado.

XV. Classe HTTP

XV.1. Objetivos

Essa classe possui métodos que permitem abaixar páginas *Web*. A partir de um objeto URL, retorna seu conteúdo na forma de um fluxo de bytes, o qual é convertido para uma *string* única.

XV.2. Principais Métodos

- HTTP: método construtor da classe.
- downloadWWWPage: método que, dado um objeto URL válido, retorna a página na forma de uma *string* única.
- getWWWPageStream: método que, sendo dados um nome de *host* e de arquivo, conecta-se a um servidor *Web* e requisita o documento do arquivo, lê a partir da

porta de comunicação configurada, e retorna um fluxo de dados com o documento requisitado.

XVI. Classe PageSaver

XVI.1. Objetivos

Essa classe tem o objetivo de interagir com os métodos da classe HTTP, para obter um documento a partir de um objeto URL.

XVI.2. Principais Métodos

- PageSaver: método construtor da classe.
- retornaPg: retorna a página como *string*.
- convertTag: obtêm uma tag html do documento da URL e, se for um link relativo, converte para um link absoluto e o retorna.
- loadPage: chama o método downloadWWWPage da classe HTTP, passando uma URL válida.
- readTag: método chamado sempre que um sinal "<", que representa uma *tag* HTML, é encontrado no fluxo de dados. Ele é responsável por ler o restante da *tag*.

XVII. Classe extraihref

XVII.1. Objetivos

Processa a *string* retornada em uma sessão de busca, extraindo todos os links encontrados.

XVII.2. Principais Métodos

- extraihref: método construtor sem parâmetros.
- searcher: procura uma *tag* <a href> e retorna o link referido pela *tag*, na variável name. Essa variável é passada para a classe cria_banco, a qual faz a interface com o banco de dados local, criando o arquivo de urls recuperadas.

XVIII. Classe cria_banco

XVIII.1. Objetivos

Faz a interface entre a classe controladora PerfilAgente e as operações de acesso aos dados da classe Banco.

XVIII.2. Principais Métodos

- cria_banco: inicializa o objeto da classe Vector vb.

  set_banco: recebe uma *string* com o nome da url a ser gravada no banco, e cria um objeto da classe URL de Java a partir desse nome. Cria uma instância da classe Banco com o objeto URL, seu nome e a data de gravação. A seguir adiciona a instância criada como um elemento do vetor vb.
- grava_banco: grava o objeto Vector com todas as instâncias da classe Banco criadas em um arquivo de objetos.
- criarq: Cria ou abre o arquivo de objetos para a gravação das urls. O nome do arquivo é um parâmetro recebido pelo método.
- le: Lê um arquivo de objetos. O nome do arquivo é passado como parâmetro do método.

XIX. Classe Usu

XIX.1. Objetivos

Representa um usuário do sistema, possuindo campos com seus dados e métodos para manipulá-los.

XIX.2. Principais Métodos

- Usu: método construtor da classe, que recebe como parâmetros valores para todos os campos da classe.
- setNome: configura o campo nome.
- testaUsu: verifica se determinados valores de *login* e senha já existem no arquivo de usuários do sistema, retornando o valor lógico *true* caso existam, e *false* caso contrário.
- adiciona: grava os dados de um usuário no arquivo de usuários do sistema.

XX. Classe CalculaSimilaridade

XX.1. Objetivos

Essa classe é responsável pelo cálculo da similaridade entre dois vetores, segundo a fórmula adotada no trabalho. O primeiro vetor corresponde aos pesos dos termos de um perfil de agente, e o segundo aos pesos dos termos de um documento obtido em uma busca, cujo valor de similaridade em relação ao perfil se deseja conhecer.

XX.2. Principais Métodos

- CalculaSimilaridade: construtor da classe, que recebe parâmetros para configurar os vetores e seus tamanhos.
- normaliza: método responsável por fazer a normalização dos dois vetores.
- calcula: método responsável pelo cálculo da similaridade, baseado no fórmula adotada no trabalho.
- ordena: ordena os vetores de termos e pesos do perfil, colocando-os em ordem decrescente de peso.

XXI. Classe CalculaPeso

XXI.1. Objetivos

Classe responsável por calcular o peso dos termos de um perfil (ou documento), baseada nos valores de frequência de termos e inverso da frequência de documento. Ela gera o vetor de pesos de um perfil (ou documento).

XXI.2. Principais Métodos

- CalculaPeso: construtor da classe, que recebe como parâmetros os valores de VetIdf e VetCont.
- getVetor: retorna o vetor de pesos calculado.

XXII. Classe CalculaIdf

XXII.1. Objetivo

Calcula o valor da frequência de documento para um perfil (ou documento), usando a fórmula normalizada adotada no trabalho.

XXII.2. Principais Métodos

- CalculaIdf: construtor da classe.
- setColeção: configura o valor do tamanho da coleção (N).
- idf: configura o vetor para o qual se deseja calcular idf.
- calcula: calcula o valor de idf dos termos do um vetor.

XXIII. Classe ConstroiVetor

XXIII.1. Objetivos

Gera o vetor de termos de um documento.

XXIII.2. Principais Métodos

- ConstroiVetor: construtor da classe, que recebe como parâmetro o documento na forma de uma *string* única.
- ExtraiTag: extrai as palavras do documento que foram identificadas como *tags* HTML.
- VerificaLista: verifica se um dado termo do documento existe em uma lista de palavras funcionais ("*stop list*").
- VerificaOcorrência: verifica se um dado termo do documento já existe no vetor de termos.
- getVetor: retorna o vetor de termos gerado.

XXIV. Classe Agent

XXIV.1. Objetivos

Representa um objeto agente do sistema, o qual será manipulado pela classe arqagente. Implementa métodos para a leitura e escrita dos dados de um agente no arquivo correspondente.

XXIV.2. Principais Métodos

- Agent: método construtor responsável pela inicialização de todos os campos de dados não estáticos da classe.
- readData: método que efetua a leitura do arquivo de agentes na forma de um registro de *strings* de tamanho fixo.
- writeData: método que grava o arquivo de agentes na forma de um registro de *strings* de tamanho fixo.

XXV. Classe threadauto

XXV.1. Objetivos

Classe que controla o modo de execução autônoma do agente.

XXV.2. Principais Métodos

- threadauto: método construtor da classe. Recebe uma referência para a classe Principal e o *login* do usuário ativo como parâmetros.
- run: método que lê o arquivo de agentes do sistema para obter os nomes de todos os agentes de um usuário, e de seus arquivos de perfil.
- lê: para cada perfil de um agente, lê todas as suas urls e carrega em um vetor. Verifica se há urls não lidas para o agente. Caso haja, emite mensagem informando o usuário. Caso todas as urls tenham sido lidas, chama o método metodo_busca.
- metodo_busca: dispara uma tarefa de busca para cada agente cujos documentos foram todos lidos pelo usuário. Recebe e filtra o resultado da busca, através de chamadas aos métodos das classes necessárias nos módulos de busca e filtragem. Informa ao usuário que há novas urls para serem visualizadas no agente.

As classes descritas a seguir pertencem ao sub-módulo de algoritmo genético do módulo de aprendizado. Embora não tenham sido implementadas, as classes foram detalhadamente especificadas durante a fase de projeto, de forma a permitir sua implementação em trabalhos futuros.

XXVI. Classe Cromossomo

XXVI.1. Objetivos

Representa um indivíduo, ou cromossomo da população, ou seja, um vetor binário indicando a presença ou ausência de termos do vetor de termos da população no vetor de termos do perfil, e a respectiva aptidão desse indivíduo na população.

XXVI.2. Principais Métodos

- Cromossomo: Construtor da classe, que recebe como parâmetros o nome de um arquivo de perfil, o vetor de termos do perfil, seu tamanho, as palavras-chaves do respectivo agente e a aptidão média da população.
- setGene: configura um gene do cromossomo, ou seja, uma posição do vetor binário.
- setNovotermo: Configura um termo do vetor de termos novos gerado pelo AG.
- set_pesos: Configura os pesos dos termos.

XXVII. Classe Populacao

XXVII.1. Objetivos

Representa a população de perfis usada para a implementação do módulo de algoritmo genético.

XXVII.2. Principais Métodos

- Populacao: método construtor da classe. Recebe como parâmetro uma referência para a classe controladora PerfilAgente.
- setTam: configura o tamanho da população, e inicializa o vetor de objetos da classe Cromossomo.
- setTermos: Configura o vetor de termos da população.
- setCromossomo: Configura um cromossomo da população, passando os valores de nome do arquivo de perfil, vetor de termos do perfil, aptidão média da população e palavras-chaves do agente correspondente. Verifica se cada elemento do vetor de termos da população está presente no vetor de termos do perfil, e configura o valor do gene correspondente no cromossomo para 0 ou 1, dependendo da presença ou ausência do termo.
- crossover: Método responsável pela operação de *crossover* entre 2 indivíduos da população. Inicialmente, ordena os cromossomos por sua aptidão. A seguir, seleciona 2 valores aleatórios como os pontos do *crossover*. Gera 2 vetores filhos novos a partir da recombinação dos elementos dos vetores pais.
- nova_geracao: Configura os novos termos da população, gerados após o *crossover*.
- evaluate: Método que faz a avaliação dos novos indivíduos da população, após as operações de *crossover* e mutação. Aqui, cada novo cromossomo faz uma busca, calcula a similaridade do vetor de termos novo em relação ao perfil, e os resultados são apresentados. Se a aptidão média da população aumentar, os resultados são salvos. Caso contrário são descartados, e mantém-se o vetor original.
- ordena_cromossomos: Ordena os cromossomos de uma população por seus respectivos valores de aptidão.
- mutacao: Seleciona aleatoriamente, usando a taxa de mutação, alguns elementos do vetor binário que representa um indivíduo, e troca seu valor de 0 para 1 ou vice-versa.
- cria_vet_peso: método que cria o vetor de pesos a partir dos termos de um indivíduo. Para isso, usa instâncias das classes que possuem as funções de calcular

tf, idf e calcular peso. A coleção considerada para os cálculos de idf é a população inteira.

- buscas: método responsável por iniciar novas buscas a fim de avaliar as aptidões dos novos indivíduos gerados através da reprodução e recombinação.

ANEXO 4 - ENDEREÇOS DAS FERRAMENTAS DE BUSCA UTILIZADAS PELO SISTEMA

ALTAVISTA - Disponível em: http://www.altavista.digital.com

LYCOS - Disponível em: http://www.lycos.com

CADÊ - Disponível em: http://www.cade.com.br

MINER - Disponível em: http://miner.uol.com.br

Referências Bibliográficas

ACKLEY, D., LITTMAN, M. “Interactions between Learning and Evolution”. **Artificial Life II**, v X, pp. 487-509. Edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen, Addison Wesley, 1992.

ALLAN, J. “Relevance feedback with too much data”. In anais do congresso **18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**. Seattle, WA USA. July, 1995.

ANUFF, Ed. **Java Sourcebook**. Whiley Computer Publishing, 1996.

ARMSTRONG, R., *et al*. “WebWatcher: A learning apprentice for the World Wide Web”. In anais do congresso **AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Resources**. Stanford, March 1995.

ASNICAR, F., TASSO, C. “ifWeb: a prototype of a user model-based intelligent agent for filtering and navigation in the world-wide web”. In UM97 Workshop on Adaptive Systems and User Modelling on the World Wide Web, **6 th International Conference on User Modelling**, Chia Laguna, Sardinia, Italy, June 1997.

BAEZA-YATES, R., RIBEIRO-NETO, B. **Modern Information Retrieval**. Addison Wesley – ACM Press. 513p. 1999.

BALABANOVIC, M. An Adaptive Web Page Recomendation Service. Stanford Universal Digital Libraries Project Working Papers SIDL - WP. 1997.

_____. Learning to Surf: Multiagent Systems for Adaptive Web Page Recomendation Service**. Tese (D.Sc) submetida ao Department of Computer Science and the Committee on Graduate Studies of Stanford University. UMI Number: 9837173. UMI Company. Stanford, 1998.**

BALABANOVIC, M., SHOHAM, Y. “Fab: Content-based, collaborative recommendation”. **Communications of the ACM**, 40 (3): 66-72, March 1997.

BARTO, A. G. **Cellular Automata as Models of Natural Systems**. The University of Michigan, 1975.

BELGRAVE, M. **The Unified Agent Architecture. A White Paper**. Disponível na Internet via WWW. URL: http://www.ee.mcgill.ca/~belmarc/uaa_paper.html, 1995. Arquivo consultado em 1999.

BELKIN, N.J., CROFT, W.B. “Information Filtering and Information Retrieval: Two sides of the same coin ?” **Communications of the Acm**, v. 35 (12):pp. 29-38, December, 1992.

BUCKLEY, C. *et al*.. “Automatic query expansion using SMART: TREC-3”. In anais do congresso **3rd Text Retrieval Conference**. Gaithersburg, Maryland, USA. November, 1994.

BURBECK, S. **Application Programming in Smalltalk-80 (TM) : How to use Model-View-Controller (MVC).** Disponível na Internet via WWW. URL: http://burks.bton.ac.uk/ burks/ language/smaltalk/mvc.htm, 1987. Arquivo consultado em 1999.

CAGLAYAN, A., HARRISON, C**. Agent Sourcebook - A Computer Guide to Desktop, Internet, and Intranet Agents**. Wiley Computer Publishing, 349p. 1997.

CHEN, H.; KIM, J. “GANNET: information retrieval using genetic algorithms and neural networks”. Center for Management of Information, College of Business and Public Administration, University of Arizona, Working Paper, CMI-WPS, 1993.

CROFT, W.B., HARPER, D.J. “Using Probabilistic Models of Document Retrieval Without Relevance Information”. Journal of Documentation, 35 (4), 285-95. 1979.

FARMER, J.D, TOFFOLI, T., WOLFRAM, S. “*Cellular Automata*”. In anais do congresso An Interdisciplinary Workshop at Los Alamos. New Mexico, North-

Holland, Amsterdam, March 7-11 1983.

FISCHER, G., STEVENS, C. “Information access in complex, poorly structured information spaces”. In anais do congresso **Human Factors in Computing Systems CHI'91 Conference**, 1991, pp. 63-70.

FOLTZ, P. W., DUMAIS, S. T. “Personalized Information Delivery: An Analysis of information filtering methods”. **Communications of ACM.** v. 35 (12), pp.29-38, December 1992.

FRAKES, W. B., BAEZA-YATES, R. **Information Retrieval – Data Structures & Algorithms**. Prentice Hall PTR, New Jersey, 1992.

GILBERT, D., APARICIO; M. A. Intelligent Agent Strategic**. Disponível na Internet via WWW. URL: http://activist.gpl.ibm.com:81/WhitePaper/ptc2.html, 1996. Arquivo consultado em 1998.**

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. Addison-Wesley, Reading, MA, 1989.

_____ “Genetic and Evolutionary Algorithms come of age”. **Communications of the ACM**, 37(3):113-119, March, 1994.

GORDON, M. “Probabilistic and genetic algorithms for document retrieval”. **Communications of the ACM**, n.31(10); p.1208-1218, October, 1988.

GRAESSER, A., FRANKLIN, S.. “Is it an agent, or just a program?: A Taxonomy for Autonomous agents”. In anais do congresso **Third International Workshop on Agent Theories, Architectures and Languages**, Springer-Verlag, 1996. (Também disponível na Internet via WWW. URL: http://zebra.msci.memphis.edu/~franklin/AgentProg.html). Arquivo consultado em 1998.)

GURALMILE, D. (.ed). **Webster's new World Dictionary of the American Language**, Second College Edition. New York: The World Publishing Company. 1970.

HARMAN, D. *et al*.. "Inverted Files". In W. Frakes and R.Baeza-Yates, editors, **Information Retrieval: Algorithms and Data Structures**, chapter 3, pages 28-43. Prentice Hall, Englewood Cliffs, NJ, USA, 1992.

HARMAN, D. "Overview of the Third Text Retrieval Conference (TREC-3)". In anais do congresso **3rd Text Retrieval Conference**. Gaithersburg, Maryland, USA. November, 1994.

HARMAN, D. "Overview of the fifth Text Retrieval Conference (TREC-5)". In anais do congresso **5th Text Retrieval Conference**, Gaithersburg, Maryland, USA. November, 1996.

HINTON , G.E., NOWLAN, S.J. "How Learning can Guide Evolution", **Complex Systems**, v 1, pp. 495-502, 1987.

HOLLAND, J.H. **Adaptation in Natural and Artificial Systems.** University of Michigan Press, Ann Arbor, 1975.

_____. "Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems". In R.S. Mishalski, J.G. Carbonell, and T.M. Mitchell, editors, **Machine Learning 2**, pp. 593-623. Kaufman, 1986.

IDE, E. "New experiments in relevance feedback"**.** In **The Smart Retrieval System - Experiments in automatic Document Processing**, pp. 337-354, Englewood Cliffs: Prentice-Hall, 1971.

INGWERSEN, P. **Information Retrieval Interaction**. Taylor Graham Press, 1992.

KAHLE, B. **Archiving the Internet**. Disponível na Internet via WWW. URL:http://www.alexa.com/~brewster/essays/_sciam_article.html, 1997. Arquivo consultado em 1998.

KETCHPEL S. P., GENESERETH, M. R.. "Software Agents". **Communications of the ACM.** V 37, Number 7, July 1994. (Também disponível na Internet via WWW. URL: http://logic.stanford.edu/sharing/papers/agents.ps. Arquivo consultado em 1998.)

KJERSTI, A.. **A Survey on Personalised Information Filtering Systems for the World Wide Web**. Dezembro 1997. Disponível na Internet via WWW. URL:http://www.nr.no/home/kjersti/agent.html. Arquivo consultado em 1999.

YANG, J., KORFHAGE, R.R. **"**Effects of query term weights modification in document retrieval: a study based on a genetic algorithm". In anais do congresso **Second Annual Symposium on Document Analysis and Information Retrieva**l, p. 271-285, Las Vegas, April, 1993.

KOZA, J.R. **Genetic programming: on the programming of computers by means of natural selection**. Cambridge: The MIT Press, 1992.

LANG, K. "NewsWeeder: Learning to filter netnews". In anais do congresso **12th International Conference on Machine Learning**. Tahoe City, California, July, 1995

LESK, M. E., SALTON, G. "Relevance assessments and retrieval system evaluation". In **The Smart System - Experiments in Automatic Document Processing**. Prentice Hall Inc. pp. 506-527. 1971.

MAES, P. **Agents that Reduce Work and Information Overload**. Disponível na Internet via WWW. URL: http://pattie.www.media.mit.edu/people/pattie/CACM-94/CACM-4. P1.html, 1994. Arquivo consultado em 1998.

MAES, P. “Artificial Life Meets Entertainment: Lifelike Autonomous Agents”. **Communications of ACM**, Special Issue on New Horizons of Commercial and Industrial AI, v. 38, No. 11, pp. 108, November, 1995 .

MCCORDUCK, P. **Machines Who Think**. Freeman Press., San Francisco, 1979.

NEWELL, A. “Physical symbol systems”. **Cognitive Science**, v. 4: pp.135-183, 1980.

NISSEN, M. *et al*.. **Intelligent Agents: a Technology and Business Aplication Analysis.** Disponível na Internet via WWW. URL: http://haas.berkeley.edu/~heilmann/agents. 1995. Arquivo consultado em 1998.

NWANA, H. S.. “Software Agents: an Overview”. **Knowledge Engineering Review** v11, no 3, September, 1996. (Também disponível na Internet via WWW. URL: http://www.cs.umbc.edu/agents/introduction/ao/. Arquivo consultado em 1999.)

OARD, D. W., MARCHIONINI, G. A.. **Conceptual Framework for Text Filtering**. Disponível na Internet via WWW. URL: http://www.glue.umd.edu/~oard/research.html. 1996. Arquivo consultado em 1999.

PAZZANI, M., MURAMATSU, J., BILLSUS, D. “Syskill & Webert: Identifying interesting web sites”. In anais do **congresso AAAI Spring Symposium on Machine Learning in Information Access**, Stanford, May 1996.

PIATETSKY-SHAPIRO, G., FRAWLEY, W.J. **Knowledge Discovery in Databases.** CA:AAAI Press, Menlo Park,1991.

PORTER, M.F. “An algorithm for suffix stripping”. **Program**, v. 14(3): pp. 130-137, 1980. (Também disponível na Internet via WWW. URL:http://www.muscat.com/~martin/stem.html. Arquivo consultado em 2000.)

PRESSMAN, R. S.. **Engenharia de Software**. Makron Books, 1995.

QUATERMAN, J., AUSTIN, T.X.. **Matrix Information and Directory Services, Inc.: Matrix News**, July, 1995. Disponível na Internet via WWW. URL: http://www.Mids.org. Arquivo consultado em 1999.

ROCCHIO, J.J. "Relevance feedback in information retrieval"**.** In **The Smart Retrieval System - Experiments in automatic Document Processing**, pp. 313-323, Englewood Cliffs: Prentice-Hall, 1971.

ROBERTSON, S.E., SPARCK JONES, K. "Relevance weighting of search terms". **Journal of the American Society for Information Sciences**, v. 27(3):pp. 129-146, 1976.

RUMBAUGH, J. *et al.*. **Object-Oriented Modeling and Design**. Prentice Hall, 1991.

SALTON, G. **Automatic Information Organization and Retrieval**, McGraw-Hill Book Co., NY, 1968.

_____. **Automatic Text Processing – The Transformation, Analysis and Retrieval of Information by Computer**. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

SALTON, G., BUCKLEY, C.. "Term-weighting Approaches in Automatic Text Retrieval", **Information Processing and Management**, v. 24 (5), pp. 513-23. 1988.

_____. "Improving Retrieval Performance by Relevance Feedback". **Journal of the American Society for Information Science**, v. 41 (4), pp. 288-97. 1990

SALTON, G., MCGILL, M. J., **Introduction to Modern Information Retrieval.** McGraw-Hill, 1983.

SALTON, G., YANG, C. S. "On the Specification of Term Values in Automatic Indexing". **Journal of Documentation**, v. 29(4), pp. 351-372, December 1973.

SALTON, G., WONG, A., YANG, C.S. “A Vector Space Model for Automatic Indexing”. **Communications of the ACM**, v. 18:11, pp. 613-620, November 1975.

SARACEVIC, T.. “Evaluation of evaluation in information retrieval”. In anais do congresso **18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**, Seattle, Washington, USA, July, 1995.

SCHUTZE, H., HULL, D. A., PEDERSEN, J. O.. “A comparison of classifiers and documents representations for the routing problem”. In anais do congresso **18 th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval**, Seattle, Washington, USA, July, 1995.

SHETH, B.. NEWT: A learning approach to personalized information filtering**. Dissertação (MSc). MIT Department of Electrical Engineering and Computer Science. s.l.:1994. Disponível na Internet via WWW. URL: http://www.cs.bham.ac.uk/~sra/ People/Stu/Sheth/index.html#ALearning. Arquivo consultado em 2000.**

SHETH, B., MAES, P. “Evolving agents for personalized information filtering”. In anais do congresso **9 th IEEE Conference on Artificial Intelligence for Applications**. Orlando, Florida, March, 1993.

SICHMAN, J., DEMAZEAU, Y. “A Model for the Decision Phase of Autonomous Belief Revision in Open Multi-Agent Systems”. **Journal of the Brazilian Computer Society**, 3 (1): pp 40-50, July, 1996.

SHIVAKUMAR, N., GARCIA-MOLINA, H. “Finding near-replicas of documents on the Web”. **Workshop on Web Databases**, Valencia, Spain, March 1998.

SHOHAM, Y. “Agent Oriented Programming”. **Artificial Intelligence**, v.60, 1993.

SPARCK JONES, K. “A Statistical Interpretation of Term Specificity and Its Application in Retrieval”. **Journal of Documentation**, v. 28:1, pp/ 11-21, March 1972.

VAN RIJSBERGEN, C. J. **Information Retrieval**, , Second Edition, Butterworths, London, 1979.

VAN RIJSBERGEN, C. J.. "A New Theoretical Framework For Information Retrieval". In anais do **congresso ACM Conference on Research and Development in Information Retrieval**, Pisa, Italy, 1986.

WOOLDRIDGE, M., JENNINGS, N.. **Intelligent Agents: Theory and Practice.** Disponível na Internet via WWW. URL:http://www.doc.mmu.ac.uk/STAFF/mike/ ker95/ bibliography3_38.html. 1994. Arquivo consultado em 1999.

YANG, J., KORFHAGE, R. R.. "Query improvement in information retrieval using genetic algorithm: a report on the experiments of the TREC project". In anais do congresso **Text Retrieval Conference (TREC-1**) p. 31-58, Gaithersburg, November, 1993.

YAO, Y. Y.. "Measuring retrieval effectiveness based on user preference of documents". **Journal of the American Society for Information Science**, v. 46(2): pp. 133-145, 1995.