9
As Instanciações do *Framework*

9.1
O *Webclipper*

A Internet vem se tornando um eficiente mecanismo de publicação de notícias. Os veículos de comunicação tradicionais, como jornais e revistas, estão cada vez mais disponibilizando seus conteúdos *online*. Pesquisas recentes realizadas nos Estados Unidos mostraram que, dentre as pessoas que têm opção de escolha entre ler jornais impressos e jornais *online*, a maioria prefere a versão *online*, o que indica uma forte aceitação desse novo paradigma de publicação por parte dos leitores. Não há como negar: o futuro da comunicação é a Internet.

Veículos de comunicação *online* têm a vantagem de serem muito mais dinâmicos que veículos impressos devido à facilidade da publicação de novas notícias - no primeiro caso, implica somente em alterar e adicionar algumas páginas HTML, ao passo que no segundo caso, é necessária uma nova edição. Isso indica que a quantidade de notícias veiculadas em uma publicação *online* é muito maior que a veiculada em uma impressa. Um bom exemplo disso é o que ocorre nos *websites* de jornais de maior circulação, como o Jornal do Brasil e O Globo, em que há uma seção do jornal chamada "plantão", que é onde são inseridas notícias ocorridas ao longo dia, logo não presentes na versão impressa.

Veículos de comunicação já realizam um certo nível de classificação na medida em que dividem suas notícias em cadernos, como o de Economia ou o de Esportes. Porém, ainda assim é possível ir além nessa classificação, disponibilizando ao usuário final um filtro maior para suas preferências, ou ainda disponibilizando uma forma diferente de filtragem. Uma área que certamente tem muito a ganhar com um sistema de classificação dessa natureza é a área de *clipping eletrônico*, também conhecido como *webclipping*.

Para entendermos o que é o *webclipping*, é necessário entender sua versão não eletrônica, o *clipping*.

PUC-Rio - Certificação Digital Nº 0024128/CA

Clipping (em inglês, "corte" ou "recorte", designando especialmente um recorte de jornal ou de revista) é o nome que se dá ao serviço de pesquisa, coleta, seleção e fornecimento de material veiculado por meios de comunicação: a imprensa escrita, o rádio, a TV e, mais recentemente, a própria Internet. O material recolhido e selecionado pelo *clipping* pode ser tanto de natureza jornalística quanto publicitária, sobre qualquer assunto veiculado pela mídia. O serviço de *clipping* é feito sob encomenda e é prestado por empresas especializadas nessa atividade: as clipadoras. De modo geral, o mercado de *clipping* no Brasil é bastante segmentado: nele, é comum que as clipadoras sejam especializadas no acompanhamento de um meio de comunicação (a mídia impressa, o rádio, a TV ou a Internet) [57].

O serviço de *clipping* é utilizado por entidades (empresas ou pessoas físicas) para que estejam sempre bem informadas, acompanhando com a máxima atenção a mídia e o mercado escolhidos, e para que possam reagir com rapidez e eficiência não apenas em momentos de crise, mas diante de informações estratégicas veiculadas todos os dias (inteligência competitiva). Com um serviço de *clipping*, é possível comprovar a divulgação de uma marca, o resultado de uma campanha publicitária ou monitorar a presença que têm, nos meios de comunicação, concorrentes diretos ou indiretos. Os clientes mais freqüentes do serviço de *clipping* são as pessoas jurídicas e físicas que têm maior exposição na mídia ou um grande interesse em acompanhar o que nela se veicula. No caso do *clipping* de mídia impressa, seus usuários mais comuns são empresas de grande e médio porte, órgãos do governo, autoridades e personalidades. As clipadoras costumam trabalhar diretamente com esses clientes ou em estreita colaboração com agências de publicidade, assessorias de imprensa, profissionais de marketing e de relações públicas.

É fácil notar que todos os conceitos de *clipping* tradicional podem ser automaticamente portados para os veículos *online*, com uma vantagem: é possível desenvolver um sistema que faça boa parte do trabalho automaticamente. Um sistema dessa natureza, em sua essência, nada mais é do que uma instância do *framework* proposto, com os *hot spots* implementados da seguinte forma:

PUC-Rio - Certificação Digital Nº 0024128/CA

- ***Os documentos processados são estruturados no formato HTML*:** apesar de existirem outras tecnologias de publicação bastante comuns, como o formato *flash* da Macromedia [57], a quase totalidade dos veículos *online* ainda utiliza HTML para a publicação de suas notícias.
- ***O algoritmo de classificação deve ser bem preciso e específico*:** o principal fundamento do *clipping* é encontrar notícias de interesse dos clientes. Dessa maneira, a classificação deve ser voltada a encontrar matérias de caráter bem específico, além de não admitir grande margem de erros.
- ***O relatório de saída deve ser flexível*:** o formato do relatório enviado aos clientes deve ser flexível. Uma forma de obter esse requisito é armazenar o resultado das pesquisas em um banco de dados a fim de ser processado posteriormente por outra ferramenta fora do escopo do classificador. Outra possibilidade seria gerar um documento no formato XML com o resultado.
- ***A ferramenta deve ser capaz de se lembrar dos documentos visitados e descobrir novos documentos*:** a inteligência dos agentes é imprescindível, pois os veículos de comunicação freqüentemente mantêm arquivos com as notícias dos últimos dias, e tais devem ser ignoradas, pois já foram processadas anteriormente – o cliente deseja receber apenas as notícias novas. Além disso, os agentes devem ser capazes de descobrir novos documentos a serem processados, que são as diversas páginas componentes do veículo – é impraticável fornecer a lista de todos os HTMLs componentes de um veículo, a ferramenta deverá processar o HTML e separar referências a outras páginas.

9.1.1 Questões de Implementação

Discutiremos, a seguir, como cada *hot spot* do *framework* foi implementado, de acordo com o subsistema em que se encontra.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.1.1.1
Subsistema Processador de Documentos

Os documentos processados estão no formato HTML. Assim, foi necessário desenvolver suporte ao processamento desse formato. Para tanto, foi criada a classe ClDocumentoHTML, que implementa a interface IDocumento com suporte ao processamento de HTML. Cada documento tem definido como seu grupo o veículo de onde foi obtido, e seu caminho como sendo a URL completa onde pode ser encontrado. Foi criada também a fábrica de documentos correspondente, denominada ClThreadFabricaHTML.

Como os documentos se encontram na Internet, é utilizado o protocolo HTTP [58] para obtenção dos mesmos. Já existe um extenso suporte a HTTP pronto na API de Java através das classes URL e HttpURLConnection, todas presentes no pacote java.net, logo este suporte foi aproveitado na criação do ClDocumentoHTML. Toda a parte de comunicação através do protocolo HTTP fica encapsulada na classe HttpURLConnection, que possui um Socket (outra classe em Java, do pacote java.net) que é onde fica encapsulada a parte da comunicação TCP/IP.

O fato da instância da classe Socket presente na HttpURLConnection ficar totalmente encapsulada traz alguns problemas sérios já documentados por diversos desenvolvedores: não há como se ter acesso ao parâmetro que define o *timeout* que o Socket deve adotar na hora de esperar pela abertura da conexão ou leitura de dados do servidor. Dessa forma, em alguns casos, uma instância da classe HttpURLConnection pode simplesmente se travar para sempre a espera de uma resposta de um servidor. Pensando em termos do *framework*, isso implica que um agente classificador entraria em *deadlock*, jamais terminando o seu trabalho e, conseqüentemente, todo o sistema entraria em *deadlock* a espera desse único agente terminar seu trabalho. De forma a resolver esse problema, foi criada uma nova classe intermediária entre a ClThreadFabricaHTML e a ClDocumentoHTML denominada ClSimpleHTMLCreationThread, cujo objetivo é abrir um novo *Thread* para cada documento processado. Assim, a fábrica de documentos não mais instancia o documento HTML diretamente, mas sim abre um novo *Thread* que é o responsável pela instanciação. Após isso, a fábrica se

PUC-Rio - Certificação Digital Nº 0024128/CA

trava a espera ou da finalização do *Thread* ou então que um certo *timeout* seja excedido. No segundo caso, o documento é ignorado. Esse *timeout* é recebido via construtor da fábrica.

A Figura 1 mostra o diagrama de classes dessa instanciação, onde fica claro que, apesar da complexidade do que foi implementado, tudo está escondido do *framework*, que continua solicitando documentos a uma fábrica, sem se preocupar como o processamento do documento é feito.

Todos os documentos HTML são verificados quanto a indicarem novos documentos a serem processados, caso a altura de classificação ainda não tenha sido excedida. Esse processo consiste unicamente em obter todos os atributos HREF das *tags* **A** de HTML, todos os atributos SRC das tags **FRAME** e o atributo VALUE das tags **OPTION**.

Outro fato relevante desse subsistema é que o método `hashcode` da classe ClDocumentoHTML é implementado como uma chamada ao método `hashcode` da String que, internamente ao documento, representa o HTML.

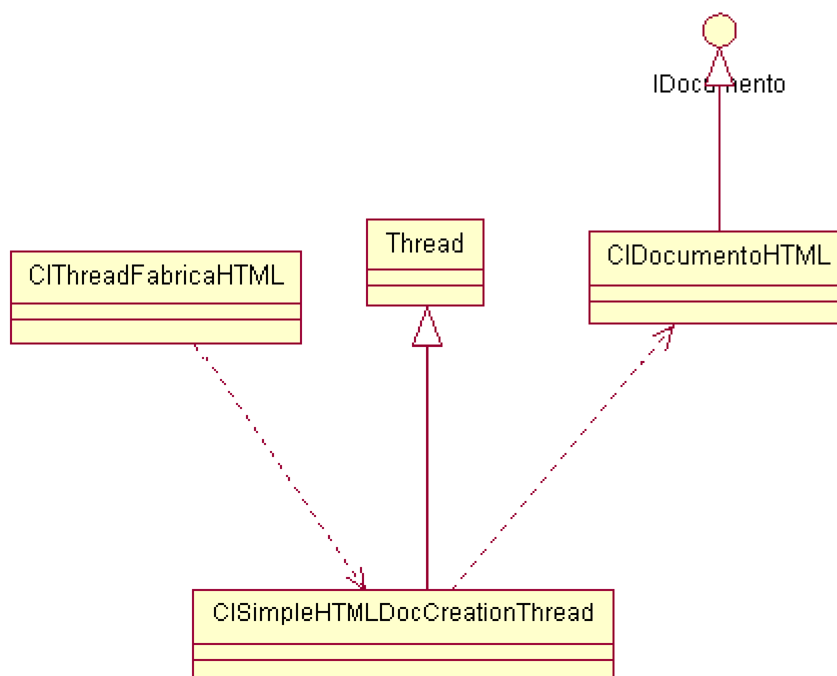

Figura 1: Diagrama de classes do subsistema processador de documentos do *Webclipper*

PUC-Rio - Certificação Digital Nº 0024128/CA

9.1.1.2
Subsistema Armazenador de Documentos a Visitar

Não foi criada nenhuma classe nova para esse subsistema, é utilizada a ClSmartStorage já descrita anteriormente, onde a lista que define os veículos a serem buscados é composta pelo *host* de cada veículo buscado (a fim de evitar que, por exemplo, um *hyperlink* de um veículo que leve para fora do *website* do mesmo seja visitado) e a lista que exclui recebe valores que variam de veículo para veículo, mas basicamente tentam excluir partes como as seções de classificados ou previsão do tempo, que não interessam à aplicação.

9.1.1.3
Subsistema de Memória

Foi criada uma nova classe com o objetivo de tratar a parte de memória não-volátil do sistema, que utiliza o código hash do documento HTML. Essa classe, denominada ClDBKeeper, tem por objetivo armazenar em banco de dados o código hash de cada documento visitado, além de outras informações que facilitam a execução de pesquisas na base, que são o nome do veículo onde o documento foi encontrado e sua URL. Armazena-se também a data de processamento, e essa data é atualizada sempre que o documento é encontrado novamente durante um processamento – isso possibilita descobrir quais os documentos que já foram removidos do veículo, pois como estes não mais serão encontrados durante os processamentos, a sua data de processamento ficará cada vez mais antiga. Obviamente, estes documentos podem ser removidos da base sem prejuízo para a pesquisa. O diagrama de classes desse subsistema encontra-se na Figura 2.

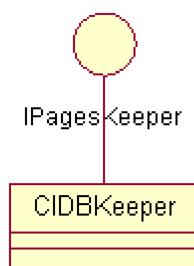

Figura 2: Diagrama de classes do subsistema de memória do *Webclipper*

PUC-Rio - Certificação Digital Nº 0024128/CA

9.1.1.4
Subsistema de Pesquisa

A fim de exemplificar como a troca de algoritmos é facilitada com a utilização do *framework*, foram efetuadas duas instanciações desse *hot spot*.

Primeira Instanciação

O subsistema de pesquisa foi instanciado com um algoritmo bastante simples conceitualmente, baseado em ocorrência de palavras, mas poderoso para essa aplicação: a interface IPesquisa foi utilizada para gerar uma classe abstrata, denominada ClPesquisa, que contém alguns métodos específicos dessa aplicação. ClPesquisa foi usada como base para uma hierarquia de classes, que têm uma estrutura bastante parecida com o *design pattern Composite* [52], por onde é possível criar pesquisas compostas. A Figura 3 ilustra essa estrutura.

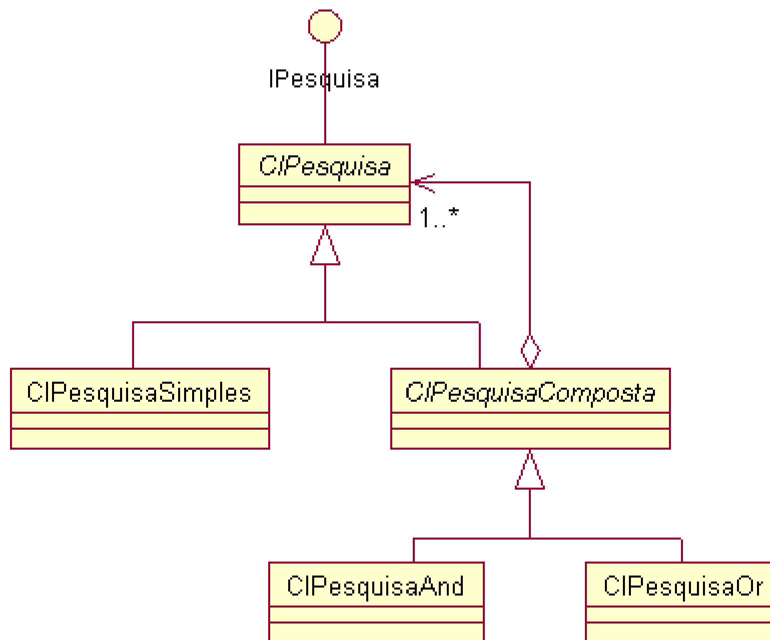

Figura 3: Diagrama de classes da primeira instanciação do subsistema de pesquisa do *Webclipper*

PUC-Rio - Certificação Digital Nº 0024128/CA

A classe ClPesquisaSimples tem seu método `pesquisar()` implementado para buscar por uma palavra no texto do documento. Caso a palavra ocorra, a pesquisa é dita como ocorrida, senão não.

As classes ClPesquisaAnd e ClPesquisaOr são utilizadas para representar os conectores lógicos E e OU, entre diversas pesquisas. Dessa forma, é possível criar árvores com expressões lógicas que devem ser avaliadas a fim de verificar se um documento pertence ou não à classe.

Segunda Instanciação

O subsistema de pesquisa foi instanciado com o conceito de tema, que é definido como sendo uma palavra aliada a um grupo de contextos definidos. Dessa forma, as pesquisas não são feitas em cima de expressões de busca, mas sim utilizando os temas definidos.

Cada tema é modelado pela classe ClTema, que é quem implementa a interface IPesquisa. Cada tema recebe, no construtor, uma String, que define a sua palavra. Os diversos contextos de um tema são modelados, cada um, por uma instância da classe ClContexto, que possui um valor inteiro, denominado *Threshold,* e uma coleção de palavras contextuais, que são definidas como uma palavra aliada a um peso. Cada palavra contextual é modelada pela classe ClPalavraContextual, que possui uma palavra e um inteiro representando seu peso. O diagrama de classes dessa estrutura encontra-se na Figura 4.

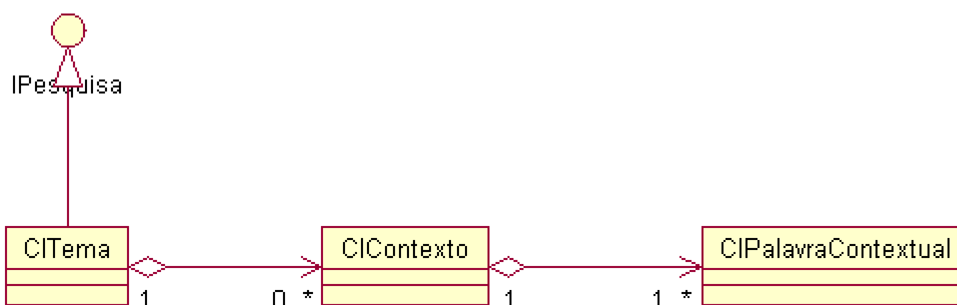

Figura 4: Diagrama de classes da segunda instanciação do subsistema de pesquisa do *Webclipper*

PUC-Rio - Certificação Digital Nº 0024128/CA

É possível notar que a multiplicidade de ClTema para ClContexto é zero ou mais. A ausência de contexto indica que a palavra que define o tema deve ser buscada em qualquer contexto.

O algoritmo tem por objetivo descobrir se um tema ocorre em um documento. Para isso, verifica-se se a palavra do tema ocorre no documento. Caso ela não ocorra, admite-se que o tema não ocorre. Caso ocorra, processam-se seus contextos. Seja C o conjunto de contextos de um tema T. Em cima desses elementos, é executado o seguinte cálculo:

- Caso C seja vazio, admite-se que o tema ocorre.
- Caso C não seja vazio, para cada contexto c pertencente a C, obtém-se seu conjunto de palavras contextuais P. Seja i um inteiro, ao qual inicialmente é atribuído o valor 0.
- Para cada palavra contextual p pertencente a P, verifica-se se a palavra que a define ocorre no documento. Caso ocorra, adiciona-se ao inteiro i o valor de seu peso.
- Compara-se o valor de i com o valor do *threshold* do Contexto. Caso i ≥ *threshold*, admite-se que o contexto ocorre.
- Ao final, se ao menos um dos contextos pertencentes a C ocorre, diz-se que o tema ocorre no documento.

Em ambas as instâncias, foi criada também uma nova classe, que estende ClRespostaPesquisa, a fim de modelar a resposta de uma pesquisa em um documento. Essa classe é denominada ClRespostaWebclipper, e adiciona algumas informações relevantes à geração do relatório.

As pesquisas somente retornam instâncias da classe ClRespostaPesquisa caso elas sejam identificadas no documento. Caso contrário, é retornado `null`. Dessa forma, o gerador de relatórios somente conterá indicações positivas.

9.1.1.5 Subsistema Gerador de Relatórios

PUC-Rio - Certificação Digital Nº 0024128/CA

Inicialmente, a geração de relatórios ficou a cargo da classe ClXMLSender, que implementa a interface ISender. A fim de deixar o formato do arquivo de relatório flexível, é utilizado um esquema de *templates* em um formato padrão. São ao todo três *templates*, que são combinados para gerar o arquivo XML de resposta final.

Posteriormente, utilizou-se o ClXMLSender para gerar arquivos no formato HTML (que pode ser encarado como um caso específico de XML), e decidiu-se que o relatório resultante deveria ser enviado por e-mail. Dessa decisão, se originou uma nova classe, de nome ClResumeSender, que estende ClXMLSender mas com a funcionalidade de enviar e-mails segundo o protocolo SMTP.

A possibilidade de continuar a realizar testes com essa instância levou a geração de uma nova classe geradora de relatórios: a ClDBSender, cujo objetivo é armazenar o relatório numa tabela de banco de dados, permitindo o processamento posterior dos resultados da pesquisa.

O diagrama de classes completo desse subsistema encontra-se na Figura 5.

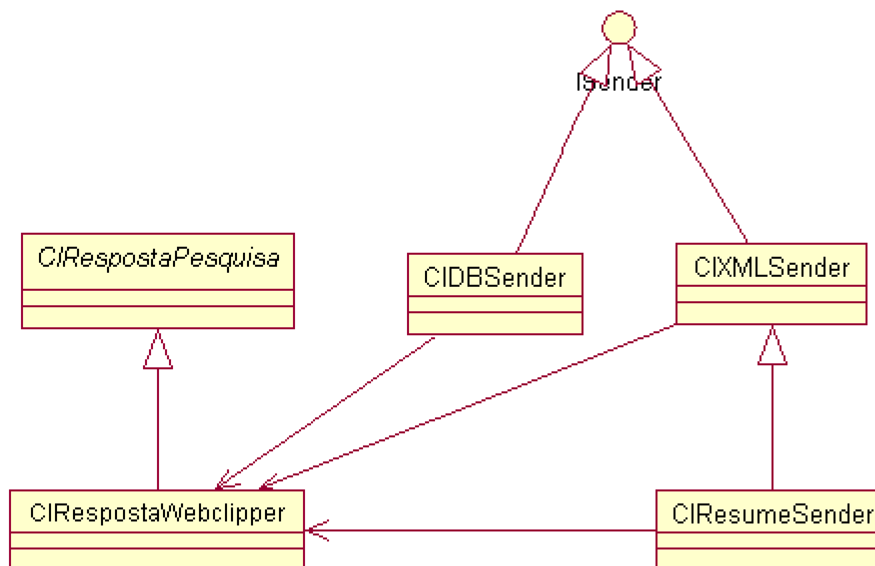

Figura 5: Diagrama de classes do subsistema gerador de relatórios do *Webclipper*

Esta aplicação é um típico caso onde é necessário utilizar vários Senders, um para cada relatório a ser gerado. Além disso, cada relatório deve conter apenas resultados referentes às suas pesquisas. Isto é um problema se considerarmos que

PUC-Rio - Certificação Digital Nº 0024128/CA

o *framework* trabalha com todas as pesquisas de uma só vez, e somente com um Sender. Quando qualquer pesquisa é validada, o Sender é avisado.

A solução está na combinação do ClXMLSender e do ClDBSender com os Senders já fornecidos no *framework*:

- Um ClGroupSender deve ser utilizado para agrupar o ClXMLSender e o ClDBSender de um relatório;
- Um ClSelectiveSender deve ser utilizado em cima do ClGroupSender de cada relatório, para filtrar as respostas indesejadas;
- Um ClGroupSender deve ser utilizado para agrupar todos os ClSelectiveSenders de cada relatório.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.1.2
Questões de Configuração

A configuração de pesquisa do *Webclipper* é dada por um arquivo XML, que contém todos os dados dos usuários que receberão os e-mails com os relatórios, bem como as expressões de pesquisa e também os veículos a serem visitados. Para processar o XML, foi utilizado o parser Jdom [59].

9.1.3
Utilização do *Webclipper*

Foi criada uma classe, denominada ClWebClipper, cujo objetivo é dar início à aplicação: é essa a classe que possui o método `main()`, e que instancia todas as demais classes do sistema para que o *framework* as utilize.

9.2
O KM *Probe*

Vamos considerar um ambiente de pesquisa e desenvolvimento, onde existem várias equipes designadas para diferentes projetos, cada uma acumulando todo tipo de informação relacionada ao seu assunto de trabalho. É fácil perceber

que, se não houver algum tipo de disciplina na organização na matéria Gerência de Conhecimento, a tendência é que haja uma *singularização intra-equipe e extra-equipe do conhecimento adquirido*: somente a pessoa que achou os documentos é que, efetivamente, vai ganhar com esse conhecimento tanto dentro de sua equipe quanto dentro da organização.

Se levarmos em consideração que dentro de uma equipe é natural que haja especialistas em determinadas áreas, é possível concluir que a singularização intra-equipe não é tão grave na medida em que o conhecimento realmente só é de interesse de um grupo restrito. Apesar disso, seria interessante que as demais áreas dentro da equipe tivessem ao menos uma boa noção do que está sendo acumulado em termos de documentação, principalmente a fonte de informações utilizada, que pode ser de interesse geral.

O problema principal ocorre com a singularização extra-equipe que, em outras palavras, significa que o *conhecimento adquirido durante a execução de um projeto fica restrito aos seus participantes, não sendo devidamente compartilhado com os demais membros da organização*. Em tal situação, pode-se concluir que o conhecimento adquirido durante os projetos não pertence à organização, mas sim a cada membro que a compõe. Isso é perigoso, pois se o detentor de um conhecimento, por algum motivo, deixa a organização, o conhecimento adquirido graças a investimentos da organização está perdido para sempre.

Uma forma de evitar tal problema é utilizar algum processo de compartilhamento do conhecimento. O exemplo mais simples de algo dessa natureza seria um repositório comum, classificado segundo assuntos, em que documentos seriam depositados e ficariam disponíveis para qualquer um consultar. Na verdade, estamos falando de uma espécie de repositório de conhecimento, bem parecido com uma biblioteca. Esse é o objetivo do projeto KM, atualmente desenvolvido no Teccomm.

O projeto KM prevê o desenvolvimento de uma ferramenta para a criação de uma aplicação hipermídia para suporte à gestão do conhecimento a partir de uma ontologia. A ontologia deve definir as categorias e relacionamentos da informação

PUC-Rio - Certificação Digital Nº 0024128/CA

a ser gerenciada. Essa aplicação deve permitir a navegação pelas informações de forma amigável, tornando mais fácil a tarefa de descoberta de conteúdo relevante sobre um determinado assunto. Em outras palavras, a ferramenta deve ser vista como um gerador de portais para gestão de conhecimento.

Ora, para que um portal gerado pela ferramenta tenha sucesso dentro de uma organização, é necessário não só disciplinar seus componentes a constantemente alimentá-la com as informações que são colhidas durante a execução dos projetos, mas também ensiná-los a classificar corretamente os documentos a fim de serem armazenados corretamente. Fica claro que esta é uma tarefa difícil de ser atingida, principalmente devido a requerer uma mudança de comportamento e atitude de seres humanos.

É nesse ponto que o KM *Probe* se encaixa. Seu objetivo é tirar o trabalho tedioso de cadastramento, bem como o trabalho muitas vezes ambíguo de classificação dos documentos. Esta instância do *framework* deve ser acoplada à ferramenta de KM, e terá a aparência de uma sonda de documentos, cujo objetivo é vasculhar locais definidos (por exemplo, páginas *web* ou diretórios compartilhados na rede) procurando por novos documentos em formato conhecido e categorizando-os para serem posteriormente armazenados. Os *hot spots* devem, então, ser implementados da seguinte forma:

PUC-Rio - Certificação Digital Nº 0024128/CA

- ***Os documentos processados são estruturados em formatos conhecidos*:** é interessante que os formatos mais comuns de documentos, como Word for Windows, PDF e Post Script, sejam suportados.
- ***O algoritmo de classificação deve ser uniforme para a organização*:** todas as equipes de desenvolvimento devem utilizar o mesmo esquema de classificação a fim de garantir a uniformidade das classificações.
- ***O relatório de saída deve ser flexível*:** o formato do relatório deve obedecer alguma interface pré-definida com a ferramenta de KM, a fim de que essa possa fazer uso corretamente dos resultados. A opção considerada foi gerar um documento no formato XML com o resultado.

- ***A sonda deve ser capaz de se lembrar dos documentos visitados*:** a sonda deve ser capaz de localizar apenas os novos documentos adicionados desde a última varredura, a fim de aumentar a performance de execução.

9.2.1 Questões de Implementação

Discutiremos, a seguir, como cada *hot spot* do *framework* foi implementado, de acordo com o subsistema em que se encontra.

9.2.1.1 Subsistema Processador de Documentos

Foi reaproveitado o subsistema processador de documentos do *Webclipper*, com pequenas modificações, e foi adicionado suporte ao processamento de arquivos texto. Para isso, foi criada a classe ClDocumentoTXT, cujo objetivo é encapsular todo o processamento desse formato de documento, e também a fábrica correspondente, chamada ClFabricaDocumentoTXT. No caso de documentos texto, não foi implementado mecanismo de descoberta de novos documentos (como foi com os documentos HTML). Cada ClDocumentoTXT tem definido como seu grupo a String "TEXTO", e seu caminho como sendo o caminho completo para onde pode ser encontrado (diretório + nome do arquivo).

Conforme já discutido anteriormente, o *framework* somente é capaz de enxergar uma única fábrica de documentos. A solução proposta foi adotar a hierarquia de classes similar a do *design pattern Composite* [52] já proposta anteriormente no capítulo de instanciação do *framework*, cuja estrutura encontra-se na **Erro! A origem da referência não foi encontrada.**

Foi criada a figura da ClFabricaChain, com esse mesmo nome, e ambas as fábricas envolvidas (ClThreadFabricaHTML e ClDocumentoTXT) passaram a herdar dela (sendo, portanto, equivalentes às fábricas ClFabrica_1 e ClFabrica_2 da figura). A estratégia para fazer com que cada uma reconhecesse seus

PUC-Rio - Certificação Digital Nº 0024128/CA

documentos, repassando os demais adiante na cadeia, foi implementar no método `processarDocumento()` o seguinte:

- A ClThreadFabricaHTML verifica se o caminho do documento a ser processado se inicia com “http://”, uma vez que todos os documentos no formato HTML vêm da *web*;
- A ClDocumentoTXT verifica se o documento tem a extensão “.txt”.

9.2.1.2 Subsistema Armazenador de Documentos a Visitar

Não foi criada nenhuma classe nova para esse subsistema, é utilizada a ClSmartStorage já descrita anteriormente, com as seguintes configurações:

- A lista que define os veículos a serem buscados é composta pelo *host* de cada veículo buscado, assim como no *Webclipper*;
- A lista que define os documentos texto a serem buscados contém o diretório onde o documento se encontra;
- A lista que exclui veículos recebe valores que variam de veículo para veículo, mas basicamente tentam excluir partes como as seções de classificados ou previsão do tempo, assim como no *Webclipper*;
- A lista que exclui documentos texto não recebe elementos.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.2.1.3 Subsistema de Memória

O subsistema de memória foi reaproveitado por completo do *Webclipper*.

9.2.1.4 Subsistema de Pesquisa

O algoritmo utilizado no subsistema de pesquisa é baseado na utilização de ontologias. Para tanto, é necessário definir uma taxonomia de conceitos

relacionados sob a forma de uma árvore, onde conceitos mais abstratos se encontram próximos à raiz, e conceitos mais concretos ficam próximos às folhas. Um exemplo de taxonomia que poderia ser utilizada para o algoritmo é ilustrada na Figura 6, onde o objetivo seria classificar documentos de economia.

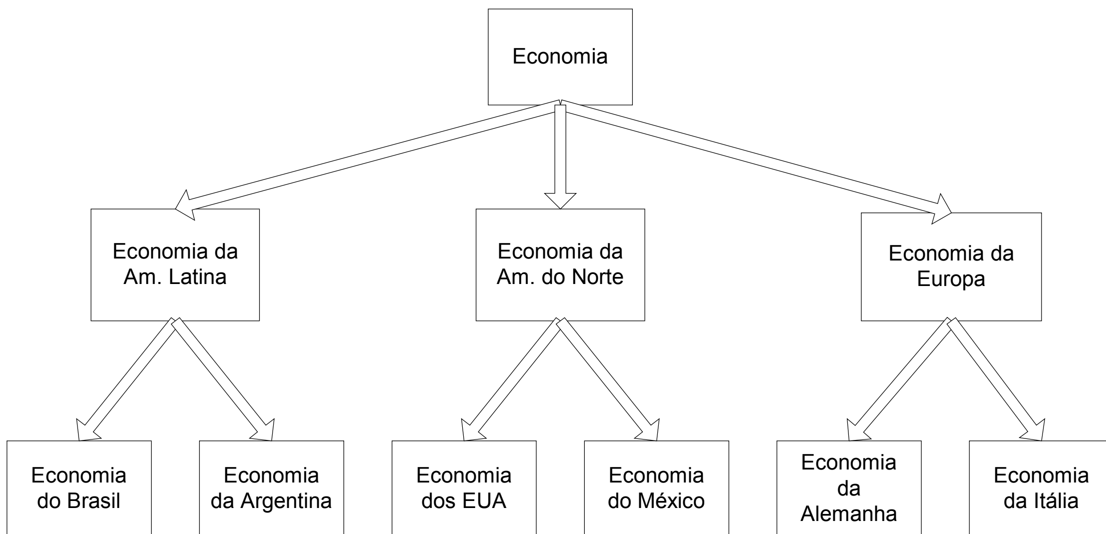

Figura 6: Exemplo de taxonomia utilizada para a KM Probe

PUC-Rio - Certificação Digital Nº 0024128/CA

Uma vez definida a taxonomia, é necessário verificar cada classe, a fim de se definir quais as condições necessárias para que um documento seja considerado como pertencente a ela. Essas condições devem se basear em ocorrência de palavras, da seguinte maneira:

- Para cada classe, é selecionado um grupo de palavras G. Essas palavras serão buscadas no texto do documento, e a quantidade de palavras que forem encontradas será utilizada para definir se o documento pertence ou não à classe.
- A cada classe, é atribuído um valor racional maior que zero e menor ou igual a 1. A esse valor daremos o nome de fator de definição, e o representaremos pela letra f.
- Seja n o número de palavras pertencentes a G que foram encontradas no texto do documento. Seja m a cardinalidade de G. Se $(n/m >= f)$, então o documento é considerado como pertencente à classe. Caso um documento d seja considerado como pertencente a uma classe C, então

todas as classes filhas de *C* também serão testadas quanto a conter *d*. Caso um documento *d* seja considerado como não pertencente a uma classe *C*, então nenhuma das classes filhas de *C* será testada quanto a conter *d*.

Podemos pensar na taxonomia em conjunto com os seus atributos como sendo uma ontologia.

A forma de implementar essa árvore foi através da classe IPesquisa: foi criada a classe ClNo, que estende de IPesquisa, e que representa um nó na ontologia. Possui como atributos um nome, a lista de palavras, o fator de definição, um ClNo que representa seu pai e uma lista de ClNo que representa seus filhos. O diagrama de classes na Figura 7 ilustra essa estrutura.

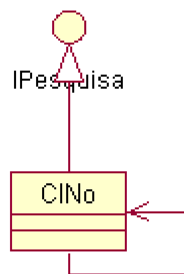

Figura 7: Diagrama de classes do subsistema de pesquisa para a KM Probe

Os métodos da interface IPesquisa responsáveis por analisar um documento foram implementados de acordo com o algoritmo definido. O subsistema de gerência, que é o coordenador do sistema, deve receber como parâmetro o ClNo raiz. Assim, todo documento será inicialmente analisado quanto a pertencer à raiz, e caberá a esta disparar outras pesquisas se for necessário.

A resposta de cada classificação é dada pela classe ClAvaliação. Cada instância de ClNó visitado durante a pesquisa gera uma instância de ClAvaliação, que estende a classe ClRespostaPesquisa adicionando um atributo booleano, recebido via construtor, que indica se a avaliação do nó correspondente foi positiva ou negativa. Além desse atributo, existe também o método `toString()`, que gera uma representação em String da avaliação, em formato XML, que obedece ao formato definido.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.2.1.5
Subsistema Gerador de Relatórios

Este subsistema é formado pela classe ClAgrupadorAvaliacoes, que implementa a interface ISender e serve como um armazenador de ClAvaliacoes. Possui o método `gerarSaida()`, que é utilizado para gerar o relatório final quando as pesquisas são encerradas.

9.2.2
Questões de Configuração

A configuração de pesquisa do KM *Probe* é dada por dois arquivos XML:

- Um deles contém todos os dados dos *links* a serem processados, bem como os arquivos texto;
- O outro possui a ontologia.

Para processar o XML, foi utilizado o parser Jdom [59].

9.2.3
Utilização do KM *Probe*

Foi criada uma classe, denominada ClKM, cujo objetivo é dar início à aplicação: é essa a classe que possui o método `main()`, e que instancia todas as demais classes do sistema para que o *framework* as utilize.

9.3
O *Site Seeker*

O *Site Seeker* é uma aplicação para busca em *websites*. Seu funcionamento é bem simples: o usuário entra com algumas palavras e o *Site seeker* procura essas palavras diretamente nos documentos HTML das páginas.

O *Site Seeker* é ideal para pequenos *websites* estáticos (que não dependam de parâmetros do usuário para gerar suas páginas, como os requeridos por

PUC-Rio - Certificação Digital Nº 0024128/CA

formulários). Como a maioria desses *websites* não possui nenhum tipo de banco de dados, a busca direta no HTML é uma saída interessante.

Para esta aplicação, os *hot spots* devem ser implementados da seguinte maneira:

- ***Os documentos processados são estruturados no formato HTML*****:** quase a totalidade dos pequenos *websites* ainda utiliza HTML para a publicação de suas páginas.
- ***O algoritmo de classificação é simples*****:** basta procurar pelas palavras diretamente. Como o *website* é pequeno, deixa-se a cargo do usuário separar manualmente as páginas que lhe interessam.
- ***O relatório de saída deve ser no formato HTML*****:** O resultado é exibido no *browser* do cliente, logo deve estar em HTML.
- ***A ferramenta não deve ser capaz de se lembrar dos documentos visitados, mas deve descobrir novos documentos*****:** Cada nova busca é totalmente independente das anteriores, logo a memória não-volátil deve estar desativada, e pode ser impraticável fornecer a lista de todos os HTMLs componentes de um *website*, logo a ferramenta deverá processar o HTML e separar referências a outras páginas.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.3.1 Questões de Implementação

Discutiremos, a seguir, como cada *hot spot* do *framework* foi implementado, de acordo com o subsistema em que se encontra.

9.3.1.1 Subsistema Processador de Documentos

Foi utilizado o mesmo subsistema do *Webclipper*, com a diferença que não é usado o conceito de grupo de um documento.

9.3.1.2
Subsistema Armazenador de Documentos a Visitar

Não foi criada nenhuma classe nova para esse subsistema, é utilizada a ClSmartStorage já descrita anteriormente, com as seguintes configurações:

- A lista que define as páginas a serem buscadas é composta pelo *host* de cada máquina onde estão hospedadas as páginas do *website*;
- A lista que exclui páginas fica vazia;

9.3.1.3
Subsistema de Memória

A parte que trata da memória não-volátil foi desativada através da utilização da classe ClDummyKeeper.

9.3.1.4
Subsistema de Pesquisa

Foi utilizada a primeira instância do mesmo subsistema do *Webclipper*, porém sem as funcionalidades de permitir pesquisas compostas – isso porque construir uma interface que tratasse desse tipo de pesquisa seria muito trabalhoso, e estaria totalmente fora do escopo desse trabalho.

9.3.1.5
Subsistema Gerador de Relatórios

Foi utilizado o mesmo subsistema do *Webclipper*, porém com uma pequena modificação: o relatório não mais é enviado por e-mail, mas sim enviado por HTTP para o *broswser* do cliente.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.3.2
Questões de Configuração

A configuração de busca do *Site Seeker* é dada por um arquivo em um formato estabelecido, denominado de arquivo .INI. Nele, ficam informações como a altura de busca no *website*, a(s) URL(s) a serem buscadas inicialmente e a lista de palavras excludentes e definidoras dos *websites*.

9.3.3
Utilização do *Site Seeker*

Foi criada uma nova classe, denominada SiteSeeker, que na verdade é um Servlet [60], que deve ser disponibilizado em um servidor *web*. Esta classe é quem instancia todas as demais classes do sistema para que o *framework* as utilize.

PUC-Rio - Certificação Digital Nº 0024128/CA

9.4
O Semantic Probe

Esta instância visa exemplificar como o *framework* desenvolvido pode ser utilizado no contexto da *web semântica*, que já foi discutida no capítulo 4.

O objetivo foi criar uma aplicação semelhante ao KM *Probe*, porém fazendo uso das meta-informações presentes em documentos que estão nos diversos formatos propostos para serem os padrões da *web semântica*. Como ainda não há padrão definido, foi escolhido SHOE para fins de exemplificação. Esta escolha se baseou principalmente no fato desse formato ser uma extensão de HTML, que já havia sido amplamente utilizado nas instâncias anteriores. Assim, o trabalho de criação de um processador para documentos SHOE foi bastante reduzido.

Documentos no formato SHOE carregam consigo informações acerca de sua categoria. Todo documento SHOE possui uma *tag* denominada CATEGORY, onde o atributo NAME possui o nome da categoria, segundo uma ontologia utilizada pelo seu autor. Há também diversas outras informações, como nome do autor, seu *website*, suas áreas de atuação, etc.

A abordagem para a geração da *Semantic Probe* foi bem simples: já que o documento carrega consigo uma classificação, basta aproveitá-la. Assim, não há processamento para descobrir a qual classe um documento pertence, apenas para descobrir qual(is) a(s) classe(s) que seu autor selecionou.

Assim, os *hot spots* devem ser implementados da seguinte maneira:

- Os documentos processados são estruturados no formato SHOE;
- O algoritmo de classificação deve buscar a classe definida pelo autor, no próprio documento;
- O relatório de saída deve ser flexível: A opção utilizada foi gerar um documento no formato XML com o resultado.
- A sonda deve ser capaz de se lembrar dos documentos visitados: a sonda deve ser capaz de localizar apenas os novos documentos adicionados desde a última varredura, a fim de aumentar a performance de execução.

9.4.1 Questões de Implementação

Discutiremos, a seguir, como cada *hot spot* do *framework* foi implementado, de acordo com o subsistema em que se encontra.

9.4.1.1 Subsistema Processador de Documentos

Foi utilizada a mesma arquitetura desse subsistema no *Webclipper*:

- Existe a classe ClDocumentoSHOE, que é análoga à classe ClDocumentoHTML;
- Existe a classe ClThreadFabricaSHOE, que é análoga à classe ClThreadFabricaHTML;
- Existe a classe ClSimpleSHOEDocCreationThread, que é análoga à classe ClSimpleHTMLDocCreationThread;

PUC-Rio - Certificação Digital Nº 0024128/CA

9.4.1.2
Subsistema Armazenador de Documentos a Visitar

Não foi criada nenhuma classe nova para esse subsistema, é utilizada a ClSmartStorage já descrita anteriormente, da mesma forma que na parte que trata dos documentos HTML do subsistema armazenador de documentos a visitar do KM *Probe*.

9.4.1.3
Subsistema de Memória

O subsistema de memória foi reaproveitado por completo do *Webclipper*.

9.4.1.4
Subsistema de Pesquisa

Foi criada a classe ClCategory, que implementa a interface IPesquisa e cujos métodos de pesquisa simplesmente extraem as categorias do documento SHOE.

Foi criada a classe ClResposta, que estende a classe ClRespostaPesquisa, adicionando um atributo para guardar a lista das categorias de cada documento.

9.4.1.5
Subsistema Gerador de Relatórios

Foi criada a classe ClSaida, que implementa a interface ISender e escreve em um arquivo no formato texto as categorias dos documentos processados, linha a linha.

9.4.2
Questões de Configuração

A configuração de busca do *Semantic Probe* é dada por um arquivo em um formato estabelecido, denominado de arquivo .INI. Nele, ficam informações como

PUC-Rio - Certificação Digital Nº 0024128/CA

a altura de busca no *website*, a(s) URL(s) a serem buscadas inicialmente e a lista de palavras excludentes e definidoras dos *websites*. Nota-se que esse padrão é idêntico ao adotado no *Site seeker*, e isso não é por acaso: o objetivo foi reutilizar essa parte, apenas com o detalhe que as URLs não mais apontam para documentos HTML, mas sim no formato SHOE.

9.4.3 Utilização do *Semantic Probe*

Foi criada uma nova classe, denominada ClSemanticProbe, que é quem instancia todas as demais classes do sistema para que o *framework* as utilize. É ela que possui o método `main()`.

PUC-Rio - Certificação Digital Nº 0024128/CA