UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

WebObjects:

Uma Plataforma de Consulta à Web

Fábio Soares Silva

CAMPINA GRANDE - PB
AGOSTO DE 2002

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

WEBOBJECTS: UMA PLATAFORMA DE CONSULTA À WEB

Fábio Soares Silva

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

Área de Concentração: Banco de Dados

Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Prof. Dr. Marcus Costa Sampaio
(Orientador)

Campina Grande, Paraíba, Brasil
Agosto de 2002

FICHA CATALOGRÁFICA

SILVA, Fábio Soares

S586W

WebObjects: Uma Plataforma de Consulta à Web

Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Agosto de 2002.

142 p. Il.

Orientador: Marcus Costa Sampaio

Palavras Chave:

1. Banco de Dados
2. Linguagens de Consulta à Web
3. Recuperação de Informação

CDU - 681.3.07B

WebObjects: Uma Plataforma de Consulta à Web

Fábio Soares Silva

Dissertação aprovada em:

Marcus Costa Sampaio

Javam C. Machado

Ulrich Schiel

Campina Grande, 29 de agosto de 2002

Agradecimentos

Aos meus pais, José Milton de Morais Silva e Maria Soares Silva, pelo apoio, carinho e dedicação nessa jornada.

À minha noiva, Liane Guanabara Guimarães, pelo companheirismo, compreensão e incentivo em todos os momentos.

Ao meu orientador, Prof. Marcus Costa Sampaio, pela confiança e esforço com que sempre encarou este trabalho.

Aos meus colegas de trabalho e aos professores que sempre foram solícitos e amigos.

Resumo

O objetivo principal da plataforma *WebObjects* é possibilitar, de forma transparente, consultas a dados disponíveis na Web, através de uma linguagem de consulta com um alto nível de abstração. Para isso, a Web é modelada como um banco de dados orientado a objeto, segundo o padrão ODMG 3.0. Este documento descreve em detalhes a plataforma, incluindo sua avaliação experimental.

Abstract

The main objective of the WebObjects Framework is to allow, in a transparent way, querying data on the Web, using a query language with a high-level abstraction. To achieve this, the Web has been modeled as an object oriented database, according to the ODMG 3.0 standard. This document describes the framework in detail, and includes an experimental evaluation of the implemented software.

Lista de Tabelas

Lista de Figuras

Sumário

Capítulo 1

I. Introdução

A partir de meados da última década, foi iniciado o processo de disseminação da Internet nos vários segmentos da sociedade. Isto fez com que a Web deixasse de ser apenas uma ferramenta de troca de informações acadêmicas, tornando-se o veículo de comunicação que revolucionou completamente as relações sociais.

A vasta utilização da Internet, por meio de uma impressionante diversidade de aplicações e técnicas de publicação de dados, fez dela o maior e mais rico banco de dados já construído na história da humanidade. Como muito bem retratado em [MEN,1997] e [GOL,2000], este banco de dados caracteriza-se principalmente pela distribuição, tamanho e diversidade de formatos, tipos, estruturas de dados e tecnologias utilizadas.

Considerando o enfoque que trata a Web como um gigantesco banco de dados, fica evidente a necessidade de incorporar a ela os variados recursos de manipulação e recuperação de informação dos Sistemas de Gerência de Banco de Dados (SGBDs). Entre esses recursos, podemos destacar a necessidade de agregar à Web um processador de consultas que possa melhor explorar os seus dados, ou seja, disponibilizar aos usuários a capacidade de manipular facilmente a riqueza e importância do seu conteúdo.

Atualmente, a principal forma de pesquisa a dados na Web são as máquinas de busca (*search engines*), tais como *AltaVista*[1] e *Google*[2]. As máquinas de busca são *websites* que constroem catálogos *off-line* de documentos na Web, e através de uma interface própria, oferecem recursos de consulta ao conteúdo desses catálogos.

As máquinas de busca atendem apenas parcialmente às necessidades do usuário, pois impõem importantes restrições na especificação das consultas. A primeira delas está associada à incapacidade de permitir, na formulação da pesquisa, que o usuário utilize todo o seu conhecimento parcial sobre os documentos de interesse. Dessa forma, dados como tamanho

[1] http://www.altavista.com
[2] http://www.google.com

mínimo e formato do arquivo, e em alguns casos a data de atualização e o domínio dos documentos, não podem ser livremente utilizadas na formulação da consulta.

Outra grande deficiência das máquinas de busca é a impossibilidade da exploração controlada da estrutura topológica dos documentos *hyperlink* da Web, fazendo com que suas consultas não ofereçam recursos que viabilizem a navegabilidade por estas referências, e a conseqüente utilização destas informações como critério de seleção [MAN,2001].

Talvez o maior dos problemas associados à pesquisa pelas máquinas de busca está na sua incapacidade de resolver consultas complexas [MAN,2001]. Esta deficiência já apontada em ponderações feitas por [ARO,1998], [MIH,1997] e [GOL,2000], torna clara a falta de poder semântico das consultas definidas nas máquinas de busca, fazendo com que as especificações sejam extremamente pobres no sentido de selecionar adequadamente os dados de interesse. Assim, os usuários destas soluções têm que resolver manualmente muitas das tarefas que poderiam, e até deveriam, ser responsabilidade da ferramenta de busca. Aliado a isso, as máquinas de busca não oferecem recursos de reestruturação da resposta, de modo que o usuário deve se adequar ao formato e ordem previamente definidos, dificultando, em parte, o processo de análise.

Outra importante restrição imposta pelas máquinas de busca está na ausência de mecanismos que facilitem a criação de aplicações que façam uso dos dados da Web. Isto ocorre porque elas oferecem aos desenvolvedores uma API com interface de utilização proprietária, e que recuperam apenas URLs que atendem aos critérios especificados na consulta. No entanto, a possibilidade de captura e manipulação das informações relativas a cada URL retornada é de responsabilidade da aplicação. Fazendo uma analogia com bases de dados relacionais, seria como se o SGBD retornasse apenas a localização da tabela, sendo de responsabilidade da aplicação a captura e recuperação das linhas que a compõem, ou seja, a recuperação do conteúdo da tabela.

Finalmente, pode-se constatar que as máquinas de busca estão pouco comprometidas com a natureza dinâmica da Web. Como os catálogos são construídos de maneira *off-line*, acontece freqüentemente que as máquinas de busca retornam URLs inválidas, isto é, fazendo referência a páginas que não existem mais. Como exemplo desta situação, uma busca por documentos

com a palavra "soteropolitano", submetida à máquina de busca *AltaVista*, retornou 135 URLs[3], dentre as quais 35 eram inválidas.

É consenso entre os pesquisadores da área que o primeiro grande desafio na direção de agregar à Web as características de um SGBD está em conceber um processador capaz de oferecer uma linguagem de consulta com alto poder de expressão — com estilo SQL[4] ou OQL[5] —, para filtrar e recuperar dados da Web de acordo com as reais necessidades de informação dos usuários.

Além de oferecer recursos de manipulação de dados semiestruturados, tais como documentos HTML e XML, o processador de consultas deve igualmente ser capaz de manipular dados estruturados [GOL,2000]. Bancos de dados relacionais, com interface Web, constituem uma parte expressiva dos dados estruturados disponibilizados na Web. Em conseqüência, impõe-se a capacidade de integrar dados semiestruturados e dados estruturados.

1.1. Objetivos da Dissertação

O objetivo principal deste trabalho foi o desenvolvimento de uma plataforma de consulta à Web, chamada de *WebObjects*, que possibilita aos seus usuários a especificação de consultas com alto poder de abstração e expressão, segundo o padrão ODMG de banco de dados orientado a objeto.

Outro requisito importante é a independência da plataforma de ambientes computacionais específicos, possibilitando que a mesma possa ser utilizada em diferentes *hardwares* e sistemas operacionais.

A seleção de objetos da Web pode ser por conteúdo, ou também por meio da exploração da sua topologia. Além disso, a plataforma permite a recuperação e manipulação dos objetos pelos desenvolvedores de maneira semelhante à forma que as aplicações atuais tratam informações recuperadas a partir de SGBDs relacionais, ou seja, implementa componentes pelos quais os comandos da linguagem de consulta são executados, sendo o resultado processado de maneira seqüencial.

---

[3] Pesquisa realizada em maio de 2002
[4] *Structured Query Language*
[5] *Object Query Language*

Os objetos Web tratados pela plataforma restringem-se a documentos HTML e a dados estruturados sob a forma de tabelas relacionais, os quais constituem a maior parte dos dados atualmente disponibilizados na Web. No entanto, a solução é extensível o suficiente para tratar novos tipos de dados.

1.2. Relevância da Dissertação

Sendo a Web um banco de dados **diferente**, por causa de sua enorme complexidade e diversidade, é crucial esconder dos usuários sua **diferença**. Mais precisamente, os usuários não deveriam distinguir se os dados são arquivos *stand-alone* HTML, ou bancos de dados Oracle (ou IBM, ou Informix, etc).

A relevância deste trabalho reside no fato de que a plataforma *WebObjects* oferece aos usuários uma visão da Web com um alto nível de abstração, encapsulando sua complexidade e diversidade. Isto quer dizer que, para os usuários, a Web não será diferente de um moderno SGBD, com evidentes ganhos de produtividade no desenvolvimento de aplicações Web. Além disso, usuários sofisticados poderão programar suas próprias consultas à Web — consultas *ad hoc* — exatamente como já o fazem a bancos de dados relacionais ou orientados a objeto.

Todos esses recursos agregados facilitam o desenvolvimento de aplicações mais bem elaboradas, com melhores e mais fáceis interfaces para o usuário, e com maior precisão das respostas das consultas.

1.3. Organização da Dissertação

O capítulo 2 é sobre a situação atual da consulta a dados na Web. São discutidos resultados de diversas pesquisas concernentes, enfatizando tanto suas vantagens quanto suas deficiências. Deste estudo, surgiram os elementos motivadores do desenvolvimento de nossa plataforma de consulta à Web.

O capítulo 3 trata dos fundamentos teóricos da plataforma *WebObjects*, especialmente o modelo padrão ODMG de banco de dados orientado a objeto.

A implementação da plataforma é discutida no capítulo 4, enfatizando a arquitetura, as estruturas de dados e os algoritmos utilizados.

O capítulo 5 é dedicado aos resultados experimentais, com considerações sobre o desempenho e a precisão das respostas das consultas testadas.

O capítulo 6 encerra o documento, com as nossas conclusões e as perspectivas do trabalho.

Adicionalmente, o documento possui 3 apêndices com informações que propiciam um melhor entendimento dos conceitos descritos neste trabalho. No apêndice A é definida, através da metalinguagem EBNF (*Extended Backus Naur Form*), a linguagem OQL/ODMG. O apêndice B mostra a BNF e os apontamentos sintáticos do subconjunto da linguagem implementado na versão atual do processador de consultas. No apêndice C são definidas em mais detalhes as instruções reconhecidas pelo interpretador de consultas (máquina virtual) *WebObjects*.

Capítulo 2

II. Linguagens de Consulta à Web: um Rápido Panorama

A Web oferece aos seus usuários um vasto repositório de informações, que contempla os mais variados temas e conteúdos. Dessa forma, é correto afirmar que nela é possível encontrar informações sobre qualquer tema de interesse, sendo necessário para isso que o usuário defina critérios de consulta que sejam capazes de filtrar e localizar os dados que atendam às suas necessidades.

Neste aspecto reside um dos grandes problemas relativos à exploração do conteúdo disponível na Web. As ferramentas de consulta mais utilizadas apresentam uma série de restrições e inconvenientes que as tornam insuficientes às necessidades dos usuários, principalmente quando comparadas às linguagens de consulta estilo-SQL dos SGBDs.

Antes de apresentar algumas soluções existentes para a consulta à Web, incluindo produtos e protótipos de pesquisa, o capítulo é iniciado com uma discussão das características da Web que geram impacto na concepção de uma solução para o problema da consulta.

2.1. Caracterização do Banco de Dados Web

A maior característica dos dados disponíveis na Web é a sua heterogeneidade, seja sob o aspecto de sua natureza, onde podemos encontrar desde simples arquivos de texto até dados multimídia, seja pela grande diversidade de plataformas computacionais ([SAV,2001], [RAJ,2001]). Grande parte dos documentos é hipertexto HTML, favorecendo a navegação por conteúdo através de *hyperlinks* [MEN,1997]. Além dos documentos HTML, cada vez mais bancos de dados relacionais se tornam disponíveis na Web. Em síntese, a Web é um banco de dados tanto semi-estruturado (documentos HTML) quanto estruturado (tabelas relacionais).

Outra peculiaridade importante do banco de dados Web está em seu volume gigantesco. Para dar uma idéia da dimensão da Web, a ferramenta de consulta, *Google*, mantinha

2.073.418.214 páginas[6] em seus índices. Estudos apresentados em [UHL,2001] apontam que a quantidade de páginas não indexadas — *Deep Web* — é cerca de 500 vezes maior. É importante enfatizar que estes números tratam apenas de documentos semi-estruturados.

Um outro aspecto importante a ser ressaltado é a distribuição física dos dados: uma miríade de plataformas diferentes, sob gerência diferente. Esta ausência de controle centralizado sobre os dados armazenados e publicados na Web é apontada em [BRI,1998] como uma das principais diferenças entre a Web e os bancos de dados convencionais.

Feitas estas considerações, torna-se claro que uma ferramenta de consulta a dados da Web deve ser suficientemente flexível para permitir o tratamento da diversidade da Web. Também, o requisito do desempenho continua premente, de modo que as consultas sejam processadas em um tempo aceitável pelos usuários.

2.2. Máquinas de Busca

Atualmente, as ferramentas de consulta à Web mais amplamente utilizadas são as chamadas máquinas de busca (*index servers)*, tais como *AltaVista* e *Google*. Máquinas de busca são *websites* que disponibilizam serviços de busca através de dois componentes básicos, o construtor de catálogos e a interface de consulta [KON,1996].

O construtor de catálogos existe para resolver o problema do desempenho das consultas face à dimensão da Web. Através de uma navegação exaustiva pelos documentos e sua estrutura de *hyperlinks*, o construtor de catálogos, normalmente implementado com *webcrawlers*[7] [PIN,1994], constrói/atualiza periodicamente um índice de palavras-chave para cada página da Web. A tarefa principal da interface de consulta é a pesquisa nos índices do catálogo. O processo de indexação é *off-line*, ou seja, um índice só é utilizado pela interface quando ele não está em processo de construção/atualização.

Esta estratégia agrega ao resultado das consultas um certo grau de imprecisão, haja vista que os catálogos *off-line* normalmente possuem informações desatualizadas/indisponíveis em relação à situação atual dos documentos da Web. No entanto, a utilização de catálogos *off-line* é a única estratégia viável em relação à dimensão e estrutura de armazenamento dos dados

[6] Este dado foi obtido em 17 de maio de 2002
[7] Agente que faz a análise de documentos da Web e seus *hyperlinks*, montando um índice para pesquisas

utilizados na Web [SPE,2000], pois uma consulta totalmente *on-line* teria um desempenho inaceitável para os usuários. Como é descrito no capítulo 4 deste documento, a *WebObjects* propõe uma solução híbrida que utiliza catálogos *off-line* e consultas *on-line* para resolver o problema da precisão das respostas.

As interfaces efetuam a consulta através da especificação de palavras-chave. O conjunto-resposta da consulta é constituído de URLs de páginas que atendem aos critérios de busca.

As máquinas de busca padecem de várias restrições importantes. Enumeremo-las:

- ✓ As interfaces oferecidas aos usuários têm um baixo poder de expressão pois possuem uma estrutura rígida e pré-definida para especificação das consultas, fazendo com que determinados critérios de filtragem de documentos não possam ser utilizados na formulação da consulta. Por exemplo, na especificação de uma consulta que busque localizar as páginas que contenham a palavra "WebOQL", não é possível solicitar a uma máquina de busca que selecione apenas aquelas páginas com data de publicação entre 2000 e 2001 ([MEN,1996], [LAK,1996], [MEN,1997]);
- ✓ Não é possível explorar a estrutura e a topologia dos documentos da Web [MEN,1997] [MIH,1997]. Consideremos a seguinte situação: em uma consulta que visa encontrar documentos técnicos sobre o carro Gol da *Volkswagen*, as palavras "Volkswagen" e "Gol" podem ser utilizadas como chaves de busca; no entanto, uma máquina de busca, com estas palavras-chave, poderia perfeitamente retornar a página de uma loja de eletrodoméstico, com uma promoção oferecendo aos participantes um carro Gol. Assim, seria interessante se pudéssemos restringir o acesso às páginas alcançadas do documento "http://www.volkswagen.com.br/dados_tecnicos/index.html"[8];
- ✓ Cada máquina de busca implementa sua própria interface, com suas particularidades, com evidentes desvantagens em relação a uma interface padronizada ([MEN,1997], [MIH,1997], [POL,1997]);
- ✓ A síndrome do "perdido no hiperespaço" ("*lost-in-hyperspace*") é mencionada em [MEN,1997]. O resultado de uma consulta é um conjunto de URLs. Grande parte do trabalho de busca é realizada pelo próprio usuário, que deve acessar cada URL para verificar se a informação é realmente a desejada. Assim, muita informação inútil é recuperada, sendo a análise da mesma um processo manual desgastante e tedioso.

[8] A página é fictícia.

- ✓ O aspecto dinâmico da Web não é levado em conta ([MEN,1996] [MEN,1997]). Por exemplo, uma URL recuperada pode apontar para uma página que se tornou inválida, ou que não existe mais;
- ✓ O conjunto de URLs retornado não obedece a critérios tais como ordenação e agrupamento, dificultando aos usuários a análise dos resultados [ARO,1998];
- ✓ Funções como manipulação de texto e data não são suportadas [LAK,1996].

Muita pesquisa tem sido realizada visando resolver os problemas descritos. Destacamos as seguintes características necessárias ([KON,1998], [GOL,2000], [ULM,2001]):

- ✓ Linguagem de consulta de alto nível com poder de expressão das linguagens estilo-SQL;
- ✓ Exploração de toda a potencialidade dos catálogos mantidos pelas máquinas de busca, possibilitando entre outras coisas: flexibilidade na consulta, utilização de funções e operadores externos e reestruturação da resposta;
- ✓ Integração de documentos semi-estruturados com dados estruturados (tabelas relacionais);
- ✓ Modelagem extensível e flexível da Web, visando tratar toda a diversidade;
- ✓ APIs para facilitar o desenvolvimento de aplicações-Web.

Segue-se a descrição de alguns protótipos de pesquisa.

2.3 Protótipos de Pesquisa

2.3.1 W3QS/W3QL

W3QS [KON,1998] é um processador de consulta que oferece uma linguagem estilo-SQL — W3QL — para exploração da estrutura e do conteúdo da Web.

As consultas de conteúdo são baseadas nos dados de um único nó do hipertexto, e resolvidas utilizando programas capazes de avaliar a cláusula `WHERE` da consulta para o tipo de documento a ser pesquisado. Por exemplo, a avaliação de documentos HTML é feita por um programa especial chamado `PERLCOND`.

Para resolver consultas de estrutura, a W3QS concebe a Web como um grafo direcionado, onde cada URL é um nó, e os *hyperlinks* entre nós são os arcos do grafo. Assim, as consultas

de estrutura são resumidas à exploração do grafo que representa a topologia dos documentos da Web.

A resposta padrão de qualquer consulta W3QL é uma tabela contendo a URL dos nós e os *hyperlinks* que satisfaçam a consulta. O resultado pode ser armazenado de diversas formas, dependendo da necessidade do usuário. Assim, é possível desde o armazenamento em um arquivo do sistema operacional, até o preenchimento de um formulário HTML que esteja preparado para receber a resposta.

Semanticamente, cada cláusula do comando de seleção tem uma função específica no contexto das consultas. A cláusula `SELECT` define a forma como o resultado deverá ser processado. A cláusula `FROM` especifica o grafo representativo da Web onde a busca deverá ser feita. Já a cláusula `WHERE` serve para impor as condições que devem ser aplicadas aos nós e arcos definidos na cláusula `FROM`, e também, as diretivas de busca para os programas responsáveis pela pesquisa. A última cláusula é específica da linguagem e define para o processador de consultas qual o procedimento a ser utilizado na pesquisa.

Como exemplo de aplicação da linguagem, o comando para localizar todos os artigos em formato *postscript* que foram escritos por Einstein é o seguinte:

```
SELECT cp n2/* result;
   FROM n1, l1, ( n2, l2 ), l3, n3;
   WHERE n1 IN {http://lycospro.lycos.com/lycospro-nojava.html};
         RUN learnform n1 cs76:0 IF n1 Unknown in LycosDof;
         FILL l1 AS IN LycosDof WITH query = "Einstein";
         l1: PERLCOND 'l1.content = ~/^FORM/I';
         n3: PERLCOND '( n3.format = ~/postscript/i ) &&
                      ( n3.content = ~/Einstein/i )';
         USING ISEARCHd -1 5000 -d 3;
```

No comando acima, a primeira linha determina que o resultado deve ser gravado no diretório do usuário, com o nome de `result`. Já a cláusula `FROM` informa que a estrutura da consulta é composta de: `n1`, representando o nó correspondente ao índice consultado (página de pesquisa do *Lycos*), `l1`, que mapeia o *hyperlink* para a página de resposta da consulta, `(n2,l2)`, definindo o documento retornado pela consulta ao *Lycos*, `l3`, representando os arcos (*hyperlinks*) para os documentos presentes no documento de resposta, e `n3`, determinando os nós de resposta.

A cláusula `WHERE` define a condição de filtro da consulta através de várias subcláusulas. A terceira linha traz a indicação do documento base da busca, que no caso, é um formulário Web que deve receber os parâmetros da pesquisa. A forma como os parâmetros devem ser tratados e informados ao formulário de entrada é definida nas linhas 4 e 5. As linhas 6, 7 e 8 especificam o programa de validação de conteúdo, bem como o filtro utilizado. Finalmente, a última linha restringe a quantidade de consultas a um máximo de 5000 conexões HTTP.

A W3QS é uma solução que permite expandir o horizonte da busca de dados na Web, possibilitando ao usuário uma linguagem de alto nível com grande poder semântico, o que viabiliza a elaboração de consultas mais bem elaboradas. Além disso, propõe um modelo cuja adição de novas funções de manipulação de dados é relativamente simples, o que passa a ser um ponto muito positivo diante da dinamicidade com que os padrões e formatos surgem na Web.

No entanto, a W3QS impõem uma série de inconvenientes. O primeiro deles está no fato de ser uma solução restrita ao ambiente UNIX, o que certamente reduz de forma considerável a gama de possíveis usuários da linguagem. Outro ponto que empobrece a solução é a necessidade do usuário conhecer programas e algoritmos utilizados na consulta, o que causa uma grande dependência da especificação da consulta com a implementação da solução, ou seja, não existe uma independência lógica. Finalmente, a última das restrições do modelo, e talvez a mais importante delas, está no fato de haver um distanciamento entre o padrão da linguagem SQL e a definida para a W3QL. Esta restrição faz com que haja uma maior dificuldade na assimilação e utilização da linguagem.

2.3.2 *WebLog*

WebLog [LAK,1996] é uma linguagem declarativa que oferece recursos para recuperação e reestruturação de informações disponíveis em documentos HTML. Através de comandos definidos por expressões lógicas, a *WebLog* é capaz de explorar conhecimentos incompletos do usuário na localização das informações a serem recuperadas.

Para concepção das consultas, a *WebLog* explora o fato das informações armazenadas em documentos HTML estarem delimitadas por *tags*, que formam grupos de informações fisicamente próximas dentro do documento. A figura 2.1 mostra um trecho de código HTML que evidencia claramente a presença dos *tags* e grupos de informações que eles delimitam.

```
<html>
  <head>
    <title>Homepage de Fábio Soares Silva</title>
    <meta http-equiv="Content-Type" content="text/html;
          charset=iso-8859-1">
  </head>
  <body bgcolor="#FFFFFF" text="#000000">
    <p>Exemplo de documento <b>HTML</b> </p>
    <p><a href="http://www.ufpb.br">Mestrado em Informática</a></p>
  </body>
</html>
```

Figura 2.1: Um trecho de código HTML

Na figura 2.1 observamos que o *tag title* delimita a informação contendo o título da página. Da mesma forma, os *tags body*, *head*, *p* e até *html* poderiam ser usados como delimitadores de outros grupos de informação.

Os vários grupos de informação são denominados *rel-infon* e constituem a unidade básica de manipulação da linguagem *WebLog*. Estas estruturas são lógicas e definidas na própria consulta pela indicação dos *tags* que irão delimitá-las. Em outras palavras, cada usuário pode definir quais *tags* serão considerados no agrupamento das informações de um documento HTML para composição da consulta, permitindo assim níveis diferenciados de granulosidade da informação. Assim, um usuário que efetua uma consulta no documento da figura 2.1 poderia definir que a *rel-infon* seria delimitada pelo *tag body*, enquanto outro poderia consultar o documento utilizando o *tag title* como delimitador das *rel-infons*.

Os *tags* presentes entre os delimitadores da *rel-infon* originam os atributos que a compõe. No exemplo da figura 2.1, se utilizarmos como delimitador da *rel-infon* o *tag a*, um dos atributos seria *href*, que é um dos *tags* definidos internamente ao *tag a*. Os valores dos vários atributos são definidos por literais contendo a informação presente no *tag* HTML.

A *WebLog* oferece uma série de predicados pré-definidos que funcionam como funções agregadas ao modelo para estender a capacidade da linguagem. Por definição, é permitido ao usuário criar novos predicados com a finalidade de atender necessidades específicas que venham a aumentar o poder de consulta da linguagem. Entre os predicados pré-definidos, poderíamos citar: *len* — recupera o tamanho de uma literal — e *substring* — obtém uma parte de uma literal.

A linguagem é formada tipicamente por expressões em lógica de primeira ordem no formato: `<url>[<rid>:<attr>→<val>]`. A `<url>` determina o documento a ser utilizado no processamento da consulta. O elemento `<rid>` é opcional e define o nome (identificador) da *rel-infon* na consulta. O atributo, que pode ser um delimitador de *rel-infon* ou um atributo manipulado na consulta, é definido pelo elemento `<attr>`. Finalmente, o elemento `<val>` define variáveis e constantes de onde valores são recuperados ou atribuídos para composição do resultado.

Seja a consulta:

```
resposta.html[ title→'citações', hlink→L, occurs→T ]
      ← http://www.unit.br[ hlink→L ], href( L,U ), U[ title→T ]
```

Funcionalmente, esta consulta gera um documento "respostas.html" contendo todos os *hyperlinks* existentes na página "http://www.unit.br". A sentença `title→'citações'` define o título do documento gerado. A especificação `hlink→L` indica que o documento gerado possui um atributo `hlink` contendo o valor obtido da variável `L` durante o processamento do comando. Uma consideração semelhante é válida para a definição `occurs→T`. Na composição do resultado, a sentença `http://www.unit.br[hlink→L]` indica que cada *hyperlinks* existente na página "http://www.unit.br" deve ser recuperado e atribuído à variável `L`. Na especificação `href(L,U)`, cada um dos *hyperlinks* atribuído a `L` deve ter a URL armazenada na variável `U` através da chamada ao predicado da linguagem `href`. Finalmente, o título do documento referenciado é recuperado e atribuído à variável `T` através da definição `U[title→T]`. Em cada *hyperlink* recuperado, os valores das variáveis `L` e `T` são utilizados na composição de um elemento da resposta.

A principal vantagem agregada pela *WebLog* é a flexibilidade oferecida pelo seu modelo de dados, que propicia ao usuário a capacidade de definir o nível de detalhamento da informação manipulada. No entanto, este mesmo modelo é extremamente complexo e de difícil entendimento, o que torna a especificação das consultas uma tarefa não trivial. Além disso, o modelo restringe, sem qualquer possibilidade de extensão, à manipulação de arquivos no formato HTML.

Por utilizar uma linguagem declarativa, a *WebLog* passa a ser uma alternativa diferente, mas não necessariamente melhor, ao padrão consagrado de recuperação e manipulação de

informações definidas por linguagens estilo-SQL. A linguagem é complexa e não apresenta a mesma rigidez sintática da maioria das linguagens utilizadas no meio comercial e científico, o que certamente é um fator capaz de dificultar a assimilação, e conseqüentemente a sua aceitação como ferramenta de consulta aos dados da Web.

2.3.3. *WebSQL*

Um grupo de pesquisa sediado na Universidade de Toronto propôs a *WebSQL* [MIH,1997], uma linguagem de consulta estilo-SQL capaz de efetuar pesquisas à informações disponibilizadas por meio de documentos HTML. Para isso, a Web foi modelada de acordo com uma estrutura relacional, onde todo o banco de dados pode ser definido por meio das relações virtuais `document` e `anchor`. Em outras palavras, toda a Web pode ser vista e consultada através de duas visões que capturam os dados dinamicamente do verdadeiro banco de dados, a própria Web. Os esquemas das duas relações virtuais são:

```
Document( url, title, text, length, modify )
Anchor( base, label, href )
```

A relação `document` tem a função de representar todos os documentos HTML presentes na Web, de modo que cada um deles é mapeado virtualmente para uma tupla da relação. Os atributos `url`, `title` e `text` são fundamentais para a pesquisa aos documentos, e representam respectivamente, o *uniform resource locator* (URL), o título e o conteúdo da página. Já os atributos `length` e `modify` são informações obtidas dos servidores da Web, e representam o tamanho do arquivo onde o documento está armazenado e a data que o mesmo foi modificado.

A relação virtual `anchor` modela as ligações existentes entre os diversos documentos HTML, ou seja, para cada *hyperlink* presente na Web, existe uma tupla mapeada na relação `anchor`. O atributo `base` serve para determinar a URL onde o *hyperlink* está localizado. Já o atributo `label` refere-se ao rótulo (título) do *hyperlink* no documento onde o mesmo está. Finalmente, a URL do documento para onde o *hyperlink* está apontando é representada pelo atributo `href`. Se existir a tupla {("http://www.unit.br", "cursos", "http://www.unit.br/cursos.htm")} na relação `anchor`, podemos afirmar que na página "http://www.unit.br" existe um *hyperlink* para a página "http://www.unit.br/cursos.htm" cujo rótulo é "cursos".

A linguagem *WebSQL* propicia aos seus usuários características de busca na Web extremamente interessantes. Além de suportar a consulta ao conteúdo dos documentos HTML, a *WebSQL* possui a capacidade de executar a navegação controlada pelos seus *hyperlinks*. Esta peculiaridade permite que as consultas sejam limitadas a parte da Web, de acordo com determinados padrões especificados na própria consulta.

Combinando todos os recursos presentes na linguagem, é possível elaborar uma série de consultas capazes de representar com maior fidelidade e correção a necessidade de pesquisa do usuário. Por exemplo, para encontrar todos os documentos que mencionem a palavra "Flamengo", deveríamos escrever a seguinte consulta:

```
SELECT d.url, d.title
   FROM document d such that d mention "Flamengo";
```

No comando acima, observamos na cláusula `FROM` a utilização do operador `mention` para filtrar os documentos que atendam à condição. Esta seleção também poderia ser definida através do operador `contains` especificado na cláusula `WHERE` do comando. O resultado desta consulta é uma outra relação contendo as URLs e títulos de todos os documentos que satisfaçam à condição de pesquisa.

Em outro exemplo, poderíamos encontrar todas as *home pages* que tratam de banco de dados e são endereçadas a partir do *website* da Universidade Tiradentes. Seja o comando:

```
SELECT a1.url, d2.title
   FROM document d1 such that "http://www.unit.br" ->*|=>* d1,
        anchor a1 such that base = d1,
        document d2 such that a1.href -> d2
   WHERE d2.text contains "banco de dados";
```

Com a utilização da relação `anchor`, torna-se possível escrever consultas que percorrem os *hyperlinks* da estrutura da Web, o que estende de maneira considerável os recursos de consulta.

De modo bem simplificado, a *WebSQL* pode ser entendida como uma biblioteca de classes Java, capaz de converter solicitações da linguagem em uma seqüência de instruções que podem ser submetidas a Web. A resposta gerada por alguma máquina de busca, ainda é processada pela biblioteca para geração do resultado final. Assim, a biblioteca de classes da

WebSQL pode ser utilizada por qualquer programa Java para criação de aplicações de consulta a dados na Web.

O mecanismo utilizado para concepção da *WebSQL* é bastante simples e permite a utilização de uma linguagem estilo-SQL, o que facilita exponencialmente a tarefa de pesquisa, pois transfere para processos automatizados, o que em um esquema de busca convencional é feito pelo usuário.

Outro ponto positivo desta solução está no fato do processador de consultas ser escrito em Java, o que propicia uma independência de plataforma em relação ao ambiente operacional, aumentando consideravelmente a possibilidade de utilização da ferramenta.

No entanto, apesar da brilhante idéia incorporada com a *WebSQL*, existem deficiências que poderiam ser aprimoradas através de uma modificação na estrutura de dados que modela a Web. Isto ocorre porque o modelo de dados da *WebSQL* é extremamente pobre e restritivo, dificultando possíveis extensões da linguagem no sentido de viabilizar a consulta de objetos da Web que não sejam necessariamente arquivos HTML.

Além disso, apesar de possuir uma linguagem mais próxima do padrão SQL do que a implementada no W3QS, a *WebSQL* ainda possui um distanciamento significativo do padrão, o que pode se tornar um fator complicador de sua aceitação.

2.3.4. *WebOQL*

Muito mais que apenas uma linguagem, a *WebOQL* [ARO,1997] é uma solução de consulta à Web que permite encarar a consulta por uma única perspectiva, pois define uma arquitetura, um modelo de dados e uma linguagem de consulta. Tal qual a *WebSQL*, a *WebOQL* permite que os usuários interajam com a Web especificando comandos em uma linguagem de mais alto nível, no caso, uma linguagem estilo-OQL.

A linguagem *WebOQL* é concebida sobre um modelo de dados cuja estrutura básica é uma árvore ordenada de arcos rotulados — *hypertree* — para a qual a estrutura de *hyperlinks* da Web pode ser mapeada. Em uma *hypertree*, cada arco representa um registro com campos definidos por literais, que é o único tipo básico disponível no modelo. Os arcos podem ser internos, quando utilizados para representar objetos estruturados, e externos, quando representam relacionamentos entre objetos.

A figura 2.2 mostra o exemplo de uma *hypertree*, onde os arcos internos são representados por setas contínuas e os arcos externos por setas tracejadas. A análise do exemplo mostra algumas características interessantes das *hypertrees*. A primeira delas está no fato de seus arcos poderem representar registros de diferentes estruturas. A outra característica fundamental é a possibilidade da estrutura suportar perfeitamente bem a representação de coleções, agrupamentos e ordenação. Esta propriedade das *hypertrees* permite que seja mapeada uma grande variedade de tipos de objetos, entre os quais podemos destacar: documentos HTML e tabelas relacionais.

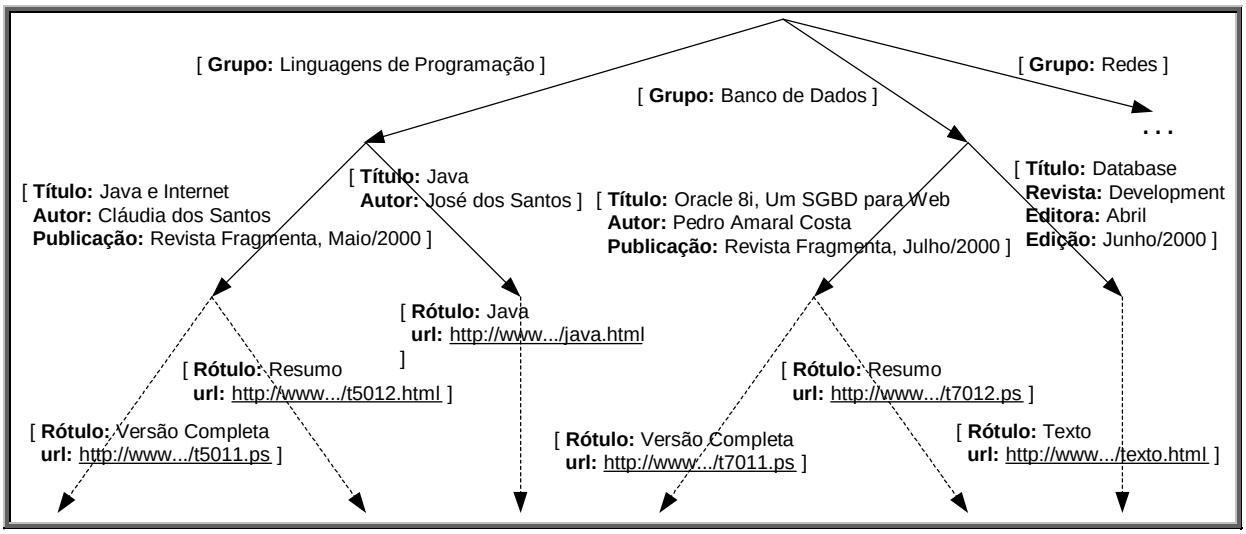

Figura 2.2: Um exemplo de *hypertree*

A *WebOQL* oferece um nível mais alto de abstração, a *web*. A finalidade da *web* é permitir a modelagem de um conjunto de *hypertrees* relacionadas, sendo definida por um par (t, F), onde t é uma *hypertree* e F é uma função de exibição (*browsing function*) que mapeia uma URL — identificador único do documento na Web — para uma *hypertree*. Os dois componentes podem ser vistos como o esquema da *web* e uma função que implicitamente define um grafo onde os nós são páginas e os arcos representam os *hyperlinks* entre estas páginas.

A *WebOQL* é uma linguagem baseada na sentença `SELECT…FROM…WHERE`. A cláusula `SELECT` define a estrutura da *hypertree* resultante da execução do comando. Já na cláusula `FROM` deve ser especificada a *web* que servirá de base para a consulta. Finalmente, a cláusula `WHERE` expressa a condição de busca aos dados.

Da mesma forma que a *WebSQL* [MIH,1997], a *WebOQL* possibilita a navegação controlada dentro das *hypertrees* através da especificação de padrões de navegação (NPs - *Navigation Patterns*), o que torna possível restringir o universo de busca especificado em uma consulta.

As NPs permitem definir a partir da raiz da *hypertree*, a estrutura dos caminhos que devem ser seguidos e considerados na composição do resultado da consulta. Esta especificação é feita através da cláusula `via` por meio dos operadores `^` e `>`, que denotam respectivamente, arcos internos e arcos externos.

Para analisarmos a forma e o poder expressivo da linguagem, vejamos a seguinte consulta:

```
Q: newWeb <- Select unique [ Nome:x, Url:x.grupo ] as schema,
                           [ y.título, y.autor ] as x.grupo,
             from x in htexemplo, y in x';
```

Este exemplo gera para cada grupo de pesquisa uma página contendo suas publicações, especificando para cada uma das publicações os campos autor e título. Ela também cria uma lista de todos os grupos de pesquisa com o *hyperlink* de cada grupo. A cláusula `unique` é semelhante ao `distinct` do SQL, pois garante que não existirão dois arcos com o mesmo rótulo na resposta, ou seja, arcos duplicados serão eliminados do resultado.

Em outro exemplo, podemos verificar o funcionamento dos padrões de navegação quando desejamos construir um índice completo para o documento "doc.html". Este, por sua vez, contém um grande número de *hyperlinks* internos entre suas partes, que são caracterizados pelo rótulo "próximo". Para isso, especificamos o seguinte comando:

```
Select [ x.url, x.text ]
   from x in browse("doc.HTML") via ( ^*[ Text ~ "Próximo" ]> )*
```

Nesta consulta, o padrão de navegação `^*` denota qualquer arco interno endereçado do documento "doc.html". A condição `Text~"próximo"` seleciona apenas os arcos cujo rótulo seja a literal "próximo".

A solução implementada para a *WebOQL* é muito semelhante à abordagem da *WebSQL*. Existe um *middleware* entre o usuário e a Web, que é responsável por receber as solicitações de consulta, convertendo-as em uma seqüência de ações capaz de capturar os dados da Web e modelar a resposta de acordo com a estrutura esperada pelo usuário. Este *middleware* é implementado através de uma API Java, o que possibilita uma série de vantagens, tais como a independência de plataforma e a facilidade de incorporação e utilização no desenvolvimento de aplicações.

A consulta propriamente dita é realizada por programas chamados *wrappers*, que exploram os recursos do sistema em busca dos dados solicitados. Basicamente, um *wrapper* mapeia uma fonte de dados específica em uma *hypertrees*.

A solução apresentada pela *WebOQL* apresenta como grande evolução um modelo de dados de grande poder expressivo para documentos estruturados e semiestruturados, o que possibilita o mapeamento de uma grande diversidade de objetos disponíveis na Internet, com diferentes estruturas e potenciais irregularidades.

Outra característica importante está no fato da *WebOQL* criar uma plataforma unificada de exploração da Web, que permite consulta de diversos tipos de documentos, viabilizando a criação de uma gama muito grande de ferramentas e aplicações.

A possibilidade de extensão do mecanismo de acesso aos dados através de programas *wrappers* também é um ponto positivo, pois permite entre outras coisas, o tratamento de qualquer objeto cujo conteúdo possa ser mapeado em uma estrutura de *hypertree*.

No entanto, a *WebOQL* sofre de alguns males já apresentados por outras soluções. O primeiro deles é um efeito colateral de uma estrutura de dados hierárquica utilizada por uma linguagem para banco de dados orientado a objetos, que impõe elementos adicionais para garantir a navegabilidade na estrutura dos documentos da Web. Este fato ocasiona um distanciamento do padrão OQL, o que dificulta a assimilação, e conseqüentemente a utilização da linguagem. Finalmente, o próprio autor aponta que muito ainda tem que ser realizado para garantir um bom desempenho à solução.

2.3.5. XML/QL

A proposta para a XML/QL [DEU,1999] define uma linguagem de consulta a dados em formato XML, cujo principal objetivo é oferecer recursos de extração, reestruturação e integração de dados contidos em documentos com este formato.

O padrão XML [MAC,1999] define um modelo de dados semiestruturado, cuja característica básica é a presença de *tags* que delimitam dados e registros. Diferentemente do HTML, os documentos XML podem possuir *tags* personalizados, que são representados entre os símbolos < >. Como exemplo, o *tag* `<NOME>` determina o início de uma informação de nome, enquanto que o *tag* `</NOME>`, determina o fim da informação. Todo *tag* de abertura de

informação deve possuir um outro de fechamento para que o documento XML seja considerado bem formado para processamento. A figura 2.3 mostra um exemplo de um documento XML composto de registros do tipo registro `Livro` cujos atributos são: `ano`, `título`, `autor` e `editora`.

```
<bib>
  <livro ano="1995">
     <!-- Um bom texto introdutório -->
     <titulo> Introdução aos Sistemas de Banco de Dados </titulo>
     <autor> Date </autor>
     <editora> Addison-Wesley </editora >
  </livro>

  <livro ano="1998">
     <titulo> Foundation for Object/Relational Databases </titulo>
     <autor> Date </autor>
     <autor> Darwen </autor>
     <editora> Addison-Wesley </editora>
  </livro>
</bib>
```

Figura 2.3: Um exemplo de um documento XML

Buscando explorar tal estrutura dos documentos XML, a linguagem XML-QL possui uma sintaxe onde a base de especificação é a sentença `WHERE...CONSTRUCT`, que gera como resultado outro documento XML com a estrutura e conteúdo definidos na cláusula `CONSTRUCTOR` do comando de seleção

Dessa forma, se no documento XML "http://www.a.b.c/bib.xml", que segue a mesma estrutura definida na figura 2.3, desejássemos encontrar todos os livros publicados pela editora "Addison-Wesley", deveríamos escrever o seguinte comando:

```
     WHERE <livro>
              <editora>Addison-Wesley</editora>
              <titulo> $t</titulo>
              <autor> $a</autor>
           </livro> IN "www.a.b.c/bib.xml"
        CONSTRUCT <resultado>
                     <autor> $a</>
                     <titulo> $t</>
                  </resultado>;
```

Na análise do documento, cada registro que atenda a condição onde o autor seja "Addison-Wesley", terá o valor definido para os campos `título` e `autor` atribuído as variáveis `t` e `a`, respectivamente. Estes valores serão utilizados na composição do resultado, que será um

documento XML onde os registros serão delimitados pelo *tag* `resultado`, e conterão os campos (*tags* XML) `autor` e `título`.

Entre as várias possibilidades de utilização da linguagem, duas aparecem com grande potencial. A primeira delas é a criação de um processador de consultas capaz de interpretar comandos da linguagem, e executar a consulta nos documentos especificados. Esta abordagem teria uma estrutura muito semelhante à adotada por *WebOQL* e *WebSQL*. Outra possibilidade seria a utilização da XML-QL como linguagem nativa de consulta para sistemas gerenciadores de banco de dados que adotassem a estrutura XML como modelo de dados.

Quando analisada apenas pelo aspecto de consulta a documentos XML, a linguagem atende satisfatoriamente aos requisitos necessários ao padrão. Ela possibilita a manipulação dos documentos através de comandos de alto nível e com grande poder semântico. No entanto, o fato da linguagem se distanciar de padrões consagrados de mercado, como SQL e OQL, pode ser uma deficiência da linguagem quando comparada com outras soluções que seguem tais padrões. O fator negativo não está na comparação das características da linguagem em si, mas na sua possibilidade de ser aceita como padrão de fato para consultas de documentos XML.

Se analisada como linguagem de consulta a dados na Web, a XML-QL possui algumas restrições importantes. As principais estão no fato de não tratar adequadamente a estrutura e a topologia dos dados da Web, além de restringir as possibilidades de consulta apenas a documentos XML, o que não atende a característica de diversidade dos dados da Web.

2.3.6. WSQ/DSQ

A WSQ/DSQ [GOL,2000] é uma proposta de arquitetura que permite combinar em uma mesma consulta as informações provenientes de bancos de dados convencionais — DSQ (*Database Supported Queries*) — com as da Web — WSQ (*Web Supported Queries*).

Esta solução manipula apenas tabelas relacionais, de modo que a Web é mapeada em termos de tabelas virtuais que tornam possível a definição de consultas através de uma única linguagem. As tabelas virtuais funcionam de maneira semelhante às visões relacionais, pois quando uma solicitação de consulta é realizada, os dados que comporão a resposta são capturados dinamicamente a partir de uma pesquisa na Web. Esta abordagem possibilita que o usuário escreva as consultas de maneira transparente, sem que seja necessário fazer distinção entre os dados que estão sendo capturados da Web e os dados obtidos das bases relacionais.

A Web é mapeada em termos de duas tabelas virtuais. São elas:

```
WebPages( SearchExp, T1, T2, ..., Tn, Url, Rank, Date )
WebCount( SearchExp, T1, T2, ..., Tn, Count )
```

Para a tabela virtual `WebPages`, o atributo `SearchExp` é uma literal de parametrização que representa a expressão de consulta. Utilizando o padrão de substituição `%n`, onde `n` é o número do atributo referente ao valor de `Tn`, uma expressão de consulta `"%1 perto de %2"`, com os valores de `T1="Sergipe"` e `T2="Alagoas"`, representa uma pesquisa a Web com a literal "Sergipe perto de Alagoas". Para uma mesma combinação de expressão de consulta e valores de parâmetros, podem existir várias linhas na tabela virtual que representam diferentes URLs que atendam ao critério de seleção. Os atributos `Url` e `Date` são retornados pela máquina de busca utilizada, enquanto que o atributo `Rank` define a ordem de importância para o qual a máquina de busca gerou as URLs resultantes.

A tabela virtual `WebCount` é estruturada de maneira semelhante a `WebPages`. No entanto, para uma determinada combinação do valor de `SearchExp` com os valores dos atributos `T1,...,Tn`, só existe uma linha na relação com o atributo `count` determinando o número de URLs que atendem a combinação.

A análise das tabelas virtuais `WebPages` e `WebCount` denota claramente a dinamicidade de estrutura e conteúdo destas tabelas. Assim, tanto o número de linhas quanto o número de colunas de cada uma delas podem variar de acordo com a consulta especificada pelo usuário.

O WSQ/DSQ permite a consulta de informações em várias máquinas de busca, sendo necessário para isso, à definição de uma tabela virtual para cada uma delas. Assim, se as máquinas de busca *AltaVista* e o *Google* podem ser utilizadas na pesquisa, devem ser definidas, por exemplo, as tabelas virtuais `WebPages_AV`, `WebCount_AV`, `WebPages_GL` e `WebCount_GL`, ou seja, um par de tabelas virtuais para cada máquina de busca utilizada.

Como o modelo de dados é implementado de acordo com uma estrutura relacional tradicional, a linguagem de acesso utilizada é o SQL padrão, com todos os seus recursos e funcionalidades. Normalmente, os SGBDs incorporam extensões ao SQL-ANSI[9] que podem

[9] Padrão definido para a linguagem SQL

também ser utilizadas como recurso de consulta. A definição das extensões permitidas na consulta irá depender do SGBD para o qual a solução WSQ/DSQ é incorporada.

Pressupondo a existência de uma tabela `States` — com os atributos `name`, `population` e `capital` — , seria possível especificar uma consulta que liste as capitais que aparecem na Web mais vezes que o seu estado da seguinte forma:

```
select s.capital, wc.count, s.name, ws.count
   from states s, webcount_av wc, webcount_av ws
   where s.capital = wc.t1 and s.state = ws.t1 and
         wc.count > ws.count;
```

O resultado processado por este comando mostra com bastante propriedade um dos problemas da pesquisa textual utilizando as máquinas de busca. Ele é decorrente do fato de palavras como "Salvador", que não estão somente associadas ao nome da capital baiana, poderem criar distorções nos resultados obtidos na computação da consulta [Gol,2000].

Em outro exemplo, poderíamos recuperar as URLs que mencionam cada estado e estejam entre as cinco primeiras referências tanto para o *AltaVista* quanto para o *Google*. Para isso, seria submetido o seguinte comando:

```
select s.name, a.url
   from states s, webpages_av a, webpages_gl g
   where s.name = a.t1 and s.name = g.t1 and a.url = g.url and
         a.rank <= 5 and g.rank <= 5;
```

Esta última consulta efetua a junção natural entre três tabelas, sendo que duas delas são virtuais. Assim, é possível o cruzamento de informações entre os dados retornados pela pesquisa de duas máquinas de busca com os dados relacionais oriundos da tabela `States`. Além disso, percebe-se a utilização transparente do atributo `Rank` como recurso para classificar o resultado recuperado das máquinas de busca.

Para definição das tabelas virtuais, e a conseqüente agregação da WSQ/DSQ, o SGBD precisa oferecer suporte a este tipo de recurso. Normalmente, as tabelas virtuais são definidas por meio de visões que capturam seus dados através de chamadas a funções criadas no próprio SGBD. Estas funções, geralmente escritas em C ou Java, devem interagir com as máquinas de busca no sentido de obter as informações da Web solicitadas pelo usuário na consulta.

Para resolver o problema da alta latência das solicitações aos dados da Web em relação à recuperação de dados em bancos de dados convencionais, o WSQ/DSQ propõe a utilização de uma técnica chamada interação assíncrona (*Asynchronous Iteration*). Basicamente, esta técnica consiste em utilizar uma área de armazenamento temporário em memória, para permitir que várias pesquisas a Web sejam disparadas e executadas simultaneamente. Assim, à medida que as várias solicitações simultâneas vão gerando seus resultados, a computação do resultado pode ser realizada. Desta forma, o WSQ/DSQ não precisa esperar que uma determinada pesquisa na Web seja finalizada para que uma outra possa ser iniciada.

A solução proposta para o WSQ/DSQ apresenta evoluções em vários aspectos. O primeiro deles está na possibilidade de especificar consultas em SQL padrão, sem nenhum tipo de artifício para incorporar características da Web na consulta, o que permite uma fácil assimilação por parte de quem já utiliza o SQL. O outro está na possibilidade de expandir o próprio SGBD, incorporando as tabelas virtuais como elemento nativo, viabilizando a utilização direta dos recursos do processador de consultas do SGBD. Além disso, esta estratégia permite o cruzamento de informações de várias máquinas de busca.

No entanto, a abordagem que cria tabelas virtuais diferentes para cada máquina de busca, não parece ser uma boa escolha, pois pode gerar dependência das consultas do usuário a uma determinada implementação. Por outro lado, o fato do recurso das tabelas virtuais não está presente em todos os SGBDs do mercado, pode ser um elemento que dificulte a utilização por parte dos usuários

Além dos aspectos positivos já comentados, a WSQ/DSQ é a primeira das soluções discutidas neste documento que possui uma proposta efetiva para o problema do desempenho das consultas, haja vista que este fator pode ser extremamente crítico quando analisarmos o universo da Web.

Como principal ponto negativo, podemos identificar um modelo de dados que por ser simples, torna-se muito restritivo em relação à possibilidade de tipos de documentos a serem consultados na Web. O modelo formado pelas tabelas virtuais `WebPages` e `WebCount`, deixa muito pouca possibilidade de extensão da solução para manipular outros tipos de dados, que não documentos HTML, além de oferecer poucos, ou nenhum, mecanismo de exploração da topologia e estrutura da Web.

Ademais, esta proposta não apresenta qualquer tipo de recurso para capturar e manipular os dados disponibilizados na Web, haja vista que a principal informação recuperada é a URL, sendo que a manipulação do conteúdo de uma URL retornada é feita sem o suporte da WSQ/DSQ.

2.3.7. *Squeal*

Tal qual a *WebSQL*, o *Squeal* [SPE,2000] propõe uma solução para a pesquisa aos dados HTML da Web mapeando seu conteúdo em termos de um esquema de banco de dados relacional, o qual pode ser consultado por meio de comandos especificados em termos da linguagem SQL. Para isso, a Web é concebida segundo as seguintes relações:

```
Page( url, contents, bytes, when )
Tag( url, tag_id, name, startOffSet, endOffSet)
Att( tag_id, name, value )
Link( source_url, anchor, destination_url, hstruct, lstruct )
Parse( url_value, component, value, depth )
```

A relação `Page` tem a função de representar todos os documentos semiestruturados presentes na Web, de modo que os documentos HTML capazes de serem consultados pela *Squeal* possuem tuplas representadas nesta relação. Os atributos `url`, `content`, `bytes` *e* `when` representam respectivamente, o *uniform resource locator* (URL), o texto associado ao documento, o tamanho em bytes da página e a data da última recuperação da página pelo *Squeal*.

Os *tags* HTML presentes em cada documento são descritos pela relação `Tag`, cujo relacionamento[10] com a relação `Page` é feito pelo atributo `url`. Para cada *tag* presente em um documento é gerado um identificador único armazenado no atributo `tag_id`. Adicionalmente, são guardadas para cada *tag* as seguintes informações: o seu nome (`name`), a posição de início do *tag* na página (`startOffset`) e a posição de final do *tag* (`endOffset`). Uma tupla válida para esta relação seria {("http://www.unit.br", 10, "IMG", 500, 720)}, indicando que no documento "http://www.unit.br" existe um *tag* "IMG" iniciado na posição 500 do documento.

[10] Chave estrangeira

A relação `Att` existe para representar os atributos existentes em cada *tag* HTML. Por exemplo, o *tag* HTML "IMG" possui um atributo "SRC" que poderia ser mapeado para a relação `Att`. Apesar de ser possível obter todos os *hyperlinks* de uma página através da pesquisa nas relações `Tag` e `Att`, a *Squeal* oferece aos seus usuários a possibilidade de uma consulta direta a este tipo de estrutura dos documentos HTML através da relação `Link`. Finalmente, a relação `Parse` traz informações sobre os dados da página na Web.

Se analisarmos com bastante frieza o esquema de dados proposto pelo *Squeal*, perceberemos que na verdade trata-se de um núcleo igual, ou na pior das hipóteses muito semelhante, ao proposto pela *WebSQL*, adicionando-se funcionalidades e detalhamento sobre as informações dos documentos HTML, ou seja, trata-se de uma extensão do esquema de dados *WebSQL*.

Tal como as idéias utilizadas por [GOL,2000], a *Squeal* define um processador de consultas que utiliza os recursos dos SGBDs para sua implementação. Assim, além de simplificar a solução, possibilita que os usuários especifiquem suas consultas em SQL padrão, ou apenas com as extensões agregadas pelo SGBD onde o *Squeal* está implantado.

Dessa forma, uma consulta que vise localizar os documentos da Web que contenham a palavra "Flamengo" e possuam pelo menos uma imagem, seria resolvida com a seguinte especificação:

```
select distinct p.url
   from page p, tag t
   where p.contents like '%flamengo%' and p.url = t.url and
         t.name = "IMG";
```

Se desejarmos localizar todos os *hyperlinks* existentes no documento da Web cuja URL é "http://www.unit.br", poderemos faze-lo especificando o seguinte comando SQL:

```
select anchor, destination_url
   from link
   where url_source = 'http://www.unit.br';
```

Para viabilizar a execução das consultas, o *Squeal* faz uso de uma técnica para capturar os dados da Web, decompondo-os e armazenando as informações em um banco de dados com o esquema proposto. Após os dados estarem armazenados no SGBD local, a consulta é submetida como uma solicitação normal ao SGBD. Para isso, o *Squeal* utiliza a técnica de

"*just-in-time database"* [STE,1998], onde os dados necessários à solução da consulta só são capturados e armazenados no SGBD local quando são efetivamente solicitados.

Esta estratégia, descrita com mais profundidade em [SPE,1998], possibilita que todos os recursos presentes no SGBD possam ser utilizados, sem que nenhuma exigência especial, tais como as tabelas virtuais definidas pelo WSQ/DSQ, seja imposta a quem deseja utilizar a solução.

Como ficou evidente, o mecanismo utilizado para concepção da *Squeal* é bastante simples, e potencializa a gama de usuários da solução, no sentido que viabiliza a utilização de comandos SQL tal qual o proposto pelo SGBD que o usuário está adaptado.

No entanto, apesar da simplicidade incorporada com o *Squeal*, este sofre dos mesmos males já apontados para a *WebSQL* por utilizar um esquema de dados relacional, que é extremamente pobre e restritivo, dificultando possíveis extensões da linguagem no sentido de viabilizar a consulta de objetos da Web que não sejam necessariamente documentos HTML.

Além disso, a solução não traz nenhuma preocupação com aspectos de desempenho associados à operação de busca dos dados da Web, com a subseqüente decomposição da estrutura para povoamento do banco de dados local. Por outro lado, a estratégia adotada pelo *Squeal* tem pouco compromisso com a dinamicidade que os documentos da Web são modificados, e em certos momentos, tornam-se indisponíveis.

2.3.8. *HtmSQL*

De maneira semelhante à proposta da XML-QL para documentos XML, a *HtmSQL* [LOT,2001] é uma linguagem estilo-SQL que busca oferecer aos seus usuários recursos de consulta e exploração do conteúdo de documentos HTML através da exploração da estrutura de *tags* deste tipo de documento.

Para que isso seja possível, a *HtmSQL* propõe uma conversão da estrutura dos *tags* HTML em uma estrutura multinível pela qual o conteúdo dos documentos possa ser acessado e referenciado através de uma consulta. Analisando uma tabela relacional, ou seja, uma estrutura em dois níveis — linhas e colunas —, as colunas poderiam ser acessados na estrutura multinível com a referência `row[0].column[1],` indicando o acesso à segunda coluna da primeira linha da tabela. De maneira análoga, os dados de um documento HTML

poderiam ser acessados com a referência `.table[0000].tr[0000].td[0002]`, denotando o conteúdo da segunda coluna (*tag* `td`), existente na primeira linha(*tag* `tr`), da primeira tabela (*tag* `table`) de um documento HTML.

Esta conversão já denota um distanciamento semântico entre o SQL, voltado para consulta em estruturas bidimensionais (relações), e a estrutura proposta pela *HtmSQL*, que propõe um esquema de dados com mais de duas dimensões.

Assim, um comando capaz de encontrar todos os *hyperlinks* presentes em um documento HTML "http://search.itworld.com:8765/query.HTML" que trate de "ejb" e foram publicados em dezembro seria:

```
select .table[0001].tr[0000]...td[0000].htxt[0000] AS linkText,
       .table[0001].tr[0000]...td[0000].text[0001] AS description,
       .table[0001].tr[0000]...td[0001].text[0000] AS date
   from http://search.itworld.com:8765/query.HTML
   withget qt = ejb
   where date matches '*dec*'
   order by date;
```

Nesta consulta, podemos perceber no argumento da cláusula `SELECT` a forma como a estrutura multinível para representação dos *tags* pode ser acessada. A cláusula `FROM` indica o documento HTML a ser utilizado na consulta. Neste caso, o documento recebe um parâmetro de consulta que é especificado pela cláusula `WITHGET`. A seleção dentro do documento dos dados que interessam é especificada na cláusula `WHERE`. Finalmente, a cláusula `ORDER BY` determina a ordenação a ser dada para o resultado final.

A *HtmSQL* é implementada por meio de uma API Java pela qual os usuários podem criar novas aplicações que façam uso dos seus recursos.

De modo geral, esta solução disponibiliza um recurso interessante associado a documentos HTML quando oferece a possibilidade de exploração, e até extração, do conteúdo disponível nesta vasta coleção de dados da Web. No entanto, a forma como a *HtmSQL* oferece este recurso impõe uma série de inconvenientes aos usuários. O primeiro deles diz respeito a uma estrutura de dados inadequada para uma linguagem estilo SQL, o que faz necessária à inclusão de outras funcionalidades à linguagem, distanciando-a do padrão. Ademais, a própria

estrutura de dados é extremamente complexa, tornando difícil à especificação de consultas que façam uso dos recursos da linguagem.

2.4. Síntese da Situação Atual da Pesquisa em Linguagens de Consulta à Web

Os dados disponíveis na Web constituem atualmente a maior e mais rica fonte de informação já produzida em todos os tempos. Quando bem utilizado, este indescritível mundo de diversidade de conteúdo pode ser um dos meios mais importantes para o desenvolvimento do conhecimento.

Para a plena utilização da Web, é necessário que exista um mecanismo automatizado que facilite o processo de consulta aos seus dados, possibilitando assim uma melhor utilização das informações.

De uma forma geral, qualquer mecanismo de consulta à Web deve atender aos seguintes requisitos:

- ✓ Automatizar o processo de consulta tratando dos problemas relacionados à utilização de catálogos *off-line*;
- ✓ Oferecer uma linguagem com grande poder de expressão, possibilitando a especificação de consultas com alta semântica;
- ✓ Seguir um padrão consagrado de busca de dados, tal qual SQL e OQL, para facilitar a utilização e a integração com outras ferramentas de desenvolvimento;
- ✓ Tratar a grande diversidade de padrões e formatos presentes nos dados da Web;
- ✓ Explorar convenientemente a topologia e a estrutura da Web;
- ✓ Ser independente de plataforma, podendo ser aplicada e utilizada por qualquer plataforma ou sistema operacional utilizado pelo usuário;
- ✓ Aumentar a capacidade de exploração dos catálogos de dados já disponíveis na Web;
- ✓ Oferecer recursos de recuperação e manipulação dos objetos disponibilizados na Web;
- ✓ Explorar os dados armazenados em bases de dados convencionais;
- ✓ Possibilitar a criação de aplicações e ferramentas que explorem o mecanismo de pesquisa, adequando a solução às necessidades do usuário.

Diante deste quadro, fica claro que o mecanismo de pesquisa implementado pelas máquinas de busca está muito distante de uma solução que atenda a necessidade atual de consulta aos

dados na Web. Esta, que certamente é a solução de consulta mais utilizada, dentre as muitas restrições, transfere boa parte da responsabilidade de busca para o próprio usuário.

Como mostrado neste documento, muitas propostas têm sido concebidas no sentido de criar uma solução objetivando atender aos requisitos de consulta aos dados da Internet. No entanto, todas elas resolveram parcialmente o problema, o que de certa forma foi o principal motivo de sua pouca aceitação pelos usuários da Web.

A tabela 2.1 evidencia através de uma comparação com os requisitos necessários, o quão distantes cada uma das soluções discutidas nesta seção está de uma solução que atenda às necessidades dos usuários.

Requisitos	**Soluções Existentes**							
	W3QS	**WebLog**	**WebSQL**	**WebOQL**	**XML/QL**	**WSQ**	**Squeal**	**HtmSQL**
Automatização da consulta tratando dos problemas relacionados à utilização de catálogos off-line	Médio	Baixo	Médio	Médio	Baixo	Médio	Médio	Médio
Poder de expressão da linguagem	Médio	Baixo	Alto	Alto	Alto	Médio	Alto	Alto
Adesão a padrões de linguagem (SQL ou OQL)	Baixo	Nenhum	Médio	Baixo	Baixo	Alto	Alto	Médio
Flexibilidade e extensibilidade do esquema de dados	Baixo	Médio	Baixo	Médio	Baixo	Baixo	Baixo	Baixo
Exploração da topologia da Web	Alto	Médio	Alto	Alto	Nenhum	Nenhum	Alto	Baixo
Exploração do conteúdo dos documentos da Web	Alto	Médio	Alto	Alto	Alto	Médio	Alto	Alto
Independência de plataforma	Baixo	Baixo	Alto	Alto	Baixo	Médio	Médio	Alto
Recuperação e manipulação dos documentos selecionados	Nenhum	Nenhum	Baixo	Baixo	Médio	Nenhum	Baixo	Baixo
Recuperação de dados de bases convencionais	Nenhum	Nenhum	Nenhum	Médio	Nenhum	Médio	Baixo	Nenhum
Recursos para criação de novas aplicações de acesso à Web	Baixo	Nenhum	Médio	Médio	Baixo	Médio	Médio	Médio

Tabela 2.1: Comparativo entre os protótipos de linguagens de consulta à Web

Capítulo 3

III. A Plataforma *WebObjects*

A concepção de um processador de consultas capaz de localizar e recuperar com precisão dados da Web, está diretamente associada à definição de uma linguagem pela qual os usuários possam especificar suas consultas com todos os detalhes necessários. De forma subjacente à linguagem, deve existir um modelo formal de dados. A riqueza semântica do modelo pode então ser devidamente explorada pela linguagem.

Com base nesta premissa, a plataforma *WebObjects* propõe uma solução que modela o espaço de busca da Web segundo um banco de dados orientado a objetos, com uma linguagem de consulta totalmente aderente ao modelo.

3.1. A Web como um Banco de Dados Virtual

Antes de conceber o universo de consulta à Web segundo um esquema conceitual de banco de dados genérico, flexível e extensível — viabilizando uma linguagem de consulta com alto poder de abstração e expressão —, é necessário tecer considerações sobre como as consultas deveriam ser processadas.

Processar uma linguagem de consulta à Web é uma tarefa de alta complexidade, que decorre principalmente dos seguintes fatores: a heterogeneidade de formatos, tipos e tecnologias utilizadas na publicação dos dados da Web, a gigantesca dimensão da sua base de dados, a sua distribuição física e a sua gerência distribuída.

Todos estes fatores conjugados impõem a um processador de consultas a necessidade de interagir com uma diversidade muito grande de serviços e dados, mantidos por uma enorme quantidade de servidores, com gerência e administração autônomas. Dessa forma, é praticamente impossível a criação de um catálogo de dados para a Web que se mantenha atualizado de maneira *on-line*. Ao invés disso, é recomendável encontrar outras soluções que garantam que as consultas retornem dados atualizados.

A solução encontrada para a plataforma *WebObjects* foi modelar a Web como um banco de dados absolutamente virtual. Como em [MIH,1997], [ARO,1998] e [GOL,2000], a recuperação dos dados segue o mecanismo utilizado pelos SGBDs relacionais para processar visões: a plataforma resolve as consultas em tempo real, com o auxílio de algum mecanismo de busca, tal como *Google* ou *AltaVista*. As URLs desatualizadas dos catálogos das máquinas de busca são então descartadas.

Esta abordagem nos permite abstrair toda a problemática inerente ao processo de criação de catálogos e exploração dos documentos da Web, sendo que o trabalho seria todo concentrado em aumentar a capacidade de utilização dos recursos já disponíveis. Assim, a *WebObjects* funciona como uma camada de mais alto nível, onde os usuários podem melhor explorar os recursos de consulta.

Mais precisamente, esta estratégia agrega à plataforma a capacidade de tratar da natureza dinâmica da Web, corrigindo em tempo real possíveis distorções dos resultados gerados pela utilização de catálogos *off-line*. Isto é possível porque os catálogos são utilizados apenas como ponto inicial da resolução das consultas, sendo validados pela plataforma, garantindo assim que informações desatualizadas não apareçam nas respostas das consultas.

Para especificação do banco de dados virtual e a conseqüente representação do espaço de busca na Web, a *WebObjects* segue fielmente o padrão de banco de dados orientado a objetos definido pelo comitê ODMG (*Object Database Management Group*), versão 3.0 [CAT,2000]. A próxima seção é dedicada a descrição deste padrão.

3.2. O Padrão ODMG para Banco de Dados Orientado a Objeto

O padrão ODMG é composto essencialmente de um modelo de objetos com sintaxe formal ODL (*Object Definition Language*), e uma linguagem formal de consulta, OQL (*Object Query Language*).

Este padrão oferece dois níveis de abstração: **interfaces** — modelam comportamento — e **classes** — modelam comportamento e estado. Além disso, define um conjunto de literais (tipos básicos) descrito na tabela 3.1.

As interfaces modelam abstratamente padrões de comportamento para os objetos, ou seja, dentro de um banco de dados não existem instâncias de objetos que seguem a estrutura de

uma interface. Ao invés disso, os objetos implementam o comportamento (operações) definidos por ela.

Natureza	Descrição	Tipo básico ODMG
Tipo atômico	Números inteiros	long, long long, short, unsigned long, unsigned short
	Números reais	float, double
	Valor booleano	boolean
	Literais	char, string
	Byte	octet
	Enumerado	enum< >
Tipo coleção	Conjuntos sem repetição	set< >
	Conjuntos com repetição	bag< >
	Listas	list< >
	Vetores	array< >
	Mapeamento de valores	dictionary< >
Tipos estruturados	Data (dia/mês/ano)	date
	Horário (hora/minuto/segundo)	time
	Data e horário	timestamp
	Intervalo	interval
	Registro	struct< >

Tabela 3.1: Tipos básicos disponíveis no padrão ODMG

Diferentemente das interfaces, as classes definem tanto comportamento quanto estado, e podem ter objetos instanciados no banco de dados. Os vários objetos de uma mesma classe são armazenados em coleções denominadas extensões (*extents*), pelas quais os usuários têm acesso aos seus dados. Fazendo uma analogia com o modelo relacional, os objetos de uma classe estão para as extensões como as linhas estão para uma tabela.

Para a especificação do estado de um objeto, ODMG oferece duas propriedades: **atributos** e **relacionamentos**.

Os atributos são definidos por um nome e um tipo, que pode ser básico ou um outro tipo definido pelo usuário. Neste último caso, o atributo é instanciado com **referências** (apontadores) para um objeto do banco de dados, sendo que a integridade referencial fica a cargo do usuário. São exemplos de atributos válidos (em sintaxe ODL):

```
attribute string nome;
readonly attribute set<string> dependentes;
attribute struct end{ string rua,
                      short numero,
                      string bairro } endereço;
attribute Professor coordenador;
```

Neste exemplo, o atributo `coordenador` só pode conter uma referência a um objeto da classe `Professor`. No entanto, se um objeto `Professor` apontado deixar de existir, a referência não será removida automaticamente pelo sistema.

Uma alternativa aos atributos com referências é definir relacionamentos entre classes. Neste tipo de associação, o padrão ODMG já agrega a restrição da integridade referencial como recurso para evitar ponteiros perdidos no banco de dados. Assim, quando um objeto que participa de um relacionamento é excluído, todas as referências a ele também o são. As seguintes definições são exemplos ODL de relacionamentos:

```
relationship set<Aluno> tem inverse Aluno::cursa;
relationship Curso cursa inverse set<Aluno>::tem;
```

Os exemplos mostram relacionamentos entre as classes `Curso` e `Aluno`, sendo o primeiro definido na classe `Curso`, e o segundo na classe `Aluno`. Lê-se: "um curso tem um conjunto de alunos, enquanto que um aluno cursa um único curso". `Tem` e `cursa` são pares de referências inversas.

O comportamento dos objetos de uma classe é especificado através da definição das **operações** (métodos) da classe. Em ODL, somente podem ser definidas **assinaturas** (tipo do valor de retorno, nome do método, parâmetros de entrada e saída, exceções) de métodos. São assinaturas ODL de métodos:

```
void processarMatricula();
long obterMediaGeral( long mat ) raises( AlunoNaoEncontrado );
boolean estaMatriculado( Aluno alu, Disciplina disc );
```

Os objetos pertencentes a uma classe são identificados de maneira unívoca no banco de dados por chaves geradas pelo sistema. Opcionalmente, os projetistas podem definir um identificador lógico formado por um conjunto de atributos da classe. Em ODL, a identificação lógica é especificada pela cláusula `key`.

Dois tipos de herança são suportados pelo padrão ODMG: herança de comportamento ou de interface, e herança de comportamento e estado (extensão de classe). Herança múltipla se dá das seguintes formas: uma interface pode herdar de várias interfaces; uma classe pode herdar de várias interfaces e de uma única classe.

3.3. Representando a Web como um Esquema ODL de Dados

A plataforma *WebObjects* utiliza a flexibilidade e o poder semântico do modelo de objetos ODMG, com sua sintaxe ODL, na definição de um esquema de banco de dados capaz de representar as informações publicadas na Web por meio de documentos HTML e tabelas relacionais. Este esquema oferece aos usuários um espaço de busca através do qual consultas OQL/ODMG (seção 3.4) podem ser formuladas.

Três diretrizes básicas nortearam a especificação do esquema de dados *WebObjects.* A primeira delas foi à necessidade de modelar objetos que permitissem a exploração do conteúdo e da estrutura de *hyperlinks*, como também possibilitassem a manipulação dos dados nos objetos de maneira fácil e intuitiva. A segunda diretriz foi incorporar nos objetos do esquema as características dos dados da Web que gerassem impacto à utilização da OQL, de modo que, a partir do esquema proposto fosse possível definir consultas no padrão da linguagem, sem qualquer tipo de personalização. A extensibilidade foi à última grande diretriz de projeto, haja vista que o objetivo era conceber um esquema capaz de tratar das informações armazenadas em documentos HTML ou tabelas relacionais, mas com suficiente abertura a extensões dos objetos modelados ou a criação de novos objetos que manipulem informações armazenadas sob outras estruturas de dados.

Pelo fato do padrão ODMG não definir um modelo formal para especificação do esquema de banco de dados em modo gráfico, e considerando a importância de tais diagramas no entendimento e utilização de um esquema, a *WebObjects* utiliza o diagrama de classes UML (*Unified Modeling Language*) [LAR,2000], cujas semelhanças conceituais com a ODL/ODMG permitem excelente compatibilidade entre as duas especificações. Assim, a figura 3.1 mostra o diagrama de classes UML que define o esquema de dados *WebObjects* para mapeamento do espaço de consulta da Web.

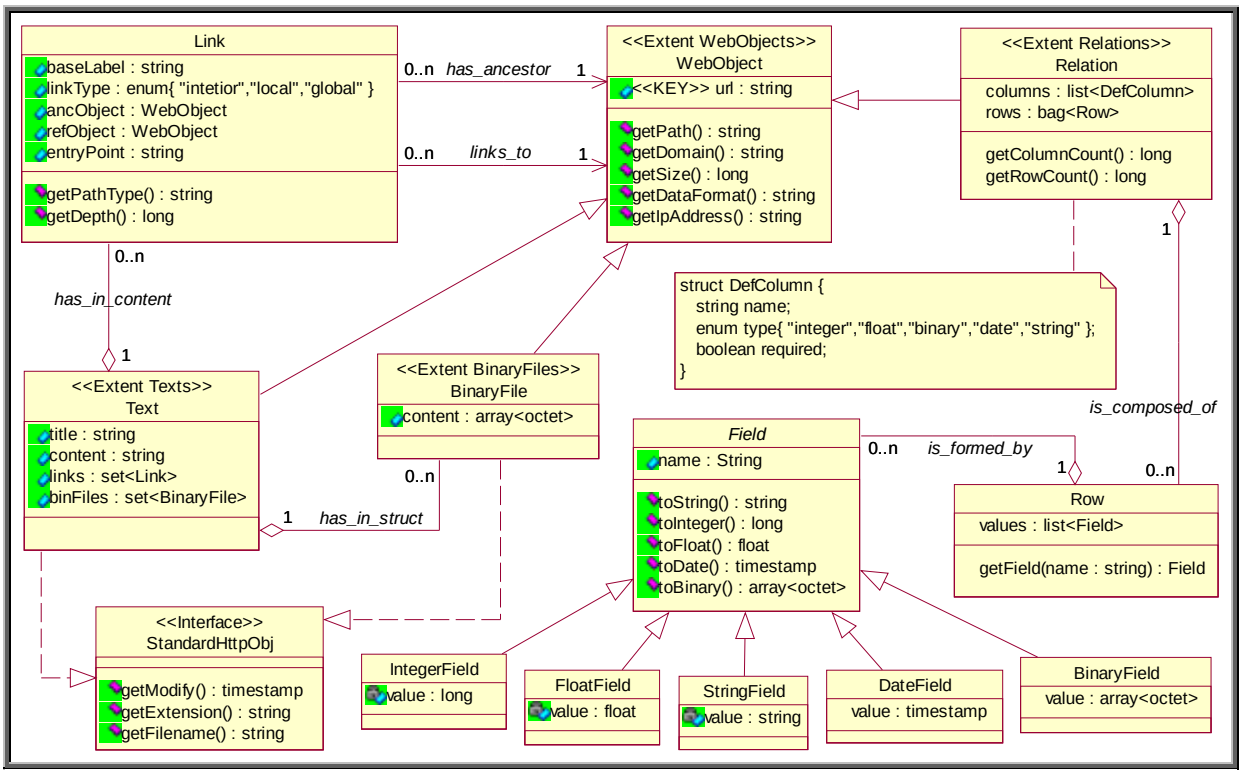

Figura 3.1: Esquema de dados UML para a plataforma *WebObjects*

A equivalente especificação ODL/ODMG do esquema de banco de dados *WebObjects* da figura 3.1 é mostrada a seguir:

```
/**
 * Definição do esquema de banco de dados para mapeamento do espaço de
 * busca da Web
 **/
module DbWeb{

  /**
   * Classe que modela o comportamento de todos os objetos consultados
   * pela plataforma, ou seja, as classes de manipulação de dados
   * específicos devem fazer herança desta classe. Como todas as
   * classes específicas herdam características de WebObject, seu
   * extent permite a consulta a qualquer objeto do banco de dados.
   **/
  class WebObject( extent WebObjects
                   key url ) {
    /**
     * Unified Resource Locator do objeto. Ex.: http://www.ufpb.br.
     * Funciona como identificador lógico dos objetos da Web. Assim,
     * cada objeto de dados tem que possuir uma URL definida para que
     * possa ser manipulado.
     **/
    attribute string url;

    /**
     * Caminho na estrutura hierárquica de Web até a localização do
     * objeto. Ex.: www.unit.br/fabiosoares/mestrado/
     **/
    string getPath();
```

```
    /**
     * Domínio onde o objeto da web está localizado. Ex.: www.unit.br
     **/
    string getDomain();

    /**
     * Tamanho em bytes ocupados pelo objeto da Web. Ex.: 1024 (bytes)
     **/
    long getSize();

    /**
     * Formato dos dados do objeto, que é recuperado a partir do
      * servidor. Ex.: text/html
     **/
    string getDataFormat();

    /**
     * Endereço IP do servidor onde o objeto está armazenado.
     * Ex.: 200.201.88.8
     **/
    string getIpAddress();
  };

  /**
   * Interface que define o comportamento padrão para objetos de dados
   * armazenados em arquivos do sistema operacional e acessados por
   * meio do protocolo http. São exemplos destes documentos, os
   * arquivos HTML e as imagens associadas às páginas.
   * As informações são recuperadas diretamente do web server.
   **/
  interface StandardHttpObj {
    /**
     * Recupera a data da última modificação do arquivo.
     **/
    timestamp getModify();

    /**
     * Recupera a extensão do arquivo de dados.
     **/
    string getExtension();

    /**
     * Recupera o nome do arquivo de dados.
     **/
    string getFilename();
  };

  /**
   * Classe que modela os arquivos binários (imagens e sons) dos
   * documentos HTML da Web. Os dados são modelados como uma coleção
   * de bytes, sendo responsabilidade das aplicações a interpretação
   * do significado dos dados.
   **/
  class BinaryFile extends WebObject: StandardHttpObj(
                                              extent BinaryFiles
                                              key url ) {
    /**
     * Vetor de bytes contendo a informação do arquivo.
     **/
    attribute array< octet > content;
  };
```

```
/**
 * Registro que define uma coluna de uma tabela relacional.
 **/
struct DefColumn {

  /**
   * Nome da coluna na tabela.
   **/
  string name;

  /**
   * Tipo da coluna. Os vários tipos existentes nos diversos SGBDs
   * do mercado devem ser mapeados segundo um destes tipos.
   **/
  enum type{ "integer", "float", "binary", "date", "string" };

  /**
   * Indica a obrigatoriedade da coluna, ou seja, se admite nulos.
   **/
  boolean required;
};

/**
 * Classe que representa as características gerais dos valores para
 * uma coluna de uma tabela relacional. As características
 * específicas de cada tipo são mapeadas para as suas subclasses.
 **/
class Field {

  /**
   * Define o nome da coluna a que o objeto pertence.
   **/
  attribute string name;

  /**
   * Retorna o conteúdo do objeto como uma string.
   **/
  string toString() raises( NotValidConversion );

  /**
   * Retorna o conteúdo do objeto como um timestamp.
   **/
  timestamp toDate() raises( NotValidConversion );

  /**
   * Retorna o conteúdo do objeto como um número real.
   **/
  float toFloat() raises( NotValidConversion );

  /**
   * Retorna o conteúdo do objeto como um array de bytes.
   **/
  array< octet > toBinary() raises( NotValidConversion );

  /**
   * Retorna o conteúdo do objeto como um número inteiro.
   **/
  long toInteger() raises( NotValidConversion );
};
```

```
/**
 * Classe instanciada para colunas que armazenam números inteiros.
 **/
class IntegerField extends Field {
  attribute integer value;
};

/**
 * Classe instanciada para colunas que armazenam campos data.
 **/
class DateField extends Field {
  attribute timestamp value;
};

/**
 * Classe instanciada para colunas que armazenam coleções de bytes.
 **/
class BinaryField extends Field {
  attribute array< octet > value;
};

/**
 * Classe instanciada para colunas que armazenam números reais.
 **/
class FloatField extends Field {
  attribute float value;
};

/**
 * Classes instanciadas para colunas que armazenam strings.
 **/
class StringField extends Field {
  attribute string value;
};


/**
 * Classe que mapeia uma linha de uma tabela relacional.
 **/
class Row {

  /**
   * Atributo que armazena os valores das várias colunas da linha.
   **/
  attribute list< Field > values;

  /**
   * Função que recupera o objeto Field que define a coluna passada
   * como parâmetro.
   **/
  Field getField( in string name ) raises( ColumnNotDefined );
};

/**
 * Classe que modela uma tabela relacional.
 **/
class Relation extends WebObject( extent Relations,
                                  key url ) {

  /**
   * Atributo que armazena a estrutura da tabela, ou seja, a
```

```
        * definição das suas colunas.
        **/
      attribute list< DefColumn > columns;

      /**
       * Atributo que armazena as linhas que compõe a tabela relacional,
       * ou seja, o seu conteúdo
       */
      attribute bag< Row > rows;

      /**
       * Retorna o número de linhas que existem na tabela.
       **/
      integer getColumnCount();

      /**
       * Retorna o número de colunas que compõe a tabela.
       **/
      integer getRowCount();
    };

    /**
     * Classe que modela um hyperlink existente em um documento HTML.
     **/
    class Link {

      /**
       * Armazena o rótulo associado ao hyperlink, na página onde ele
       * está localizado.
       **/
      attribute string baseLable;

      /**
       * Armazena a relação de localização entre o documento apontado
       * pelo hyperlink e o documento base. Pode ser:
       *   interior - trata-se do mesmo documento;
       *   local - são documentos diferentes de um mesmo servidor;
       *   global - trata-se de documentos de servidores diferentes;
       **/
      attribute enum linkType{ "interior", "local", "global" );

      /**
       * Apontador para o objeto onde hyperlinks está localizado.
       **/
      attribute WebObject ancObject;

      /**
       * Apontador para o objeto destino do hyperlink, ou seja, o
       * documento para onde a navegação pelo hyperlink irá.
       **/
      attribute WebObject refObject;

      /**
       * String com o rótulo HTML de hyperlinks que apontam para um
       * ponto específico do documento que não o início.
       **/
      attribute string entryPoint;

      /**
       * String com o caminho percorrido entre o objeto base da coleção
       * de hyperlinks até o documento de destino.
```

```
        * Ex.: local->local->global
        **/
       string getPathType();

       /**
        * Retorna o número de documentos acessados entre o objeto base e
        * o destino do hyperlink.
        **/
       integer getDepth();
     };

     /**
      * Classe que mapeia documentos texto com estrutura de hyperlinks.
      */
     class Text extends WebObject: StandardHttpObj( extent Texts
                                                    key url ) {

       /**
        * Armazena o título do documento
        */
       attribute string title;

       /**
        * Armazena o conteúdo do documento, ou seja, o conjunto de
        * palavras que compõem o seu texto.
        */
       attribute string content;

       /**
        * Armazena a coleção de todos os hyperlinks acessados direta ou
        * indiretamente a partir do documento
        */
       attribute set< Link > links;

       /**
        * Armazena a coleção de todos os objetos binários (imagem, som,
        * etc) associados ao documento
        */
       attribute set< BinaryFile > binFiles;
     };
   }
```

Para ilustrar a pertinência do esquema de dados *WebObjects*, imaginemos que a Web fosse restrita a um único *website* estruturado em árvore, como na figura 3.2. A instância do banco de dados para o *website* segundo o esquema ODL definido nesta seção e o equivalente esquema UML da figura 3.1 é mostrada na figura 3.3. O conteúdo do esquema se encontra unicamente na Web. Mais precisamente, nenhum dado da Web é copiado para o espaço local à *WebObjects*, ou o banco de dados *WebObjects* é virtual.

O conteúdo da Web é definido pelas extensões virtuais `WebObjects`, `Relations`, `Texts` e `BinaryFiles` das classes `WebObject`, `Relation`, `Text` e `BinaryFile`, respectivamente.

Estas três últimas classes são subclasses da ancestral `WebObject`, o que implica que a extensão desta última compreende todo o espaço de busca da Web[11].

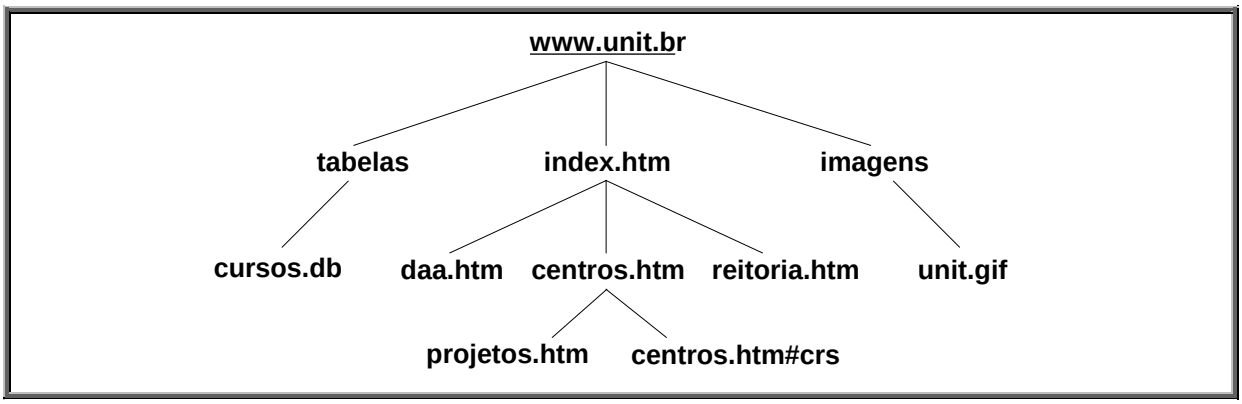

Figura 3.2: Esquema de uma estrutura de objetos de um *website* da Web

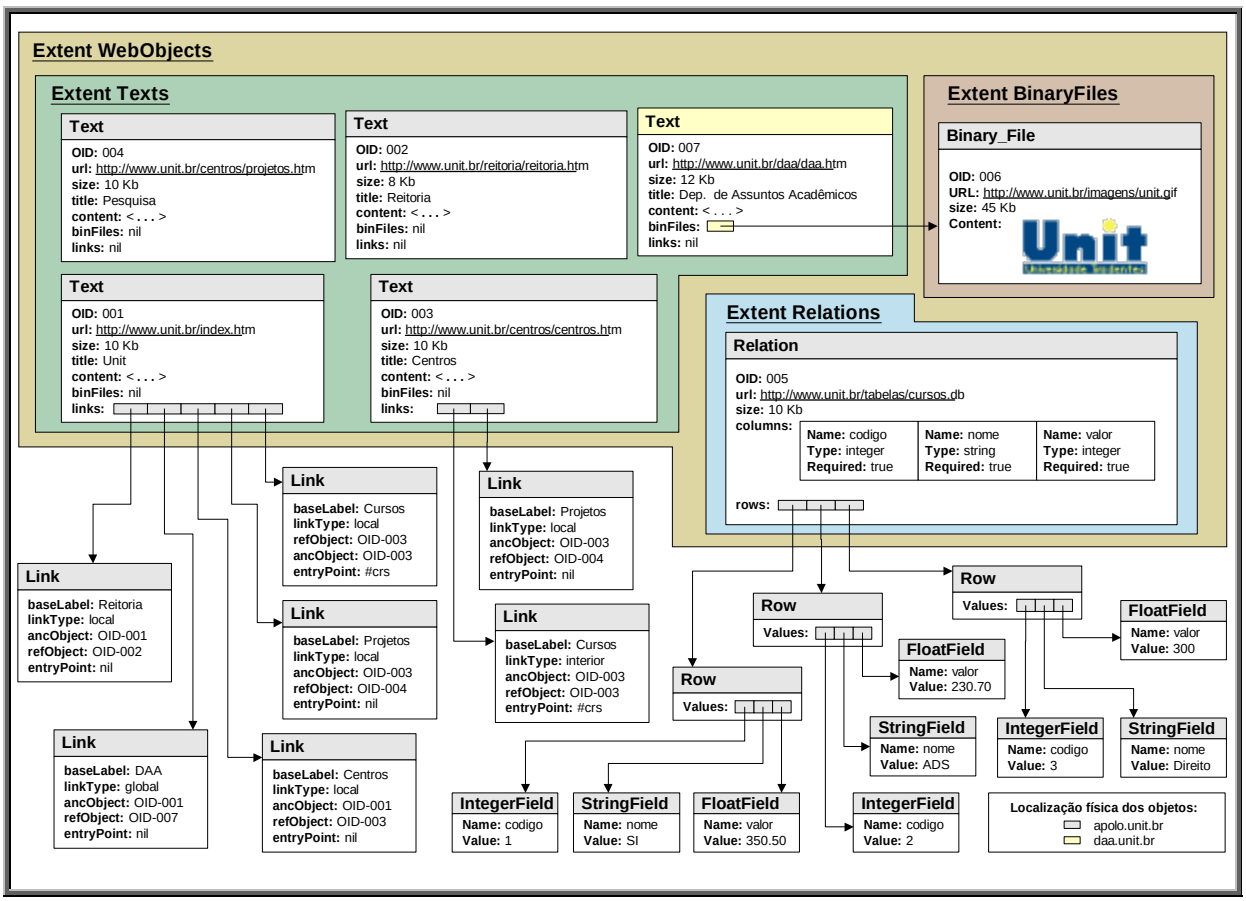

Figura 3.3: Uma instância do banco de dados virtual *WebObjects*

Outra importante peculiaridade do esquema de dados proposto está no fato de ele não distinguir a localização física dos objetos para a composição do espaço de consulta do usuário, ou seja, a Web é vista como um único banco de dados. Esta característica pode ser

[11] Com a ressalva que os objetos ou são documentos HTML ou são tabelas relacionais.

observada na figura 3.3 com os objetos "www.unit.br/index.htm" ("OID-001") e "www.unit.br/daa/daa.htm" ("OID-007"), que pertencem à mesma extensão, embora estejam localizados em diferentes servidores. A legenda da figura explica que objetos com tarja cinza estão armazenados no servidor "apolo.unit.br", enquanto que os de tarja amarela estão no servidor "daa.unit.br"[12].

As ligações entre as classes são definidas em termos de atributos e não de relacionamentos, já que a validação da integridade referencial incorporada pelos relacionamentos ODMG não precisa ser garantida. Isto acontece porque a instância do banco de dados é virtual e *read-only*, ou seja, não sofre manutenção por meio da plataforma.

A classe `WebObject` define as características gerais — atributos e comportamento — de qualquer objeto da Web, e através dos mecanismos de herança e polimorfismo, a classe é estendida para definir os objetos especializados. Este fato garante a extensibilidade ao esquema, haja vista que a adição de novas estruturas de publicação de dados pode ser conseguida com a extensão desta classe.

O atributo fundamental para a classe `WebObject` é a `url`, cuja principal função é identificar univocamente todos os objetos acessados pela plataforma. Assim, páginas HTML, arquivos de imagem, tabelas relacionais, ou qualquer outro objeto deve possuir um valor válido para este atributo, tornando viável a localização e manipulação dos seus dados. Além do atributo `url`, a classe oferece alguns métodos que aplicados ao documento HTML "projetos.htm" (figura 3.2) — com identificador físico "OID-004" (figura 3.3) — recuperam as seguintes informações:

- ✓ `getPath()` => retorna "www.unit.br/centros/", que descreve a localização do documento na estrutura hierárquica da Web;
- ✓ `getDomain()` => devolve "www.unit.br", ou seja, o domínio do servidor de onde o documento está armazenado;
- ✓ `getSize()` => recupera 1064, representando o tamanho (quantidade de bytes) do objeto;

[12] Obviamente, as ‘tarjas’ não fazem parte do banco de dados.

- ✓ `getDataFormat()` => retorna "text/html", ou seja, o tipo de arquivo recuperado do *web server* para o objeto;
- ✓ `getIpAddress()` => recupera "200.241.51.5", que define o endereço IP de onde o objeto está localizado.

Com o objetivo de viabilizar o acesso a bases de dados relacionais, o esquema *WebObjects* propõe uma solução capaz de mapear toda tabela relacional para a classe `Relation`. A estrutura e o conteúdo da tabela são representados através dos atributos `columns` e `rows`, respectivamente. Além destes, a classe `Relation` disponibiliza os métodos `getColumnCount()` — recupera o número de colunas da tabela — e `getRowCount()` — obtém o número de linhas da tabela.

O atributo `columns` é modelado como uma coleção de registros que define a estrutura básica de cada coluna. Para isso, cada registro armazena as informações sobre o nome, o tipo e a obrigatoriedade das colunas nos atributos `name`, `type` e `required`, respectivamente. O campo `type` de cada coluna é definido com tipos genéricos — `integer`, `float`, `binary`, `date` e `string` — para os quais os tipos de dados específicos dos vários SGBDs podem ser mapeados. Assim, uma coluna de uma tabela *Oracle* com o tipo `number(10)` seria mapeada para um tipo `integer`, enquanto que uma coluna `varchar2(40)` seria mapeada para um tipo `string`.

As linhas de uma tabela são representadas pelo atributo `rows`, cuja composição se dá através de uma coleção de objetos da classe `Row`, agregada à classe `Relation`. Um objeto da classe `Row` é composto por uma coleção de objetos — atributo `values` — da classe `Field`, ou uma das subclasses `BinaryField`, `IntegerField`, `FloatField`, `StringField` e `DateField`. A idéia para mapeamento dos dados das colunas nas subclasses de `Field` é o mesmo do utilizado para o tipo de coluna, ou seja, os valores armazenados nos vários SGBDs do mercado serão instanciados adequando-se a uma das subclasses de `Field`.

O acesso aos valores das colunas de uma relação pode ser feito através do atributo `values`, ou por uma chamada ao método `getField()` passando o nome da coluna a ser recuperada. Além disso, os objetos da classe `Field` permitem a conversão de tipos através de chamadas aos métodos `toInteger()`, `toString()`, `toDate()`, `toFloat()` e `toBinary()`.

Para ilustrar a forma como os dados relacionais são mapeados na instância do banco de dados *WebObjects*, verifiquemos o objeto cuja URL é "http://www.unit.br/tabelas/cursos.db" (figura 3.2) — identificador "OID-005" (figura 3.3) — que representa uma tabela "cursos" com estrutura e conteúdo definidos na figura 3.4.

Column	Null?	Type
CODIGO	NOT NULL	NUMBER(3)
NOME	NOT NULL	VARCHAR(40)
VALOR	NOT NULL	NUMBER(6,2)

(a)

Código	Nome	Valor
001	SI	350.50
002	ADS	230.70
003	Direito	300.00

(b)

Figura 3.4: Estrutura (a) e conteúdo (b) da tabela cursos

Objetivando modelar o comportamento padrão de objetos da Web que são publicados em arquivos do sistema operacional com acesso via protocolo HTTP — classes `Text` e `BinaryFile` —, a plataforma define a interface `StandardHttpObj`, cujos métodos obtêm informações sobre o arquivo de dados onde o objeto está armazenado. São eles: `getModify()` — recupera a data da última modificação —, `getExtension()` — obtém a extensão do arquivo — e `getFilename()` — recupera o nome do arquivo.

Dentre todas as classes da plataforma *WebObjects*, a classe `BinaryFile` é a que possui menos recursos de manipulação. O único atributo desta classe é do tipo coleção, mais precisamente um vetor de bytes. Cabe aos aplicativos a interpretação da seqüência de bytes para que a informação possa ser exibida ao usuário ou manipulada por um programa. Uma instância desta classe pode ser observada na figura 3.3 pelo objeto de identificador "OID-006" e cuja URL é "http://www.unit.br/imagens/unit.gif".

A classe `Text` define documentos texto com estrutura de *hyperlinks*, ou documentos HTML. Os atributos específicos desta classe são: `title` (título do documento), `content` (conteúdo do documento), `links` (coleção de *hyperlinks* que podem ser acessados a partir do documento) e `binFiles` (arquivos binários, tais como imagens e sons, vinculados ao documento). O atributo `links` é modelado como uma coleção de objetos da classe `Link`, que define as características de cada *hyperlink* dos documentos e existe apenas como uma agregação da classe `Text`, ou seja, seus objetos só são instanciados como parte dos objetos `Text`.

3.3.1. Links Redundantes

Na concepção do atributo `links` da classe `Text`, a redundância foi um importante recurso utilizado para facilitar a especificação das consultas por parte dos usuários. Assim, este atributo possui a coleção de todos os objetos que podem ser alcançados de um dado objeto texto, mesmo que não seja de maneira direta. Se analisarmos um documento HTML "A" onde existe um *hyperlink* para um outro documento "B", e neste documento "B" existe um *hyperlink* para um documento "C", o atributo `links` do objeto representativo de "A" conterá tanto um apontador para o objeto "B", quanto um apontador para o objeto "C".

Na figura 3.3, podemos observar que o documento "http://www.unit.br/index" ("OID-001") possui objetos da classe `Link` para os documentos "centros.htm" ("OID-003"), "daa.htm" ("OID-007"), "reitoria.htm" ("OID-002"), "projetos.htm" ("OID-004") e "centros.htm#crs" ("OID-003"). Os dois últimos objetos são acessados indiretamente a partir do documento "index.htm", ou seja, para atingirmos o documento "projetos.htm" a partir de "index.htm", é necessário passarmos pelo documento "centros.htm".

Outro aspecto importante a ser analisado é o fato do documento "centros.htm" ("OID-003") possuir dois objetos da classe `Link` associados a ele, com diferenças quanto à localização e ao tipo dos *hyperlinks*. Esta diferença denota a existência de *hyperlinks* que apontam para um ponto interior ao próprio documento, ou seja, são internos e representados por rótulos HTML que identificam o ponto de acesso à página. No exemplo da figura 3.3, o primeiro está localizado no documento "index.htm" ("OID-001") e aponta para o início do documento "centros.htm". O segundo *hyperlink* ("centros.htm#crs") está localizado no próprio documento "centros.htm" e aponta para o local do documento identificado pelo rótulo (*tag* HTML) "crs".

A redundância utilizada pela plataforma *WebObjects* é fundamental para criação de uma linguagem de consulta à *Web* que não utilize recursos complexos como recursividade, ou seja, de fácil utilização, e que esteja em conformidade com o padrão OQL proposto pelo consórcio ODMG [CAT,2000]. Por exemplo, se desejássemos explorar a estrutura topológica da Web encontrando todos os *hyperlinks* de documentos localizados no servidor "apolo.unit.br", endereçados a partir do documento "http://www.unit.br/index.htm", e que estivessem apontando para documentos inválidos, precisaríamos agregar à linguagem OQL recursividade ou elementos adicionais que tratem da exploração dos *hyperlinks*, o que ocasionaria um distanciamento do padrão, e conseqüentemente, um afastamento dos objetivos da *WebObjects*.

Agregar, através da redundância, este tratamento ao esquema de dados, torna possível utilizar uma linguagem que atende ao padrão OQL e permite percorrer e consultar facilmente os documentos hipertexto e sua topologia, resolvendo assim o problema de pesquisa para uma boa parte das informações disponíveis na Web.

A mesma característica de um banco de dados virtual disponibilizado apenas para operações de consulta (*read-only*) já mencionada para justificar a ausência dos relacionamentos ODMG, justifica a ampla utilização da redundância para representação do atributo `links` da classe `Text`, haja vista que ela não existe fisicamente, mas somente na montagem dos resultados para o usuário e na forma como o mesmo concebe suas consultas. No entanto, esta característica irá impor uma série de requisitos e peculiaridades na implementação da plataforma, cuja discussão mais detalhada será realizada na seção dedicada à descrição da arquitetura da plataforma.

Para análise dos vários atributos existentes na classe `Link`, foram utilizadas na figura 3.3 as instâncias da classe associada, através do atributo `links`, ao objeto cuja URL é "http://www./unit.br/index.htm" ("OID-001"). A instância escolhida é aquela cujos valores para o `baseLabel` é "Projetos" e para o `refObject` é "OID-004".

O atributo `refObject` é um apontador que endereça o documento de destino do `Link`, ou seja, o objeto que será acessado em uma navegação pelo *hyperlink*. Já o atributo `ancObject` (abreviação de *ancestor Object*) de um `Link` endereça o objeto onde o *hyperlink* está efetivamente localizado. No exemplo proposto, o valor do `refObject` é "OID-004", indicando que a navegação pelo *hyperlink* irá acessar o documento identificado na Web pela URL "http://www.unit.br/centros/projetos.htm". Sempre enfatizando que o atributo `links` é uma coleção de todos os objetos acessíveis direta ou indiretamente a partir de um documento texto, o atributo `ancObject` do exemplo possui o valor "OID-003", determinando que o *hyperlink* está localizado no documento HTML cuja URL é "http://www.unit.br/centros/centros.htm". Esta informação nos permite identificar entre outras coisas, que o acesso ao objeto referenciado é indireto. O texto associado ao *hyperlink* no documento onde ele está localizado na Web é obtido através do atributo `baseLabel`.

Para [MEN,1997], a capacidade de exploração controlada da estrutura de *hyperlinks* é fundamental para qualquer mecanismo de consulta à *Web*. Por isso, além dos recursos já

citados, a plataforma *WebObjects* oferece o atributo `linkType` e os métodos `getPathType()` e `getDepth()`.

O atributo `linkType` representa a relação de localização entre o objeto `Text` e o objeto endereçado pelo atributo `refObject` de seus *hyperlinks*. Os valores permitidos para o atributo são:

- ✓ **Interior:** representado pela literal "interior", indica que o objeto endereçado pelo *hyperlink* está no mesmo documento que o objeto;
- ✓ **Local:** representado pela literal "local", indica que o objeto endereçado pelo *hyperlink* está no mesmo servidor do objeto;
- ✓ **Global:** representado pela literal "global", indica que o objeto endereçado pelo *hyperlink* está em um servidor diferente daquele onde o objeto está.

Na figura 3.3, o *hyperlink* "Cursos" do objeto "www.unit.br/index.htm" ("OID-001") possui para o `linkType` o valor "local", indicando que os dois objetos estão armazenados no mesmo servidor mas em documentos diferentes. No mesmo `Link`, observamos que o valor retornado pelo método `getPathLink()` — recupera a relação de localização entre o objeto e os *hyperlinks* acessados até o objeto referenciado — é "local;interior", indicando que entre o objeto `Text` e aquele apontado pelo *hyperlink* existe um endereçamento indireto, de modo que para acessar o objeto referenciado, devem ser acessados dois objetos intermediários, sendo um deles local e outro interior. Neste mesmo exemplo, o valor obtido com a execução do método `getDepth()` — recupera a quantidade de nós acessados até o objeto destino — é 2.

Finalmente, os *hyperlinks* que apontam para um ponto específico do documento são mapeados pela classe `Link` no atributo `entryPoint`. Quando o destino for o início da página, este atributo possui o valor nulo (*nil*), caso contrário, possui o rótulo HTML que representa o local do documento onde ele deve ser acessado. Na figura 3.3, o valor do `entryPoint` para o *hyperlink* definido pela URL "http://www.unit.br/centros/centros.htm#crs" é "crs".

3.4. Consultando a Web com a Linguagem OQL

Sobre o esquema de dados orientado a objeto, ODL/ODMG, da plataforma *WebObjects*, os usuários podem explorar o conteúdo e a estrutura de *hyperlinks* da Web através de consultas

especificadas por comandos que seguem estritamente a sintaxe OQL (*Object Query Language*) do padrão ODMG [CAT,2000].

O principal alicerce da linguagem OQL é a sua aderência ao modelo de dados ODL/ODMG. Oferecendo uma linguagem estilo SQL-99 [SQL,1999] a OQL agrega o tratamento dos conceitos da orientação a objetos. O apêndice A traz a descrição formal da linguagem OQL, utilizando a metalinguagem EBNF (*Extended Backus Naur Form*).

Uma vez conhecida à sintaxe da OQL, a próxima seção é dedicada a uma descrição semântica da linguagem, mostrando como os comandos são especificados no sentido de recuperar informações de uma base de dados orientada a objetos.

3.4.1 Forma Geral de uma Consulta OQL

A base para especificação das consultas OQL é a sentença `SELECT...FROM...WHERE`, definindo, respectivamente, as informações a serem recuperadas, a origem destas informações, e a condição de seleção para a consulta.

Na cláusula `SELECT` é especificado quais informações devem compor o resultado do comando de seleção, e de que forma elas estão organizadas no resultado. Para isso, é permitido definir três tipos de elementos na cláusula `SELECT`: constantes, itens de dados e expressões. As constantes são definidas como literais válidas da linguagem para os tipos básicos. São exemplos de constantes OQL as seguintes literais: "Mestrado" (`string`), 'A' (`char`), 10 (`integer`), '01/01/2001' (`date`) e '01/05/2001 15:04:10' (`timestamp`). Os itens de dados são os elementos capturados a partir da seleção dos atributos e métodos que compõe os objetos recuperados. Por fim, as expressões são elementos que envolvem constantes e itens de dados em operações válidas da linguagem, como por exemplo, uma divisão.

Seja o comando OQL:

```
select 20, w.url, w.getSize() * 1.1
    from WebObjects w;
```

Funcionalmente, o exemplo recupera um `bag` de objetos composto pela constante 20 — `integer` —, de um item de dado — `string` com o valor do atributo `url` do objeto — e de uma expressão — `float` com o resultado obtido da execução do método `getSize()` tamanho do objeto, acrescido de 10% —.

Os valores para o atributo `url` e `getSize()` são recuperados a partir da variável `w` especificada na cláusula `FROM`. Este tipo de variável é utilizado com o propósito de varrer coleções de objetos, ou seja, cada um dos objetos da coleção é atribuído à variável `w` para que o comando seja processado. No contexto do exemplo, cada um dos objetos pertencente à coleção `WebObjects` é atribuído à variável `w`, e para cada um deles, é criado um elemento na resposta composto e organizado como a especificação da cláusula `SELECT`.

É importante ressaltar que a resposta de um comando de seleção OQL é sempre uma coleção de objetos.

Um dos pontos fortes da linguagem OQL é a possibilidade de navegação por referências (ponteiros), coleções e estruturas de forma fácil e intuitiva. Isto é possível com a utilização das expressões de caminho (***Path Expressions***) definidas pelos símbolos "." ou "->". Assim, as variáveis para varredura de coleções utilizadas como argumento da cláusula `FROM` não se limitam às extensões, permitindo que sejam varridas coleções de objetos acessadas a partir de outras variáveis de coleção.

Seja o comando OQL que recupera todos os *hyperlinks* existentes no documento Web "http://www.unit.br/fabiosoares/opcoes.htm":

```
select l.baseLabel, l.refObject.url, l.refObject.getSize()
   from Texts t, t.links l
   where t.url = "http://www.unit.br/index.html" and
         l.getDepth() <= 1;
```

Podemos observar claramente a fácil navegabilidade oferecida pela linguagem OQL. Na referência `l.refObject.url` da primeira linha, fica evidente a capacidade de partir de um objeto `l`, navegar pela referência `refObject` até a respectiva instância `WebObject`, e dela obtermos o valor do atributo `url`. A segunda linha possui na cláusula `FROM` a especificação `t.links l`, que indica a necessidade de para cada objeto `Text` recuperado em `t`, varrer cada *hyperlink* presente na coleção de objetos `links` com a variável `l`.

O exemplo também especifica uma condição na cláusula `WHERE` para filtrar os objetos a serem recuperados como resposta do comando. Cada par de objetos (`Text`, `Link`) recuperados no par de variáveis (`t`, `l`) devem ser avaliados segundo a condição definida na cláusula `WHERE`. Se o resultado da avaliação for verdadeiro, o par de objetos é utilizado para criação de um dos

elementos da resposta. Caso contrário, o par de objetos é descartado, e o próximo par é analisado.

Assim, um algoritmo conceitual para resolução deste último exemplo seria:

```
Para cada objeto Text presente no extent Texts faça
   Atribuir objeto atual ao ponteiro t
   Para cada objeto Link presente na coleção t.links faça
      Atribuir o objeto atual da coleção ao ponteiro l
      Se ( t.url = "http://www.unit.br/index.html" ) e
         ( l.getDepth() <= 1 ) então
         adicionar ao resultado( l.baseLabel, l.refObject.url );
```

Percebe-se neste algoritmo que cada coleção existente, e portanto varrida na cláusula `FROM`, gera uma estrutura de repetição para recuperar os objetos das coleções. A condição da cláusula `WHERE` é validada através de estruturas condicionais, e somente aqueles pares de objetos que atendem à condição são utilizados na montagem do resultado. No algoritmo, a avaliação da condição `t.url="http://www.unit.br/index.html"` poderia ser antecipada, de modo que os objetos da coleção `links` só seriam recuperados se esta avaliação fosse verdadeira. Isto certamente melhoraria o desempenho do algoritmo.

Subselect e Ordenação

A linguagem OQL permite que o resultado da consulta (um `bag` de objetos) seja ordenado segundo o critério definido pelo usuário, tornando-se então uma lista de objetos. Para isso, bastam especificar na cláusula ORDER BY do comando, os elementos que irão definir a ordem do resultado.

Outro importante recurso está na possibilidade de utilizar o resultado de um comando de seleção em outro, em um tipo de sentença denominada *subselect*. A OQL permite que seja usado um comando de seleção em qualquer ponto de outro comando que se refira a uma coleção, pois um *subselect* nada mais é que a definição dinâmica de um `bag` de objetos. A única exigência está na compatibilidade de variáveis e operadores utilizados em conjunto com o *subselect*. Isto está de acordo com um importante princípio de boa linguagem de programação, que é o **princípio da ortogonalidade**.

Vejamos um exemplo de um comando que faz uso de *subselect* e gera uma resposta em uma ordem específica:

```
select rw.getField("mat_alu").toString() as Matricula
   from Relations r, r.rows rw
   where r.url="http://www.unit.br/alunos.db" and
         rw.getField("cod_curso").toInteger() in(
                select rw.getField("cod_curso").toInteger()
                   from Relations r1, r.rows rw1
                    where r1.url="http://www.unit.br/cursos.db" and
                          rw1.getField("cod_centro").toInteger()=03 )
  order by rw.getField("mat_alu").toString();
```

Este consulta recupera a matrícula de todos os alunos — URL "http://www.unit.br/alunos.db" — que pertencem a cursos — URL "http://www.unit.br/cursos.db" — do centro 03, sendo o resultado ordenado pela matrícula do aluno. O *subselect* iniciado na quinta linha, recupera uma coleção com todos os códigos de cursos que pertencem ao centro 03. O operador `in`, que pode ser utilizado com coleções, serve para verificar se o campo `cod_curso` da linha atual `rw` do `SELECT` mais externo é um dos componentes da coleção retornada pelo *subselect.* Na primeira linha, ainda podemos observar a utilização de apelidos para as colunas retornadas por um comando OQL.

Fragmentação e Consultas Agrupada

Tal como a linguagem SQL, a OQL oferece recursos de consulta agrupada aos seus dados através das cláusulas `GROUP BY` e `HAVING`.

A cláusula `GROUP BY` serve para dividir os dados de uma origem — cláusula `FROM` — em grupos cujo critério de agrupamento é definido pelo usuário, e pelo qual pode-se ter acesso a consultas agrupadas aos seus elementos. Cada agrupamento dá origem a um fragmento implícito — cláusula `partition` — por onde os usuários podem obter informações sobre quantidade (`count`), menor valor (`min`), maior valor (`max`), média (`avg`) e valor total (`sum`) entre os elementos de um grupo.

Conjugada à cláusula `GROUP BY`, pode ser especificada uma condição que restrinja os grupos que devem fazer parte do resultado final. Para isso, a cláusula `HAVING` funciona com os mesmos princípios da cláusula `WHERE`, sendo que as expressões condicionais de grupo só podem possuir expressões lógicas e booleanas que envolvam um item de grupo ou uma função agregada.

Como exemplo de consultas agrupadas, podemos recuperar a quantidade de tabelas disponibilizadas em cada domínio, sendo que só devem ser retornados domínios com menos de 100 tabelas. O comando para recuperar estas informações é o seguinte:

```
select dominio, tot_tab: count( select p.r.url from partition p )
   from Reletions r
   group by dominio: r.getDomain()
   having count( select p.r.url
                 from partition p ) < 100;
```

Neste comando, o construtor presente na cláusula `GROUP BY` (linha 4) garante que os dados da extensão `Relations` serão agrupados por domínio, sendo que cada um deles dará origem a um fragmento virtual. Na quinta linha, as fragmentações de cada grupo são acessadas para serem contados o número de tabelas existentes, sendo que somente aqueles com quantidades maiores que 50 serão selecionados. Os elementos projetados são: o item de grupo e a quantidade de elementos de cada um dos fragmentos.

Obviamente, a semântica da linguagem OQL é muito mais rica e detalhada do que a descrição apresentada nesta subseção, sendo que sua completa apresentação foge ao escopo deste trabalho. Ela pode ser obtida em [CAT,2000].

3.5. Mais Exemplos de Consultas OQL à Web

Utilizando o modelo conceitual composto do esquema de dados *WebObjects* e a linguagem OQL, torna-se possível à especificação de consultas que permitam explorarmos os dados da Web em seu conteúdo e estrutura, em um formato semelhante àqueles oferecidos pelos SGBDs convencionais. Esta subseção é dedicada a formulação de mais exemplos que demonstrem o potencial de consulta da *WebObjects.*

O primeiro e mais simples nível de acesso às informações da Web se dá através da recuperação dos dados gerais dos objetos, ou seja, os dados que são herdados da classe `WebObject`. Este tipo de consulta é realizado normalmente com a indicação, através da URL, de qual objeto deve ter os dados recuperados. Assim, para obter informações como o tamanho do documento de URL "http://www.unit.br/index.html" e o endereço IP do servidor onde ele está armazenado, deve-se escrever o seguinte comando OQL:

```
select w.getSize() as Tamanho, w.getIpAddress() as IP
   from WebObjects w
   where w.url = " http://www.unit.br/index.html";
```

Em uma forma de acesso um pouco mais elaborada, poderíamos utilizar a plataforma *WebObjects* para recuperar objetos da Web explorando as informações presentes em seus conteúdos. Para este tipo de consulta é possível localizarmos, por exemplo, todos os documentos da Web que tenham em seu conteúdo as palavras "Flamengo" e "Zico", fazendo uma pesquisa por objetos que tragam informações sobre o maior ídolo da história do Flamengo. Para isso, deveríamos escrever o seguinte comando:

```
select t.url, t.getIpAddress(), t.getSize()
   from Texts t
   where t.content like "Flamengo" and t.content like "Zico";
```

Nesta consulta fica evidente a facilidade pela qual as expressões condicionais envolvendo operadores lógicos e booleanos podem ser especificadas. Além disso, denota a forma de funcionamento do operador `LIKE` na *WebObjects*, que quando especificado sem o caractere de macro-substituição "%", procura pela existência da palavra no conteúdo. Este último exemplo poderia ser incrementado se desejássemos localizar documentos que façam referência ao apelido de Zico, o "Galinho", e estivessem localizados em um domínio específico, o "www.flamengo.com.br". Vejamos o comando:

```
select t.url, t.getIpAddress(), t.getSize()
   from Texts t
   where ( t.content like "Flamengo" ) and
         ( ( t.content like "Zico" ) or
           ( t.content like "galinho" ) ) and
         ( t.getDomain() = "www.flamengo.com.br" );
```

As consultas definidas para a plataforma *WebObjects* também podem explorar a estrutura dos documentos da Web através dos atributos `links` e `binFiles` da classe `Text`. Dessa forma, para recuperarmos no *website* da Universidade Tiradentes, todos os documentos texto que possuem algum *hyperlink* e imagens no formato GIF, JPEG ou JPG, deveríamos submeter o seguinte comando:

```
select distinct t.url
   from Texts t, t.binFiles f
   where ( t.getDomain() = "www.unit.br" ) and
         ( t.links != nil ) and ( t.binFiles != nil ) and
         ( t.getExtension() in( "gif","jpeg","jpg" );
```

O modelo de consulta também oferece a possibilidade de mesclar em uma mesma consulta recursos de exploração de estrutura e conteúdo. Para localizar no *website* da Universidade Tiradentes todos os documentos diretamente acessados a partir da página principal, e que possuam no seu conteúdo ou título a palavra NEAD, seria suficiente a seguinte consulta:

```
select o.url
   from Texts t, t.links l, l.refObject o
   where ( t.url = "http://www.unit.br/index.htm" ) and
         ( l.linkType in( "local","internal" ) ) and
         ( o.content like "nead" or o.title like "nead" ) and
         ( t.getDepth() = 1 );
```

A consulta à estrutura dos documentos pode ser extremamente útil no processo de manutenção de *websites*. É comum neste tipo de atividade a identificação de *hyperlinks* inválidos, ou seja, que apontam para documentos inexistentes. Este levantamento poderia ser facilmente realizado no *website* da Universidade Tiradentes através do seguinte comando:

```
select l.ancObject.url, l.baseLabel
   from Texts t, t.links l
   where ( t.url = "http://www.unit.br/index.htm" ) and
         ( l.refObject = nil ) and
         ( not( l.getPathType() like "global" ) );
```

Este exemplo utiliza o apontador `ancObject` para encontrar a URL do documento que contém o *hyperlink* perdido, enquanto que o `refObject` é utilizado para determinar se o *hyperlink* está apontando para um objeto inválido. Os atributos `url` e `links` serviram para restringir a busca a um único *website*.

A *WebObjects* possibilita o acesso às informações das tabelas relacionais de duas formas: pela estrutura (esquema) ou pelo conteúdo. A consulta à estrutura das tabelas é possível acessando o atributo `columns` da classe `Relation`. Dessa forma, para recuperarmos quais as colunas que compõe a relação "alunos", deveríamos submeter o seguinte comando:

```
select c.name, c.type, c.required
   from Relations r, r.columns c
   where r.url = "http://www.unit.br/alunos.db";
```

A consulta ao conteúdo das relações pode ser realizada através do atributo `rows`. Assim, é possível listar o código, o nome, o código do fornecedor e o nome do fornecedor daqueles produtos que custam mais que R$ 300,00 com o seguinte comando:

```
select r1.codigo, r1.nome, r2.codido, r2.nome
   from Relations r1, Relations r2
   where ( r1.url = "http://www.unit.br/produtos.db" ) and
         ( r2.url = "http://www.unit.br/fornecedores.db" ) and
         ( r1.cod_forn = r2.codigo ) and ( r1.preco > 300 );
```

Neste exemplo, os dados das tabelas produtos e fornecedores são recuperados por meio de objetos da classe `Relation` definidos na Web segundo uma URL. Alem disso, percebe-se o recurso de macro-substituição OQL incorporado para as relações, ou seja, a referência `r1.codigo` é equivalente a `r1.rows.getField("codigo")`.

Por fim, a *WebObjects* possibilita a especificação de consultas capazes de recuperar informações de tabelas relacionais e dados de documentos HTML sob um único mecanismo de consulta, ou seja, os dados oriundos destas duas estruturas são encarados apenas como objetos do esquema conceitual. Para obter o domínio, o endereço IP do servidor e o tamanho das *home pages* dos professores da UNIT vinculados ao centro 01, deve ser submetido o seguinte comando:

```
select r.urlProf, w.getDomain(), w.getIpAddress(), w.getSize()
   from Relations r, WebObjects w
   where ( r.url = "http://www.unit.br/professores.db" ) and
         ( r.urlProf = w.url ) and ( r.cod_centro = 01 );
```

Aqui, a *home page* de cada professor é armazenada na coluna `urlProf` da tabela "Professores", que é utilizada em uma junção com a extensão *WebObjects*, para que os dados relativos ao documento da Web possam ser recuperados.

3.6. Considerações Finais

Como ficou evidente neste capítulo, o modelo conceitual definido pela *WebObjects* oferece aos seus usuários uma estrutura para especificação de consultas que agrega um grande poder

semântico, ou seja, possibilita aos usuários a utilização de uma linguagem declarativa com o poder de especificação semelhante à SQL. Esta característica permite uma maior capacidade de definição de consultas, atendendo à diversidade e complexidade dos dados contidos na Web.

Outro aspecto relevante a ressaltar é que, modelando a Web como um SGBD homogêneo e orientado a objeto, toda a heterogeneidade da Web fica completamente encapsulada. Dessa forma, a decisão de adotar o modelo padrão de banco de dados orientado a objeto na plataforma *WebObjects* representa uma evolução na recuperação de dados da Web, por possibilitar aos usuários da plataforma seguir um padrão de mercado, oferecer recursos de extensibilidade e, principalmente, aumentar a capacidade semântica para a especificação de consultas à Web.

Capítulo 4

IV. O Processador de Consultas

No capítulo anterior, discutimos: (1) como modelar a Web como um banco de dados ODL/ODMG, virtual; e (2) como consultar dados na Web por meio da linguagem OQL/ODMG, operando sobre o esquema ODL. O propósito deste capítulo é descrever o processador de consultas OQL da plataforma *WebObjects*.

O principal requisito do processador é, evidentemente, ser capaz de reconhecer e executar um comando OQL. Os outros requisitos estabelecidos para o processador são: independência de ambiente operacional (*hardware* e *software*), e recursos para facilitar o desenvolvimento de aplicações-Web.

4.1. Arquitetura do Sistema

A plataforma *WebObjects* implementa o processador de consultas como uma biblioteca de classes Java, capaz de reconhecer, interpretar e executar consultas escritas na linguagem OQL sobre um particular esquema de dados ODL/ODMG. Dessa forma, qualquer programa Java pode fazer uso dos recursos oferecidos pela plataforma, através da utilização de sua API. Uma vez recebido um comando OQL, o mesmo é processado, retornando como resultado da consulta uma coleção de objetos, ou então uma mensagem de erro, em caso de problema quando do processamento da consulta.

Esta estratégia possibilita que o usuário possa desenvolver suas aplicações em qualquer ambiente operacional, sob as mais variadas arquiteturas de sistemas, por exemplo, cliente/servidor ou multicamada (*n-tier*), através da incorporação dos pacotes Java que implementam a *WebObjects*.

A figura 4.1 mostra a arquitetura da plataforma *WebObjects*, seus componentes e a interação entre eles.

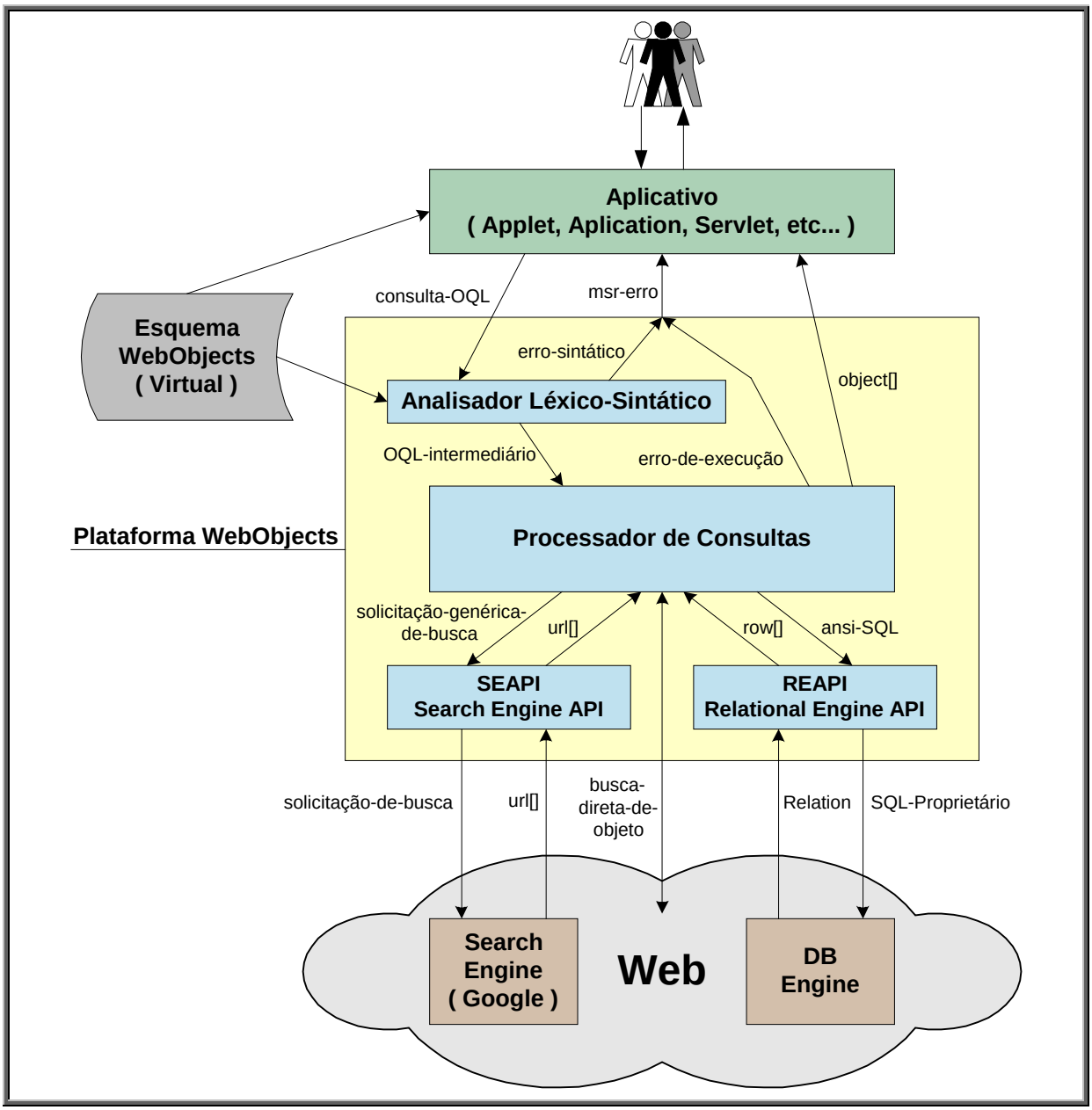

Figura 4.1: Arquitetura da plataforma *WebObjects*

O componente da plataforma que inicia o processo de execução de uma consulta OQL chama-se analisador Léxico-Sintático. Sua tarefa básica é reconhecer o comando como sendo compatível com a sintaxe OQL, bem como validar tipos e gerar um código intermediário. O código intermediário, sem erro de sintaxe, serve de base para o processador de consulta propriamente dito.

O componente Processador de Consultas implementa por sua vez dois sub-componentes: Gerador de Código e Máquina Virtual[13].

O componente Gerador de Código é responsável por analisar a tabela de símbolos — estrutura de dados montada a partir da decomposição do comando OQL pelo analisador Léxico-Sintático —, e a partir dela gerar um plano de consultas especificado em termos de instruções para o componente Máquina Virtual. Neste processo, é analisada a dependência entre o acesso às extensões `WebObjects`, `Relations`, `BinaryFiles` e `Texts` do esquema de dados, e definida a seqüência pela qual os dados serão acessados.

O gerador de código também faz uma análise do custo da consulta, interrompendo a execução daquelas que sejam consideradas proibitivas para serem resolvidas, ou seja, não resolve consultas que não sejam capazes de filtrar um subconjunto dos dados da Web a partir do acesso a um catálogo. Isto ocorre porque resolver consultas sem esta restrição, implicaria em uma exploração de todo conteúdo da Web, o que teria um custo computacional inviável. Nestas situações, o usuário deve reformular seu comando de seleção no sentido de restringir a consulta a um subconjunto dos dados da Web.

O plano é então executado pelo componente Máquina Virtual, por meio de um conjunto de classes que acessam informação na Web, instanciando objetos do esquema de dados com as informações coletadas.

A máquina virtual implementa uma pilha de execução, realizando tarefas como avaliação de condições e montagem do resultado, ou seja, ela funciona como um coordenador das tarefas realizadas durante a execução da consulta. A tarefa de busca das informações na Web e o tratamento de toda a complexidade inerente aos vários tipos de dados ficam sob responsabilidade das classes que tratam de acessar as informações. A máquina reconhece o tipo de dado a ser acessado, e através de chamadas aos métodos disponíveis nas classes de objetos e nas APIs, obtém as informações necessárias à validação e composição dos resultados.

Os dados da Web podem ser capturados de duas maneiras distintas: o acesso direto a objetos através do protocolo HTTP ou através de chamadas às APIs de acesso SEAPI e REAPI. O

[13] Para não sobrecarregar a figura 4.1, esses módulos não aparecem.

acesso direto é feito por meio da API oferecida pela própria linguagem Java para acesso às informações disponíveis nos *web servers*. As APIs de acesso da plataforma foram desenvolvidas basicamente para viabilizar a comunicação com as máquinas de busca (SEAPI) e com os serviços de acesso a dados relacionais (REAPI).

A SEAPI (*Search Engine API*) é o módulo da plataforma responsável pela integração da *WebObjects* com os vários *search engines* (máquinas de busca). Este recurso é possível pois a SEAPI define uma interface genérica de requisição, que é utilizada pelo processador de consultas para gerar suas solicitações. Uma vez solicitada uma busca, a SEAPI converte a solicitação para um formato proprietário do *search engine*, e a envia para que seja executada pela máquina de busca. O resultado é automaticamente repassado para o processador de consultas através de uma coleção de URLs que atendem à condição de pesquisa. Dessa forma, incorporar um novo *search engine* a solução exige apenas uma modificação da SEAPI, para que a mesma seja capaz de efetuar solicitações no novo padrão.

O mesmo princípio de projeto motiva a criação da REAPI (*Relational Engine API*), que recebe solicitações para acesso a bases relacionais em SQL-ANSI e a converte para um SQL proprietário do mecanismo que deva ser acessado. A mesma característica de extensibilidade de novos *engines* oferecida pela SEAPI pode ser alcançada com a REAPI para os vários SGBDs disponíveis no mercado.

Sumarizando o processo de execução de uma consulta OQL, o primeiro passo realizado pela plataforma é a montagem da tabela de símbolos através da decomposição dos componentes presentes no comando OQL. Após isso, a plataforma realiza a análise da condição especificada na consulta para verificar se a mesma aplica um filtro válido na Web, e em caso positivo, determinar a ordem que as extensões devem ser acessados. Neste ponto, as APIs de acesso são acionadas no sentido de recuperar todas as URLs que atendem a solicitação de consulta. Para cada URL recuperada, o respectivo objeto é instanciado pelo interpretador, que avalia a condição aplicada no filtro para confirmar se os dados retornados pelos mecanismos de busca ainda são válidos para o objeto especificado.

No processo de análise e recuperação das informações, a plataforma *WebObjects* utiliza a premissa de somente acessar os dados quando os mesmos são efetivamente necessários. Para isso, utiliza técnicas de avaliação de condição em curto circuito, além de antecipar as validações para que dados desnecessários não sejam recuperados. As classes de recuperação

dos dados da Web são implementadas com o objetivo de somente capturar as informações quando elas são realmente solicitadas. Por exemplo, um objeto `Text` não precisa ter seu conteúdo recuperado e processado, se o usuário consultar apenas o tamanho do arquivo onde o objeto está armazenado.

Além disso, a plataforma utiliza os recursos oferecidos pelas *threads* Java para viabilizar a captura simultânea de informações na Web. Esta técnica permite que documentos com alta latência de recuperação possam ser capturados ao mesmo tempo em que os de baixa latência, permitindo que estes últimos possam ser obtidos da Web mais rapidamente, possibilitando assim um menor tempo de resposta para o usuário.

4.2. Descrição Detalhada dos Módulos

Após a descrição do esquema geral de funcionamento e da arquitetura da plataforma *WebObjects* para resolução de suas consultas, são detalhados nesta subseção a estrutura e o funcionamento de cada componente principal da arquitetura, enfatizando as técnicas e algoritmos empregados e as classes implementadas.

4.2.1. Analisador Léxico-Sintático

Este componente do processador de consultas *WebObjects* foi desenvolvido utilizando o gerador de compiladores JavaCC [JCC,2001], cujo código gerado é puro Java, e possui o aval da *Sun*, a empresa responsável pelo desenvolvimento da tecnologia Java.

Através da especificação da BNF da linguagem em uma sintaxe que mistura construções Java com comandos próprios do JavaCC, o gerador de compiladores interpreta o arquivo de definição e gera um conjunto de classes Java que implementam um analisador léxico-sintático cuja técnica de reconhecimento é a tradução dirigida pela sintaxe [SET,1995]. Além disso, ele permite que os desenvolvedores façam apontamentos sintáticos na árvore de reconhecimento, a fim de executar tarefas, tais como validação de tipos e manutenção da tabela de símbolos, que não possam ser especificadas na definição padrão. Estes apontamentos sintáticos são criados por meio de comandos válidos para a linguagem Java. A definição JavaCC do analisador léxico-sintático do processador de consultas *WebObjects* pode ser observada no apêndice B deste documento.

Por padrão, o JavaCC gera 5 classes com funções específicas na tarefa de reconhecimento léxico-sintático. A primeira delas — `ParseException` —, define uma classe de exceção Java que será utilizada quando algum erro for encontrado no reconhecimento. As literais que representam as palavras reservadas da linguagem OQL são definidas na classe `ParserWebObjectsConstants`. Os *tokens* — literais válidas como elementos da linguagem, tais como identificadores e palavras reservadas — são reconhecidos por uma outra classe — `WebObjectsTokenManager` — que implementa o próprio gerenciador de *tokens* (*Token Manager*). Ele faz uso da classe `SimpleCharStream`, cuja função é converter o comando em um fluxo de caracteres que facilitem a tarefa de reconhecimento dos *tokens*. Finalmente, a classe `ParserWebObjects` contém a lógica para avaliação sintática do comando e todos os apontamentos que realizam as tarefas específicas definidas pelo projetista.

No processo de reconhecimento léxico-sintático do comando OQL, os apontamentos sintáticos do analisador *WebObjects* realizam duas tarefas adicionais: a validação de tipos e a montagem da tabela de símbolos. A validação de tipos é realizada por chamadas a métodos disponíveis na classe estática `WebObjectsTypes`. A montagem da tabela de símbolos é feita por meio de métodos disponíveis na classe `TabSymbols`, cuja implementação oferece dupla funcionalidade dentro da plataforma. A primeira função está diretamente relacionada ao processo de reconhecimento do comando, pois armazena a estrutura de dados que viabiliza a definição do plano de acesso da consulta. A segunda está vinculada à fase de processamento da consulta, haja vista que ela armazena o método responsável por transformar a estrutura de dados da tabela de símbolos em um algoritmo capaz de ser executado pelo interpretador *WebObjects*.

Todas estas classes, adicionando-se algumas outras de suporte, compõe o pacote Java `webobjects.parser`.

4.2.2. Processador de Consultas

Antes de analisarmos o funcionamento do processador de consultas *WebObjects*, é necessário uma perfeita compreensão de sua tabela de símbolos, haja vista que ela é uma estrutura fundamental para que seja possível a definição da estratégia de acesso, ou seja, a especificação do algoritmo a ser utilizado para a execução da consulta. As classes que a implementam estão definidas no diagrama de classes UML da figura 4.2.

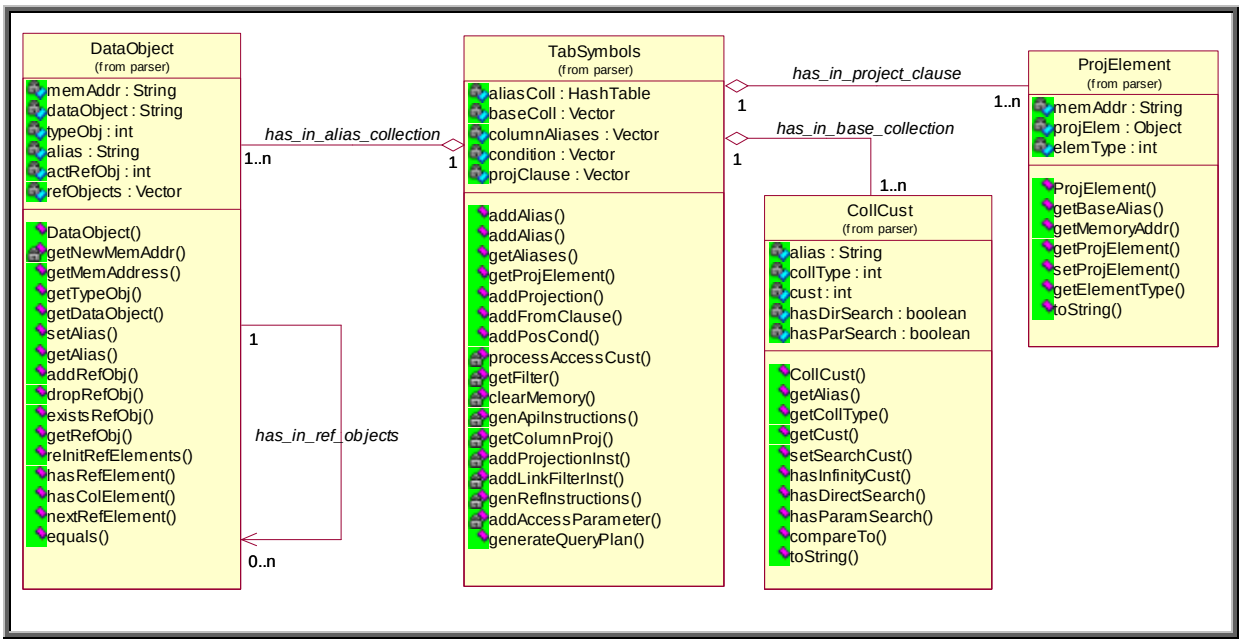

Figura 4.2: Classes que implementam a tabela de símbolos *WebObjects*

Na classe `TabSymbol`, o atributo `columnAliases` armazena uma coleção de `strings` com os apelidos definidos para os dados recuperados no comando de seleção, ou a própria descrição dos itens projetados, para os casos onde os apelidos não são definidos. O atributo `condition` armazena a equivalente expressão em ordem posfixa para a condição definida na cláusula `WHERE`, de modo que seu conteúdo irá permitir a definição de dependência da consulta em operações de junção, além de viabilizar a checagem da capacidade que o comando apresenta em selecionar os dados da Web.

A tabela de símbolos da plataforma *WebObjects* é projetada visando facilitar o processamento das coleções e referências dos comandos OQL. Para isso, cada coleção ou referência a objetos de dados do comando de seleção possui uma instância da classe `DataObject` que o define. Estes objetos possuem um endereço de memória na máquina virtual — atributo `memAddr` — que o interpretador *WebObjects* irá utilizar para localizar e armazenar o objeto na memória *cache* do processador de consultas. O atributo `dataObject` é uma literal válida para o dicionário de dados *WebObjects* e descreve a classe do objeto de dados a ser acessado. O atributo `typeObj` define as características da classe cujas informações devem ser recuperadas, ou seja, identifica o tipo de referência ou coleção que o objeto representa.

As várias instâncias da classe `DataObject` são organizadas segundo uma hierarquia (árvore) de acesso, onde no topo está sempre uma coleção oriunda de uma extensão do esquema de dados *WebObjects*. Por exemplo, se a cláusula `FROM` de um comando OQL apresenta a

definição `Relations r` e a cláusula `SELECT` a referência `r.columns.required`, existirá uma instância da classe `DataObject` para cada uma das coleções, sendo que elas estarão hierarquizadas com a coleção `Relation` no topo e `columns` como sua filha. Esta hierarquia é construída através do atributo `refObjects`. O atributo `alias` contém o apelido da coleção ou referência, quando a mesma é especificada na cláusula `FROM`.

Através do atributo `aliasColl`, a tabela de símbolos endereça apenas as instâncias que estão no topo de cada hierarquia, ou seja, as coleções que representam as extensões acessadas pelo comando de seleção. Este atributo é definido como uma tabela *hash* onde a chave de acesso é o apelido da extensão no comando, o que facilita o processo de localização da instância da classe `DataObject` a partir do seu apelido.

O atributo `projClause` é uma coleção de objetos da classe `ProjElement` que define os dados que serão recuperados pelo interpretador na montagem do resultado da consulta. Cada instância desta classe possui o endereço de memória onde o objeto fonte dos dados está localizado na *cache* da máquina virtual *WebObjects* — atributo `memAddr` —, a descrição do dado a ser recuperado do objeto — atributo `projElem` — e o tipo da informação recuperada — atributo `elemType` —.

Finalmente, cada extensão do esquema de dados acessada pelo comando de seleção possui uma instância da classe `CollCust` armazenada no atributo `baseColl`. Estas informações são essenciais para que o processador de consultas seja capaz de determinar se uma consulta é viável ou não de ser resolvida, bem como a ordem de acesso às várias extensões.

A classe `CollCust` define algumas informações essenciais para o cálculo do custo de uma consulta. O atributo `hasDirSearch` possui o valor verdadeiro se para a respectiva extensão existe na expressão da cláusula `WHERE` uma condição que selecione os dados da Web sem a necessidade de dados de outras extensões, ou seja, o filtro é direto e sem junção. A indicação que a seleção precisa de dados oriundos de outras extensões é definida no atributo `hasParSearch`. Com base nestas informações e no tipo da coleção acessada, é calculado um valor para o custo que é armazenado no atributo `cust`.

A figura 4.3 mostra a tabela de símbolos *WebObjects* para a seguinte consulta:

```
select l.ancObject.url, l.baseLabel, l.refObject.url as refUrl
   from Texts t, t.links l
   where t.url="http://www.unit.br/fabiosoares/opcoes.htm" and
         l.getDepth() <= 1;
```

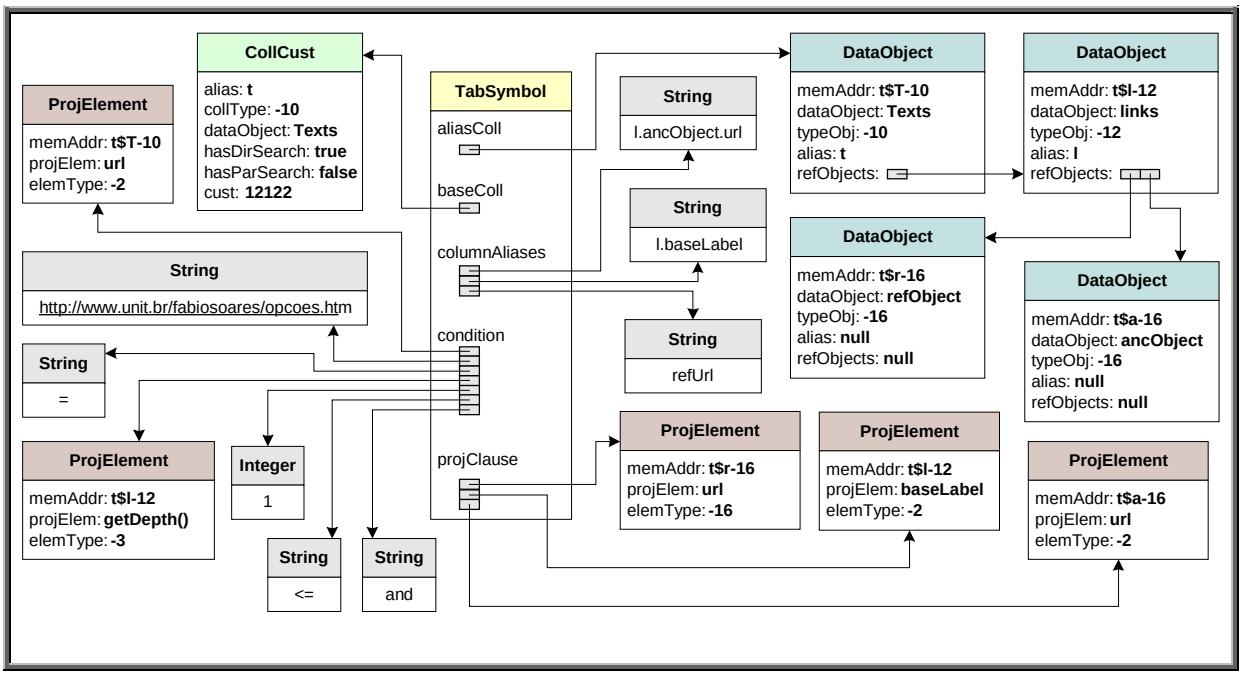

Figura 4.3: Exemplo de uma tabela de símbolos do processador *WebObjects*

Sobre a tabela de símbolos da figura 4.3, algumas características devem ser enfatizadas. O atributo `aliasColl` inicia uma árvore de dependência entre as referências e coleções acessadas pelo comando, sendo que apenas as coleções podem ter filhos. Esta característica irá ser de fundamental importância para a montagem do algoritmo de recuperação dos dados. Além disso, cada um dos objetos `dataObject` possui um endereço de memória gerado durante o processo de reconhecimento que indica a sua localização na memória *cache* do processador de consultas. Estes endereços são os mesmos utilizados nos objetos `projElement` para apontar a posição de memória onde os dados projetados devem ser recuperados. Finalmente, o atributo `condition` é um vetor genérico de objetos, de forma que a condição em ordem posfixa da cláusula `WHERE` é definida por termos armazenados em objetos de classes diferentes.

A tarefa de montagem do algoritmo de acesso é realizada através de uma chamada ao método `generateQueryPlan()` da classe `TabSymbol`, o qual faz uso de vários métodos definidos nas classes `WebObjectsTypes` e `Condition`. A figura 4.4 mostra a definição UML destas

últimas e mais as classes `Instructions` — define as instruções reconhecidas pelo interpretador — e `Interpreter` — classe que executa o algoritmo gerado para a consulta.

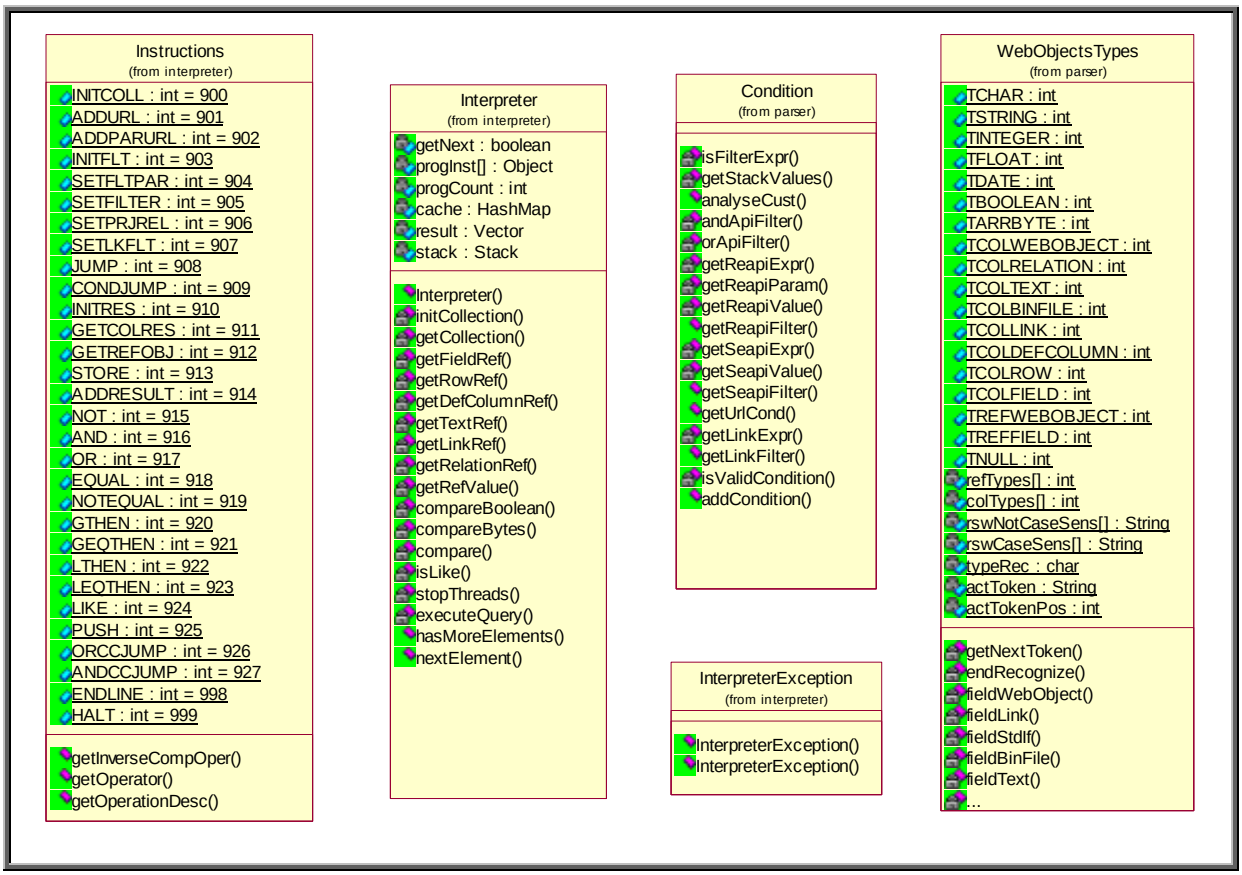

Figura 4.4: Principais classes responsáveis pela geração e execução da consulta

A transformação da tabela de símbolos em um algoritmo capaz de recuperar as informações necessárias à resolução da consulta passa necessariamente por uma série de etapas que estão diretamente associadas à capacidade de análise e decomposição da cláusula `WHERE` do comando de seleção, ou seja, das informações armazenadas no atributo `condition` da tabela de símbolos.

Redução de Expressões Condicionais

A redução das expressões condicionais é uma técnica utilizada no processo de análise da condição de consulta no sentido de extrair informações necessárias à antecipação de testes para montagem do resultado ou para análise do custo associado à consulta. Por exemplo, se em uma condição de busca é especificada a seguinte expressão:

```
t.url = "http://www.unit.br/index.htm" and l.getDepth() <= 1
```

Quando analisada em relação ao objeto de dados `t`, esta expressão pode ser reduzida à condição `t.url=http://www.unit.br/index.html`, pois se a mesma for falsa, o segundo teste não precisa ser executado, ou seja, as informações relativas ao objeto `l` não devem ser recuperadas para que saibamos que o resultado final da avaliação é falso. Isto não quer dizer que para a montagem do resultado final a segunda expressão possa ser desconsiderada. No entanto, poderíamos primeiramente restringir o universo de objetos recuperados na variável `t` àqueles cuja primeira condição seja verdadeira, e depois avaliar a segunda em uma coleção de dados limitada.

A redução para o operador lógico `OR` só pode ser realizada em situações de avaliação condicional em curto circuito, pois se o primeiro termo for verdadeiro, o segundo não precisa ser avaliado para que possamos definir o resultado final da expressão. No entanto, a antecipação do universo de objetos recuperados não pode utilizar a redução para o operador `OR`, haja vista que para uma dada condição, se o primeiro termo é falso, o resultado final não pode ser computado até que o segundo termo tenha sido avaliado. Neste tipo de redução, uma expressão que envolve condições de um mesmo objeto é a própria expressão. Por exemplo, a redução da expressão `t.content like "zico" or t.content like "flamengo"` é ela própria, pois os dois termos envolvem seleções sobre o objeto `t`.

Esta técnica pode ser utilizada repetidas vezes em expressões que envolvam vários termos lógicos ou que possuam precedência modificada por parênteses, de modo que ao final do processo tenhamos uma expressão cujo filtro se aplique apenas a um objeto de dados. Esta tarefa é realizada por vários métodos disponíveis na classe estática `Condition`.

Análise do Custo de Acesso para uma Consulta

A primeira tarefa no sentido de gerar um algoritmo que resolva uma consulta OQL aplicada sob o esquema de dados *WebObjects* é determinar o custo associado à consulta, ou seja, a capacidade que ela tem de filtrar as informações recuperadas da Web. Esta tarefa torna-se crucial quando observamos a grande dimensão e a escassez de recursos para recuperação de informações na Web, o que faz de uma consulta a todo seu conteúdo praticamente impossível.

Esta análise utiliza basicamente a premissa de que qualquer consulta à Web deve possuir pelo menos uma extensão cujos dados recuperados sejam limitados por um filtro direto, ou seja, uma condição que limite os objetos da extensão por expressões cujos itens de dados sejam

somente comparados com constantes, e não com dados obtidos a partir de outros objetos. Além disso, não pode existir nenhuma extensão que não seja filtrada de maneira direta ou indireta. Esta verificação é possível através da técnica de redução da condição para as extensões definidas na cláusula `FROM`.

A redução neste nível ainda descarta condições incapazes de restringir os dados recuperados da Web. Por exemplo, uma condição que utilize o tamanho dos objetos `Text` para filtrar os elementos da sua respectiva extensão, não deve ser considerada na redução, pois este tipo de condição não pode ser submetido a uma máquina de busca para selecionar um conjunto de objetos da Web.

Assim, pelo processo de redução de condições do atributo `condition` da tabela de símbolos, torna-se possível à valoração dos atributos `hasDirSearch` e `hasParSearch` de cada objeto `CollCust` associado a cada extensão acessada pelo comando. Depois disso, é calculado um valor para o custo de acesso à extensão, levando-se em consideração que o menor custo está associado aos seguintes critérios:

1. A extensão possuir um filtro direto (`hasDirSearch=true`);
2. A extensão possuir um filtro parametrizado (`hasParSearch=true`);
3. A ordem de prioridade para as extensões é: `Relations`, `Texts` e `BinaryFiles`.

A definição do custo serve para que as extensões sejam acessadas ordenadamente em relação ao custo, pois aquelas de menor valor têm uma maior capacidade de restringir o universo de objetos considerados, sendo as de maior custo normalmente utilizadas com filtros que obtêm dados oriundos de outros objetos recuperados.

Algoritmo Básico de Recuperação

Com a tabela de símbolos totalmente montada, ou seja, com o custo associado a cada extensão definido, é possível recuperarmos a informação através da utilização de um algoritmo básico que explore por ordem de custo a árvore de dependências iniciada a partir do atributo `aliasColl`. Assim, cada coleção encontrada na hierarquia dá origem a um laço que recupera todos os seus elementos, armazenando-os na posição de memória definida para a coleção. As referências geram apenas acessos e armazenamento dos objetos na *cache* do processador de consultas. No centro destes laços encadeados, estaria a avaliação da condição de busca para determinar se a informação deve ou não ser adicionada ao conjunto resposta da consulta.

Teoricamente, este algoritmo seria capaz de recuperar as informações requisitadas por qualquer consulta OQL submetida à plataforma. No entanto, as características do banco de dados da Web impõem sérias restrições de desempenho a esta estratégia. Assim, para tornar este algoritmo viável, o processador *WebObjects* utiliza a técnica de antecipar os filtros das coleções, fazendo com que os dados a serem acessados já estejam parcialmente em conformidade com a solicitação da consulta.

Além disso, o algoritmo antecipa os testes que envolvem dados de um único objeto, fazendo com que a continuidade no encadeamento nos laços não ocorra se uma condição falsa for encontrada. Dessa forma, a recuperação de uma informação da Web só será realizada se os filtros aplicados aos dados recuperados no laço anterior estejam corretos em relação à expressão de consulta.

O algoritmo básico de consulta utilizado pelo processador *WebObjects* é o seguinte:

```
Obter a primeira coleção a ser acessada
Aplicar o filtro aos dados da coleção
Para cada objeto recuperado faça
   Colocar o objeto na cache
   Para cada objeto do tipo referência faca
      Colocar objeto na cache
   Se condição específica da coleção for verdadeira então
      Obter a próxima coleção
      Aplicar o filtro aos dados da coleção
      Para cada objeto da coleção faça
        ...
        Se condição for verdadeira faça
           Adicionar os elementos projetados no resultado
```

Este algoritmo pode ser utilizado para consultas que envolvam junção, sendo que no momento em que a última coleção de uma extensão for acessada, a coleção seguinte é a próxima extensão da junção, com o filtro sendo montado a partir de dados recuperados da extensão anterior. É importante ressaltar que a cada informação recuperada, o filtro específico é validado, de modo que o processamento do laço seguinte só é realizado, se a condição anterior for verdadeira.

A escolha deste algoritmo de processamento está associada à ausência de estruturas de indexação dos dados da Web que viabilizem a utilização de outros algoritmos mais elaborados, e que propiciem um melhor desempenho em relação aos laços encadeados.

A Máquina Virtual WebObjects

A execução do algoritmo de acesso — plano de acesso — é feita pela máquina virtual *WebObjects* implementada pela classe `Interpreter` da figura 4.4. Seus atributos são detalhados na tabela 4.1.

Atributo	Descrição
`progInst`	Vetor de objetos que contém o algoritmo ou as instruções a serem executadas pela máquina virtual.
`progCount`	Índice para a próxima instrução do algoritmo a ser executada.
`cache`	Área de memória onde os objetos recuperados pela máquina virtual são armazenados.
`result`	Vetor de objetos que contêm as informações da próxima linha retornada como resposta à consulta.
`stack`	implementa a pilha genérica de execução

Tabela 4.1: Descrição dos atributos da classe *Interpreter*

A função básica do interpretador é funcionar como um gerente do acesso aos dados, coordenando as atividades realizadas pelos vários objetos responsáveis por acessar os mecanismos de busca, recuperar as informações da Web e instanciar os objetos de dados. Esta estratégia faz com que o interpretador seja aberto a adição de novas classes de objetos, bastando para isso que ele reconheça a interface de chamada.

Esta característica fica bastante clara quando analisamos as instruções executadas pela máquina virtual, cuja definição está na classe `Instructions` da figura 4.4, e a descrição é detalhada no apêndice C deste documento.

Com o objetivo de analisarmos um algoritmo executado pela máquina virtual *WebObjects*, a figura 4.5 traz o plano de acesso gerado a partir da tabela de símbolos de uma consulta que recupera os *hyperlinks* diretamente acessados de páginas que contenham as palavras "Mestrado" e "Fabio Soares" ou do documento de URL "http://www.unit.br/index.htm". O comando para esta consulta é o seguinte:

```
select l.ancObject.url, l.baseLabel, l.refObject.url as refUrl
   from Texts t, t.links l
   where ( ( t.url="http://www.unit.br/index.htm" ) or
           ( t.content like "Mestrado" and
             t.content like "Fabio Soares" ) ) and
         ( l.getDepth() <= 1 );
```

```
01 - INITCOLL( "t$T-10#api", "Seapi" )
02 - ADDURL( "t$T-10#api", "http://www.unit.br/index.htm" )
03 - INITFLT( "t$T-10#flt",
              "( :$$:url:$$: = "http://www.unit.br/index.htm" ) or
               ( :$$:text:$$: like "Mestrado" and :$$:text:$$: like "Fabio Soares" )" )
04 - SETFILTER( "t$T-10#api", "t$T-10#flt" )
05 - GETCOLRES( "t$T-10#api", "t$T-10", "Text:true" )
06 - CONDJUMP( 56 )
07 - INITFLT( "t$T-10#lkflt", "getDepth() 1 <=" )
08 - SETFILTER( "t$T-10", "t$T-10#lkflt" )
09 - GETREFOBJ( "t$T-10", "sourceUrl" )
10 - PUSH( "http://www.unit.br/fabiosoares/opcoes.htm" )
11 - EQUAL
12 - ORCCJUMP( 22 )
13 - GETREFOBJ( "t$T-10", "content" )
14 - PUSH( %Mestrado% )
15 - LIKE
16 - ANDCCJUMP( 21 )
17 - GETREFOBJ( "t$T-10", "content" )
18 - PUSH( "%Fabio Soares%" )
19 - LIKE
20 - AND
21 - OR
22 - CONDJUMP( 05 )
23 - GETCOLRES( "t$T-10", "t$l-12", "links" )
24 - CONDJUMP( 05 )
25 - GETREFOBJ( "t$l-12", "refObject" )
26 - STORE( "t$r-16" )
27 - GETREFOBJ( "t$l-12", "ancObject" )
28 - STORE( "t$a-16" )
29 - GETREFOBJ( "t$T-10", "sourceUrl" )
30 - PUSH( "http://www.unit.br/fabiosoares/opcoes.htm" )
31 - EQUAL
32 - ORCCJUMP( 42 )
33 - GETREFOBJ( "t$T-10", "content" )
34 - PUSH( "%Mestrado%" )
35 - LIKE
36 - ANDCCJUMP( 41 )
37 - GETREFOBJ( "t$T-10", "content" )
38 - PUSH( "%Fabio Soares%" )
39 - LIKE
40 - AND
41 - OR
42 - ANDCCJUMP( 47 )
43 - GETREFOBJ( "t$l-12", "getDepth()" )
44 - PUSH( 1 )
45 - LEQTHEN
46 - AND
47 - CONDJUMP( 23 )
48 - INITRES
49 - GETREFOBJ( "t$a-16", "url" )
50 - ADDRESULT
51 - GETREFOBJ( "t$l-12", "baseLabel" )
52 - ADDRESULT
53 - GETREFOBJ( "t$r-16", "url" )
54 - ADDRESULT
55 - ENDLINE( 23 )
56 - HALT
```

Figura 4.5: Exemplo de plano de acesso para uma consulta *WebObjects*

No algoritmo da figura 4.5, as instruções das linhas 01 a 04 realizam a seleção dos objetos da extensão `Texts` que atendam à condição de pesquisa. A linha 01 inicializa um objeto `Seapi` — faz a interface com as máquinas de busca — na posição de memória "t$T-10#api" da máquina virtual a fim de recuperar as URLs que atendam à condição de seleção. A linha 02 faz uso de um recurso que permite a adição direta de uma URL ao resultado devolvido pela `Seapi`. As linhas 03 e 04 aplicam um filtro obtido a partir da redução de expressões condicionais em relação ao objeto `t` para `Seapi`. Este filtro é definido em uma sintaxe genérica, que será modificado pela `Seapi` para a sintaxe específica da máquina de busca acessada, de modo que possa ser recuperada uma coleção de URLs que atendam ao filtro especificado.

O processamento da coleção de URLs é feito por um laço iniciado na instrução da linha 05, para obter da `Seapi` definida na posição de memória "t$T-10#api" o objeto `Text` identificado pela próxima URL da coleção, armazenando-o na posição de memória "t$T-10". Se esta instrução conseguir recuperar um objeto, é adicionado no topo da pilha de execução o valor verdadeiro, caso contrário, é adicionado o valor falso. A instrução da linha 06, recupera valor do topo da pilha e salta para a instrução da linha 56 se o valor recuperado for falso, ou seja, executa a instrução `HALT` — fim de execução — se não existir mais URLs a serem recuperadas como resposta.

Na recuperação dos objetos `Text` está o primeiro ponto onde a plataforma faz uso das *threads* Java, pois no momento que a `Seapi` finaliza a composição da coleção de URLs retornadas pela máquina de busca, a plataforma *WebObjects* dispara um conjunto de *threads* que recuperam os objetos `Text` em um *buffer*. Assim, simultaneamente à execução da consulta, as *threads* capturam os objetos `Text`, fazendo com que documentos de alta latência de recuperação não impeçam a continuidade na execução da consulta, pois à medida que os objetos de baixa latência são recuperados no *buffer*, a interpretador pode usá-los para continuar o processamento dos dados. Esta técnica é baseada nas idéias da interação assíncrona proposta por [GOL,2000].

As instruções das linhas 07 e 08 definem o filtro a ser aplicado à coleção de *hyperlinks* do objeto `Text` recuperado no laço mais externo.

Nas instruções das linhas 09 a 21 é validada a condição específica do objeto `Text`, ou seja, a coleção do segundo laço só é processada se o filtro aplicado aos objetos do laço anterior for

válido. O salto condicional da linha 22 garante esta característica quando desvia no fluxo de execução para a instrução 05 — obtém o próximo elemento do laço anterior — quando o resultado da expressão condicional é falso. Sobre esta condição devem ser enfatizados dois aspectos importantes. O primeiro está no uso da avaliação em curto circuito da condição, o que faz com que o processamento do resultado final seja antecipado. O segundo está na precisão que ela oferece, pois neste ponto os dados da Web são recuperados, e as informações retornadas pela máquina de busca são validadas. Dessa forma, páginas indisponíveis ou que foram modificadas deixando de atender a solicitação de busca, são retiradas do resultado.

As instruções das linhas 23 a 47 realizam um processamento semelhante ao descrito para a extensão `Texts`, só que aplicado à coleção `links` de cada objeto `Text` corrente. É importante observar que a condição aplicada no último laço se refere à avaliação da expressão condicional completa, sem que isso cause problemas de desempenho, pois este teste envolve dados em memória, já que nas avaliações anteriores os dados foram recuperados e disponibilizados na *cache* do processador de consultas.

O resultado da consulta é montado nas instruções das linhas 48 a 55, cuja tarefa é acessar os dados da *cache*, adicionando-os ao vetor que define um elemento do resultado (`bag` de objetos). A instrução `ENDLINE` da linha 55 causa a finalização da montagem do resultado, e um salto para a instrução que recupera o resultado do laço mais interno.

4.2.3. APIs de Acesso

Como ficou evidente no algoritmo exemplificado na subseção anterior, o ponto de partida para a execução de uma consulta é o acesso aos dados disponíveis nas várias extensões. Esta tarefa é realizada pelas classes do pacote `webobjects.api`, cuja definição pode ser observada no diagrama de classes UML da figura 4.6.

As características comuns a todos os mecanismos de busca são definidas na classe abstrata `StandardApi`, ou seja, ela define o comportamento padrão que é utilizado pelo interpretador de consultas para fazer acesso a um mecanismo de busca. Isto garante que na adição de novos mecanismos não será necessária a modificação na estrutura do interpretador *WebObjects*, haja vista que ele já esta preparado para reconhecer a interface padrão.

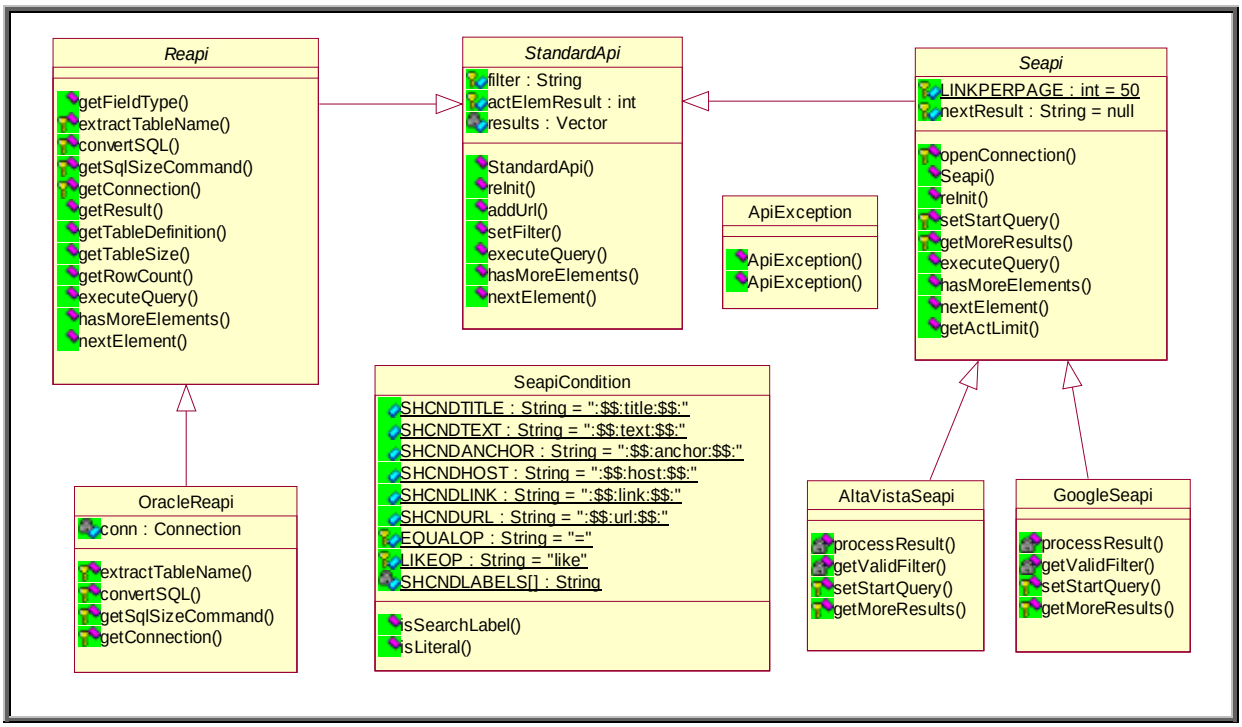

Figura 4.6: Diagrama de classes UML do pacote *webobjects.api*

Através da herança, as classes abstratas `Seapi` e `Reapi` especializam as características para dois tipos de mecanismos de busca: o primeiro para as máquinas de busca como *Google* e o segundo para SGBDs relacionais.

A `Seapi` define os métodos que todos as classes de acesso às máquinas de busca devem possuir. Ela está preparada para reconhecer uma consulta especificada em uma sintaxe genérica, realizando a busca e devolvendo um conjunto de URLs que atendam à condição especificada. As constantes válidas para esta solicitação genérica são definidas na classe estática `SeapiCondition`.

Para que uma máquina de busca seja reconhecida pela plataforma, é necessária a definição de uma nova subclasse de `Seapi` que redefina basicamente 4 métodos. O método `getValidFilter()` deve converter a solicitação genérica em uma condição válida para a máquina de busca específica. O método `setStartQuery()` monta para a condição específica um comando que é submetido para que a máquina de busca compute a primeira coleção com os resultados da busca. Para efetuar a recuperação da próxima coleção de URLs, deve ser redefinido o método `getMoreResults()`. Finalmente, o método `processResult()` tem a função de extrair do resultado as URLs, ou seja, somente a informação útil à plataforma.

Nesta versão inicial, a plataforma *WebObjects* reconhece as máquinas de busca *Google* e *AltaVista* através das classes `GoogleSeapi` e `AltaVistaSeapi`. A classe `GoogleSeapi` tem a vantagem de utilizar uma interface de acesso específica para o desenvolvimento de aplicações, ou seja, recupera o resultado através de uma API que utiliza a tecnologia de *web service*. No entanto, o *Google* não oferece grandes recursos de exploração do seu catálogo, restringindo principalmente a utilização de expressões condicionais que envolvam parêntese e mudança de precedência. Já o *AltaVista* disponibiliza a utilização de tais expressões, mas oferece um acesso mais difícil ao resultado, pois a recuperação das URLs é realizada através de um programa CGI, que devolve uma página HTML como resposta. Esta página deve ser decomposta para capturar as URLs.

De maneira análoga à `Seapi`, a classe abstrata `Reapi` define o comportamento padrão de acesso aos dados de bases relacionais. O tratamento ao SGBD específico é realizado por classes que façam herança da `Reapi` e redefinam os métodos `extractTableName()` — a partir da URL que define a tabela, extrai o nome da tabela no SGBD específico —, `convertSQL()` — transforma o SQL-ANSI genérico em um SQL proprietário do SGBD —, `getSqlSizeCommand()` — devolve o comando de seleção que obtém o tamanho de uma tabela através da consulta ao dicionário de dados do SGBD — e `getConnection()` — recupera uma conexão válida para o SGBD —. Nesta versão, a *WebObjects* é capaz de acessar dados oriundos do SGBD *Oracle*.

Esta estratégia de acesso aos dados garante a *WebObjects* completa independência em relação aos mecanismos de busca, pois o interpretador utiliza apenas a interface disponibilizada pela classe básica `StandardApi`, enquanto que as classes de acesso utilizam recursos oferecidos pela `Seapi` e `Reapi`. Assim, o interpretador está independente em relação aos mecanismos de acesso, enquanto que as classes de dados em relação a produtos específicos.

4.2.4. Classes de Dados

A estratégia de execução de consultas definida para o processador *WebObjects* está fortemente galgada na premissa de transferir o tratamento das particularidades dos objetos da Web para as classes de dados. Esta característica foi evidenciada com bastante propriedade no algoritmo exibido na figura 4.5, quando se percebe que a execução da consulta está

basicamente associada à chamada e armazenamento dos resultados dos métodos disponibilizados nas várias classes de dados.

As classes de dados são implementadas no pacote Java `webobjects.database` e seguem basicamente a estruturação do diagrama de classes definido para o esquema de dados da plataforma. A exceção a esta consideração está nas classes responsáveis por recuperar as informações das coleções internas dos objetos, como são os dados do atributo `links` da classe `Text`.

Esta diferença estrutural é originada da necessidade de melhorar o desempenho das consultas, fazendo com que tais dados sejam capturados sob demanda, ou seja, somente no momento que eles são verdadeiramente solicitados. Assim, as classes que recuperam este tipo de informação devem possuir elementos adicionais que controlem esta recuperação. Além disso, as classes de manipulação dos documentos HTML devem controlar a execução concorrente das *threads* Java e dos recursos compartilhados.

A figura 4.7 traz o diagrama UML das classes que recuperam as informações dos documentos HTML disponibilizados na Web.

As classes que representam as extensões — `Text` e `BinaryFile` — são definidas de maneira análoga ao esquema conceitual e implementam os métodos com a lógica de recuperação e preenchimento dos atributos com os dados obtidos da Web.

Já para o preenchimento de informações que demandam mais recursos computacionais, como é o caso dos atributos `links` e `content` da classe `Text`, os dados dos objetos só são recuperados quando ocorre uma solicitação a partir de uma chamada ao seu respectivo método `get`. Especificamente para o caso das coleções, é utilizado o processamento concorrente para melhorar a taxa de recuperação das informações.

A técnica se resume à definição de um *buffer* de objetos de onde serão capturados os dados retornados pelo método `get`, e para onde um conjunto de *threads* Java disparadas no momento que a primeira solicitação `get` é emitida, irá adicionar os objetos recuperados da Web. O controle do *buffer* é idêntico ao esquema clássico do produtor-consumidor, ou seja, quando o *buffer* está cheio, as *threads* dormem até que o processador de consultas *WebObjects* consuma um dos resultados produzidos e libere espaço no *buffer*. Quando o

buffer está vazio, o processador dorme até que uma das *threads* produza algum resultado que possa ser consumido.

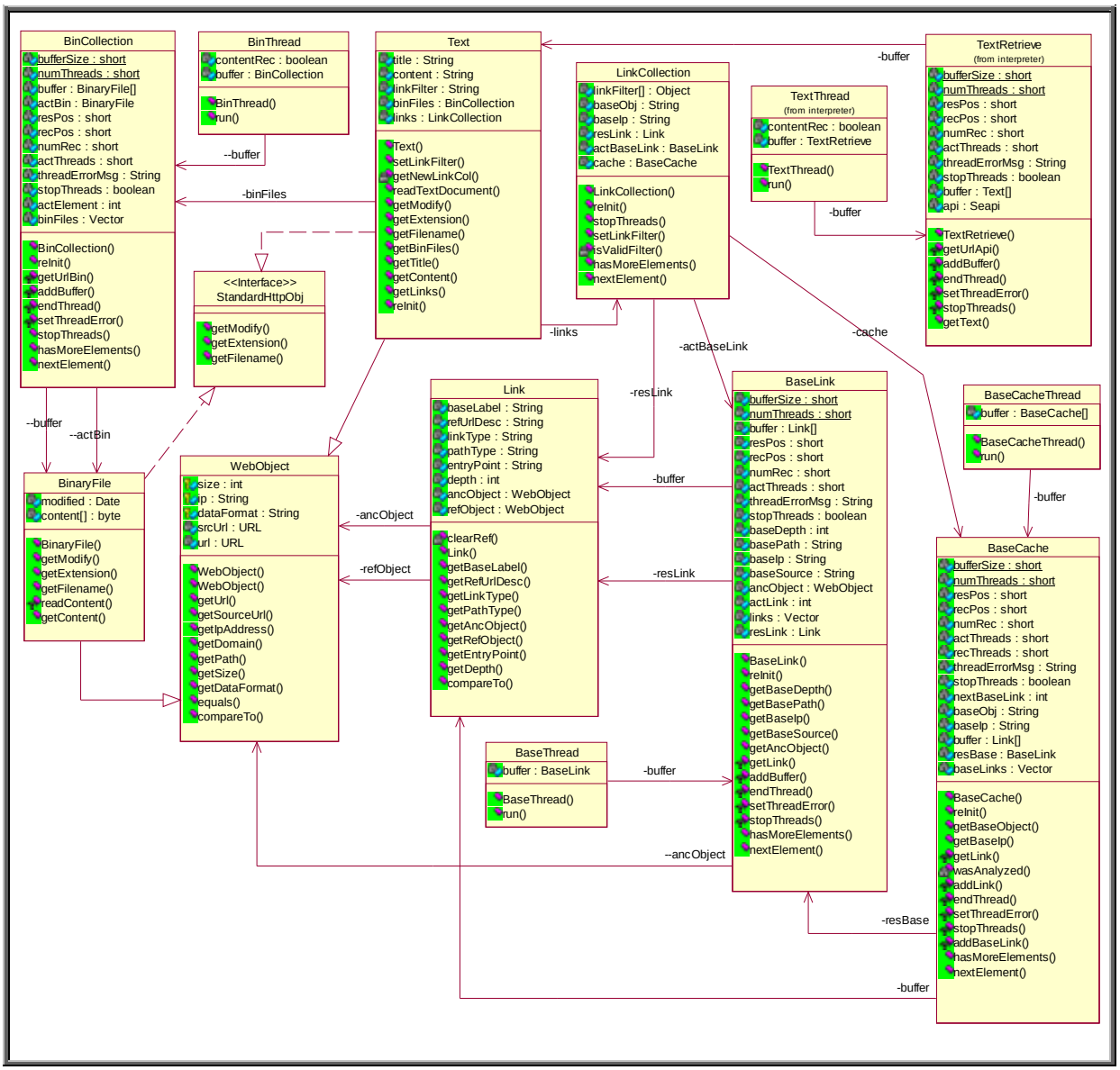

Figura 4.7: Classes que mapeiam as informações recuperadas dos documentos HTML

Seguindo este esquema, o conjunto de classes `TextRetrieve` e `TextThread` implementa um *buffer* para a recuperação concorrente de objetos `Text`. As classes `LinkCollection`, `BaseLink`, `BaseCache`, `BaseThread` e `BaseCacheThread` implementam o esquema para a recuperação dos *hyperlinks* dos documentos, ou seja, no preenchimento das informações do atributo `links`. Finalmente, a recuperação dos objetos binários — atributo `binFiles` — é realizada pelas classes `BinCollection` e `BinThread`.

Para [GOL,2000], esta técnica pode reduzir consideravelmente o tempo de captura das informações na Web, pois propicia que documentos com alta latência de recuperação não

causem um entrave no processamento do resultado da consulta. Isto é possível pois as *threads* que estiverem recuperando documentos de baixa latência irão recuperar informações no *buffer* que poderão ser consumidas e manipuladas pelo processador na montagem do resultado da consulta. Além disso, o tempo que uma *thread* está esperando por uma resposta a uma solicitação de dados de um *web server*, é utilizado por outra na efetiva recuperação de informações.

A validação da técnica foi confirmada por experimentos comparativos entre execuções lineares da consulta em relação a execuções concorrentes. Em média, a execução concorrente reduziu em 60% o tempo de processamento total da consulta, de modo que o tempo final se resumiu em alguns casos, ao tempo gasto para a recuperação do documento de maior latência. O pior caso ocorre quando o documento de maior latência é o último retornado pelo mecanismo de busca, e conseqüentemente, o último recuperado da Web. O melhor caso ocorre quando o documento de maior latência está no início da recuperação.

Para que seja possível uma perfeita noção da importância da utilização das *threads* Java, uma consulta que recuperou 134 URLs[14] da Web, gastou 15 minutos em um processamento linear, enquanto que utilizando 10 *threads*, o tempo gasto foi de apenas 2 minutos e meio.

Na recuperação das informações relacionais, são utilizadas as classes definidas no diagrama UML da figura 4.8. Neste nível, a única preocupação foi à recuperação sob demanda das linhas da tabela, cuja tarefa é desempenhada pela classe `RowCollection`. Para este tipo de informação, a latência de recuperação não apresentou um entrave ao desempenho, sendo dessa forma desnecessária a utilização de *threads*.

4.2.5. Pacote Library

O pacote `webobjects.library` implementa um conjunto de classes que oferecem recursos gerais de tratamento dos dados manipulados pela plataforma.

Este pacote é composto pelas classes `Library`, `LinkHtml`, `HtmlParser` *e* `HtmlCallBack`. A classe estática `Library` implementa um elenco de métodos de propósito geral, dentre os quais poderíamos destacar os que desempenham funções de formatação de dados e tratamento de URLs. As classes `LinkHtml`, `HtmlParser` e `HtmlCallBack` implementam

[14] Consulta pela palavra soteropolitano, realizada em 15/05/2002

conjuntamente um *parser* HTML, muito útil nas operações de extração de *hyperlinks*, documentos binários e conteúdo de um documento HTML. A recuperação dos dados para os objetos `Text` e o processamento do resultado do *AltaVista* fazem uso dos recursos oferecidos por este *parser*.

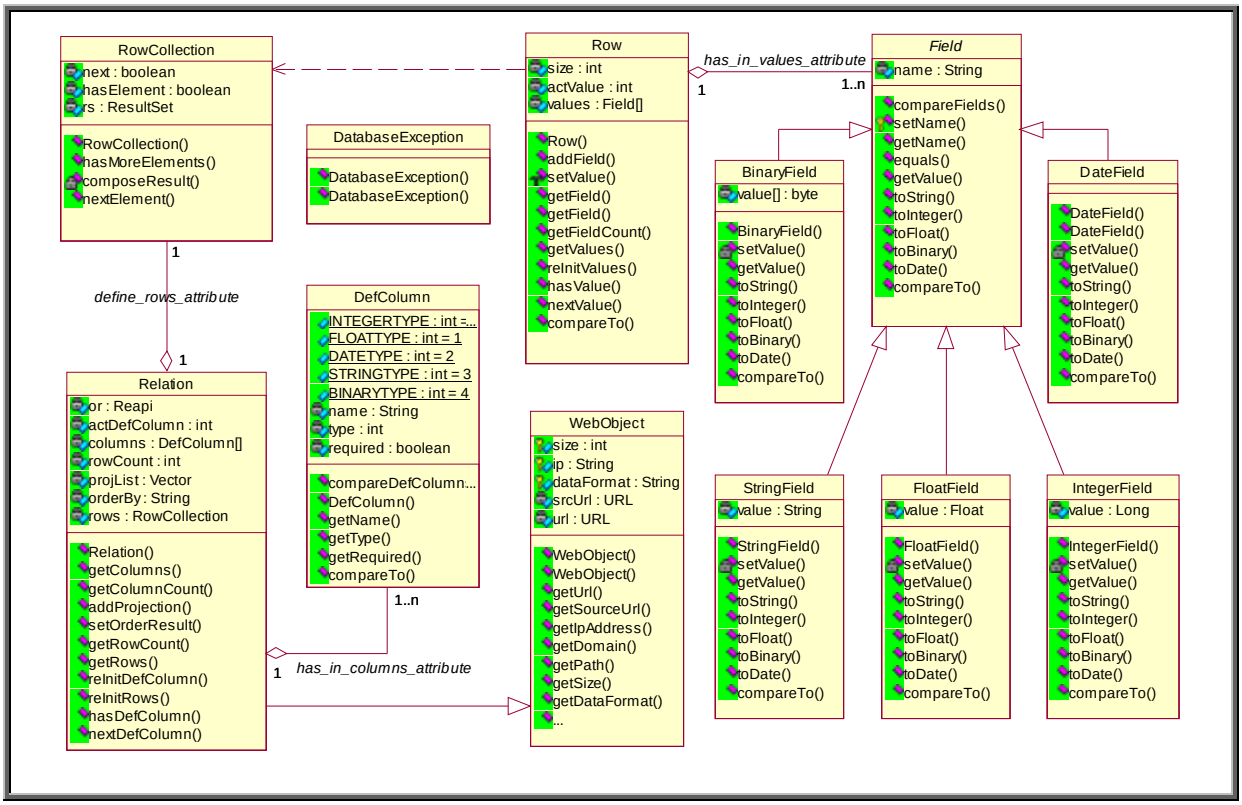

Figura 4.8: Classes que recuperam os dados relacionais

4.3. Interface de Utilização

Para que o usuário possa fazer uso dos recursos oferecidos pela plataforma *WebObjects*, é definida a classe `Query`, que basicamente implementa um cursor para a plataforma. Assim, a partir de um comando OQL, os usuários podem processar unidirecionalmente o conjunto resposta recuperado para o comando de seleção.

A interface de utilização desta classe é extremamente simples e próxima da sintaxe definida para a classe `Iterator` do Java. O construtor da classe recebe como parâmetro o comando OQL a ser executado. Para o processamento do resultado da consulta são disponibilizados os métodos `nextElement()` e `hasMoreElements()`. O método `nextElement()` retorna um vetor de objetos com um elemento do resultado da consulta, sendo que se este método for chamado e não exista mais nenhum resultado a ser retornado, é gerada uma exceção Java `NoSuchElementException`. O método `hasMoreElements()` retorna verdadeiro se ainda

existe resultado a ser recuperado pelo método `nextElement()`. Assim, a combinação dos dois métodos permite que através de um laço possamos recuperar todos os elementos retornados por uma consulta.

Além da exceção `NoSuchElementException`, a manipulação da classe `Query` pode gerar as exceções de `ParseException` — geradas por erros sintáticos e semânticos na especificação do comando OQL —, `ApiException` — gerados por erros na APIs de acesso a dados —, `InterpreterException` — gerados por erros durante a fase de execução da consulta — e `DatabaseException` — erros gerados durante à recuperação das informações pelas classes de dados.

Para ilustrarmos a utilização deste componente, a figura 4.9 traz um exemplo de aplicação Java que imprime no dispositivo de saída padrão a definição do esquema de uma tabela.

```
import webobjects.Query;
import java.util.Vector;

public class Exemplo {

   public static void main( String[] args ) {
     try {
       /* Define o comando que recupera o esquema da tabela */
       String oql =
           "select c.name, c.type, c.required \n" +
               "from Relations r, r.columns c \n" +
               "where r.url=\"http://www.unit.br/tabelas/centros.oracledb\";";

       /* Abre um cursor para o comando */
       Query qRes = new Query( oql );

       /* Enquento existir resposta, obtém o próximo resultado */
       while ( qRes.hasMoreElements() ) {
         /* Recupera o próximo resultado */
         Vector res  = qRes.nextElement();

         /* Decompõe os elementos do vetor de resposta em Strings */
         String name = String.valueOf( res.elementAt( 0 ) );
         String type = String.valueOf( res.elementAt( 1 ) );
         String req  = String.valueOf( res.elementAt( 2 ) );

         /* Imprime o resultado na saída padrão */
         System.out.println( name + " - " + type + " - " + req );
       }
     } catch ( Exception e ) {
       /* Tratamento dos erros ocorridos na execução da consulta */
       System.out.println( "Erro: " + e.getMessage() );
     }
   }
}
```

Figura 4.9: Exemplo de aplicação com a plataforma *WebObjects*

4.4. Recursos e Restrições da Versão Atual

O estágio atual de implementação da plataforma *WebObjects* oferece aos seus usuários os recursos básicos de consulta a um banco de dados orientado a objetos, ou seja, propicia a execução de comandos utilizando somente a sentença `SELECT...FROM...WHERE` em sua concepção mais simplificada.

Sob esta sentença existem algumas limitações no que tange à plena utilização da sintaxe da linguagem. A primeira limitação diz respeito aos elementos projetados, que na versão atual são restritos aos itens de dados, ou seja, não admitem projeção de constantes e expressões. A experiência em desenvolvimento de aplicações mostrou que esta restrição não é um fator tão importante, haja vista que as expressões e constantes podem ser definidas no aplicativo que faz uso da plataforma. As mesmas restrições são aplicadas à cláusula WHERE, que só admite filtros com expressões lógicas e booleanas que envolvam itens de dados.

Na cláusula FROM não é admitido o produto cartesiano entre objetos presentes nas extensões `Texts`, devido ao alto custo que envolve tais consultas. As cláusulas `GROUP BY` e `ORDER BY`, os *subselect* e o *typecast* de classes ainda não estão disponíveis na plataforma, além de operações de combinação de resultados de `SELECTs`, tais como `UNION` e `MINUS`.

No que diz respeito à manipulação das informações, a restrição está no componente da plataforma responsável por analisar e extrair os dados dos documentos HTML, que está preparado apenas para reconhecer *tags* HTML, não reconhecendo *hyperlinks* incorporados nas páginas por meio de *java script* ou outros recursos que não sejam padrão HTML. Além disso, o reconhecimento dos arquivos binários está restrito àqueles incorporados com a *tag* IMG.

4.5. Considerações Finais

A plataforma *WebObjects* é uma biblioteca composta por 55 classes distribuídas em 6 pacotes que implementam aproximadamente 13.000 linhas de especificação Java envolvendo comandos e comentários.

Este código apresenta alta complexidade pois realiza tarefas que envolvem processos computacionais não triviais, dos quais podemos destacar a implementação da máquina virtual, a interface com os mecanismos de busca, a captura dos dados da Web com o gerenciamento da programação concorrente e a administração dos *buffers* compartilhados.

Estes fatores, conjugados ao pouco tempo de uso da API, impõem uma natural necessidade de amadurecimento da solução, que só será atingido à medida que mais usuários utilizem os recursos oferecidos pela plataforma *WebObjects*. Assim, à proporção que a mesma for utilizada, ocorrerá uma natural depuração de problemas e deficiências, fazendo com que a plataforma ganhe em robustez, desempenho e funcionalidade.

Capítulo 5

V. Resultados Experimentais

Este capítulo é dedicado à descrição dos testes experimentais com a plataforma *WebObjects* cujo principal objetivo foi avaliar a plataforma sob quatro aspectos principais:

- ✓ Desempenho das consultas;
- ✓ Precisão dos resultados das consultas em relação à condição de busca;
- ✓ Facilidade de desenvolvimento de aplicações Web;
- ✓ Comparação com ferramentas convencionais de busca de documentos na Web.

5.1. Modelo de Testes

Os testes efetuados para a avaliação experimental da plataforma *WebObjects* buscaram simular situações cotidianas dos usuários, onde as necessidades de consulta a informações da Web fossem representadas por meio de comandos OQL submetidos à plataforma. Além disso, as necessidades específicas de projetistas de aplicações também foram avaliadas, no sentido de diagnosticar o verdadeiro potencial de desenvolvimento oferecido pela *WebObjects.*

Com este propósito, a avaliação experimental foi dividida em fases, sendo cada uma delas dedicada a um objetivo específico. São elas:

Fase 1: Estes testes foram dedicados a avaliação de desempenho das consultas submetidas à plataforma. Para isso, as consultas foram divididas por tipo de acesso, de modo que o desempenho de cada um dos tipos fosse mensurado isoladamente;

Fase 2: Nesta fase foi verificada a precisão dos resultados obtidos pelo uso da plataforma. A definição de precisão neste contexto está associada à paridade entre a condição de consulta e o conteúdo da informação recuperada, ou seja, se os dados recuperados estão disponíveis e realmente de acordo com a condição

especificada. O valor de referência buscado para a precisão foi de 100%, haja vista que um dos objetivos da plataforma é corrigir as possíveis imprecisões de resultados recuperados a partir das máquinas de busca. Os testes foram realizados para provar que todos os dados retornados por uma consulta *WebObjects* referem-se a documentos disponíveis e de acordo com a condição de busca;

Fase 3: O objetivo desta fase foi determinar o quanto que a plataforma *WebObjects* agrega de valor ao processo de desenvolvimento de aplicações que fazem uso do banco de dados da Web. Para isso, foram desenvolvidas várias aplicações utilizando a plataforma. São elas: *HtmlExplorer*, *TableExplorer* e *WebExplorer*. Além disso, pode-se mensurar o comportamento, inclusive de desempenho, da plataforma quando utilizada em um ambiente de produção com uma aplicação destinada ao usuário final;

Fase 4: Esta fase foi dedicada a uma análise comparativa de uma das ferramentas desenvolvidas, a *WebExplorer*, com a máquina de busca *Google*.

Este modelo de testes permitiu um perfeito diagnóstico do valor agregado pela plataforma *WebObjects* ao processo de consulta aos dados da Web, bem como possibilitou a compreensão de suas deficiências e necessidades de evoluções.

5.2. Fase 1 - Avaliação de Desempenho das Consultas

O desempenho das consultas submetidas à execução pela plataforma *WebObjects* foi avaliado basicamente pela medição dos tempos de resposta obtidos por uma aplicação Java que de forma simples e direta submetia o comando OQL à execução pela plataforma. Esta estratégia torna possível aproximar os valores obtidos no sentido de mensurar mais precisamente o desempenho da plataforma, descartando o tempo associado ao processamento das tarefas das aplicações, tais como montagem da interface com o usuário.

A aplicação foi executada em um microcomputador *Pentium* II 400 Mhz, com 256 Mb de memória RAM, disco IDE de 20 Gb, placa de rede de 10Mbps e sistema operacional *Windows NT Workstation 4.0*. O canal de comunicação utilizado para capturar os dados da Web possuía a capacidade de 1 Mbps, sendo que o mesmo era compartilhado por todo *website* da Universidade Tiradentes, ou seja, ele é utilizado como canal de comunicação para cerca de

900 computadores e diversas aplicações Web disponibilizadas pela Universidade aos seus alunos.

Para avaliação de desempenho, as consultas foram classificadas nas seguintes categorias em relação ao tipo de dado recuperado: características gerais do documento, conteúdo dos documentos, *hyperlinks*/arquivos binários (imagens e sons) e acesso a dados relacionais.

5.2.1. Consulta a Informações sobre as Características Gerais dos Objetos

Este tipo de consulta está associado à recuperação de informações como tipo/tamanho do documento e domínio a que pertence, ou seja, a todas os dados obtidos através de uma solicitação direta ao *web server* e que não necessite da captura e processamento do conteúdo do objeto. São exemplos destas consultas, comandos OQL com a seguinte forma:

```
select w.url, w.getDomain(), w.getIpAddress(), w.getModify()
   from WebObjects w
   where w.url in( "http://www.unit.br", "http://www.altavista.com",
                   "http://www.google.com", "http://www.ufpb.br",
                   "http://www.terra.com.br/index.htm" );
```

Esta consulta recupera os dados básicos dos documentos através de suas URLs. Neste caso, a plataforma *WebObjects* não necessita utilizar os catálogos para obtenção dos objetos, isto porque a partir da URL de cada documento é possível acessar diretamente do *web server* os dados solicitados.

Tais consultas podem ser realizadas através de qualquer uma das extensões, haja vista que a plataforma realiza a recuperação sob demanda das informações. Assim, se o usuário não fizer nenhuma referência a informações que necessitem a obtenção do conteúdo do documento, a plataforma só irá recuperar as informações gerais do objeto, ou seja, aquelas que são absolutamente necessárias para que o objeto seja instanciado.

O desempenho médio apresentado para recuperação deste tipo de informação foi da ordem de 2 objetos por segundo, denotando uma excelente capacidade de execução destes tipos de consulta.

5.2.2. Consulta ao Conteúdo dos Documentos HTML

Esta categoria abrange as consultas cuja computação do resultado necessita a pesquisa aos catálogos de uma máquina de busca, com uma posterior recuperação e processamento do conteúdo dos documentos. Isto ocorre porque a *WebObjects* necessita utilizar como ponto de partida da execução as URLs que atendem a condição de consulta e que são recuperadas com o acesso aos catálogos, fazendo em seguida a recuperação do documento de cada URL para validação de sua disponibilidade, ou para verificar se o mesmo ainda possui o conteúdo sincronizado com o catálogo *off-line* da máquina de busca. É exemplo típico deste tipo de consulta o seguinte comando OQL:

```
select t.url, t.getSize(), t.getDataFormat()
   from Texts t
   where ( t.content like "UFPB" and t.content like "Mestrado" ) or
         ( t.content like "UNIT" and t.content like "Fabio Soares" );
```

Perceba que neste comando o usuário não filtra os dados da Web por URLs de seu conhecimento. Ao invés disso, busca recuperar os documentos por características que são internas ao seu conteúdo. Daí a necessidade de acessar os catálogos das máquinas de busca e validar o seu conteúdo atual.

O desempenho destas consultas está diretamente associado à quantidade de *threads* utilizadas no processo, número este que depende da capacidade do servidor. Na avaliação experimental foram utilizadas 20 *threads*, somando-se todas os pontos do fluxo de execução da consulta onde as mesmas são necessárias.

Na medição experimental, esta categoria de consultas obteve uma taxa média de recuperação do conteúdo de aproximadamente 50 Kbps. Isto implica em dizer que a plataforma leva 0,16 segundos para processar 1 Kbyte de um documento disponível na Web.

A exceção a este desempenho está em consultas onde no processo de execução, a ordem das URLs retornadas a partir da máquina de busca faça com que documentos com alta latência de recuperação sejam processados no final da composição do resultado. Isto ocorre porque neste ponto a plataforma não obtém as vantagens da utilização das *threads*, pois o documento de alta latência passa a ser o último a ter os dados recuperados, fazendo com que a plataforma não possa continuar a execução até que seja finalizada a recuperação do documento.

5.2.3. Consulta aos Hyperlinks e Arquivos Binários dos Documentos HTML

Nesta categoria estão as consultas que necessitam recuperar informações sobre os *hyperlinks* e arquivos binários (imagens e sons) de um documento HTML por meio da varredura e análise dos *tags* que representam tal informação. Assim, o processamento destas consultas passa necessariamente por uma consulta a conteúdo. Adicionalmente, para cada *tag* localizado, a plataforma recupera seu respectivo objeto para validar sua disponibilidade.

Um exemplo típico destas consultas possui o seguinte formato:

```
select t.url, l.baseLabel, l.refObject.url
    from Texts t, t.links l
    where ( t.url = "http://www.unit.br/index.htm" ) and
          ( l.getDepth() <= 1 );
```

Neste exemplo, apesar da URL da página base para a consulta ser informada, a *WebObjects* precisa recuperar o conteúdo do documento para extrair as informações referentes aos *hyperlinks* existentes na página.

Para estas consultas, a *WebObjects* apresentou um desempenho médio de aproximadamente 0,2 segundo por objeto recuperado — *hyperlink* ou arquivo binário —, ou seja, uma média de 3 objetos por segundo.

A medição experimental mostrou uma restrição para aplicações que utilizem o atributo `links` do objeto `Text` para recuperação de *hyperlinks* indiretamente acessados a partir do objeto. Isto ocorreu quando a consulta necessita obter *hyperlinks* com uma profundidade superior a 3, ou seja, o tempo de resposta obtido com estas consultas torna proibitiva a criação de aplicações *on-line*, sendo indicado nestes casos um processamento em *batch* ou *off-line*.

5.2.4. Consulta aos Dados de Tabelas Relacionais

O desempenho para processamento dos dados obtidos de tabelas relacionais foi totalmente compatível com qualquer aplicação Web que acesse uma base de dados, tais como sistema de comercio eletrônico ou *home bank*.

A plataforma obteve uma taxa média de 5 linhas recuperada por segundo de processamento. Esta taxa aplicada a uma aplicação típica de acesso a banco de dados que mostra os dados recuperados em blocos de linhas propiciaria um excelente desempenho. Por exemplo, se

utilizássemos a métrica de exibir os dados em blocos de 13 linhas, teríamos um tempo médio de resposta na ordem de 3 segundos.

5.3. Fase 2 - Avaliação da Precisão

O aumento da precisão das consultas a documentos da Web foi um resultado positivo obtido pelo uso da plataforma *WebObjects.* Isto foi possível primeiramente porque o aumento do poder semântico das consultas viabiliza a especificação de pesquisas que melhor representem a necessidade do usuário. Além disso, na avaliação da consulta é realizada uma validação dos dados obtidos a partir da máquina de busca no sentido de corrigir duas distorções de resultados muito comumente enfrentadas pelos usuários: a indisponibilidade dos documentos ou a não conformidade do catálogo com o conteúdo atual do documento.

A indisponibilidade de documentos recuperados por uma consulta está principalmente associada à forma dinâmica com que os *websites* tornam-se disponíveis e indisponíveis no decorrer do tempo. Por outro lado, o fato do catálogo da máquina de busca ser *off-line* faz com que o reconhecimento da remoção definitiva de documentos da Web não seja ágil.

Este mesmo catálogo *off-line* faz com que mudanças em páginas da Web não sejam rapidamente refletidas no resultado da consulta, propiciando que as máquinas de busca retornem em muitas situações documentos que não estão em conformidade com os termos especificados. Este problema foi muito observado em páginas de freqüente atualização, como são aquelas oriundas de jornais e revistas.

Estas deficiências deixam de existir com a utilização da plataforma *WebObjects,* pois o seu esquema de funcionamento elimina tais distorções. Para comprovar esta afirmação, a tabela 5.1 traz o resultado comparativo de uma consulta realizada no *AltaVista* e com o uso da plataforma *WebObjects*. O objetivo da consulta foi encontrar os documentos que possuam em seu conteúdo as palavras "Universidade Tiradentes" e "Mestrado". No *AltaVista* os dois termos foram utilizados na pesquisa, enquanto que na plataforma *WebObjects* foi utilizado o seguinte comando OQL:

```
select t.url, l.getSize(), t.getModify()
    from Texts t, t.links l
    where ( t.content like "Universidade Tiradentes" ) and
          ( t.content like "Mestrado" );
```

AltaVista	WebObjects
vulcano.dsc.ufpb.br/copin/unit/index.htm	Documento recuperado
vulcano.dsc.ufpb.br/copin/unit/avaldisc.htm	Documento recuperado
www.unit.br/posgraduacao/posgraduacao.htm	Documento recuperado
www.unit.br/posgraduacao/capacitacao.htm	Documento recuperado
vulcano.dsc.ufpb.br/copin/unit/assuntosmono.htm	Documento não recuperado: não possui a palavra mestrado
www.unit.br/guiaacademico/daahtm.htm	Documento não recuperado: não possui a palavra Universidade Tiradentes
www.squema.com.br/links.htm	Documento recuperado
usuarios.netdados.com.br/barros/daniel6.htm	Documento não recuperado: indisponível
www.unit.br/engamb/apresentacao.htm	Documento recuperado
www.unit.br/efreire/	Documento recuperado
www.ufs.br/frames/quem_e_quem/ccsa.htm	Documento recuperado
www.scv-araraquara.com.br/universidades.htm	Documento recuperado
www.ulbrajp.com.br/~contabeis/pbenedito.html	Documento recuperado
www.icb.ufmg.br/~romeucg/Portugues/index.html	Documento recuperado
www.ibict.br/biblioteca/capacitacao.htm	Documento recuperado
usuarios.netdados.com.br/advogado/links.html	Documento não recuperado: indisponível
www.colegiojoliveira.com.br/joliveira/links.html	Documento não recuperado: indisponível
www.unit.br/zemaria/Curriculo.htm	Documento recuperado
www.unit.br/ebc/curriculo.html	Documento recuperado
www.cnpq.br/noticias/index.htm	Documento não recuperado: Não possui nenhuma das palavras
www.cesur.br/Facjar/historico/hist_index.html	Documento não recuperado: indisponível
allchemy.iq.usp.br/sintetizando/superior.html	Documento recuperado
www.peq.coppe.ufrj.br/docentes/jcarlos/	Documento recuperado
www.cofecon.org.br/homepage/hp_var/pre_bras/pre2000.html	Documento recuperado
www.jfse.gov.br/mag-curriculum.html	Documento não recuperado: indisponível
cienciadaeducacao.vilabol.uol.com.br/faculdades.htm	Documento não recuperado: indisponível
www.ull.es/publicaciones/latina/n/99/94jornal.htm	Documento recuperado
www.irdeb.ba.gov.br/internetlinks.htm	Documento recuperado
www.soutomaior.eti.br/mario/paginas/dicfij.htm	Documento recuperado
www.pbnet.com.br/openline/fsatiro/links.html	Documento recuperado
www.unit.br/boletim/pag4.html	Documento não recuperado: não possui a palavra mestrado
cev.ucb.br/pipermail/cevpg-l/2001-July/000132.html	Documento não recuperado: indisponível
www.ifi.unicamp.br/jornal-da-ciencia/msg00661.html	Documento recuperado
www.sbq.org.br/publicacoes/beletronico/bienio2/boletim245.htm	Documento recuperado

Tabela 5.1: Uma consulta a documentos com *AltaVista* e *WebExplorer*

O resultado da pesquisa apresentou os problemas mencionados em 11 dos 34 resultados obtidos a partir do *AltaVista*, ou seja, neste exemplo específico, aproximadamente 32% dos elementos retornados pela máquina de busca não atendem mais à condição de consulta definida pelo usuário. Os testes experimentais apontaram que em média 20% das URLs retornadas pelas máquinas de busca não atendiam mais à condição de consulta. Em casos extremos, este percentual chegou a 50%.

5.4. Fase 3 - Aplicativos Desenvolvidos com a Plataforma *WebObjects*

Nesta fase da avaliação experimental foram desenvolvidas 4 aplicações que fazem uso dos recursos da plataforma *WebObjects*: *HtmlExplorer*, *TableExplorer* e *WebExplorer* nas versões JSP e cliente/servidor. As próximas subseções apresentam detalhadamente cada uma das aplicações.

Estas aplicações podem ser classificadas por dois critérios distintos: a arquitetura e a natureza das consultas. Quanto à arquitetura, as consultas podem ser multicamadas (*n-tier*) com uso de JSPs ou cliente/servidor através de aplicações Java convencionais que usam o *swing*[15]. Quanto à natureza das consultas, as aplicações poderiam ser classificadas em: (a1) consultas com estrutura pré-definida na aplicação que acessam dados de documentos semi-estruturados HTML; (a2) consultas com estrutura pré-definida na aplicação que manipulam dados estruturados oriundos de tabelas relacionais; e (a3) que executam qualquer consulta especificada pelos usuários a dados semi-estruturados ou estruturados, cuja resposta seja um `bag` de `strings`.

Para as aplicações multicamada implementadas através de JSPs, foram utilizados dois servidores de aplicações, o *WebLogic* e o *TomCat*. A versão do *WebLogic* foi a 5.5, cuja restrição está associada ao fato de não funcionar com a versão 1.4 do JDK, que é um recurso exigido para o perfeito funcionamento da API de acesso à máquina de busca *Google*. Por isso, foi utilizado também o *TomCat* na versão 4.0, que apresentou um desempenho de até 50% menor do que o *WebLogic* em determinados tipos de consulta.

5.4.1. Aplicação *HtmlExplorer*

HtmlExplorer é uma aplicação multicamada da categoria **a1**. A partir da especificação de uma URL válida para um documento da Web, o *HtmlExplorer* obtém informações gerais sobre o documento — como tamanho, formato e data de modificação —, seus *hyperlinks* e os arquivos de imagem vinculados. A figura 5.1 mostra a interface utilizada pelo usuário.

Para recuperação das informações da Web e a posterior formatação da interface, o *HtmlExplorer* utiliza o seguinte código Java:

---

[15] Pacote Java para criação de interfaces gráficas com o usuário

```
Query q = new Query( "select t " +
                          "from Texts t " +
                          "where t.url = \"" + urlDesc + "\";" );
```

Este comando recupera o objeto `Text(t)` para a URL obtida pela aplicação — variável `urlDesc` — e a partir dele são obtidas todas as informações para a montagem da primeira parte da interface de resultado da figura 5.1, que exibe as informações iniciadas com o título e finalizadas o tamanho em bytes do arquivo HTML. Estes dados são recuperados através de chamadas aos métodos da classe `Text`, tais como `t.getTitle()` para o título e `t.getIpAddress()` para o endereço IP do servidor.

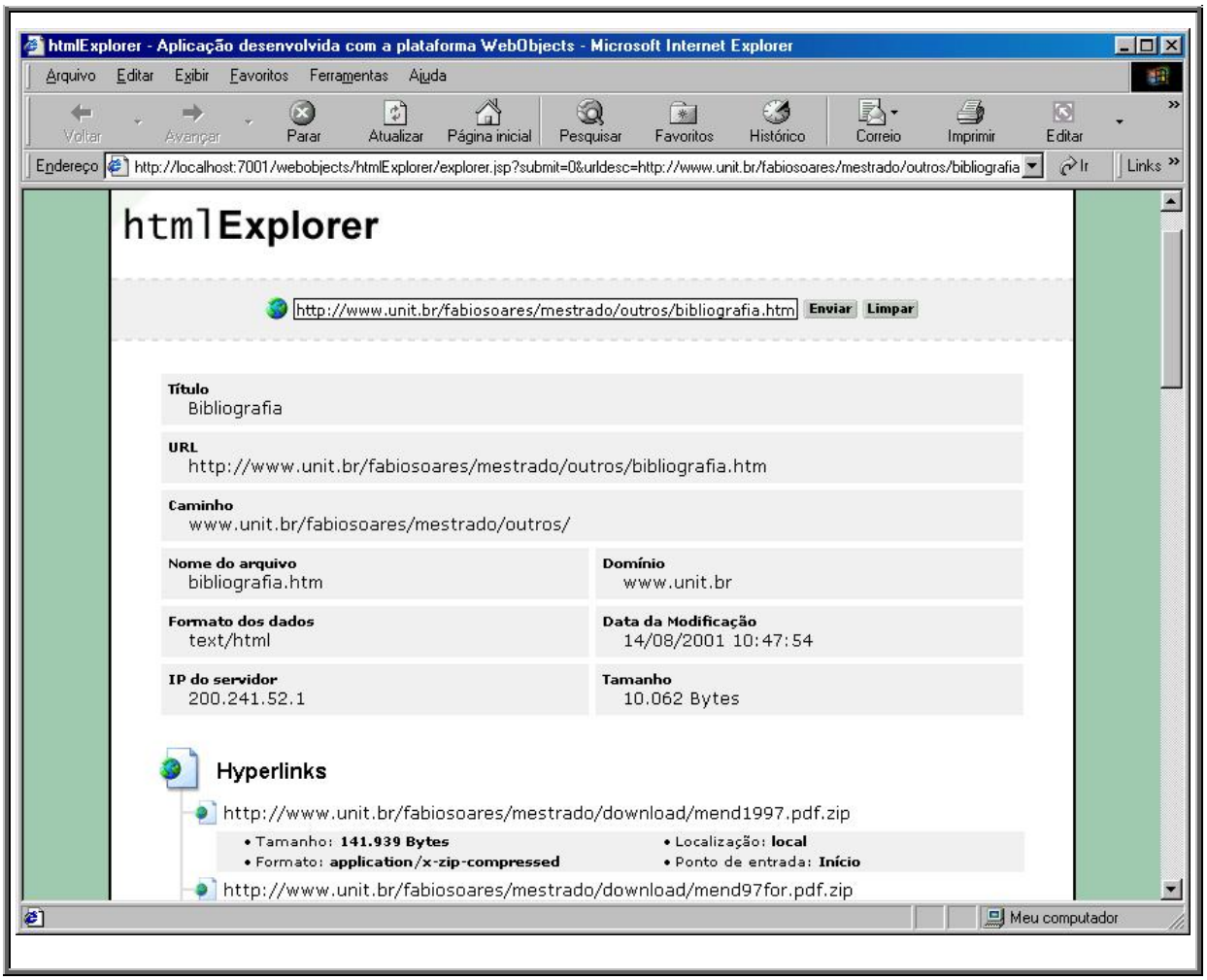

Figura 5.1: Aplicativo para exploração de estrutura de documentos HTML

Na segunda parte da interface, o *htmlExplorer* mostra todos os *hyperlinks* acessados diretamente a partir de um documento HTML. Como vimos no capítulo 3, estes dados podem ser obtidos restringindo os *hyperlinks* apenas aqueles cuja profundidade — resultado do método `getDepth()` — seja igual a 1. Para cada *hyperlink* recuperado a interface mostra o

texto associado a ele no documento (rótulo) — se não houver mostra a URL —, o tamanho e formato do objeto referenciado, o tipo do *hyperlink* e o ponto de entrada no documento referenciado.

O processo de montagem da segunda parte do formulário é iniciado por uma chamada ao método `t.getLinks()` do objeto `Text(t)` para recuperar uma instância da classe `LinkCollection`, cuja funcionalidade permite acessar todos os *hyperlinks* existentes em um determinado documento. Para este objeto é aplicado um filtro que restringe a recuperação aos *hyperlinks* acessados diretamente a partir do documento. Daí, um laço faz uso dos métodos `NextElement()` e `hasMoreElements()` para obter cada *hyperlink* direto — objeto da classe `Link` — existente na página, e através dos métodos da classe, recuperar todas as informações necessárias à montagem da interface.

Para cada *hyperlink* recuperado, a aplicação oferece duas opções. Com um clique no ícone à esquerda do rótulo do *hyperlink* — no exemplo da figura 5.1 é o texto com a URL "http://www.unit.br/fabiosoares/mestrado/download/mend97for.pdf.zip" —, o usuário pode abrir em uma outra janela o documento referenciado. Já com um clique sobre o próprio rótulo é possível executar a aplicação passando como parâmetro a URL que define o documento de destino do *hyperlink*. Em outras palavras, um clique no rótulo equivale ao usuário digitar na interface da aplicação a URL do documento de destino do *hyperlink* e clicar no botão de enviar para explorar a estrutura do documento. Obviamente, está opção só está habilitada para *hyperlinks* cujo documento de destino tenha o formato HTML.

Quando o *hyperlink* está indisponível, o ícone associado a ele apresenta uma tonalidade acinzentada tal como exibido na figura 5.2. Além disso, são desabilitadas as opções de visualização do conteúdo e da estrutura do documento. Para estes casos, o detalhamento das informações do objeto referenciado é substituído por indicações claras da indisponibilidade do *hyperlink*. Esta funcionalidade pode ser bastante útil para administradores de *websites* na tarefa de localizar *hyperlinks* perdidos em suas *home pages*.

A figura 5.2 ainda exibe a parte final da interface com as informações sobre os arquivos de imagens (`BinaryFiles`) associados ao documento. A montagem desta parte da interface é muito semelhante à parte dos *hyperlinks*. Ela é feita por uma chamada ao método `t.getBinFiles()` do objeto `Text(t)` para recuperar uma instância da classe `BinCollection` cuja funcionalidade permite acessar todos os arquivos de imagens

associados ao documento. A partir deste ponto o processamento se dá por um laço que em cada passagem recupera um objeto `BinaryFile`, e com chamadas a seus métodos monta os dados exibidos na interface.

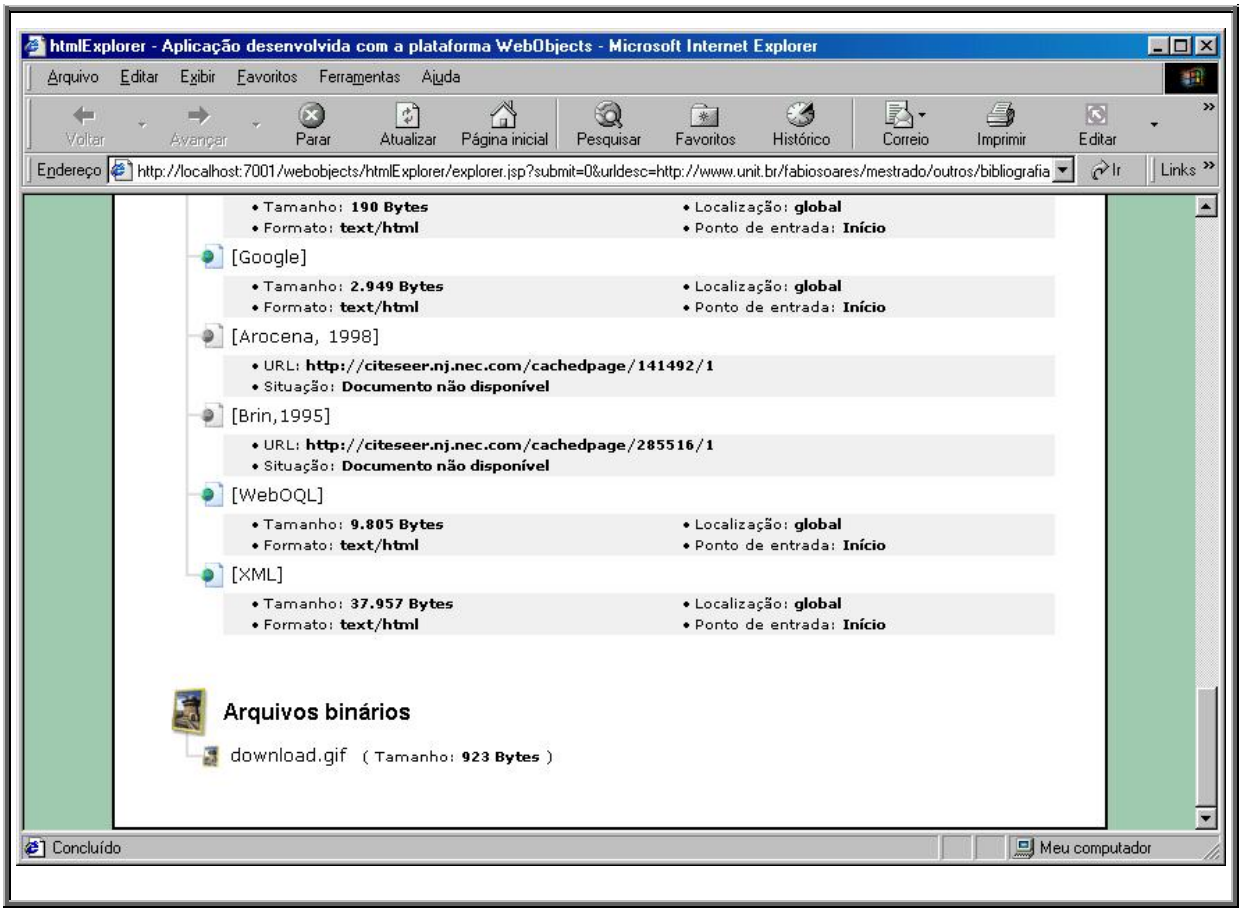

Figura 5.2: Parte final da interface do *HtmlExplorer*

Este aplicativo apresentou um tempo médio de resposta de aproximadamente 4 segundos para recuperação das informações de um documento contendo em média 10 objetos associados (*hyperlinks* ou imagens).

5.4.2. Aplicação *TableExplorer*

O *TableExplorer* é uma ferramenta muito similar às oferecidas por fabricantes de SGBDs e de ferramentas de desenvolvimento de sistemas para visualização de conteúdo e estrutura de tabelas relacionais. Ela pode ser classificada como uma aplicação multicamada da categoria **a2**.

Com o *TableExplorer*, o usuário especifica a URL que define a tabela a ser explorada e o aplicativo recupera seu conteúdo. Como podemos observar na figura 5.3, a estrutura da tabela

pode ser visualizada através de uma outra janela acessada a partir do *hyperlink* rotulado pela literal "Estrutura da Tabela".

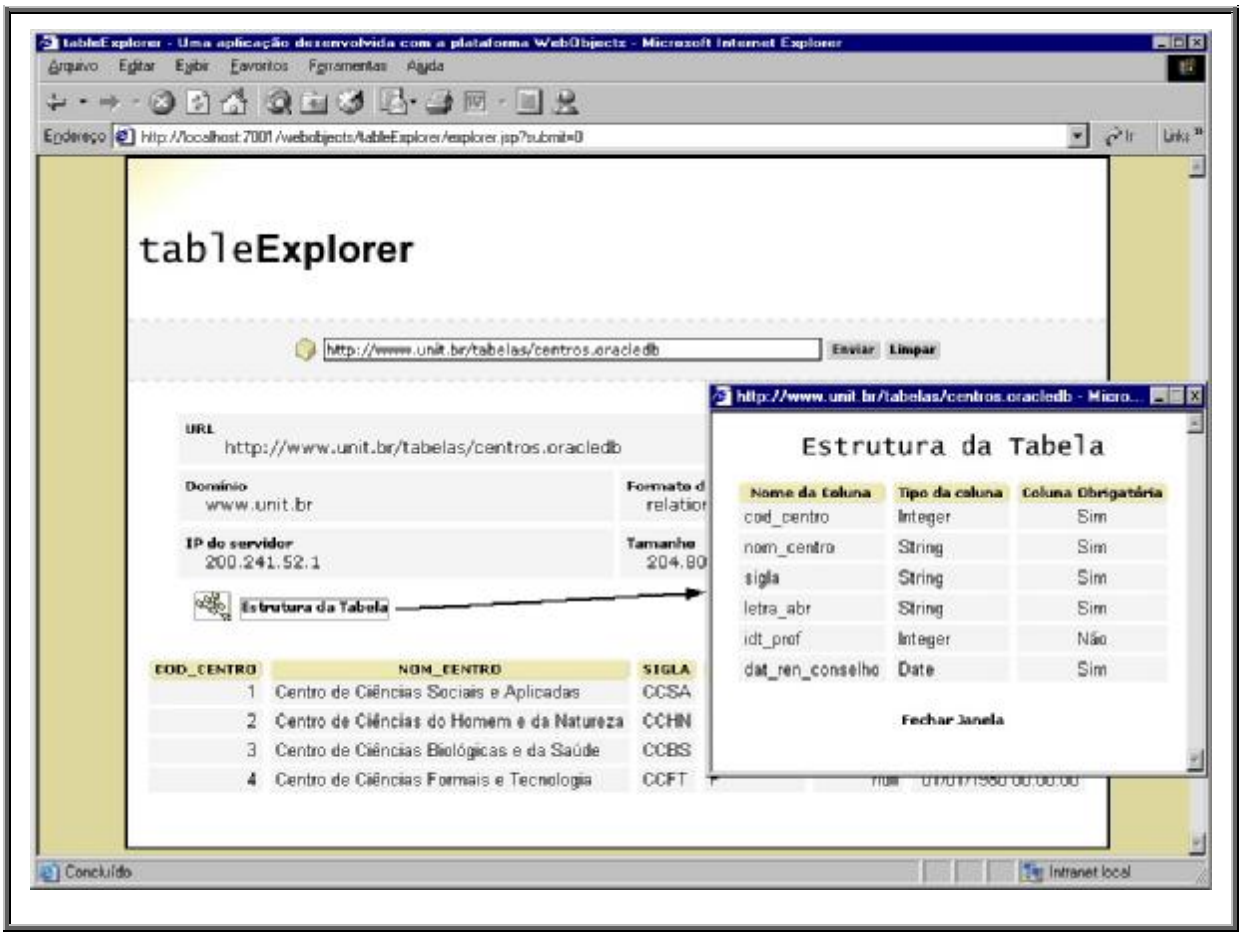

Figura 5.3: Aplicativo para exploração de dados em tabelas relacionais

A estratégia para recuperação dos dados da tabela é semelhante à utilizada no *HtmlExplorer*, sendo que o objeto é recuperado da extensão `Relations`, sendo portanto uma instância da classe `Relation`. As informações das linhas da tabela são obtidas através do processamento de um objeto da classe `RowCollection`, obtido por uma chamada ao método `getRows()`. A partir deste ponto, a montagem do *layout* de reposta é realizada por um laço que processa cada uma das linhas recuperadas da coleção. O cabeçalho da resposta é construído por uma chamada ao método `getColumns()`, cujo resultado é um array de registros `DefColumn` — implementado no Java como uma classe — do qual pode-se extrair ordenadamente o nome das colunas que compõem uma tabela.

Para esta aplicação foi realizado um controle adicional, pois a mesma exibe os dados da tabela em páginas com 10 linhas cada. Assim, quando uma tabela possui mais que 10 linhas, a aplicação exibe um *hyperlink* rotulado pela literal "próximo", por onde é possível a

visualização das próximas 10 linhas da tabela. Para viabilizar este recurso, a aplicação armazena na sessão HTTP o objeto `RowCollection`, de forma que quando a página é chamada a partir do *hyperlink* "próximo", o objeto é recuperado da sessão e a exibição dos dados da tabela é reiniciada a partir do ponto onde a página anterior terminou.

Na exibição da estrutura da tabela, a janela *popup* recebe como parâmetro a URL e obtém os dados da estrutura da tabela com o seguinte comando:

```
Query q = new Query( "select c.name, c.required, c.type " +
                         "from Relations r, r.columns c " +
                         "where t.url = \"" + urlDesc + "\";" );
```

Para a montagem do resultado, é necessário apenas o processamento dos elementos retornados pela classe `Query`.

A utilização do aplicativo *TableExplorer* executando no servidor de aplicações *TomCat*, cujo desempenho foi muito inferior ao obtido pelo *WebLogic*, obteve um tempo médio de resposta de 3 segundos para recuperação de uma página com os dados gerais do objeto e mais 10 linhas do seu conteúdo. A recuperação dos dados da estrutura da tabela levou em média 5 segundos.

5.4.3. Aplicação *WebExplorer*

Este aplicativo pode ser enquadrado na categoria **a3**, sendo disponibilizado nas versões cliente/servidor e multicamada. A figura 5.4 mostra a interface JSP para a aplicação, sendo que a forma e o estilo são muito próximos do definido para a versão cliente/servidor.

A funcionalidade básica desta aplicação é oferecer aos usuários a possibilidade de interagir diretamente com a plataforma *WebObjects* através de comandos OQL. Para isso, a interface disponibiliza uma caixa de texto pela qual é especificado o comando, e este é submetido à execução pela plataforma através do botão "executar". A aplicação instancia um objeto da classe `Query` (cursor) com o comando especificado, processando-o seqüencialmente para montagem do resultado.

Esta aplicação converte automaticamente todo item de dado projetado no resultado para literais (tipo `String` do Java), que são os únicos tipos possíveis de serem exibidos por uma interface deste tipo. Assim, se um objeto complexo é projetado em uma consulta, o resultado

exibido para o objeto será o resultado da execução do método padrão `toString()`. Além disso, de maneira semelhante ao *TableExplorer*, o *WebExplorer* utiliza a sessão do usuário para permitir que o resultado seja exibido em páginas que contêm no máximo 10 elementos.

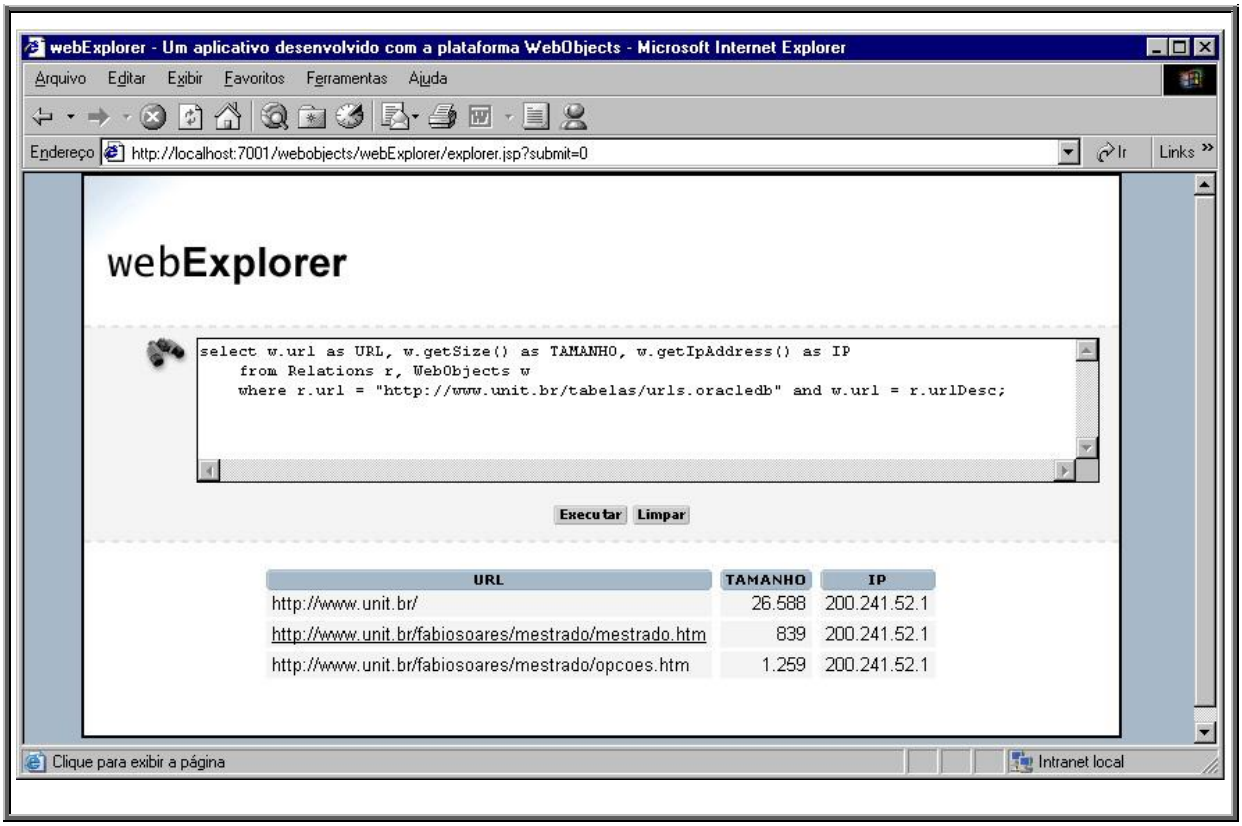

Figura 5.4: Aplicativo que executa comandos OQL

A dificuldade associada à criação deste aplicativo está diretamente relacionada à formatação do resultado na interface, isto porque o acesso aos dados da Web é facilmente realizado pela criação do objeto `Query` com o comando especificado, e o posterior processamento da resposta através dos métodos `hasMoreElements()` e `NextElement()`.

O tempo médio de resposta deste aplicativo foi de 13 segundos para montagem de um página de resultados contendo 10 elementos. Este desempenho é totalmente aceitável quando analisamos aplicações que envolvam dados obtidos a partir da Internet.

5.5. Fase 4 - Comparação de Ferramentas: *Google* x *WebExplorer*

Com a finalidade de tornar claro o potencial de consulta da plataforma *WebObjects*, foi realizado um comparativo entre o *WebExplorer* e o *Google*, uma das ferramentas mais utilizadas para consultas aos dados da Web.

Quando analisada sob a ótica do potencial de exploração do banco de dados da Web, o *Google* está em grande desvantagem pois é uma ferramenta restrita à recuperação de documentos por conteúdo, enquanto que o *WebExplorer* oferece recursos de exploração de topologia, consulta à base de dados relacionais e a documentos da Web. Além disso, o *Google* não oferece recursos de reestruturação do resultado da consulta.

Para delinearmos os aspectos de desempenho, clareza das consultas e precisão das respostas obtidas, foi realizada uma pesquisa visando encontrar a página do colega de "mestrado" "Nilton Oliveira". Para isso, estas duas palavras-chave foram submetidas ao *Google* através de sua interface, e ao *WebExplorer* com o seguinte comando:

```
select t.url as URL, t.getSize() as Tamanho, t.getModify() as Data
   from Texts t
   where ( t.content like "Nilton Oliveira" ) and
         ( t.content like "Mestrado" );
```

O primeiro fato a ser observado está na riqueza semântica do comando OQL quando comparado à busca por palavra-chave. Especificamente o *Google* não oferece recursos de especificação de consultas booleanas mais elaboradas, ou seja, consultas que necessitem de utilização de operadores `AND` e `OR`, muitas vezes com precedência modificada por parênteses. Além disso, o *WebExplorer* permite a recuperação de qualquer dado do documento que seja de interesse do usuário.

A figura 5.5 traz o resultado obtido pelos dois aplicativos[16]. Neste ponto podemos observar a já enfatizada vantagem da *WebExplorer* sobre o *Google* no aspecto precisão. Como os catálogos são *off-line*, *websites* que sofreram modificação ou estão indisponíveis no momento da busca continuam a ser retornados pelo *Google*, enquanto que com o uso da plataforma *WebObjects* estas informações não são mais recuperadas. No exemplo, o documento cuja URL é "http://www.sigaconsultoria.com.br/consultores/consultores.html" — rótulo Consultores — não possui mais a literal "Nilton Oliveira".

Além disso, o *WebExplorer* permite que o usuário recupere dados e aplique filtros que não são possíveis pelo *Google*. Se na consulta do exemplo o usuário precisasse restringir os documentos àqueles armazenados em servidores na faixa de endereço IP "128.241", poderia

[16] Consulta realizada em 15 de junho 2002

faze-lo no *WebExplorer* adicionando a condição `t.getIPAddress() like "128.241%"`, enquanto que no *Google* isto seria impossível.

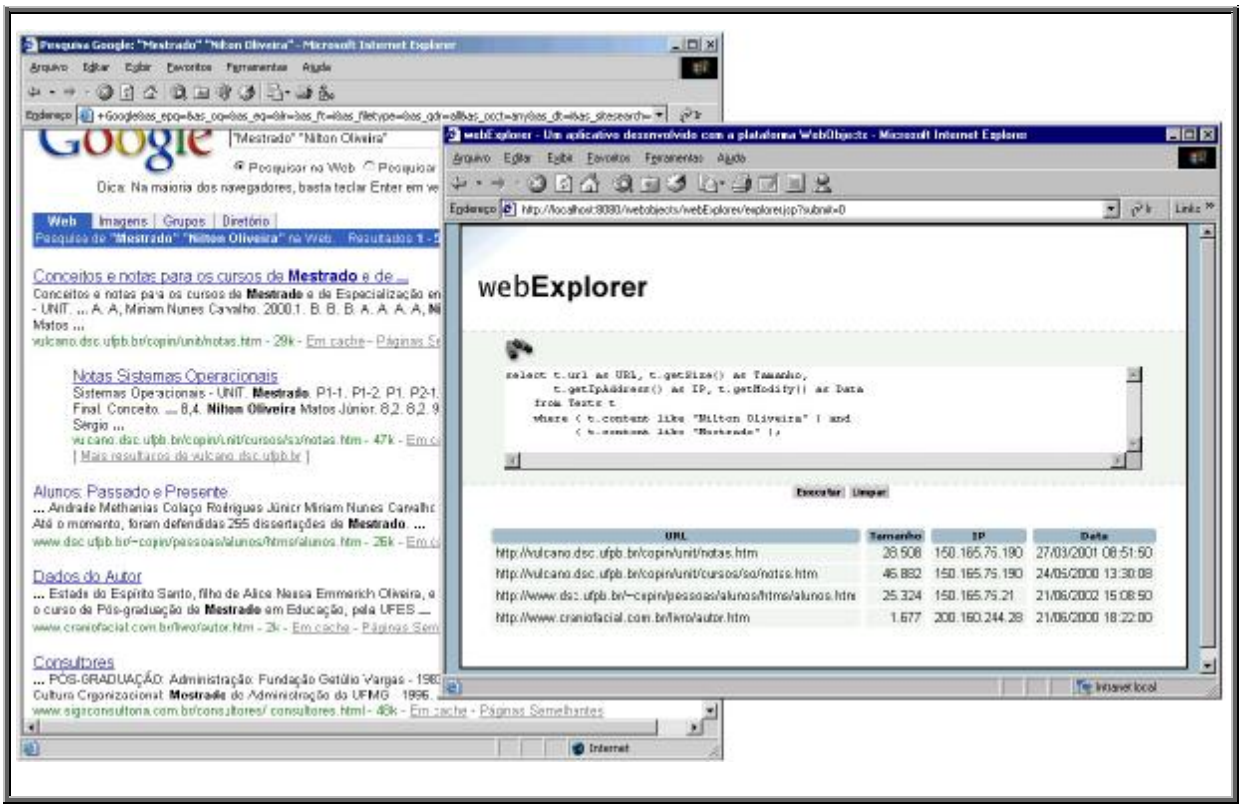

Figura 5.5: Consulta submetida para o *Google* e *WebExplorer*

Obviamente, o desempenho apresentado pelo *Google* foi superior ao do *WebExplorer* — uma média de 4 segundos —, pois o primeiro não tem o *overhead* associado à validação do dado na Web. No entanto, o desempenho do *WebExplorer* está de acordo com a maioria das aplicações de acesso a informações pela Internet.

5.6. Considerações Finais

De modo geral, a avaliação experimental da plataforma *WebObjects* superou as expectativas iniciais que apontavam para um baixo desempenho em relação às consultas realizadas nas máquinas de busca, haja vista que o custo associado à recuperação/validação dos dados na Web propiciava grandes perdas de desempenho.

Inicialmente este cenário se confirmou, quando foi observado que uma consulta para processar dados de 134 documentos gastou em média 15 minutos. Este panorama foi completamente modificado pela adição das *threads* que, nesta situação específica, reduziu o tempo de processamento para 2 minutos e meio. A programação concorrente foi o fator

principal para os ganhos de desempenho, melhorando o tempo médio de processamento da consulta em 60% quando comparada com o processamento linear.

Assim, se comparamos os tempos médios de resposta do *Google* e da *WebExplorer* perceberemos uma diferença de apenas 5 segundos na montagem de uma página contendo 10 elementos na resposta.

Outro fator animador está no fato da plataforma ter sido avaliada em um equipamento com baixa capacidade de processamento. A expectativa é que se a mesma for executada em um ambiente com maior poder de processamento, os resultados de desempenho obtidos sejam ainda melhores, principalmente porque o número de *threads* utilizadas pode ser aumentado consideravelmente.

Sob o aspecto precisão, a avaliação experimental validou a meta de 100% para as informações recuperadas pela plataforma, pois todos os dados indisponíveis e desatualizados foram identificados e descartados na resposta. Além disso, constatou por meio de dados reais a incapacidade das máquinas de busca de tratar a natureza dinâmica com que a Web muda.

A combinação precisão/desempenho oferecida pela plataforma *WebObjects* mostrou que é possível a criação de uma ferramenta automatizada para resolver o problema da consulta aos dados da Web.

A plataforma *WebObjects* apresentou uma extrema sensibilidade às mudanças nas condições de tráfego de informação na Web. Este resultado era previsível, isto porque a estratégia utilizada pelo processador de consultas está diretamente associada à recuperação de informações da Web, o que depende intrinsecamente da capacidade de trafegar bytes pela rede. As máquinas de busca sofrem muito menos com a variação do tráfego, pois seus catálogos são atualizados de forma *off-line*, de modo que a variação nas condições do tráfego não causa grande impacto para o usuário no momento da consulta *on-line*.

Finalmente, sob o aspecto de facilidade de criação de aplicações de acesso aos dados da Web, a plataforma *WebObjects* apresentou uma excelente avaliação, haja vista que apenas com a utilização de comandos OQL embutidos em um componente de processamento seqüencial do resultado — classe `Query` —, é possível o desenvolvimento de qualquer tipo de aplicação que faça uso dos seus recursos. Neste requisito, a *WebObjects* superou inclusive os novos serviços disponibilizados pelas máquinas de busca, que oferecem uma interface proprietária que

agrega certa complexidade na especificação das solicitações de consulta e no tratamento da resposta.

Os testes experimentais apontaram alguns problemas intermitentes em relação à aceitação de solicitação HTTP por alguns *web servers*. Em contatos realizados com alguns *websites*, foi apontado que o erro poderia estar associado à configuração de *firewall* e política de segurança. Em virtude de não possuirmos nenhum tipo de interferência na administração destes *websites*, o diagnóstico e resolução destes problemas fugiram da nossa alçada, e conseqüentemente do escopo deste trabalho.

Capítulo 6

VI. Conclusões e Perspectivas do Trabalho

Dar a Web as desejáveis características de um banco de dados convencional é uma tarefa que ainda demandará muito esforço de pesquisa, pois são inúmeros os problemas que envolvem tal abordagem e cuja solução é bastante complexa. No entanto, é consenso a necessidade de viabilizar mecanismos de consulta que potencializem a capacidade de recuperação e manipulação dos dados presentes nesta incomensurável fonte de conhecimento.

O modelo de consulta oferecido pelas ferramentas convencionais mais utilizadas, as máquinas de busca como *AltaVista* e *Google*, está aquém das necessidades dos usuários por vários motivos: resultados desatualizados, pouca flexibilidade no acesso/manipulação dos dados e complexidade na especificação de consultas mais elaboradas, entre outros. Em conseqüência, os usuários não dispõem de facilidades para especificar consultas com o poder semântico necessário para selecionar estritamente objetos de seu interesse. Além disso, os processos atuais de consulta à Web fazem com que os usuários desprendam muito esforço em um processo lento e tedioso de análise e seleção dos resultados obtidos com as máquinas de busca.

Por outro lado, uma gama de projetistas de aplicação gostaria de acessar a Web como já o fazem em bancos de dados convencionais, ou seja, utilizando uma linguagem de alto poder semântico. Assim, seria possível a criação de novas aplicações que explorassem este banco de dados com a forma e estrutura particulares de cada grupo de usuários. Ademais, é um requisito importante para o desenvolvimento de aplicações a necessidade de manipulação dos dados da Web com recursos mais abrangentes que o tratamento de URLs que atendam a uma condição de consulta.

Nossa proposta vai ao encontro destas necessidades, quando oferece aos usuários da Web um processador de consultas capaz de resolver pesquisas especificadas em termos de comandos OQL/ODMG sobre um esquema conceitual de banco de dados que modela a Web — documentos HTML e tabelas relacionais — segundo o modelo orientado a objetos padrão ODL/ODMG.

6.1. Síntese dos Resultados

O trabalho desenvolvido na construção da plataforma *WebObjects* atingiu completamente os seus objetivos iniciais. O primeiro, e talvez o mais importante entre todos, foi o fato da plataforma oferecer aos seus usuários a possibilidade de especificar consultas em uma linguagem com alto poder de abstração e expressão, a OQL. O segundo objetivo diz respeito à independência de ambiente operacional, que foi cumprido pela utilização da linguagem Java na implementação da *WebObjects*, possibilitando a sua execução nas mais variadas plataformas de *hardware* e *software*.

Outros objetivos são relativos à forma e tipos de objetos manipulados e tratados pela plataforma. Estes foram rigorosamente cumpridos quando a *WebObjects* viabiliza consultas a conteúdo e estrutura de *hyperlinks* dos documentos HTML, como também permite o acesso a bases de dados relacionais. O mecanismo pelo qual os dados são acessados também estão de acordo com os objetivos traçados, haja vista que oferece, de modo simples e objetivo, a manipulação e tratamento dos objetos de dados capturados a partir da Web.

Finalmente, a possibilidade de um esquema de dados extensível foi obtida através da utilização de um modelo de dados orientado a objetos, que permite a adição de novas estruturas de dados, que não as já propostas neste trabalho, através da extensão das características de sua classe básica `WebObject`. Isto é possível porque o esquema de dados ODL e a própria arquitetura da solução foram projetados para permitir a extensão da solução. O esquema de dados ODL, que por definição já permite a extensão através da herança, foi definido incorporando na classe `WebObject` todas as características que devem ser comuns a qualquer objeto manipulado pela plataforma. A arquitetura também foi definida no sentido de transferir a responsabilidade do tratamento das particularidades de cada informação para suas respectivas classes de dados, de modo que o processador *WebObjects* recupera os dados necessários ao processamento da consulta através de chamadas a métodos das classes de dados. Assim, a extensão do processador para o reconhecimento de um novo padrão, resumi-se a implementação da classe de dados e a atualização do dicionário do processador para reconhecer a interface com a nova classe.

Tanto o modelo de dados quanto à linguagem de acesso seguem fielmente o padrão ODMG, cujas características permitem ao banco de dados independência em relação ao SGBD, portabilidade, interoperabilidade e integração com outras bases de dados.

Linguagens estilo-SQL, como o é a OQL, são um avanço significativo na forma como as informações dos bancos de dados são acessadas, constituindo atualmente de um patrimônio indispensável para a comunidade de usuários de bancos de dados. Assim, a linguagem de consulta OQL da plataforma *WebObjects* possibilita a especificação de consultas com forte poder semântico, o que permite aos usuários a definição de um filtro capaz de selecionar de maneira mais consistente e precisa os dados a recuperar. Além disso, como ferramenta de busca, a *WebObjects* oferece ao usuário um diferencial importante ao garantir precisão na resposta, transferindo para o computador a tarefa de validação sobre a conformidade dos dados recuperados pela máquina de busca em relação à condição de consulta especificada.

A flexibilidade de mesclar dados oriundos de bancos de dados relacionais com dados publicados em documentos HTML presentes na Web, também é um fato que merece ser ressaltado, principalmente porque diferentemente de soluções como o WSQ/DSQ [GOL,2000], a *WebObjects* não faz nenhum tipo de exigência quanto aos recursos ou fabricante do SGBD utilizado.

A respeito do processador de consultas, a característica a ser enfatizada está na otimização da execução do plano de consulta gerado e interpretado pela máquina virtual *WebObjects*. Técnicas como validação das condições em curto circuito e antecipação da avaliação de condições fazem com que informações desnecessárias não sejam recuperadas, propiciando assim um melhor desempenho para as consultas.

No que tange ao desenvolvimento de aplicações, o processamento seqüencial do resultado de comandos OQL oferecido pela classe `Query` da plataforma *WebObjects*, propicia aos projetistas de sistemas um mecanismo trivial para desenvolvimento de novos aplicativos que explorem o conteúdo da Web. Por ser uma técnica já amplamente disseminada no ambiente de desenvolvimento, ela faz como que os programadores de aplicação se sintam familiarizados e bastante à vontade na utilização dos recursos da plataforma.

No que diz respeito aos recursos e restrições impostos pela plataforma *WebObjects* aos usuários na especificação das consultas, a grande maioria deles está associada à pobreza de recursos oferecidos pelas APIs de acesso aos catálogos das máquinas de busca. Assim, apesar da informação existir no catálogo, ela não está disponível para acesso externo, fazendo com que as consultas definidas na plataforma sejam limitadas aos recursos de exploração dos catálogos oferecidos nas APIs de acesso. O melhor exemplo deste fato ocorre com o *Google*,

que não dispõe em sua API recursos para elaboração de consultas boolenas complexas. Assim sendo, se a plataforma fosse incorporada como parte integrante de uma máquina de busca, os recursos e o potencial de consulta oferecido cresceria exponencialmente, como também seu desempenho, isto porque o custo associado ao acesso aos catálogos seria reduzido sensivelmente.

Outra vantagem da utilização da plataforma *WebObjects* está na possibilidade de utilizar qualquer máquina de busca que possua uma API de acesso ao seu catálogo. Isto é possível porque o interpretador envia solicitações genéricas de busca, sendo tarefa da SEAPI a conversão para a máquina específica. A adição ou modificação da máquina de busca utilizada pela plataforma pode ser realizada sem que os usuários sejam afetados.

Na comparação direta com as máquinas de busca, a *WebObjects* possui a desvantagem de possuir um desempenho pior. Isto ocorre porque os dados recuperados a partir dos catálogos da Web são tratados pela *WebObjects* no sentido de verificar se os mesmos estão disponíveis e validar se o conteúdo do documento ainda está de acordo com a condição de busca, ou seja, se os dados do documento estão atualizados em relação ao catálogo. Este processamento gera um custo adicional para o processamento da consulta, que é compensado pela alta precisão das respostas. Além disso, a plataforma *WebObjects* oferece uma série de outros recursos, tais como acesso a bases relacionais e manipulação dos objetos de dados, que não são oferecidos pelas máquinas de busca.

Quando comparados com soluções similares, tais como *WebOQL* e *WebSQL*, a plataforma *WebObjects* possui como grande diferencial o esquema de dados e a linguagem de consulta, que na nossa concepção, são muito mais adequados ao tratamento das irregularidades e diversidade de dados disponíveis na Web. Além disso, a plataforma *WebObjects* segue fielmente um padrão, enquanto que a maioria das outras soluções foge de maneira considerável dos padrões das linguagens a que procuram seguir. Uma comparação mais detalhada, incluindo os aspectos mencionados no capítulo 4 sobre avaliação de desempenho, precisão e facilidade de desenvolvimento, não foi realizada porque as soluções discutidas no capítulo 2 não possuíam protótipos em funcionamento e disponíveis para utilização.

6.2. Trabalhos Futuros

A avaliação experimental mostrou que a plataforma *WebObjects* é um protótipo capaz de resolver de forma básica o problema da consulta aos dados da Web. No entanto, muito trabalho deve ser realizado no sentido de agregar ao produto outras características desejáveis a um bom processador de consultas.

A primeira tarefa diz respeito à agregação dos elementos da linguagem OQL ainda não reconhecidos na versão atual. Entre eles, os que demandarão mais esforço estão associados a cláusulas como `GROUP BY` e `ORDER BY`, que necessitam a disponibilidade de todos os dados filtrados pela cláusula `WHERE` para que o resultado final seja computado.

Como ocorre em qualquer SGBD, outro aspecto que merece pesquisa constante está associado à otimização das consultas. O algoritmo atual faz uso de uma técnica extremamente simples para recuperação de dados, cuja base de processamento são os laços aninhados para varredura das várias coleções e conseqüente montagem do resultado, o que gera como principal deficiência o baixo desempenho para operações de junções. Assim, a busca por novos algoritmos de execução e análise de custo deve ser pesquisada no sentido de melhorar o desempenho obtido pela versão atual do processador de consultas.

Outro ponto a ser agregado a plataforma é a capacidade de utilizar várias máquinas de busca em uma mesma consulta, realizando no momento da execução o cruzamento das informações recuperadas de cada uma delas. Pode-se propor algoritmos inteligentes para a combinação transparente das informações recuperadas das várias máquinas de busca no sentido de obter informações mais relevantes.

Uma tarefa importante seria agregar a plataforma à capacidade de manipulação e consulta a documentos XML, que atualmente vem se tornando uma importante estrutura de dados para publicação de informações na Web. Sua característica de flexibilidade de formato e estrutura dos registros, agregado ao fato de utilizar texto simples para armazenamento das informações, faz do XML um importante padrão em aplicações de intercâmbio e compartilhamento de informações.

Bibliografia

[ARO,1997] AROCENA, Gustavo O. *WebOQL: Exploiting Document Structure in Web Queries*. Tese(Mestrado em Ciência da Computação) - Department of Computer Science in the University of Toronto, 1997.

[ARO,1998] AROCENA, Gustavo O., MENDELZON, Alberto O. *WebOQL: Reestructuring Documents, Databases e Webs*, in 14th Intl. Conf. on Data Engineering (ICDE 98), Florida, USA, 1998.

[ATV,1995] AltaVista Company. *The AltaVista Home Page*, "http://www.altavista.com", 1995.

[BRI,1998] BRIN, Sergey, PAGE, Lawrence. *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, in The 7th International World Wide Web Conference, Brisbane, Austrália, 1998.

[CAT,2000] CATTELL, R. G. G., BARRY, Douglas, BERLER, Mark, EASTMAN, Jeff, JORDAN, David, RUSSEL, Craig, SCHADOW, Olaf, STANIENDA, Torsen, VELEZ, Fernando. *The Object Data Standard: ODMG 3.0*. San Diego: Academic Press, 2000.

[DEU,1999] DEUTSCH, Alin, FERNANDEZ, Mary, FLORESCU, Daniela, LEVY, Alon, SUCIU, Dan. *XML-QL: A Query Language for XML*, in 8th Internacional Conference on the World Wide Web, Toronto, Canadá, 1999.

[GOL,2000] GOLDMAN, Roy, WIDOM, Jennifer. *WSQ/DSQ: A Pratical Approach for Combined Querying of Databases and the Web*, in SIGMOD/PODS, Dallas, USA, 2000.

[GOM,2001] GOMES, Edeyson A. *Fidus: Uma Ferramenta para Busca de Informações Personalizadas na Web*. Tese(Mestrado em Informática) - Coordenação de Curso de Pós-Graduação em Informática da Univeridade Federal da Paraíba, 2001.

[GOO,1998] Google Company. *The Google Home Page*. "http://www.google.com", 1998.

[JCC,2000] WebGain Company. *The Java Compiler Compiler (JavaCC) Home Page*. "http://www.webgain.com/products/java_cc/", 2000.

[KON,1996] KONOPNICKI, David, SHMUELI, O. *W3QS: A Query System for the World Wide Web*, in 21th Int. Conf. on Very Large Databases, Zurich, 1996.

[KON,1998] KONOPNICKI, David, SHMUELI, Oded. *Information Gathering in the WWW: The W3QL Query Language and the W3QS System*, in ACM TODS, Seatle, USA, 1998.

[LAE,2000] LAENDER, Alberto H. F., NETO, Berthier Ribeiro, SILVA, Altigran S., SILVA, Elaine S. *Representing Web Data as Complex Objects*, in Electronic Commerce and Web Technologies (EC-Web), Londres, Inglaterra, 2000.

[LAK,1996] LAKSHMANAN, Laks V. S., SADRI, Fereidoon, SUBRAMANIAN, Iyer N. *A declarative Language for Querying and Restructuring the Web*, in 6th Workshop on Research Issues in Data Engineering, New Orleans, USA, 1996.

[LAR,2000] LARMAN, Craig. *Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientado a Objetos*. Porto Alegre, Bookman, 2000.

[LOT,2001] LOTON, Tony. *Access the World's Biggest Database with Web Database Connectivity*, in JavaWorld, 2001.

[MAC,1999] MACGRATH, Sean. *XML Aplicações Práticas - Como Desenvolver Aplicações de Comércio Eletrônico.* Rio de Janeiro: Editora Campus, 1999.

[MAN,2001] MANOCHA, Nitish, COOK, Diane J., HOLDER, Lawrence B. *Structural Web Search Using a Graph-Based Discovery System*, in Proceedings of the Florida Articial Intelligence Research Symposium, Florida, USA, 2001.

[MEN,1996] MENDELZON, Alberto O., MILO, Tova. *Formal Models of Web Queries*, in PODS, Arizona, USA, 1997.

[MEN,1997] MENDELZON, Alberto O., MIHAILA, George A., MILO, Tova. *Querying the World Wide Web*, in Journal of Digital Libraries, 1997.

[MIH,1996] MIHAILA, George A. *WebSQL - An SQL-like Query Language for the World Wide Web*. Tese(Mestrado em Ciência da Computação) - Department of Computer Science in the University of Toronto, 1997.

[MIH,1997] AROCENA, Gustavo O., MENDELZON, Alberto O., MIHAILA, George A. *Applications of a Web Query Language*, in 6th Int. WWW Conference, California, USA, 1997.

[ODM,1993] Object Data Management Group. *The ODMG Home Page*, "http://www.odmg.org".

[PIN,1994] PINKERTON, B. *Finding What People Want: Experiences With WebCrawler*, in Electronic Proceedings of the 2nd World Wide Web Conference, Chicago, USA, 1994.

[POL,1997] POLLOCK, Annable, HOCKLEY, Andrew. *What's Wrong with Internet Searching*, in D-Lib Magazine, 1997.

[RAJ,2001] RAJARAMAN, Anand, ULLMAN, Jeffrey D. *Querying Websites Using Compact Skeletons*, in ACM Sigmod/Pods 2001 Conference, California, USA, 2001.

[SAV,2001] SAVNIK, Iztok, TARI, Zahir. *Query System for Web Data Source (Working Notes)*, 2001.

[SET,1995] SETHI, Ravi, AHO, Alfred V., ULLMAN, Jeffrey D. *Compiladores: Princípios, Técnicas e Ferramentas.* Rio de Janeiro: LTC S.A., 1995.

[SPE,1998] SPERTUS, Ellen. *ParaSite: Mining Structural Information on the Worls Wide Web*. Tese (PhD) - Department of EECS of MIT, 1998.

[SPE,2000] SPERTUS, Ellen, STEIN, Lynn A. *Squeal: A Structured Query Language for the Web*, in Proceedings of the 9th International World Wide Web Conference, Amsterdam, Holanda, 2000.

[STE,1998] SPERTUS, Ellen, STEIN, Lynn Andrea. *Just-In-Time Databases on the World Wide Web*, in Proceedings of the 7th International ACM Conference on Information Knowledge Management, November 1998.

[SQL,1999] ISO, International Organization for Standardization, ANSI, American National Standards Institute. *ISO International Standard: Database Language SQL - Part 2: Fundation (SQL/Foundation),* 1999.

[UHL,2001] UHL, Axel. *The Future of Internet Search*, in 3rd International Symposium on Distributed Objects & Applications, 2001.

[WQL,1997] AROCENA, Gustavo O. *The WebOQL Home Page*, "http://www.db.toronto.edu/~weboql".

[WSL,1997] MIHAILA, George A. *The WebSQL Home Page*, "http://www.cs.toronto.edu/~websql".

Apêndice A

EBNF da linguagem OQL/ODMG

```
query                  ::=  select [distinct] projectionAttributes
                                fromClause
                               [whereClause]
                               [groupClause]
                               [orderClause]
projectionAttibutes    ::=  projectionList | *
projectionList         ::=  projection{, projection}
projection             ::=  field | expr [as identifier]
fromClause             ::=  from iteratorDef {, iteratorDef}
iteratorDef            ::=  expr[[as] identifier] | identifier in expr
whereClause            ::=  where expr
groupClause            ::=  group by fieldList [havingClause]
havingClause           ::=  having expr
orderClause            ::=  order by sortCriteria
sortCriteria           ::=  sortCriterion{, sortCriterion}
sortCriterion          ::=  expr[(asc|desc)]
expr                   ::=  castExpr
castExpr               ::=  orExpr | (type) castExpr
orExpr                 ::=  orelseExpr {or orelseExpr}
orelseExpr             ::=  andExpr {orelse andExpr}
andExpr                ::=  quantifierExpr {and quantifierExpr}
quantifierExpr         ::=  andthenExpr |
                            for all inClause:andthenExpr |
                            exists inClause:andthenExpr
inClause               ::=  identifier in expr
andthenExpr            ::=  equalityExpr {andthen equalityExpr}
equalityExpr           ::=  relationalExpr
                            {(=|!=)[compositePredicate] relationalExpr} |
                            relationalExpr {like relationalExpr}
relationalExpr         ::=  additiveExpr
                             {(<|<=|>|>=)[compositePredicate] additiveExpr}
additiveExpr           ::=  multiplicativeExpr
                                {+ multiplicativeExpr} |
                            multiplicativeExpr
                                {- multiplicativeExpr} |
                            multiplicativeExpr
                                {union multiplicativeExpr} |
                            multiplicativeExpr
                                {except multiplicativeExpr} |
                            multiplicativeExpr
                                {|| multiplicativeExpr}
```

```
multiplicativeExpr      ::= inExpr {* inExpr} |
                            inExpr {/ inExpr} |
                            inExpr {mod inExpr} |
                            inExpr {intersect inExpr}
inExpr                  ::= unaryExpr [in unaryExpr]
unaryExpr               ::= + unaryExpr |
                            - unaryExpr |
                            abs unaryExpr |
                            not unaryExpr |
                            postfixExpr
postfixExpr             ::= primaryExpr{[index]) |
                            primaryExpr{(.|->)identifier[argList]}
index                   ::= expr{, expr} | expr:expr
argList                 ::= ([valueList])
primarExpr              ::= conversionExpr |
                            collectionExpr |
                            aggregateExpr |
                            undefinedExpr |
                            objectConstriction |
                            structConstruction |
                            collectionConstruction |
                            identifier[argList] |
                            queryParam |
                            literal |
                            ( query )
conversionExpr          ::= listtoset( query ) |
                            element( query ) |
                            distinct( query ) |
                            flatten( query )
collectionExpr          ::= first( query ) |
                            last( query ) |
                            unique( query ) |
                            exists( query )
aggregateExpr           ::= sum( query ) |
                            min( query ) |
                            max( query ) |
                            avg( query ) |
                            exists( ( query | * ) )
undefinedExpr           ::= is_undefined( query ) |
                            is_defined( query )
objectConstruction      ::= identifier( fieldList )
structConstruction      ::= struct( fieldList )
fieldList               ::= field{, field}
field                   ::= identifier:expr
collectionConstruction  ::= array( [valueList] ) |
                            set( [valueList] ) |
                            bag( [valueList] ) |
                            list( [valueList] ) |
```

```
                            list( listRange )
valueList                ::= expr{, expr}
listRange                ::= expr .. expr
queryParam               ::= $ longLiteral
type                     ::= [unsigned] short |
                             [unsigned] long |
                             long long |
                             float |
                             double|
                             char |
                             string |
                             boolean |
                             octet |
                             enum [identifier.]identifier |
                             date |
                             time |
                             interval |
                             timestamp |
                             set< type > |
                             bag< type > |
                             list< type > |
                             array< type > |
                             dictionary< type, type > |
                             identifier
identifier               ::= letter {letter | digit | _}
literal                  ::= booleanLiteral | longLiteral | doubleLiteral |
                             charLiteral | stringLiteral | dateLiteral |
                             timeLiteral | timestampLiteral | nil |
                             undefined
booleanLiteral           ::= true | false
longLiteral              ::= digit {digit}
doubleLiteral            ::= digit{digit}.digit{digit}
                                 [(E|e)[+|-]digit{digit}]
charLiteral              ::= 'character'
stringLiteral            ::= "{character}"
dateLiteral              ::= date 'longLiteral-LongLiteral-LongLiteral'
timeLiteral              ::= time 'longLiteral:longLiteral:longLiteral'
timestampLiteral         ::= timestamp 'lonkLiteral-longLiteral-longLiteral
                                        longLiteral:longLiteral:longLiteral'
character                ::= letter | digit | special-character
letter                   ::= A | B | ... | Z | a | b | ... | z
digit                    ::= 0 | 1 | ... | 9
special-character        ::= ? | _ | * | % | \
```

Apêndice B

Definição JavaCC da linguagem OQL para a *WebObjects*

Este apêndice traz a BNF JavaCC dos construtores da linguagem OQL reconhecidos pela plataforma *WebObjects*, bem como os apontamentos sintáticos feitos para manutenção da tabela de símbolos utilizada no processo de geração do código que é interpretado pela máquina virtual.

```
/*
 * This is the grammar for the OQL user for WebObjects
 * Grammar written by Fábio Soares Silva
 */

options {
  LOOKAHEAD = 1;
  CHOICE_AMBIGUITY_CHECK = 2;
  OTHER_AMBIGUITY_CHECK = 1;
  STATIC = false;
  DEBUG_PARSER = false;
  DEBUG_LOOKAHEAD = false;
  DEBUG_TOKEN_MANAGER = false;
  ERROR_REPORTING = true;
  JAVA_UNICODE_ESCAPE = false;
  UNICODE_INPUT = false;
  IGNORE_CASE = false;
  USER_TOKEN_MANAGER = false;
  USER_CHAR_STREAM = false;
  BUILD_PARSER = true;
  BUILD_TOKEN_MANAGER = true;
  SANITY_CHECK = true;
  FORCE_LA_CHECK = false;
}

PARSER_BEGIN( ParserWebObjects )

package webobjects.parser;

/**
```

```
 * Title:       WebObjects - Uma Plataforma de Acesso aos Dados da Web
 * Description: Api Java que oferece recursos de pesquisa aos dados da Web através de uma linguagem OQL
 * Copyright:   Copyright (c) 2002
 * Company:     UFPB
 * @author:     Fábio Soares Silva
 * @version:    1.0
 */

import webobjects.interpreter.Instructions;
import java.util.Vector;
import java.util.Date;

/**
 * Classe que representa efetua o parse de um comando OQL, apontando os dadpos na tabela de símbolos
 * e gerando o plano de execução da consulta
 */
public class ParserWebObjects {

   /**
    * Tabela de símbolos que armazena os dados extraídos do comando OQL
    */
   private TabSymbols tabSymbols = null;

   /**
    * Armazena o atributo atual reconhecido no comando OQL
    */
   private AttrElement attr;

   /**
    * Vector que contém todos os elementos projetados no comando OQL
    */
   private Vector projClause;

   /**
     * Concatena as informações do token de origem no token de destino
     * @param tSrc Token a ter seus dados concatenados no destino
     * @param tDes Token para onde os dados da origem serão concatenados
     */
   private void concatTokens( Token tSrc, Token tDes ) {
     tDes.endColumn = tSrc.endColumn;
     tDes.endLine = tSrc.endLine;
     tDes.image = tDes.image.concat( tSrc.image );
     tDes.next = tSrc.next;
   }
```

```
/**
 * Verifica se existem espaços em branco entre dois tokens
 * @param line Linha para qual o erro deve ser informado
 * @param column Coluna para qual o erro deve ser reportado
 * @param type1 Primeiro tipo
 * @param type1 Segundo tipo
 */
private void throwException( int line, int column, String msg ) throws ParseException {
  throw new ParseException( "Error in line " + line + " column " + column + ": " + msg );
}

/**
 * Verifica se existem espaços em branco entre dois tokens
 * @param line Linha para qual o erro deve ser informado
 * @param column Coluna para qual o erro deve ser reportado
 * @param type1 Primeiro tipo
 * @param type1 Segundo tipo
 */
private void validateTypes( int line, int column, int type1, int type2 ) throws ParseException {
  if ( !WebObjectsTypes.isCompatible( type1, type2 ) )
    throw new ParseException( "Error in line " + line + " column " + column +
                              ": Incompatible types in value list" );
}

/**
 * Verifica se o parâmetro para o operador like é valido
 */
private void validateLikeArq( int beginLine, int beginCol, String objDesc, String likeValue ) throws ParseException {
  if ( objDesc.equals( "url" ) && ( likeValue.startsWith( "%" ) ) )
    throwException( beginLine, beginCol, "Field URL can't have \"%\" caracter on start on like argument" );
  else if ( ( objDesc.equals( "getDomain()" ) || objDesc.equals( "content" ) || objDesc.equals( "title" ) ) &&
            ( likeValue.length() < 4 || likeValue.startsWith( "%" ) ) )
    throwException( beginLine, beginCol, "Invalid like argument for this field" );
  else if ( !objDesc.equals( "getDomain()" ) && !objDesc.equals( "content" ) &&
            !objDesc.equals( "title" ) && !likeValue.startsWith( "%" ) && !likeValue.endsWith( "%" ) )
    throwException( beginLine, beginCol, "Like argument for this field must have \"%\" caracter" );
}

/**
 * Verifica se existem espaços em branco entre dois tokens
 * @param tAnt Primeiro token
 * @param tIdt Segundo token
 */
```

```
private void checkSpaceBetweenTokens( Token tAnt, Token tIdt ) throws ParseException {
  // Verifica se os dois tokens são consecutivos
  if ( ( tAnt.beginLine != tIdt.beginLine ) || ( tAnt.endLine != tIdt.endLine ) ||
       ( tAnt.endColumn+1 != tIdt.beginColumn && tIdt.endColumn+1 != tAnt.beginColumn ) ) {

    if ( tAnt.image.equals( "." ) )
      if ( ( tAnt.beginLine < tIdt.beginLine ) ||
           ( tAnt.beginLine == tIdt.beginLine && tAnt.beginColumn < tIdt.beginColumn ) )
        throw new ParseException( "Expecting an identifier after token \".\" in line " +
                                  tAnt.beginLine + " column " + tAnt.beginColumn );
      else
        throw new ParseException( "Expecting an identifier before token \".\" in line " +
                                  tAnt.beginLine + " column " + tAnt.beginColumn );
    else
      throw new ParseException( "Expecting a token \"(\", without white space, after that identifier " +
                                "in line " + tAnt.beginLine + " column " + tAnt.beginColumn );
  }
}

/**
 * Adiciona a conversão de tipos para uma referência direta a coluna de uma relação
 * @param type Tipo utilizado para comparação
 * @param attElem Tokens que definem o acesso direto à relação de uma tabela
 * @return Elemento de projeção resultante da adição da referência direta
 */
private ProjElement addTypeConversion( int type, AttrElement attElem ) throws ParseException {
  Token t = new Token();
  switch ( type ) {
    case WebObjectsTypes.TSTRING : t.image = "toString()";
                                   break;
    case WebObjectsTypes.TINTEGER: t.image = "toInteger()";
                                   break;
    case WebObjectsTypes.TFLOAT  : t.image = "toFloat()";
                                   break;
    case WebObjectsTypes.TDATE   : t.image = "toDate()";
                                   break;
    case WebObjectsTypes.TBOOLEAN: t.image = "toBoolean()";
                                   break;
    default                      : t.image = "toBinary()";
  }
  attElem.addToken( t );
  return tabSymbols.getProjElement( attElem );
}
```

```
   /**
    * Recupera os apelidos das colunas para o comando de seleção
    * @return Um vetor com strings que possuem os elementos das colunas
    */
   public Vector getColumnNames() {
     if ( tabSymbols == null )
       return null;
     else
       return tabSymbols.getAliases();
   }

   /**
    * Limpa os dados utilizados pelo parser
    */
   public void clear() {
     tabSymbols = null;
     projClause = null;
   }

}

PARSER_END( ParserWebObjects )

SKIP : { " " | "\t" | "\n" | "\r" }

/* Palavras reservadas da linguagem */
TOKEN [IGNORE_CASE] : {
  < ENDQUERY: ";" > |
  < SELECT: "SELECT" > |
  < FROM: "FROM" > |
  < IN: "IN" > |
  < WHERE: "WHERE" > |
  < OR: "OR" > |
  < AND: "AND" > |
  < LIKE: "LIKE" > |
  < NOT: "NOT" > |
  < FLOAT: "FLOAT" > |
  < TRUE: "TRUE" > |
  < FALSE: "FALSE" > |
  < NIL: "NIL" > |
  < AS: "AS" >
}

/* Símbolos especiais */
```

```
TOKEN : {
  < COMMA: "," > |
  < COLON: ":" > |
  < LPAREN: "(" > |
  < RPAREN: ")" > |
  < DOT: "." > |
  < DOTDOT: ".." > |
  < APOST: "'" > |
  < ASPAS: "\"" > |
  < SLASH: "/" > |
  < COMPOP: <EQUAL> | <NOTEQUAL> | <LTHAN> | <LETHAN> | <GTHAN> | <GETHAN> > |
  < #EQUAL: "=" > |
  < #GTHAN: ">" > |
  < #LTHAN: "<" > |
  < #GETHAN: ">=" > |
  < #LETHAN: "<=" > |
  < #NOTEQUAL: "!=" >
}

/* Identificador de tipos */
TOKEN : {
  < INTEGER: <DIGIT> ( <DIGIT> )* > |
  < TKDOUBLE: <INTEGER> <DOT> <INTEGER> > |
  < TKTIMEST: <APOST> <TKDATELIT> " " <TKTIMELIT> <APOST> > |
  < TKDATE: <APOST> <TKDATELIT> <APOST> > |
  < IDENTIFIER: <LETTER> ( <DIGIT> | <LETTER> | "_" )* > |
  < CHARLIT: <APOST> ( ( ~["'","\\","\n","\r"] ) | ( "\\" ( ["n","t","b","r","f","\\","'","\""] |
                                                           ["0"-"7"] ( ["0"-"7"] )? |
                                                           ["0"-"3"] ["0"-"7"] ["0"-"7"] ) ) )
             <APOST> > |
  < STLITERAL: <ASPAS> ( ( ~["\"","\\","\n","\r"] ) | ( "\\" ( ["n","t","b","r","f","\\","'","\""]  |
                                                             ["0"-"7"] ( ["0"-"7"] )? |
                                                             ["0"-"3"] ["0"-"7"] ["0"-"7"] ) ) )*
               <ASPAS> > |
  < #TKDATELIT: <DIGIT> <DIGIT> <SLASH> <DIGIT> <DIGIT> <SLASH> <DIGIT> <DIGIT> <DIGIT> <DIGIT> > |
  < #TKTIMELIT: <DIGIT> <DIGIT> <COLON> <DIGIT> <DIGIT> <COLON> <DIGIT> <DIGIT> > |
  < #DIGIT: ["0"-"9"] > |
  < #LETTER: ["a"-"z", "A"-"Z"] >
}

/**
 * Efetua a análise da consulta passada como parâmetro
 * @return Retorna um vetor com o plano de execução da consulta
 */
```

```
Vector analyseQuery() :
{}
{
  {
    tabSymbols = new TabSymbols();
    projClause = new Vector();
  }
  query() <ENDQUERY> <EOF>
  {
    return tabSymbols.generateQueryPlan();
  }
}

/**
 * Reconhece uma consulta OQL, segundo o conjunto de comandos disponíveis na plataforma WebObjects
 */
void query() :
{}
{
  <SELECT> projectionList()
       <FROM> fromClause()
       whereClause()
}

/**
 * Reconhece os elementos projetados como argumento da cláusula SELECT
 */
void projectionList() :
{}
{
  { projClause.clear(); }
  projectionItem() ( <COMMA> projectionItem() )*
}

/**
 * Reconhece uma sequência de tokens que define um elemento projetado
 */
void projectionItem() :
{}
{
  {
    attr = new AttrElement();
    Token t = null;
  }
```

```
  fieldDescriptor() [ <AS> t = <IDENTIFIER> ]
  {
    projClause.add( attr );
    if ( t == null )
      tabSymbols.addAlias( attr.toString() );
    else
      tabSymbols.addAlias( t );
  }
}

/**
 * Reconhece a descrição completa de um objeto ou atributo das classes que compõem a plataforma WebObjects
 */
void fieldDescriptor() :
{
  Token tIdt, tDot;
}
{
  { tDot = this.getToken(0); }
  tIdt = fieldIdentifier() { if ( tDot.image.equals( "." ) )
                               checkSpaceBetweenTokens( tDot, tIdt );
                               attr.addToken( tIdt );
                           }
              ( tDot = <DOT> {
                               checkSpaceBetweenTokens( tDot, tIdt );
                             }
                tIdt = fieldIdentifier() {
                                           checkSpaceBetweenTokens( tDot, tIdt );
                                           attr.addToken( tIdt );
                                         }
              )*

}

/**
 * Reconhece um identificador de atributo ou método das classes que compõem a plataforma WebObjects
 */
Token fieldIdentifier() :
{
  Token tRes;
  Token tAct;
}
{
  tRes = <IDENTIFIER> [ tAct = <LPAREN> {
```

```
                                        checkSpaceBetweenTokens( tAct, tRes );
                                        concatTokens( tAct, tRes );
                                      }
                        [ tAct = <STLITERAL> { concatTokens( tAct, tRes ); } ]
                        tAct = <RPAREN> {
                                        concatTokens( tAct, tRes );
                                      }
                      ]
  { return tRes; }
}

/**
 * Reconhece os vários elementos da cláusula FROM e faz o apontamento sintático
 */
void fromClause() :
{}
{
  { attr = new AttrElement(); }
  fromDescriptor() ( <COMMA> fromDescriptor() )*
  {
    tabSymbols.addProjection( projClause );
    projClause.clear();
  }
}

/**
 * Reconhece um elemento recuperado na cláusula FROM
 */
void fromDescriptor() :
{
  Token t;
}
{
  { attr.reInitTokens(); }
  fieldDescriptor() t = <IDENTIFIER>
  { tabSymbols.addFromClause( attr, t ); }
}

/**
 * Reconhece uma expressão que restringe os dados recuperados por uma consulta OQL
 */
void whereClause() :
{}
{
```

```
  <WHERE> conditionExpr()
}

/**
 * Reconhece a expressão condicional com símbolo OR
 */
void conditionExpr() :
{}
{
  logicalTerm() ( <OR> logicalTerm() { tabSymbols.addPosCond( new Integer( Instructions.OR ) ); } )*
}

void logicalTerm() :
{}
{
  logicalFactor() ( <AND> logicalFactor() { tabSymbols.addPosCond( new Integer( Instructions.AND ) ); } )*
}

void logicalFactor() :
{
  boolean notValue = false;
}
{
  [ <NOT> { notValue = true; } ]
       ( LOOKAHEAD( selectionExpr() ) selectionExpr() | <LPAREN> conditionExpr() <RPAREN> )
       {
         if ( notValue )
           tabSymbols.addPosCond( new Integer( Instructions.NOT ) );
       }
}

void selectionExpr() :
{
  ProjElement pjAnt = null;
  ProjElement pjPos = null;
  Token t;
  boolean hasTerm = false;
  AttrElement oldAttr;
}
{
  pjAnt = fieldSelDesc() {
                           t = this.getToken(0);
                           oldAttr = attr;
                         }
```

```
      [ ( t = <COMPOP> (
                          pjPos = literal() |
                               { attr = new AttrElement(); }
                          pjPos = fieldSelDesc()
                               {
                                 if ( pjAnt.getBaseAlias().equals( pjPos.getBaseAlias() ) )
                                   throwException(  t.beginLine,  t.beginColumn,  "The  arguments  of  boolean  expression  must  have
differents object sources" );
                               }
                         ) |
          t = <LIKE> pjPos = stringLiteral() |
          t = <IN> <LPAREN> valueList( pjAnt ) <RPAREN> ) { hasTerm = true; } ]
  {
    String oper = t.image.toLowerCase();
    if ( !oper.equals( "in" ) ) {
      // Para o caso de Referência direta a campos de uma relação, faz a conversão de tipos
      if ( pjAnt.getElementType() == WebObjectsTypes.TREFFIELD && pjPos.getElementType() == WebObjectsTypes.TREFFIELD ) {
        pjAnt = addTypeConversion( WebObjectsTypes.TSTRING, oldAttr );
        pjPos = addTypeConversion( WebObjectsTypes.TSTRING, attr );
      } else if ( pjAnt.getElementType() == WebObjectsTypes.TREFFIELD ) &&
                ( pjPos == null || pjPos.getElementType() == WebObjectsTypes.TBOOLEAN ) ) )
        throwException( t.beginLine, t.beginColumn, "Boolean values aren't accepted for Oracle Databases" );
      else if ( pjAnt.getElementType() == WebObjectsTypes.TREFFIELD &&
                pjPos.getElementType() != WebObjectsTypes.TNULL &&
                pjPos.getElementType() != WebObjectsTypes.TREFFIELD )
        pjAnt = addTypeConversion( pjPos.getElementType(), oldAttr );
      else if ( pjPos.getElementType() == WebObjectsTypes.TREFFIELD &&
                pjAnt.getElementType() != WebObjectsTypes.TNULL &&
                pjAnt.getElementType() != WebObjectsTypes.TREFFIELD )
        pjPos = addTypeConversion( pjAnt.getElementType(), attr );

      // Verifica se os tipos de dados são válidos
      if ( !hasTerm && pjAnt.getElementType() != WebObjectsTypes.TBOOLEAN )
        throwException( t.beginLine, t.beginColumn, "Boolean expression is required" );
      else if ( !hasTerm ) {
        tabSymbols.addPosCond( pjAnt );
        tabSymbols.addPosCond( new ProjElement( null, new Boolean( true ), WebObjectsTypes.TBOOLEAN ) );
        tabSymbols.addPosCond( new Integer( Instructions.EQUAL ) );
      } else {
        validateTypes( t.beginLine, t.beginColumn, pjAnt.getElementType(), pjPos.getElementType() );
        if ( oper.equals( "like" ) )
          validateLikeArq( t.beginLine, t.beginColumn, String.valueOf( pjAnt.getProjElement() ),
                           String.valueOf( pjPos.getProjElement() ) );
        tabSymbols.addPosCond( pjAnt );
```

```
        tabSymbols.addPosCond( pjPos );
        if ( !oper.equals( "in" ) )
          tabSymbols.addPosCond( new Integer( Instructions.getOperator( t.image ) ) );
      }
    }
  }
}

ProjElement fieldSelDesc() :
{}
{
  { attr.reInitTokens(); }
  fieldDescriptor()
  { return tabSymbols.getProjElement( attr ); }
}

void valueList( ProjElement term ) :
{
  ProjElement pj;
  Token t;
}
{
  pj = literal() {
                  t = this.getToken(0);
                  // Para o caso de Referência direta a campos de uma relação, faz a conversão de tipos
                  if ( term.getElementType() == WebObjectsTypes.TREFFIELD && pj.getElementType() == WebObjectsTypes.TBOOLEAN )
                    throwException( t.beginLine, t.beginColumn, "Boolean values aren't accepted for Oracle Databases" );
                  else if ( pj.getElementType() == WebObjectsTypes.TNULL )
                    throwException( t.beginLine, t.beginColumn, "Null values aren't accepted for in operator" );
                  else if ( term.getElementType() == WebObjectsTypes.TREFFIELD &&
                            pj.getElementType() != WebObjectsTypes.TREFFIELD )
                    term = addTypeConversion( pj.getElementType(), attr );
                  // Efetua a validação dos tipos e adição da condição
                  validateTypes( t.beginLine, t.beginColumn, term.getElementType(), pj.getElementType() );
                  tabSymbols.addPosCond( term );
                  tabSymbols.addPosCond( pj );
                  tabSymbols.addPosCond( new Integer( Instructions.EQUAL ) );
                }
          ( <COMMA> pj = literal()
                {
                  t = this.getToken(0);
                  validateTypes( t.beginLine, t.beginColumn, term.getElementType(), pj.getElementType() );
                  tabSymbols.addPosCond( term );
                  tabSymbols.addPosCond( pj );
```

```
                  tabSymbols.addPosCond( new Integer( Instructions.EQUAL ) );
                  tabSymbols.addPosCond( new Integer( Instructions.OR ) );
                }
          )*
}

ProjElement literal() :
{
  ProjElement pj;
}
{
  (  pj=booleanLiteral() | pj=longLiteral() | pj=floatLiteral() | pj=stringLiteral() | pj = dateLiteral() | pj=nullLiteral() )
  { return pj; }
}

ProjElement booleanLiteral() :
{}
{
  <TRUE> { return new ProjElement( null, new Boolean( true ), WebObjectsTypes.TBOOLEAN ); } |
  <FALSE> { return new ProjElement( null, new Boolean( false ), WebObjectsTypes.TBOOLEAN ); }
}

ProjElement longLiteral() :
{
  Token t;
}
{
  t = <INTEGER> { return new ProjElement( null, new Long( t.image ), WebObjectsTypes.TINTEGER ); }
}

ProjElement floatLiteral() :
{
  Token t;
}
{
  t = <TKDOUBLE> { return new ProjElement( null, new Float( t.image ), WebObjectsTypes.TFLOAT ); }
}

ProjElement stringLiteral() :
{
  Token t;
}
{
  t = <STLITERAL> { return new ProjElement( null, t.image.substring( 1, t.image.length()-1 ), WebObjectsTypes.TSTRING ); }
```

```
}

ProjElement dateLiteral() :
{
  Token t;
  Date d;
}
{
  ( t = <TKTIMEST> { d = WebObjectsTypes.convertDate( t.image, false ); } |
    t = <TKDATE> { d = WebObjectsTypes.convertDate( t.image, true ); } )
     {
       if ( d == null )
         throwException( t.beginLine, t.beginColumn, "Incorrect date value" );
       else
         return new ProjElement( null, d, WebObjectsTypes.TDATE );
     }
}

ProjElement nullLiteral() :
{}
{
  <NIL> { return new ProjElement( null, null, WebObjectsTypes.TNULL ); }
}
```

Apêndice C

Instruções da Máquina Virtual *WebObjects*

Instrução	INITCOLL
Descrição	Incializa um objeto coleção
Ação	O objeto inicializado é armazenado na cache de objetos da consulta
Parâmetros	Memória - Posição de memória na cache de objetos onde a coleção deve ser armazenada. Objeto - String que descreve a coleção a ser inicializada. Ex: "Reapi".

Instrução	ADDURL
Descrição	Adiciona uma URL ao elenco gerenciado pela API
Ação	A API é obtida da cache, e o valor especificado para a url é adicionado ao elenco da API.
Parâmetros	Memória - Posição de memória onde a API está armazenada. URL - Descrição da URL a ser adicionada diretamente a API.

Instrução	ADDPARURL
Descrição	Adiciona uma URL obtida de um elemento de outro objeto ao elenco recuperado pela API.
Ação	A API é obtida da cache, o valor é obtido do objeto de referência e o valor para a url é adicionado ao elenco da API.
Parâmetros	Memória (API) - Posição de memória onde a API está armazenada. Memória (Objeto) - Posição de memória onde o objeto a ter o valor obtido está armazenado. Valor - Atributo do objeto a ter o valor recuperado para ser atribuído na API.

Instrução	INITFLT
Descrição	Inicializa o filtro a ser aplicado a uma API.
Ação	Inicializa o filtro da API. Esta função deve ser utilizada em laços aninhados onde a cada interação do laço mais externo, o filtro da API mais interna deve ser inicializado e parametrizado.
Parâmetros	Memória - Posição de memória onde o filtro deve ser armazenado. Filtro - Valor inicial para o filtro.

Instrução	SETFLTPAR
Descrição	Substitui um parâmetro em um filtro.
Ação	Recupera o filtro da cache, substitui o parâmetro com um valor recuperado da pilha de execução, devolvendo o filtro para a cache com o valor já substituído.
Parâmetros	Memória - Posição de memória onde o filtro deve ser armazenado. Parâmetro - String com o parâmetro a ser substituído.

Instrução	SETFILTER
Descrição	Atribui um filtro a uma API.
Ação	Recupera o filtro de uma posição de memória, atribuindo-o a uma API.
Parâmetros	Memória (API) - Posição de memória da API a ter o filtro aplicado. Memória (Filtro) - Posição de memória onde o filtro a ser aplicado está localizado.

Instrução	SETPRJREL
Descrição	Adiciona uma coluna de projeção a um objeto Relation.
Ação	Recupera o objeto Relation armazenado na cache e adiciona o elemento de projeção especificado.
Parâmetros	Memória - Posição de memória onde o objeto Relation está localizado. Projeção - String que define a coluna a ser projetada.

Instrução	SETLKFLT
Descrição	Define o limite para a profundidade máxima de links explorados em um objeto Text.
Ação	Recupera o objeto Text armazenado na cache e atribui o limite de profundidade especificado.
Parâmetros	Memória - Posição de memória onde o objeto Text está armazenado. Memória (Filtro) - Posição de memória onde o filtro a ser aplicado está localizado.

Instrução	JUMP
Descrição	Salta incondicionalmente o fluxo de execução do programa.
Ação	Recupera a instrução para onde o fluxo deve ser desviado, e efetua a mudança no fluxo.
Parâmetros	Instrução - Índice da instrução para onde o fluxo deve ser desviado.

Instrução	CONDJUMP
Descrição	Salto condicional do fluxo de execução do programa.
Ação	Recupera um valor da pilha de execução, e caso este valor seja false, existe um salto na execução da consulta.
Parâmetros	Instrução - Índice da instrução para onde o fluxo deve ser desviado.

Instrução	INITRES
Descrição	Inicializa o vetor que armazena os campos de uma resposta para a consulta.
Ação	Recupera a instrução para onde o fluxo deve ser desviado, e efetua a mudança no fluxo.
Parâmetros	---

Instrução	GETCOLRES
Descrição	Recupera o próximo resultado de uma coleção a ser utilizado na composição do resultado.
Ação	Obtém da cache o objeto que contém a coleção, recupera sua próxima resposta e armazena em uma posição de memória. Caso a operação seja realizada com sucesso, um booleano TRUE é colocado na pilha. Caso contrário, um valor FALSE é empilhado.
Parâmetros	Memória (Origem) - Posição de memória do objeto onde o próximo elemento deve ser recuperado. Memória (Destino) - Posição de memória onde o resultado deve ser armazenado. Resultado - String que descreve o objeto a ser recuperado da origem.

Instrução	GETREFOBJ
Descrição	Obtém de um objeto um valor de atributo.
Ação	Obtém da cache o objeto que contém o dado a ser obtido, recupera a informação e armazena-a na pilha.
Parâmetros	Memória (Origem) - Posição de memória do objeto onde o dado a ser recuperado está. Resultado - String que descreve o dado a ser recuperado.

Instrução	STORE
Descrição	Armazena um objeto na memória cache.
Ação	Recupera o objeto do topo da pilha e armazena-o em uma posição de memória.
Parâmetros	Memória - Posição de memória onde o objeto deve ser armazenado.

Instrução	ADDRESULT
Descrição	Adiciona um valor ao vetor de resultados.
Ação	Recupera o objeto do topo da pilha e adiciona-o ao vetor de resultados de uma consulta.
Parâmetros	---

Instrução	NOT
Descrição	Inverte o valor de um resultado booleano.
Ação	Recupera o valor do topo da pilha e efetua a inversão do valor, armazenando o resultado na pilha.
Parâmetros	---

Instrução	AND
Descrição	Efetua o E lógico de dois valores booleanos.
Ação	Recupera os dois elementos do topo da pilha e efetua o E lógico armazenando o resultado na pilha.
Parâmetros	---

Instrução	OR
Descrição	Efetua o OU lógico de dois valores booleanos.
Ação	Recupera os dois elementos do topo da pilha e efetua o OU lógico armazenando o resultado na pilha.
Parâmetros	---

Instrução	EQUAL
Descrição	Verifica se dois objetos são iguais.
Ação	Recupera os dois objetos do topo da pilha e verifica se os dois são iguais. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	NOTEQUAL
Descrição	Verifica se dois objetos são diferentes.
Ação	Recupera os dois objetos do topo da pilha e verifica se os dois são diferentes. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	GTHEN
Descrição	Verifica se um objeto é maior que outro.
Ação	Recupera os dois objetos do topo da pilha e verifica se o primeiro é maior que o segundo. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	GEQTHEN
Descrição	Verifica se um objeto é maior ou igual a outro.
Ação	Recupera os dois objetos do topo da pilha e verifica se o primeiro é maior ou igual ao segundo. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	LTHEN
Descrição	Verifica se um objeto é menor que outro.
Ação	Recupera os dois objetos do topo da pilha e verifica se o primeiro é menor que o segundo. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	LEQTHEN
Descrição	Verifica se um objeto é menor ou igual a outro.
Ação	Recupera os dois objetos do topo da pilha e verifica se o primeiro é menor ou igual ao segundo. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	LIKE
Descrição	Verifica se uma String segue o padrão definido por outra.
Ação	Recupera as duas Strings do topo da pilha e verifica se a primeiro segue o padrão da segunda. O resultado booleano é armazenado na pilha de execução.
Parâmetros	---

Instrução	PUSH
Descrição	Empilha um valor.
Ação	Empilha um valor capturado diretamente do vetor de instruções.
Parâmetros	Valor - Valor a ser empilhado.

Instrução	ORCCJUMP
Descrição	Efetua o salto na avaliação de uma instrução OR quando a segunda expressão não precisa ser avaliada para a computação do resultado final, ou seja, realiza o curto circuito de uma expressão OR.
Ação	Obtém da pilha o resultado e se for verdadeiro empilha o valor TRUE e salta para o final da expressão.
Parâmetros	Instrução - Instrução para onde o fluxo de controle da consulta deve ser desviado.

Instrução	ANDCCJUMP
Descrição	Efetua o salto na avaliação de uma instrução AND quando a segunda expressão não precisa ser avaliada para a computação do resultado final, ou seja, realiza o curto circuito de uma expressão AND.
Ação	Obtém da pilha o resultado e se for falso empilha o valor FALSE e salta para o final da expressão.
Parâmetros	Instrução - Instrução para onde o fluxo de controle da consulta deve ser desviado.

Instrução	ENDLINE
Descrição	Finaliza a composição de uma linha de resultado.
Ação	Causa a interrupção da montagem do resultado e um salto na execução da consulta.
Parâmetros	Instrução - Índice da instrução para onde o fluxo deve ser desviado.

Instrução	HALT
Descrição	Finaliza a execução da consulta.
Ação	Causa o final da execução da consulta.
Parâmetros	---