Universidade Federal de Santa Catarina - UFSC

Programa de Pós-Graduação em Ciência da Computação

Luciano Nakano

UMA PROPOSTA DE EXTENSÃO À API XQUERY DO FRAMEWORK .NET

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de mestre em Ciência da Computação

Prof. Murilo Silva de Camargo, Dr.

Orientador

Florianópolis, Fevereiro de 2003

UMA PROPOSTA DE EXTENSÃO À API XQUERY DO FRAMEWORK .NET

LUCIANO NAKANO

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO - SISTEMAS DE COMPUTAÇÃO E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO.

_____

Prof. , Dr. Fernando A. O. Gauthier
Coordenador do Curso

BANCA EXAMINADORA:

_____

Prof. Murilo Silva de Camargo, Dr.
Orientador

_____

Prof. Rosvelter João Coelho da Costa, Dr.

_____

Prof. Vitório Bruno Mazzola, Dr.

"Dê um peixe a um homem faminto e você o alimentará por um dia.
Ensine-o a pescar, e você o alimentará pelo resto da vida."
Provérbio Chinês

A todos os professores, mestres e doutores, que contribuíram para minha formação pós-acadêmica.

AGRADECIMENTOS

Agradeço aos amigos, familiares e professores pela compreensão e apoio nesta difícil jornada.

Aos companheiros de trabalho, Aparecido Ferreira da Silva, Ivonei Freitas da Silva e Valmir Gonçalves que colaboraram nesta caminhada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A tecnologia XML vem ganhando nos últimos anos um espaço considerável nas comunidades de pesquisa de sistemas de banco de dados e aplicações voltadas para a World Wide WEB.

O sucesso de tal tecnologia pode ser verificado na adoção desta como um recurso adicional, de extrema necessidade, para a manipulação e o intercâmbio de informações em sistemas fornecidos pelas grandes corporações tanto na área de banco de dados como na área de aplicações WEB. Na Conferência Internacional de Engenharia de Software (ICSE 2002) realizada em Orlando/Flórida/USA, no período de 19 a 26 de maio de 2002, Jim Cassell apresentou relatório do Gartner Group que analisa tendências em informática nesse início de século, e uma das tendências apresentadas é a consolidação de XML como linguagem universal de integração entre aplicações.

O presente trabalho apresenta uma visão de convergência que a XML propicia entre as áreas de banco de dados e a WEB, permitindo que funções altamente estruturadas na área de banco de dados tais como localizar, filtrar, e analisar informações sejam possíveis no ambiente WEB. Algumas linguagens de consulta XML, com ênfase à linguagem XQuery, são estudadas com o propósito de apresentar tal convergência. A API XQuery do Framework .NET é apresentada e estendida a fim de demonstrar o uso da linguagem de consulta XQuery.

ABSTRACT

The XML technology comes winning in the last years a large space in the communities of research of database systems and applications to the World Wide WEB.

The success of such technology can be verified in the adoption of this as an additional resource, of extreme need, for the manipulation and the exchange of information in systems supplied so much by the great corporations in the database area as in the area of applications WEB. In the International Conference of Engineering of Software (ICSE 2002) accomplished in Orlando/Flórida/USA, in the period of 19 to May 26, 2002, Jim Cassell presented report of Gartner Group that analyzes tendencies in computer science in that century beginning, and one of the presented tendencies is the consolidation of XML as universal language of integration among applications.

The present work presents a convergence vision that XML propitiates between the database areas and the WEB, allowing that functions highly structured in the database area such as locating, filtering, and analyzing information is possible in the WEB. Some XML query languages, with emphasis to the XQuery, is studied with the purpose of presenting such convergence. Framework .NET'S API XQuery is presented and extended in order to demonstrate the use of the XQuery query language.

CAPÍTULO I

INTRODUÇÃO

1.1 Introdução

Grande número dos atuais sistemas gerenciadores de banco de dados tem sua fundamentação teórica baseada nos estudos matemáticos realizados através de modelos relacionais.

Os conceitos oriundos da matemática e definidos há anos por grandes estudiosos foram potencialmente aproveitados para construção de poderosos métodos e formalismos na estruturação de modelos de dados.

À medida que estes conceitos evoluíram na área de sistemas gerenciadores de banco de dados, evoluiu-se também a área de abrangência na perspectiva de utilização dos mesmos dentro de dois principais enfoques atuais: as *Intranets* e a *World Wide WEB*. Desta evolução surgiu-se a necessidade de integração de sistemas preexistentes, o que caracterizou o despontamento do termo “Banco de Dados Distribuído”.

Aliados às tecnologias mais atuais, os principais fundamentos destes sistemas podem fornecer diversas soluções de integração de ambientes e sistemas de banco de dados distribuídos com ênfase nessa linha de pesquisa. Uma destas tecnologias atuais que vêm ganhando espaço considerável neste ramo e que possui diversos trabalhos interessantes em desenvolvimento é a XML (http://www.w3.org/XML/).

Um problema que surge com os diversos trabalhos e protótipos propostos para os sistemas de banco de dados com suporte específico a XML e utilitários de banco de dados relacionados a XML é a falta de um padrão para a linguagem de consulta em dados semi-estruturados e, em particular, à tecnologia XML.

1.2 Objetivos

Este trabalho tem como propósito geral apresentar os principais conceitos que envolvem a tecnologia XML na área de trabalho de sistemas de banco de dados e na WEB.

Têm-se como objetivos específicos:

- Apresentar um estudo crítico da principal linguagem de consulta existente atualmente em XML, a qual é proposta pelo W3C para adoção como a linguagem padrão (*XQuery*).

- Analisar os conceitos existentes em relação às técnicas de consulta a dados semi-estruturados (XML).

- Avaliar resultados práticos obtidos pela utilização da linguagem em estudo.

- Propor uma extensão à API XQuery do Framework .NET visando suprir determinadas especificações descritas pelo Working Draft do W3C não

contempladas pela API, realizando assim de maneira prática o estudo da linguagem XQuery.

1.3 Importância do Trabalho

A importância deste trabalho concentra-se no fato de fornecer uma avaliação de resultados das técnicas utilizadas pelas linguagens de consulta a dados semi-estruturados, em especial àquelas utilizadas em consultas a dados XML.

A avaliação de resultados realizada sobre as técnicas de linguagens de consulta a dados semi-estruturados e, em particular a XML se faz necessária devido aos seguintes pontos: (i) modelos de dados semi-estruturados estão surgindo com grande impacto causado pela *WEB*, da perspectiva de bancos de dados, (ii) XML tem sido reconhecida amplamente como um dos mais importantes formatos na *WEB* e uma linguagem de consulta é essencial para dados na *WEB*, (iii) é possível representar bancos de dados como documentos XML, mas o tamanho do documento resultante pode proibir sua transmissão como uma única página na *WEB*, além disso, um grande documento XML provavelmente seria uma visão de um grande banco de dados, e os usuários normalmente desejam apenas um fragmento dos dados.

1.4 Estrutura do Trabalho

Este trabalho está estruturado em seis capítulos. O primeiro apresenta uma introdução, os objetivos, a importância e a estrutura do trabalho.

No segundo capítulo é feito um estudo da teoria de sistemas de bancos de dados distribuídos ligada à *WEB*, caracterizando o problema surgido no qual o trabalho é enfatizado. Esse estudo engloba os seguintes aspectos: definição dos conceitos necessários e importância da *World Wide WEB*; arquitetura e protocolos envolvidos na *WEB*; acesso de bancos de dados via *WEB*; dados semi-estruturados; arquiteturas para

integração de informações; e apresentação de alguns projetos de pesquisas existentes no contexto de dados semi-estruturados.

O terceiro capítulo apresenta um estudo sobre a tecnologia XML. O enfoque é dado sobre a sintaxe da linguagem XML sua relação com dados semi-estruturados, bem como a tipificação de dados semi-estruturados.

No quarto capítulo são apresentadas as principais linguagens de consulta em estudo atualmente no contexto do capítulo anterior e as técnicas envolvidas em tais linguagens.

O quinto capítulo apresenta um estudo da *API XQuery* do *Framework .NET* e uma extensão para esta *API* com uso dos recursos dispostos pelo *Framework*, pela linguagem de programação C# (*C-Sharp*) e pela linguagem XSL.

O sexto capítulo apresenta as conclusões e recomendações deste trabalho.

CAPÍTULO II

2. BANCO DE DADOS DISTRIBUÍDO E WEB

2.1 Introdução

Este capítulo do trabalho faz uma revisão literária dos principais tópicos em comum nas duas comunidades de pesquisa inter-relacionadas: a comunidade *WEB* e a comunidade de banco de dados. Estes pontos são abordados com o propósito de demonstrar a necessidade de convergência das visões de estrutura da informação definida em cada comunidade em virtude da evolução dos sistemas de informação.

O grande crescimento da *WEB* através do surgimento de novas tecnologias torna cada vez mais comum a publicação de informações neste ambiente por pessoas do mundo inteiro a qualquer instante. A expressiva facilidade, flexibilidade, produtividade e o poderoso recurso de compartilhamento de informações que a *WEB* permite são marcantes no contexto mundial.

Algumas características naturais do ambiente *WEB*, tais como o constante aumento de volume de informações disponíveis, a grande diversidade de informações e as constantes atualizações destas informações fizeram crescer a demanda por aplicativos com modos distintos para a entrega/acesso de dados (Özsu e Valduriez, 1999).

A área de pesquisa criada com o surgimento destas alternativas de entrega/acesso de dados pode ser caracterizada de várias maneiras (Liu et al., 1998). Segundo Özsu e Valduriez (1999), "*Dois pontos importantes nesta área de pesquisa são os sistemas baseados em pull, nos quais os clientes acessam servidores de dados para obter os dados necessários e os sistemas baseados em push, nos quais os servidores "entregam" dados aos clientes sem esperar que eles sejam solicitados*".

O acesso a dados na *WEB* é comumente baseado na técnica de *pull*. A idéia do projeto inicial da *WEB* foi fornecer um sistema de hipertexto distribuído na Internet. Devido a sua interface gráfica (*browsers*) facilitar a navegação nos documentos distribuídos, este sistema teve grande aceitação e difusão. É possível perceber que as organizações estão usando em escala crescente a *WEB* em *intranets* (redes privadas de servidores da *WEB*) para várias aplicações, tais como publicação de informações, *workflow* (gerenciamento eletrônico de informações), *groupware*[1] e acesso a sistemas de informações.

A *WEB* é orientada a documentos e fornece mecanismos de recuperação de informações para pesquisar documentos distribuídos. De acordo com Özsu e Valduriez (1999), "*A indústria de banco de dados, desenvolveu gateways para realizar a interface entre documentos da WEB e bancos de dados. Com isso, o número de origens de dados heterogêneos que podem ser acessadas a partir da WEB cresceu exponencialmente. Desta forma, o acesso integrado a várias origens de dados distribuídas se tornou um desafio técnico bastante grande e complexo*".

Neste contexto, tecnologias de bancos de dados distribuídos são extremamente úteis, mas precisam ser ampliadas a fim de lidar com questões relativas ao ambiente *WEB*. Uma delas é a ampla heterogeneidade das origens de dados e de seus recursos computacionais, que variam desde bancos de dados altamente estruturados até

[1]Groupware é um software que permite que um grupo de usuários de uma rede colabore em determinado projeto. O groupware pode oferecer serviços para comunicação (por exemplo, correio eletrônico), desenvolvimento de documentos colaborativos, agendamento e acompanhamento. Os documentos podem conter texto, imagens e outros tipos de informação.

arquivos sem qualquer estrutura, com documentos semi-estruturados sendo um interessante ponto intermediário.

Em uma outra cultura, tecnologias baseadas em *push* são propostas devido ao fluxo de informação exibido na comunicação de muitos aplicativos, como por exemplo, sistemas de transmissão de notícias, distribuição de software e informações de tráfego. Nesses ambientes, a direção do fluxo de informação pode ser mais interessante dos servidores para os clientes, ou seja, pode fazer mais sentido “empurrar” os dados dos servidores sem esperar que os clientes “puxem” estes dados, conforme pode ser observado na figura abaixo.

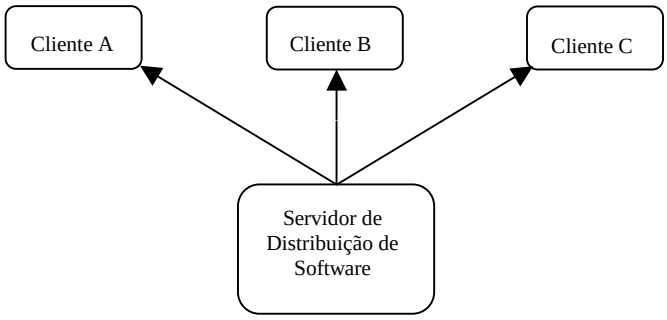


Figura 1 – Tecnologia baseada em push.

2.2 Paradigma WEB *versus* Banco de Dados Distribuído

A grande rede de computadores denominada comumente como *WEB* cresceu exponencialmente e sem limites de fronteira, dando um grande impulso para a formação da nova era da sociedade da informação. Da mesma forma, expandiu-se também a quantidade e diversidade de usuários e aplicações que dela fazem uso e que, de alguma forma, necessitam de informações contidas nesta grande rede mundial para seu auto-desenvolvimento e crescimento.

2.2.1 Os Problemas na WEB

Com o veloz crescimento desta nova sociedade da informação, veloz também foi o agravamento de problemas de ordem prática, tais como a segurança e o acesso às informações. Da mesma forma, é possível notar que à medida que a rede mundial cresce, crescem também as dificuldades para a localização de informações relevantes dentro de uma determinada área de pesquisa desejada, bem como a análise criteriosa destas informações.

A estrutura de dados é um item difícil de ser explorado neste contexto, visto que a unidade de informação é tipicamente um arquivo criado por um usuário da *WEB* e compartilhado com outros por meio da disponibilização de seu nome na forma de um URL (*Uniform Resource Locator*).

A principal estrutura contida nos documentos da *WEB* (derivada do desenvolvimento da HTML – *Hipertext Markup Language*) é uma estrutura de texto para apresentação visual. Este é o padrão simples e universal usado para a troca de informação.

2.2.2 O Paradigma da WEB

Da perspectiva de um banco de dados, a *WEB* gerou demanda por arquiteturas de bancos de dados recentemente desenvolvidas, tais como *envoltórias* e *sistemas de mediação* para integração de bancos de dados, levando às pesquisas e desenvolvimento de modelos de dados semi-estruturados com linguagens de consulta adaptadas a este modelo.

Em comparação com sistemas convencionais de gerenciamento de bancos de dados, a comunicação com dados na *WEB* apresenta uma mudança essencial de paradigma. A abordagem padrão de bancos de dados é baseada em uma arquitetura cliente/servidor como pode ser vista na figura abaixo (Özsu e Valduriez, 1999).

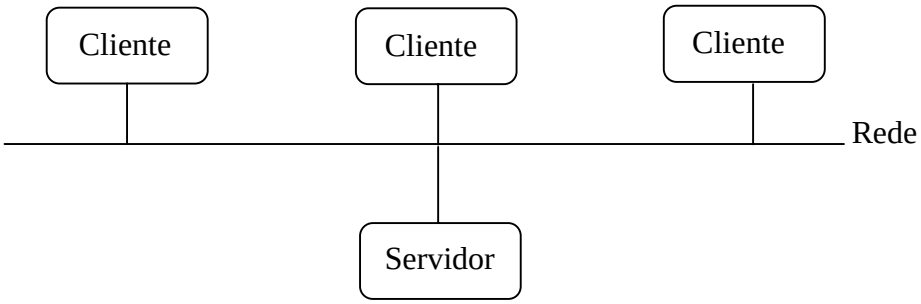


Figura 2 – Arquitetura tradicional de banco de dados cliente servidor.

O cliente (uma pessoa ou um programa) emite uma consulta que é processada, compilada em um código otimizado e executada. Dados de resposta são devolvidos pelo servidor.

Em contrapartida, o processamento de dados no contexto *WEB* é baseado em uma abordagem de múltiplas camadas como pode ser verificado na figura abaixo (Özsu e Valduriez, 1999).

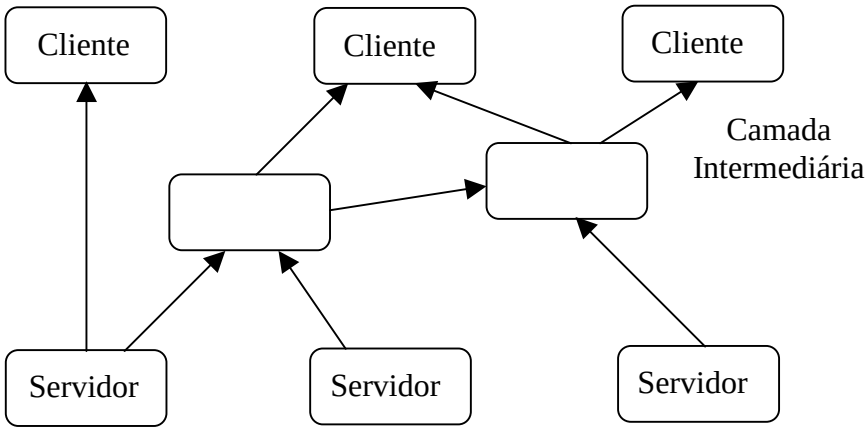


Figura 3 – Arquitetura de aplicação com base na WEB.

A camada mais baixa é composta por fontes de dados, as quais são conhecidas como servidores de dados. Estes podem ser servidores de bancos de dados convencionais, servidores de arquivos, ou qualquer aplicação que forneça dados. A fim de fornecer as informações de uma forma regular (conforme regras) para os clientes, os servidores de dados traduzem suas informações para um modelo de dados de lógica comum e um formato comum (atualmente é muito provável que se utilize o formato

XML). A camada mais alta, a camada do cliente, consiste em interfaces ou aplicações com o usuário. Entre as duas pode haver uma enorme coleção de camadas intermediárias, freqüentemente chamadas *middleware* (o software que transforma, integra ou adiciona valor aos dados).

2.2.3 O Paradigma em Banco de Dados Distribuído

Várias pesquisas acerca de bancos de dados distribuídos foram realizadas para a integração de dados no contexto do *middleware*.

2.2.3.1 Data Warehousing

Uma das abordagens destas pesquisas é o *datawarehousing* como pode ser visualizado na figura 4 (Özsu e Valduriez, 1999). O *datawarehouse* é considerado como um banco de dados, geralmente muito grande, capaz de acessar todas as informações de uma empresa.

Apesar de poder ser distribuído entre diversos computadores e poder conter diversos bancos de dados e informações em diversos formatos, o acesso a essa área de armazenamento de informações deve ser feito através de um servidor. Portanto, o acesso ao *datawarehouse* é transparente para o usuário, que pode utilizar comandos simples para simplificar e analisar todas as informações.

O *datawarehouse* contém ainda dados sobre a forma como está organizado, sobre onde as informações podem ser encontradas e sobre as conexões entre os dados. Geralmente utilizado para o apoio à decisão dentro de uma empresa, o *datawarehouse* também permite à instituição organizar seus dados e verificar relacionamentos entre informações obtidas de diferentes partes da organização.

Através desta abordagem, o *middleware* realiza a tarefa de importação dos dados da fonte e armazena os mesmos num banco de dados intermediário construído e

elaborado especialmente para isso (o *warehouse*), o qual é consultado pelo cliente. O grande desafio nesta abordagem é a tarefa de manter o referido banco de dados atualizado (em dia) no momento em que as fontes de informação são atualizadas.

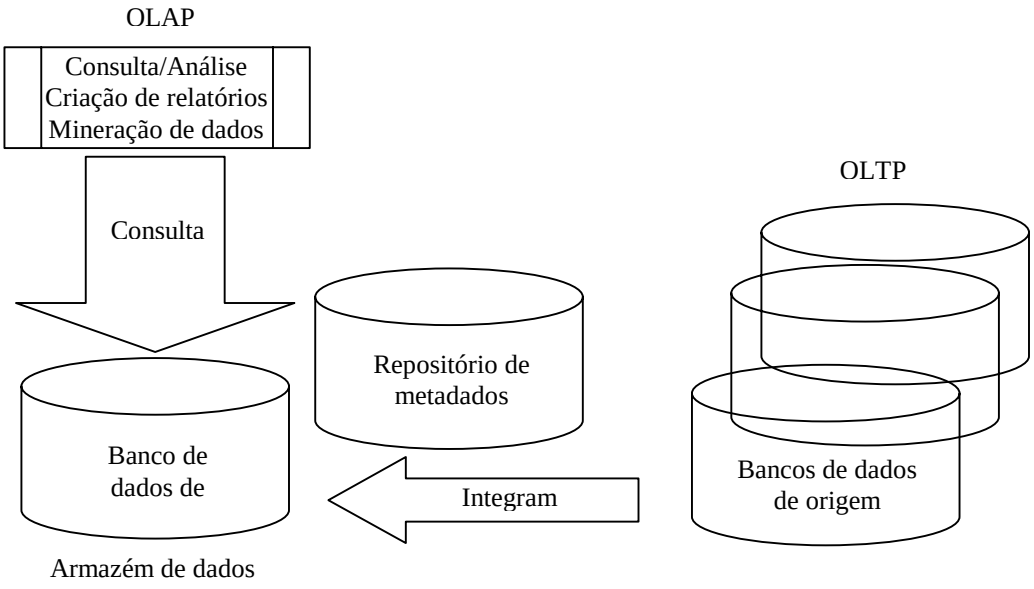


Figura 4 – Arquitetura de um data warehouse.

Um ou mais bancos de dados de origem, contendo dados operacionais atualizados por aplicativos de OLTP (*On-Line Transaction Processing*) estão integrados em um único banco de dados de destino (ou armazém de dados).

O OLTP é um sistema usado para processar as transações assim que o computador as recebe, atualizando imediatamente os arquivos mestres de um sistema de gerenciamento de banco de dados. O processamento OLTP é útil no acompanhamento de registros financeiros e no controle de inventário.

O banco de dados de destino é acessado através de consultas por aplicativos de *desktop* tais como ferramentas de consulta e análise, emissão de relatório e mineração de dados. Os aplicativos de *desktop* populares para análise de dados são programas de planilhas eletrônicas.

Como o próprio nome sugere, o repositório de *metadados* é um banco de dados, separado, contendo informações e executando o controle dos dados armazenados

atualmente no armazém de dados. A composição típica dos *metadados* inclui as descrições de tabelas de destino com suas definições de origem. Basicamente o repositório de *metadados* atua como um projeto/esquema lógico com a função de isolar o armazém de dados das mudanças que porventura possam ocorrer no esquema de bancos de dados de origem.

Desta maneira, caso ocorra alguma mudança no esquema de um/algum banco de dados de origem, o administrador do armazém de dados pode simplesmente atualizar o repositório de *metadados* que a mudança será propagada automaticamente para o banco de dados de destino bem como para os aplicativos OLAP (*On-line Analytical Processing*).

O OLAP é um sistema de banco de dados relacional capaz de tratar consultas mais complexas que àquelas tratadas por bancos de dados relacionais padrões, através do acesso multidimensional aos dados (exibição dos dados através de diferentes critérios), da capacidade de cálculo intensivo e de técnicas de indexação especializadas.

2.2.3.2 Sistema Mediador

O processo de integração de informações de várias origens de dados na Internet cria a necessidade de alguma forma de visão interligada dos dados a fim de permitir a formulação de consultas distribuídas. Dentro do contexto da Internet surgem algumas questões mais complexas do que aquelas dos sistemas de bancos de dados distribuídos. A primeira questão é que a quantidade de origens de informações pode ser extremamente elevada, consequentemente a integração de visões e a resolução de conflitos torna-se um problema ainda maior.

Uma segunda questão é a dinâmica do espaço de origens de dados, devido a este fator a inclusão ou eliminação de uma determinada origem de dados deve ser realizada com o menor impacto possível sobre a visão global integrada.

Uma outra questão menos visível, porém não menos importante, é a utilização de recursos computacionais que podem ser bem distintas de uma origem de dados para outra, variando desde sistemas gerenciadores de bancos de dados completos até simples arquivos de texto. Esta característica é um tanto quanto distinta da característica de sistemas de bancos de dados distribuídos, onde subentende origens de dados no mínimo com uma interface SQL. E por fim, uma última consideração é com relação à estruturação das origens de dados, que podem ser bem estruturadas, não-estruturadas ou até mesmo semi-estruturadas, nas duas últimas não oferecendo praticamente nenhuma informação para a integração de visões.

A fim de resolver estes problemas, surge uma segunda abordagem para a integração de dados no contexto do *middleware*, a qual é composta da utilização de um sistema *mediador-envoltória* (figura 5), no qual as consultas do cliente são modificadas e decompostas a fim de serem expressas diretamente em consultas junto à fonte de dados (Özsu e Valduriez, 1999). Os resultados parciais de várias fontes são integrados pelo mediador em tempo real, resolvendo-se o problema de atualização, porém degradando o desempenho nas tarefas de comunicação e transformação da consulta.

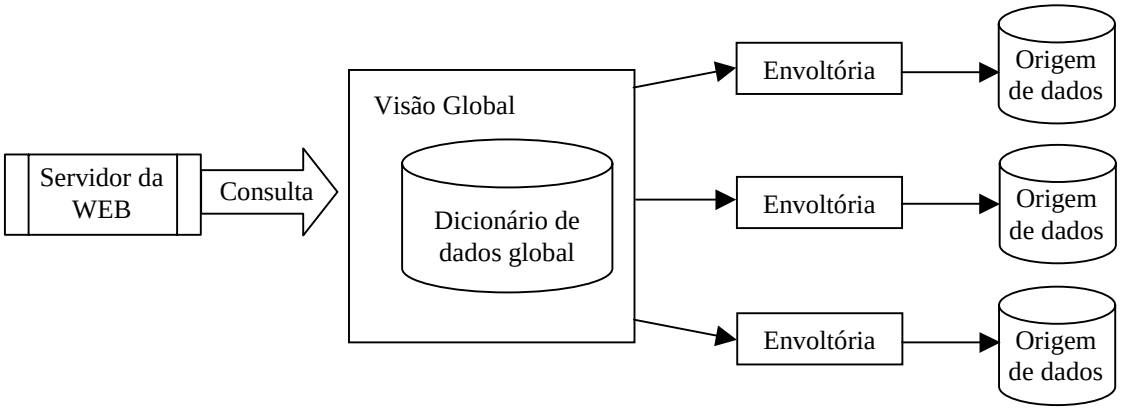


Figura 5 – Arquitetura de mediador-envoltória.

Para cada origem de dados, uma envoltória exporta determinadas informações sobre o esquema, dados e recursos de processamento de consultas de origem. O mediador centraliza as informações fornecidas pelas envoltórias através de uma visão unificada de todos os dados disponíveis, armazenando-as no dicionário de dados global.

A consulta requisitada pelo usuário (no caso da figura 5 advinda de um servidor da WEB) é então decomposta também pelo mediador em consultas menores (executáveis pelas envoltórias) e, por fim, os resultados parciais contidos em cada envoltória são reunidos pelo mediador que calcula e fornece a resposta à consulta do usuário.

O modelo de *mediador-envoltória* é uma abstração com ampla aceitação para o problema da integração de informações (Özsu e Valduriez, 1999). A arquitetura de *mediador-envoltória* difere fundamentalmente de um armazém de dados no fato de que os dados integrados não são materializados. O mediador pode ser usado como um banco de dados de origem para um armazém de dados.

Algumas vantagens podem ser claramente observadas na arquitetura *mediador-envoltória*. A primeira delas é que os componentes especializados da arquitetura permitem que problemas e preocupações de diferentes tipos de usuários podem ser tratados em separado (modularidade).

Uma outra vantagem é que os mediadores em geral são especificados de forma a serem especializados num conjunto inter-relacionado de origens de dados, onde os dados têm características semelhantes (algo em comum), e assim trabalham com esquemas e semânticas determinadas para um domínio específico.

A especialização dos componentes tem por objetivo tornar o sistema distribuído flexível e extensível. Estes objetivos podem ser abstraídos através da figura abaixo (Özsu e Valduriez, 1999), a qual demonstra uma hierarquia de mediadores especializados, onde um mediador IR (*Information Request*) integra vários mecanismos de pesquisa, um mediador DB (*Database*) para bancos de dados heterogêneos integra dois bancos de dados de arquiteturas distintas e um mediador de um nível hierárquico superior denominado IR/DB a fim de oferecer recursos tanto de IR como consultas a bancos de dados.

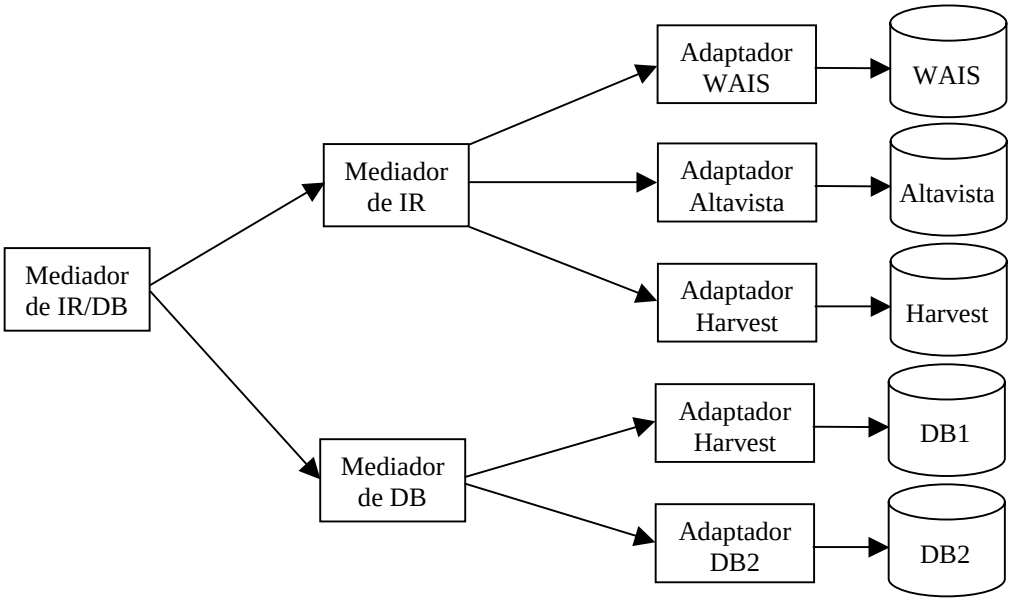


Figura 6 – Hierarquia de mediadores.

2.3 Necessidade de Convergência

As tecnologias da informação têm apresentado grandes avanços na habilidade de gerar, distribuir e armazenar informações. Infelizmente, as ferramentas para localizar, filtrar, e analisar essas informações, não têm acompanhado o mesmo ritmo de avanço. Uma solução emergente é a convergência da cultura da *WEB* com a cultura de sistemas de bancos de dados.

A tecnologia XML se apresentou como o primeiro passo na direção da convergência destas duas visões sobre estrutura da informação. Uma vez que a XML é sintaticamente relacionada à HTML, ferramentas foram desenvolvidas para converter XML em HTML. No entanto, a idéia principal aqui não é descrever formatos textuais tal como a HTML o faz, mas sim transmitir dados estruturados. Neste sentido, a XML tem por finalidade descrever linguagens de marcação de uma forma estruturada. Segundo Abiteboul (Abiteboul et al., 1999): "*Enquanto a WEB proporciona:*

- *uma infra-estrutura global e conjuntos de padrões para sustentar a troca de documentos;*

- *um formato de apresentação para hipertexto (HTML);*

- *interfaces com o usuário bem construídas para recuperação de documentos (técnicas de recuperação de informação);*

- *um novo formato, XML, para a troca de dados de uma forma estruturada;*

a tecnologia de banco de dados fornece:

- *técnicas de armazenamento e linguagens de consulta que fornecem acesso eficiente a grandes conjuntos de dados altamente estruturados;*

- *modelos de dados e métodos para estruturar dados;*

- *mecanismos para a manutenção da integridade e consistência de dados;*

- *um novo modelo, o de dados semi-estruturados, que abranda os rigores dos sistemas de bancos de dados altamente estruturados”*.

2.4 Dados semi-estruturados

Várias estruturas tradicionais da teoria de banco de dados precisam ser reinventadas no contexto da *WEB*. As informações na *WEB* não são bem ajustadas ao esquema de tabelas como em sistemas de banco de dados, pelo contrário, as informações neste cenário são auto-descritivas e irregulares, contendo uma pequena

diferença entre o esquema e os dados. Esta diferença tem sido formalizada através do conceito de dados semi-estruturados.

Os esquemas de dados semi-estruturados são bem distintos daqueles tradicionalmente conhecidos em tabelas no contexto de sistemas de banco de dados ou até mesmo dos esquemas orientados a objeto mais complexos (Vianu V., 2001). É possível perceber nitidamente que um dado semi-estruturado carece de um esquema fixo e rígido, embora o dado possua uma estrutura implícita.

Enquanto a falta de um esquema fixo torna a extração de dados semi-estruturados bastante fácil e objetivamente atrativa, a apresentação e consulta de tais dados é muito prejudicada (Vianu V., 2001). Portanto, um problema crítico é decifrar a estrutura que está implícita no dado semi-estruturado e, sequencialmente, reformular os dados em termos desta estrutura.

As linguagens de consulta em dados semi-estruturados também diferem significativamente de suas correspondentes em sistemas de banco de dados relacionais. A falta de um esquema bem definido conduz as linguagens para uma abordagem de navegação, onde a informação é explorada através de pontos de entrada específicos (Vianu V., 2001). A estrutura aninhada dos dados determina o uso da recursão em consultas, na forma de expressões de caminho.

2.4.1 Dados semi-estruturados e XML

Os dados semi-estruturados podem ser expressos por grafos rotulados. Os nodos são vistos como objetos e possuem identificadores. Estes objetos podem ser atômicos ou complexos.

Objetos complexos são aqueles que têm a característica de serem ligados a outros objetos por arestas rotuladas. Objetos atômicos possuem valores associados a eles mesmos. A intenção é produzir um formalismo mais poderoso e flexível para

descrever a informação de uma maneira unificada e integrada. A figura abaixo mostra um grafo de dados (Abiteboul et al., 1999).

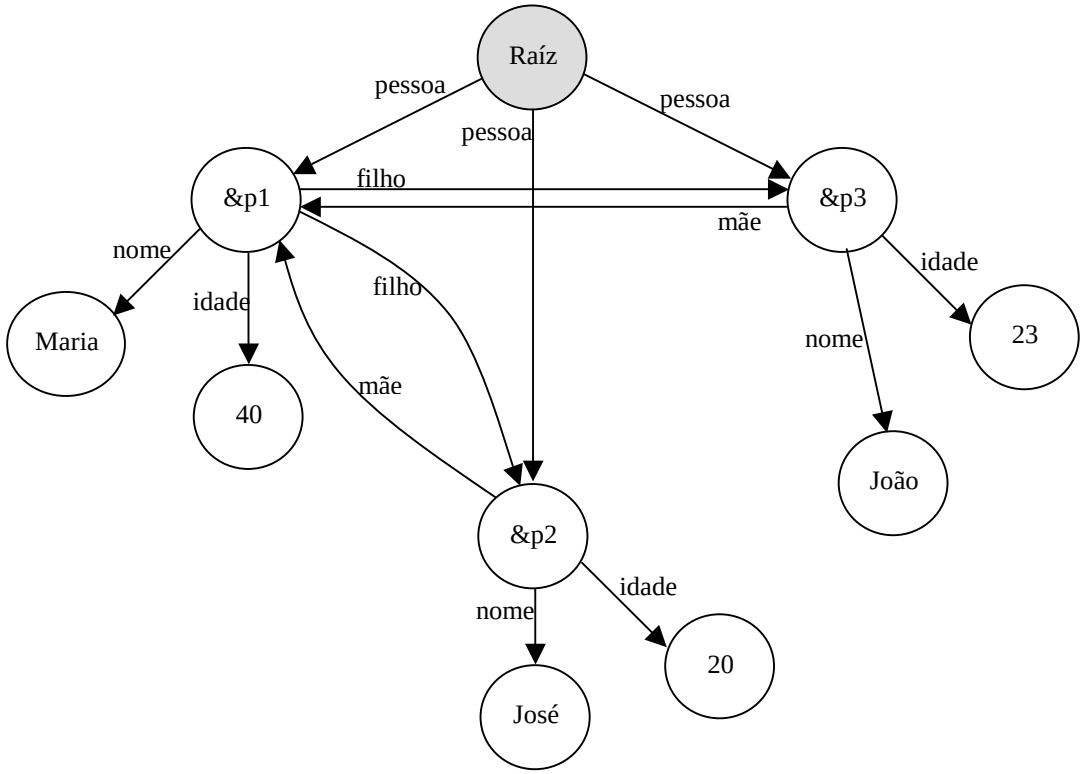


Figura 7 – Um grafo de dados semi-estruturado.

Bancos de dados relacionais ou orientados a objeto também podem ser expressos por grafos. Por exemplo, a base de dados da tabela 1 é representada pelo grafo da figura 8.

R	***A***	***B***	***C***
	1	*1*	*2*
	2	*1*	*3*

Q	***C***	***D***
	2	*1*
	1	*0*

Tabela 1 – Um banco de dados relacional expresso em tabelas.

Pela definição das tabelas acima é possível perceber os seguintes esquemas relacionais: R = {A, B, C} e Q = {C, D}. Já pela representação através de grafos é importante observar que não existe uma distinção explícita entre dados e esquema.

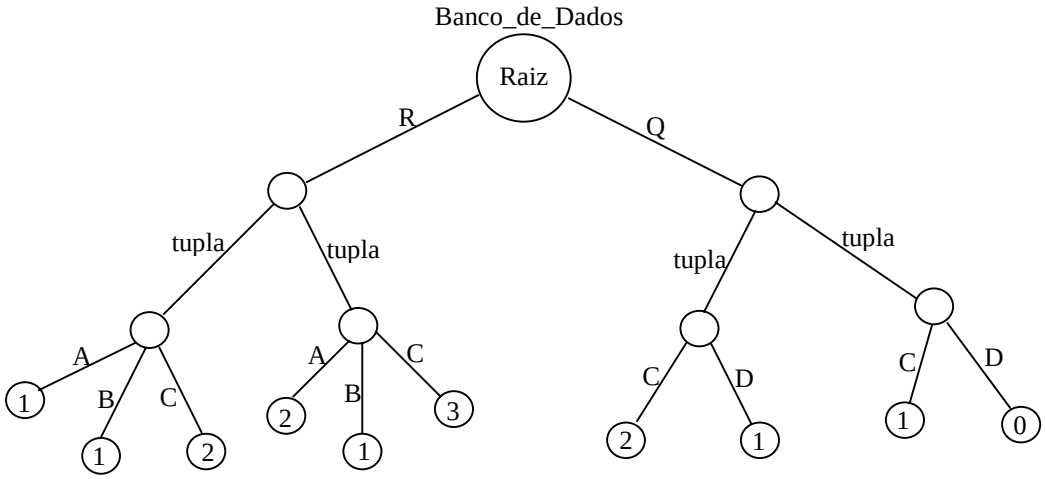


Figura 8 – Um grafo de dados para as relações R e Q.

Muitas variantes do modelo de dados semi-estruturado têm sido propostas, com diferenças mínimas no formalismo. O primeiro modelo de dados semi-estruturado foi o *OEM* (*Object Exchange Model*), introduzido no projeto *Tsimmis* como um veículo de integração de origens heterogêneas (Chawathe et al., 1994). Este modelo foi seguido também pelo sistema *Lore* (McHugh et al., 1997). Um outro modelo, *UnQL*, foi desenvolvido na Universidade da Pensilvânia, motivado pelo modelo *OEM* e pelo modelo de grafos *AceDB* usado em bancos de dados biológicos.

Diferentemente dos modelos de dados semi-estruturados, XML não nasceu na comunidade de banco de dados. Ela foi introduzida na comunidade de pesquisas à tecnologia de integração de documentos, mais especificamente na *WEB*, como um subconjunto da linguagem SGML. A XML é, de uma certa forma, um acréscimo a HTML de forma a permitir dados explanados com informação sobre o seu significado além de sua apresentação.

Um documento XML consiste de elementos aninhados, com sub-elementos ordenados. Cada elemento tem um nome (também conhecido como *tag* ou *label*). Um exemplo de documento XML, usado para armazenar anúncios de carros novos e usados, pode ser visto na figura 9-a e sua abstração pode ser representada por uma árvore rotulada, a qual pode ser vista na figura 9-b (os valores na árvore são omitidos).

a)

```
<Revendedora>
  <CarrosUsados>
    <anúncio>
      <modelo>Honda civic</modelo>
      <ano>2001</ano>
    </ anúncio >
  </CarrosUsados>
  <CarrosNovos>
    < anúncio >
      <modelo>BMW 325i</modelo>
    </ anúncio >
  </CarrosNovos>
</Revendedora>
```

b)

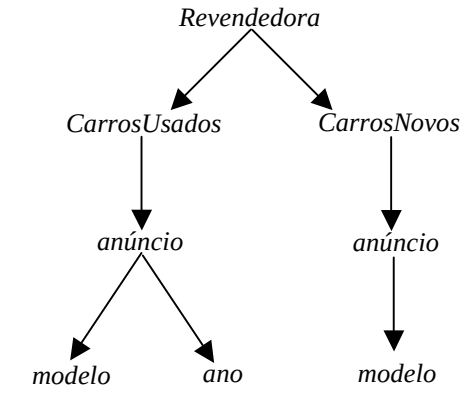


Figura 9 – Um documento XML simples.

A ascensão da tecnologia XML aumentou a importância da abstração da estrutura de documentos XML através de árvores rotuladas. Além disso, a XML fornece um mecanismo de referência entre elementos que permite a simulação de grafos arbitrários e, portanto, de dados semi-estruturados.

É importante mencionar que a XML pode ser vista como um modelo de objeto. Isto é ilustrado por uma API (*Aplication Program Interface* - Interface para programação de aplicações) padrão para XML proposta pelo W3C, onde documentos XML são descritos em termos do modelo DOM (*Document Object Model*).

2.4.2 Esquemas para dados semi-estrutudados

A flexibilidade obtida com o conceito de dados semi-estruturados tem um preço: a carência do conceito de esquema. Este conceito apresenta-se muito útil, visto que descreve a inter-relação entre os dados auxiliando na consulta dos mesmos, permitindo a otimização de consultas mais complexas e o armazenamento eficiente de informações. A fim de manter algumas destas vantagens, existem esforços na pesquisa direcionada à recuperação da informação do esquema em dados semi-estruturados.

Um esquema para dados semi-estruturados leva ao conceito de *paths* (caminho; mais precisamente as seqüências de rótulos geradas ao longo do

caminhamento em um grafo). Esta é uma extensão natural para dados semi-estruturados dos esquemas relacional e orientado a objeto. A linguagem XML marca o retorno do conceito de esquema em dados semi-estruturados, através da forma de seus *DTDs* (*Data Type Definitions*).

2.4.3 Restrições em dados semi-estruturados

As restrições são ingredientes fundamentais no contexto de bancos de dados clássicos. Enquanto suas regras primárias funcionam como um filtro para informações inválidas, suas funcionalidades se estendem também para a otimização de consultas, o projeto de esquemas e a escolha de métodos eficientes de armazenamento e acesso. A maioria das restrições em bancos de dados comuns é realizada em termos de dependências funcionais e dependências de inclusão. Este conceito continua sendo importante no contexto de dados semi-estruturados e XML. No entanto, a diferença entre as estruturas dos conceitos (BD e Dados Semi-Estruturados) conduz a diferenças significantes em como as restrições são especificadas e nas suas propriedades (Vianu V., 2001).

Assim como em dados semi-estruturados, existe uma necessidade natural de se expressar dependências de inclusão em documentos XML. Além disso, restrições de chave são inclusas em várias propostas, tal como em *XML Schema*[2]. Ambos os tipos de restrições também aparecem em documentos XML gerados a partir de bancos de dados.

Em XML, restrições de chave e dependências de inclusão envolvem valores de dados associados às folhas dos documentos XML (ou aos valores dos atributos vistos como elementos folhas), enquanto que em dados semi-estruturados as dependências de inclusão referem-se aos próprios nodos (valores de dados podem ser facilmente modelados como nodos, apesar de que fazer isso em XML destruiria a estrutura de

árvore dos documentos). As dependências de inclusão podem ser expressas em XML de forma análoga como é expressa em dados semi-estruturados, utilizando-se expressões de caminho.

Existe uma interação intrínseca entre restrições XML e *DTDs*. O impacto de *DTDs* e outros formalismos de esquema sobre o conceito de restrições interessa tanto à teoria quanto à prática, e se mantém recentemente pouco explorado. Uma pesquisa mais aprofundada de restrições em dados semi-estruturados e em XML pode ser vista em (Fan e Libkin, 2001). Restrições em dados semi-estruturados também são discutidas em (Abiteboul et al., 1999).

2.4.4 Projetos no contexto de dados semi-estruturados

Muitos projetos de pesquisa sobre integração de dados dentro do contexto da WEB foram realizados. Alguns desses projetos foram demonstrados e bons protótipos foram criados. Alguns desses projetos podem ser citados como principais para o estudo de importantes questões, por exemplo, o *TSIMMIS* (Chawathe et al., 1994) da Universidade de Stanford (EUA), o *Garlic* (Haas et al., 1997) do *Almaden Research Laboratories* da IBM (EUA), o *Information Manifold* (Levy et al., 1996) e o *Strudel* (Fernandez et al., 1998) na *AT&T Research Laboratories* (EUA) e o *Disco* (Tomasic et al., 1998) em *Inria* (França).

O *TSIMMIS* segue a arquitetura de um sistema *mediador-envoltória*, permitindo o uso de hierarquias de envoltórias e mediadores. Os componentes se comunicam com a utilização do modelo de dados semi-estruturados e de uma linguagem de consulta associada chamada MSL (*Mediator Specification Language*). As instruções MSL são regras lógicas que podem lidar com objetos. O *TSIMMIS* tem como característica principal a geração automática de mediadores e envoltórias através do uso

---

[2]XML Schema é uma linguagem de definição que permite definir a estrutura e os tipos de dados em documentos XML. Um XML Schema define os elementos, atributos e tipos de dados que se adequam ao

de uma linguagem livre de contexto, facilitando a descrição de recursos de consulta. O projeto *TSIMMIS* concentra-se principalmente na otimização de consultas do tipo seleção-projeção, dando pouca ênfase às junções, as quais são consideradas pouco prováveis no contexto da *WEB* (Chawathe et al., 1994).

O *Garlic* pressupõe que os recursos de consulta das origens são desconhecidos para o mediador; encontrar um plano de execução significa negociar com as origens a maior proporção possível de um plano que seja possível manipular. A estratégia do *Garlic* pode levar a um tráfego de rede desnecessário entre o mediador e as envoltórias. O *Garlic* lida apenas com consultas conjuntivas (Haas et al., 1997).

O projeto *Information Manifold* (*IM*) proporciona acesso uniforme a grandes coleções heterogêneas de origens de dados através da *WEB*. O *IM* fornece um mecanismo para descrever de forma declarativa o conteúdo e os recursos de consulta das origens de dados disponíveis. O *IM* tem várias características inovadoras. Primeiro, ele fornece um mecanismo no qual o conteúdo das origens de dados é descrito sob a forma de consultas sobre um conjunto de relações e classes. Desse modo, é possível modelar as distinções mais minuciosas entre o conteúdo de diferentes origens, bem como é fácil adicionar e eliminar origens.

A modelagem dos recursos de consulta de origens de dados é crucial para a integração com muitas origens existentes. Em segundo lugar, o *IM* emprega um algoritmo eficiente que usa as descrições de origens para criar planos de consulta que podem acessar diversas origens de dados para responder a uma consulta. O algoritmo seleciona as origens acessadas para responder à consulta e considera os recursos das diferentes origens (Levy et al., 1996).

Strudel é outro projeto de integração de informações realizado pela *AT&T Research* com foco em dados semi-estruturados. O sistema incorporado ao projeto *Strudel* aplica conceitos de sistemas de gerenciamento de bancos de dados ao processo

---

W3C XML Schema Parte 1: Estruturas Recomendadas para a Linguagem de Definição XML Schema.

de construção de *sites* da *WEB*. A idéia chave deste projeto envolve a separação do gerenciamento dos dados do *site* da criação e gerenciamento da estrutura do *site* bem como da apresentação visual das páginas do *site*. Primeiramente o construtor do *site* cria um modelo uniforme de todos os dados disponíveis no *site*. Depois o construtor utiliza esse modelo para definir de forma declarativa a estrutura do *site*. E por fim o construtor especifica a apresentação visual das páginas na linguagem de modelos de HTML do *Strudel*. O modelo de dados subjacente do *Strudel* é um modelo de grafos orientados semi-estruturados identificados (Fernandez et al., 1998).

No projeto *Disco*, mediadores e envoltórias operam de modo independente: um mediador acessa uma envoltória simplesmente através de uma descrição da envoltória semelhante a um URL. Essa característica faz com que as envoltórias possam ser compartilhadas facilmente entre vários mediadores. Além disso, as envoltórias opcionalmente exportam estatísticas de custo e equações de custo que descrevem o tamanho dos dados nas origens subjacentes e o custo de acessar essas origens (Naacke et al., 1998). Os mediadores do *Disco* usam essas informações de custo para executar uma sofisticada otimização de consultas baseada no custo.

O processamento de consultas realizado no *Disco* pode continuar a funcionar mesmo quando algumas origens de dados estão indisponíveis. Durante o processamento de consultas, as origens de dados indisponíveis são detectadas e o processamento de consultas continua sendo executado para as origens de dados disponíveis, armazenando os resultados parciais dessas consultas. Quando as origens de dados indisponíveis se tornam disponíveis, seus resultados são integrados aos resultados anteriormente armazenados para produzir a resposta final à consulta solicitada (Tomasic et al., 1998).

Através dos projetos citados, foi possível uma melhor compreensão da complexidade embutida no contexto da integração de informações de origens de dados distribuídos heterogeneamente. É possível notar a necessidade de um trabalho intenso e complexo a fim de facilitar o desenvolvimento de mediadores e envoltórias em vários domínios de aplicações.

2.5 Conclusão

Neste capítulo foi possível observar que, apesar das necessidades que surgem sobre o trabalho na informação tanto num ambiente altamente estruturado (como é o caso de um sistema de banco de dados) como num ambiente semi-estruturado (como é o caso do ambiente *WEB*) serem praticamente as mesmas, as características que distinguem estes dois ambientes tornam os métodos para execução de tal trabalho completamente distintos. Por exemplo, a teoria matemática aplicada aos bancos de dados na forma de esquema, e que facilita enormemente o trabalho de acesso, consulta e armazenamento eficiente da informação contida em tais bancos de dados, é precária em dados semi-estruturados. A linguagem XML torna-se útil e de extrema importância para determinar um esquema, sendo fundamental para o retorno do conceito de esquema em dados semi-estruturados, através da forma de seus *DTDs* (*Data Type Definitions*) como é abordado no capítulo posterior.

CAPÍTULO III

A TECNOLOGIA XML

3.1 Introdução

Neste capítulo serão abordados tópicos referentes às principais características da tecnologia XML, a qual está ganhando espaço considerável tanto na comunidade de pesquisa WEB como na comunidade de pesquisa de banco de dados, já que é considerada um ponto de partida essencial na convergência de ambas as visões de estrutura de informação.

O crescimento massivo da Internet demonstrou um grande problema da tecnologia WEB baseada em HTML (*HyperText Markup Language*). A HTML foi projetada para a simples apresentação de conteúdo (alguns transformaram esta linguagem num tipo de arte) e para navegação na rede manualmente. O problema é que esta linguagem não satisfez ao processamento automatizado de informação (Jung F., 2000).

Este problema pode ser verificado num exemplo simples: um navegador WEB (*Internet Explorer, Netscape, ...*) "*sabe*" que o termo encontrado numa determinada página HTML "*<h1>SUN</h1>*" deverá ser apresentado para o usuário

como um cabeçalho. Mas qual o significado real desta informação? É claro que os humanos podem supor através do contexto qual o significado disso, porém o computador não pode fazer suposições, a não ser que seja programado para isto, o que é uma idéia ainda não pesquisada neste contexto (Myllymaki J., 2001).

O padrão HTML foi criado baseado no padrão SGML (*Standard Generalized Markup Language* – Linguagem Padrão de Marcações Genéricas), esta, por sua vez, foi um modelo criado em 1986 (ISO 8879) para ser um padrão de marcação generalizada. De acordo com *Richard Light*, a SGML é um padrão muito usado para codificar documentos estruturados, variando em tamanho e complexidade (Light R., 1999).

O problema da SGML é que ela fornece um padrão complexo para ser usado na Internet, e por isso criou-se a HTML, a qual se mostrou uma linguagem bem mais simples tendo por base o padrão SGML, e "*perfeitamente*" própria para ser usada na Rede Mundial. Portanto, a HTML surgiu a fim de fornecer um modo fácil, rápido e prático para formatação de textos na Internet (Jung F., 2000).

Em 1996, um novo grupo de *experts* liderado por Jon Bosak da *Sun Microsystems* e apoiado pelo W3C (*World Wide WEB Consortium*) iniciou um trabalho sobre um novo padrão (Oppel K., 1999). Este novo padrão tinha que ser simples, extensível e legível por computadores e humanos. Finalizado em fevereiro de 1998, este novo padrão foi chamado de XML (*eXtensible Markup Language* – Linguagem de Marcação Estendida).

No mesmo ano, o mundo comercial iniciou o uso deste novo padrão. Do segundo para o terceiro quadrimestre de 1998 a percentagem da indústria de TI (tecnologia da informação) nos EUA usando XML em páginas *WEB* pulou de 1% para 16% (Oppel K., 1999). O novo padrão foi rapidamente adotado pelos líderes da indústria tais como *Sun*, *Microsoft*, *DataChannel*, *NetScape*, *IBM*, *SAP*, *Adobe* e *Software AG*.

A introdução de XML marca uma nova fase na história da Internet, a transformação da *WEB* de uma rede de informação para uma base de conhecimentos e uma plataforma de computação global.

3.2 Características Gerais da XML

A XML é um subconjunto da SGML que permite que uma marcação específica seja criada para especificar idéias e compartilhá-las na rede. Esta recente tecnologia visa principalmente facilitar as pesquisas em documentos disponibilizados na WEB pelo fato de impor uma estrutura a estes documentos.

Desta forma, ao serem processadas pesquisas a determinadas informações na rede, é possível que estas pesquisas sejam efetuadas em uma estrutura pré-definida ao invés de serem simplesmente executadas buscas textuais (como acontece em documentos escritos na linguagem HTML).

Conforme a citação, a sigla XML significa *eXtensible Markup Language* (Linguagem de Marcação Extensível). Isto quer dizer que a XML, ao contrário da HTML, permite aos usuários definir suas próprias *tag's* de marcação de texto. Portanto, enquanto num documento HTML pode-se apenas definir a formatação dos elementos, num documento XML pode-se definir o que estes dados significam, agregando mais informação a um documento.

A XML permite que usuários da Internet criem dados estruturados e definam informações sobre a estrutura destes dados. Desta forma, é possível trabalhar os dados de maneira que eles se mantenham como informações legíveis e compreensíveis aos olhos de um usuário. A tabela 2 (Light R., 1999) compara trechos de documento em HTML e XML.

<!—- Trecho de documento HTML --> <h1>Invoice</h1> <p>From: Joe Bloggs <p>To: A. Another <p>Date: 1 Feb 1999 <p>Amount: $100.00 <p>Tax: 21% <p>Total Due: $121.00	<!—Trecho de documento XML --> <Invoice> <From>Joe Bloggs</From> <To>A. Another</To> <Date year = ‘1999’ month = ’2’ day = ‘1’></Date> <Amount currency = ‘Dollars’>100.00</Amount> <TaxRate>21</TaxRate> <TotalDue currency = ‘Dollars’>121.00</TotalDue> </Invoice>

Tabela 2 – Trechos de documentos HTML e XML

Comparando os dois trechos de documentos HTML e XML acima, é possível notar que a linguagem XML é auto-explicativa, isto é, sua estrutura revela mais a respeito do significado do documento do que a linguagem HTML. Esta última preocupa-se apenas com a formatação dos dados para uma visualização final mais agradável.

A possibilidade de se definir as próprias *tag's* abre diversas possibilidades para a publicação de dados na *WEB*. Uma página HTML carrega consigo somente informações sobre parágrafos, cores, fontes, etc. conforme ilustrado na figura abaixo.

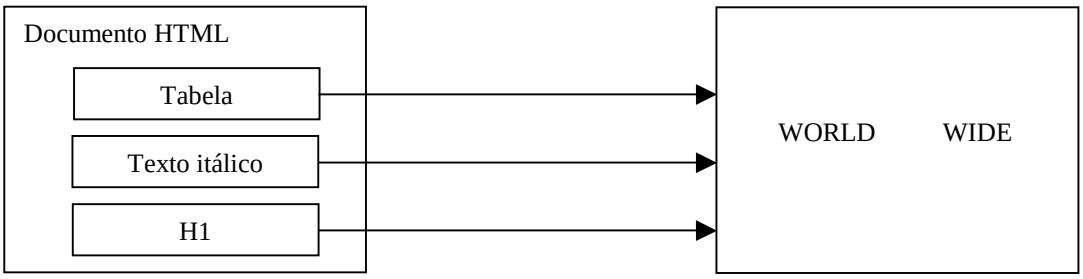


Figura 10 – Exibição de documento HTML na WEB.

Segundo *McGrath* (McGrath S., 1999), como consequência, quando este documento se torna disponível na *WEB*, os diferentes utilitários de pesquisa e usuários vêem somente um conjunto de níveis, tabelas, texto em itálico, cabeçalho, etc.

Por outro lado, uma página XML carrega informações sobre os dados contidos nela mesma, também conhecidos como metadados (figura 11).

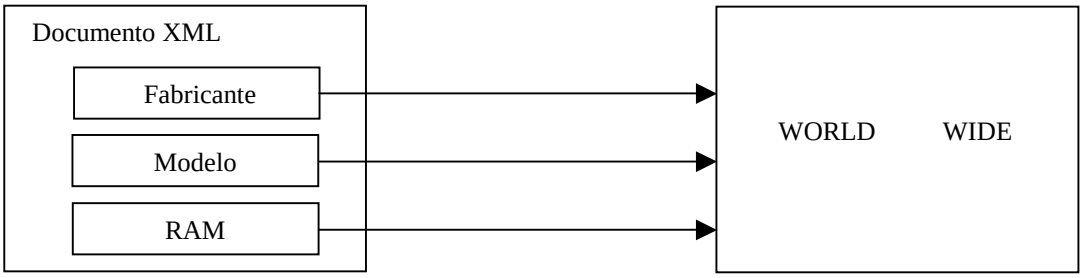


Figura 11 – Exibição de documento XML na WEB.

McGrath (McGrath S., 1999) diz que a filosofia da essência da XML apareceu como resultado de uma análise longa e cuidadosa do que realmente significa o termo "*documento*" no mundo digital.

O termo documento compreende três itens distintos reunidos num só, a saber: conteúdo, apresentação e estrutura. É possível notar que estes três itens estão relacionados diretamente ao conceito de banco de dados. Muitas vantagens podem ser obtidas ao separar estes três itens. Por exemplo, é possível obter apresentações distintas para uma mesma informação. Também é possível realizar a alteração dos dados de várias apresentações simplesmente alterando um só arquivo.

3.3 Pontos Fortes da XML

Alguns pontos principais desta nova linguagem podem ser citados em favor dos benefícios trazidos. O primeiro e principal ponto que é possível observar é que a XML apresenta-se como uma meta-linguagem, fazendo com que ela possa descrever qualquer tipo de linguagem com a capacidade de agregar informações.

Alguns exemplos de linguagens desenvolvidas com base na XML são:

- MathML: *Mathematical Markup Language*, a qual permite especificar a estrutura de expressões matemáticas (W3C1999a);

- OFX: *Open Financial Exchange*, linguagem criada por instituições financeiras para especificar o formato de documentos a serem intercambiados, tais como faturas, recibos e extratos;

- CML: *Chemical Markup Language*, a qual se apresenta como uma linguagem que define o formato de documentos a fim de permitir o intercâmbio de informações sobre produtos químicos, incluindo suas propriedades, usuários e fornecedores;

- SMIL: *Synchronized Multimedia Integration Language*, uma linguagem que viabiliza a autoria de documentos capaz de representar a integração de um conjunto de dados de mídia independentes em uma apresentação multimídia sincronizada. Trata-se de uma recomendação do W3C para formato e comportamento temporal de documentos multimídia (W3C1998a);

- XHTML: *Extensible Hypertext Markup Language*, uma linguagem padrão para a WEB que tornou-se uma recomendação do W3C em janeiro de 2000 (W3C2000a).

Um segundo ponto marcante sobre a linguagem XML é a possibilidade de várias adaptações. Marcações personalizadas podem ser criadas para um vasto campo de necessidades. As soluções para troca de dados atuais são extremamente complexas e caras para serem implementadas e mantidas. A flexibilidade da XML em definir vocabulários específicos abre a possibilidade para que pequenas e médias empresas tenham acesso ao intercâmbio de dados usando um padrão simples da Internet.

O terceiro ponto de destaque da linguagem XML está ligado ao fato de que ela contém somente idéias e marcações por ter sido criada com o intuito de estruturar, armazenar e compartilhar a informação pela WEB. Esta característica faz com que os documentos XML possuam uma manutenção relativamente fácil. Folhas de estilo e *links* são agregados separadamente do documento XML a fim de determinar uma

apresentação ao usuário final. Cada um destes conceitos pode ser modificado separadamente quando necessário.

A XML possui ainda embutido um conceito de ligação mais sofisticado que a HTML. Enquanto esta última faz uma ligação simples de um objeto a outro, a XML ainda pode ligar dois ou mais pontos a uma única idéia. Outra importante característica da linguagem XML é sua simplicidade. Sua antecessora, a SGML, possui trezentas páginas de especificação, enquanto a XML possui apenas trinta e três. As idéias desnecessárias foram retiradas em favor de idéias concisas (http://www.w3.org/XML/).

Duas grandes razões da existência da linguagem XML é a força de sua estrutura e a portabilidade. Enquanto a SGML foi trabalhada em termos da força de estrutura e a HTML em termos de portabilidade, a XML surgiu com ambas características. A XML pode ser navegada com ou sem um DTD (*Document Type Definition* – Definição de Tipo de Documento; normas que definem como as *tags* podem ser estruturadas nos documento XML). Tudo que um navegador precisa para processar um documento XML é saber que a folha de estilos é quem controla a aparência da apresentação dos dados contidos no documento. Se uma validação mais específica é necessária, o DTD pode acompanhar o documento XML e fornecer detalhes exatos da estrutura de marcação.

3.4 Document Type Definition (DTD)

Os documentos XML definem uma estrutura dos dados a serem exibidos. Esta estrutura também pode ser validada através de um documento separado denominado DTD – *Document Type Definition* – ou Definição de Tipo de Documento. Portanto um DTD nada mais é do que uma descrição formal usado para validar a sintaxe de declaração XML de um documento em particular (Light R., 1999).

O DTD define as estruturas de dados que deverão ser utilizadas dentro do documento XML associado. Neste caso, o programa interpretador fará uma análise do

documento XML com base nas estruturas definidas no DTD associado e emitirá uma mensagem de erro caso encontre alguma inconsistência.

Neste ponto nota-se uma diferença básica entre os formatos HTML e XML. Enquanto o primeiro dificilmente emite mensagens de erro, o último é mais rígido quanto à consistência e validade da estrutura do documento. Isto se torna uma vantagem à medida que, em documentos grandes, o número de erros involuntários tende a crescer. Assim, uma linguagem que auxilie na correção destes erros vem a ser de grande ajuda (Light R., 1999).

A validação de documentos XML acima descrita através do uso de um DTD é opcional. Assim, define-se uma distinção entre o conceito de documentos válidos e o conceito de documentos bem-formados. Estes últimos não são submetidos a um DTD. Neste caso, basta que estes documentos tenham seus elementos internos (também chamados de *tag's*) bem aninhados, formando uma estrutura de árvore. Já os documentos válidos são aqueles que foram validados através de um DTD.

O DTD então estabelece que nome será usado para os diferentes tipos de elementos e onde eles podem ocorrer, além de estabelecer como eles se ajustam juntos no documento XML. Por exemplo, caso seja necessário um tipo de documento capaz de descrever listas as quais são compostas por itens, a parte relevante ao DTD deve conter algo parecido com a descrição abaixo:

```
<!ELEMENT Lista (Item)+>
<!ELEMENT Item (#PCDATA)>
```

A primeira linha deste trecho de um DTD define o tipo de elemento *Lista* contendo um ou mais *Item* (representado pelo sinal de mais +), já a segunda linha define o tipo de elemento *Item* contendo apenas caracteres do tipo texto (*Parsed Character DATA* ou *PCDATA*). Analisadores de validação lêem o DTD relacionado ao documento XML, identificam onde cada tipo de elemento pode aparecer na estrutura do documento XML e como cada elemento se relaciona com os outros, de tal forma que, aplicações que necessitem consultar estas informações (a maioria dos editores, máquinas de procura, navegadores, bancos de dados), possam ser executados corretamente.

O exemplo do DTD supracitado permitiria a criação de um documento XML contendo uma lista com a seguinte estrutura:

<Lista><Item>Batata</Item><Item>Feijão</Item><Item>Arroz</Item></Lista>

Uma DTD apresenta-se, portanto, precisamente como uma gramática livre de contexto para o documento. Considerando o exemplo abaixo:

<!ELEMENT biblio (livro)>*

<!ELEMENT livro (titulo,autor)>

<!ELEMENT titulo (#PCDATA)>

<!ELEMENT autor (sobrenome,prenome)>*

<!ELEMENT sobrenome (#PCDATA)>

<!ELEMENT prenome (#PCDATA)>

Aqui, *livro** é uma expressão regular, significando qualquer número de elementos *livro*. Outras expressões regulares são *livro+* (uma ou mais ocorrências de *livro*), *livro?* (zero ou uma ocorrência de *livro*), *livro,livro* (concatenação) e *((titulo,autor)* | *(autor,titulo))* (alternância). O exemplo ainda impõe que *<titulo>*, *<autor>* apareçam nesta ordem em um elemento *livro*.

As gramáticas podem ser recursivas, como na DTD abaixo que descreve árvores binárias:

<!ELEMENT nó (folha | (nó, folha))>
<!ELEMENT folha (#PCDATA)>

Um documento XML relacionado ao DTD acima pode ser o seguinte:

```
<nó>
    <nó>
        <nó><folha>elemento1</folha></nó>
        <nó><folha>elemento2</folha></nó>
    </nó>
    <nó>
        <nó><folha>elemento3</folha></nó>
        <nó><folha>elemento4</folha></nó>
    </nó>
</nó>
```

DTDs também podem ser utilizadas, até certo ponto, como esquemas (McGrath S., 1999). Por exemplo, considerando o seguinte esquema relacional: r1={a,b,c} e r2={c,d}, os seguintes valores para as relações:

R1	***A***	***B***	***C***
	a1	*b1*	*c1*
	a2	*b2*	*c2*

R2	***C***	***D***
	c2	*d2*
	c3	*d3*
	c4	*d4*

e a seguinte representação em XML:

```
<bd>
    <r1><a>a1</a><b>b1</b><c>c1</c></r1>
    <r1><a>a2</a><b>b2</b><c>c2</c></r1>
    <r2><c>c2</c><d>d2</d></r2>
    <r2><c>c3</c><d>d3</d></r2>
    <r2><c>c4</c><d>d4</d></r2>
</bd>
```

Nesta representação os nomes das relações *r1* e *r2* são tomados como marcas para as tuplas na relação. Uma DTD para tais dados é dada abaixo:

```
<!DOCTYPE bd [
   <!ELEMENT bd (r1*,r2*)>
   <!ELEMENT r1 (a,b,c)>
   <!ELEMENT r2 (c,d)>
   <!ELEMENT a (#PCDATA)>
   <!ELEMENT b (#PCDATA)>
   <!ELEMENT c (#PCDATA)>
   <!ELEMENT d (#PCDATA)>
]>
```

A DTD restringe de forma correta elementos *r1* a conter três componentes *a,b,c* bem como elementos *r2* a conter os componentes *c,d*. Porém, a DTD também força que os componentes *c* e *d* ocorram nesta ordem. É claro que a ordem *d, c* é igualmente aceitável para dados relacionais, portanto é necessário modificar a linha que especifica a ordem dos elementos *c, d* no DTD para:

```
<!ELEMENT r2 ((c,d) | (d,c))>
```

Isto se torna inconveniente a partir do momento em que sejam necessárias várias combinações de posicionamento dos elementos para suprir tal deficiência demonstrada no DTD, por exemplo, para o caso de *r1*, onde seria necessária a listagem de seis ordens possíveis para os elementos (permutação de 3 elementos = fatorial de 3). Outro problema que surge é que as tuplas *r1* ficam restritas a aparecer antes de elementos *r2*. A seguinte modificação no DTD supriria tal deficiência:

<!ELEMENT bd ((r1 | r2))>*

Por outro lado, um DTD torna fácil descrever componentes opcionais ou repetidos. Por exemplo, é possível modificar *r1* para:

<!ELEMENT r1 (a,b?,c+)>

e assim, a linguagem expressaria que exatamente um elemento *a* é solicitado, que *b* é opcional e que *c* é obrigatório podendo haver várias ocorrências. A partir daí o esquema definido pode ser armazenado em um arquivo, por exemplo, *esquema.dtd* e referenciado dentro de um arquivo XML como:

<!DOCTYPE bd SYSTEM "http://esquema.com/esquema.dtd">

Isto permitiria que vários sites na WEB compartilhassem o mesmo esquema, facilitando assim o intercâmbio de dados.

3.5 Extensible Style Language (XSL)

Assim como a definição dos tipos usados no documento é separada do documento em si, a definição da formatação dos elementos também é separada. A XML define uma linguagem de estilo para ser usada na formatação de seus documentos, denominada XSL (*XML Style Language* – Linguagem de Estilo da XML) ou XS (*XML Style*). Da mesma forma que a XML é um subconjunto do Padrão Internacional da SGML (*ISO 8879*), a XSL é um subconjunto simplificado da linguagem de estilo padrão Internacional conhecida como DSSSL (*Document Style Semantics and Specifications*

Language – linguagem de transformação e formatação para SGML - *ISO/IEC 10179*) conforme pode ser observado na figura abaixo (McGrath S., 1999).

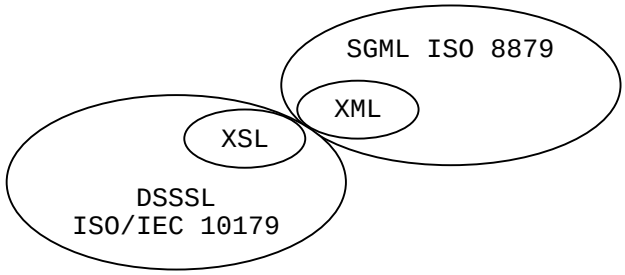


Figura 12 – Relação entre os padrões de estilo e documentação.

A principal função da XSL é permitir aos usuários escreverem transformações de XML para HTML, descrevendo desta forma a apresentação do conteúdo do documento XML.

Os documentos XML caracterizam-se pela separação de *dados* (o documento XML propriamente dito), da *formatação destes dados* (XSL) e da *estrutura dos dados* (DTD). Através da junção destes três itens determina-se a composição de um documento XML completo (figura 13) (McGrath S., 1999).

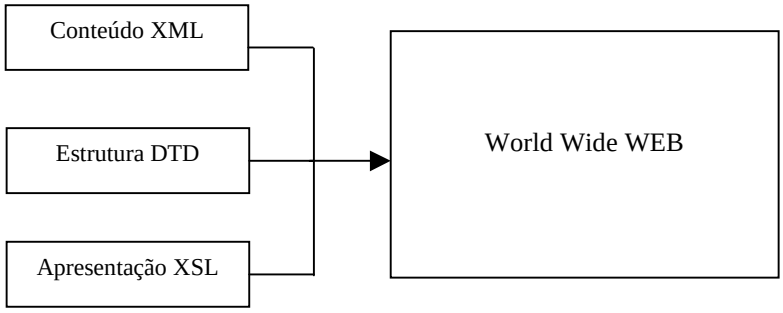


Figura 13 – Elementos XML integrados para exibição na WEB.

O modelo de dados para XSL é uma árvore ordenada. Um programa XSL é um conjunto de regras de modelo (em inglês, *template rules*). Cada regra consiste em um padrão e um modelo.

A XSL começa do elemento raiz e tenta aplicar um padrão a este nó. Se bem-sucedida, executa o modelo correspondente. A execução deste modelo normalmente instrui a XSL a produzir algum resultado em HTML e a aplicar os

modelos recursivamente nos nós secundários. Neste ponto o mesmo processo é repetido. Desta forma, um programa XSL age como uma função recursiva.

3.6 Regras de formação XML

Existem algumas convenções sugeridas pelos grandes fabricantes de software no sentido de se escrever código HTML bem-formado. Para a linguagem HTML elas são apenas uma sugestão. Já os arquivos XML devem seguir regras básicas para que se caracterize um documento válido e para que não gere mensagens de erro no programa processador do arquivo XML (por exemplo, no Internet Explorer versão 5.0 ou superior). Abaixo serão listadas as regras necessárias.

1ª.Regra: Todas as *tag's* precisam ser fechadas (tabela 3).

HTML	XML
<P>Parágrafo HTML <P>Outro parágrafo HTML	<P>Parágrafo XML </P> <P>Outro parágrafo XML</P>

Tabela 3 – Fechamento de Tag's XML.

Caso seja necessário não colocar nenhum elemento entre as *tag's* basta escrever uma *tag* e colocar uma barra no final (tabela 4).

Tabela 4 – Tag XML sem elemento interno.

2ª.Regra: As *tag's* não podem ser entrelaçadas, isto é, se você abriu duas *tag's*, aquela que foi aberta por último deve ser fechada primeiro (tabela 5).

HTML	XML
<B>Olá<I>Meus</B>Irmãos</I>	<B>Olá</B><I><B>Meus</B>Irmãos</I>

Tabela 5 – Posicionamento de Tag's XML.

3ª.Regra: Manter consistência quanto a caixa (alta ou baixa) de uma *tag*. Não é necessário escrever sempre em maiúscula, mas se foi iniciado uma *tag* com letras maiúsculas, é interessante encerrá-la com maiúsculas (tabela 6).

HTML	XML
<B><I>Olá Meus Irmãos</I>	<B><I>Olá Meus Irmãos</I></B>

Tabela 6 – Tag's XML maiúsculas/minúsculas.

4ª.Regra: Todos os atributos devem ser colocados entre aspas (ver tabela 7).

HTML	XML
<IMG src=sample.gif width=10 height=20>	<IMG src="sample.gif" width="10" height="20"/>

Tabela 7 – Atributos definidos em XML.

5ª.Regra: Utilização de um único elemento raiz (ou *root*) (ver tabela 8).

HTML	XML
<TITLE> Marcações desleixadas </TITLE> <BODY> Este arquivo não está bem formado </BODY>	**<HTML>** <TITLE> Marcações desleixadas </TITLE> <BODY> Este arquivo não está bem formado </BODY> **</HTML>**

Tabela 8 – Utilização correta do elemento root em XML.

6ª.Regra: A linguagem HTML define algumas entidades que são usadas para a internacionalização da linguagem. Um exemplo do uso destas entidades é o uso de caracteres acentuados no português. As entidades são delimitadas por um & (e comercial) e um ; (ponto-e-vírgula) (tabela 9).

< (<) > (>) & (&) &quote; (") ' (´)

Tabela 9 – Entidades usadas em XML.

7ª.Regra: *Scripts* em HTML podem conter caracteres especiais como < e &. Por este motivo é aconselhável colocar estes blocos de código em seções *CDATA* para que elas não sejam processadas. As seções *CDATA* servem justamente para isto: impedir o processamento do código em seu interior (tabela 10).

HTML	XML

<SCRIPT>
function less_than_seven(n){
return n<7;
}
</SCRIPT>
	<SCRIPT><![CDATA[
	function less_than_seven(n){
	return n<7;
	}
	]]></SCRIPT>

Tabela 10 – Scripts HTML Versus Scripts XML.

3.7 O processamento de documentos XML

Um documento XML pode ser criado, lido, atualizado ou manipulado através de um programa processador conhecido como *parser* XML. O *parser* XML lê o documento XML e verifica se este está de acordo com as regras de formação apresentadas anteriormente. No caso em que o documento XML possua uma DTD, o *parser* também verifica se o documento é válido conforme as especificações de sua DTD.

A fim de manipular um documento XML, o *parser* XML carrega inicialmente este documento na memória. Após carregado, as informações contidas no documento XML podem ser recuperadas e manipuladas através do acesso do *parser* a um determinado modelo de representação de dados. Um dos modelos de representação de dados mais utilizados atualmente é o DOM (*Document Object Model*), o qual representa uma visão do documento XML em forma de árvore de diretórios.

A Microsoft desenvolveu o *parser* MSXML e acoplou este ao seu *browser* de navegação Internet Explorer a partir da versão 5.0. O modelo DOM é uma API independente de plataforma e linguagem, permitindo que programas acessem e alterem o conteúdo, a estrutura e o estilo de um documento. A especificação do DOM é separada e realizada em diferentes níveis, conforme descrito a seguir:

- Nível 0 – apenas define as funcionalidades equivalentes àquelas encontradas nos *browsers* de navegação *Netscape Navigator* 3.0 e Microsoft Internet Explorer 3.0. Não é uma especificação do W3C.

- Nível 1 - concentra-se em modelos de documentos HTML e XML. Contém as funcionalidades para navegação e manipulação de documentos. Tornou-se uma recomendação do W3C em outubro de 1998.

- Nível 2 - adiciona um modelo de folha de estilos ao DOM nível 1. Além disso, define funcionalidades para manipular a informação de estilo associada ao documento. Também define um modelo de evento. Tornou-se recomendação do W3C em novembro de 2000.

- Nível 2 HTML - especifica uma API para manipular a estrutura e o conteúdo de um documento HTML. Esta especificação ainda está em andamento.

- Nível 2 *Views* - especifica uma API para acessar e atualizar dinamicamente a visão de um documento, onde uma visão é considerada uma representação alternativa de um documento.

- Nível 2 *Style* - especifica uma API para acessar e atualizar dinamicamente o conteúdo de folhas de estilo.

- Nível 3 - especifica modelos de conteúdo (DTD e esquemas XML) e validação de documentos. Também especifica o carregamento e armazenamento de documentos, visões de documentos e formatação de documentos. Também encontra-se em andamento.

- Nível *Core* - especifica uma API para acessar e atualizar o conteúdo e a estrutura de documentos.

Este trabalho concentra-se nos níveis 1 e 2 da especificação do DOM. Alguns detalhes mais específicos a respeito da utilização do DOM são abordados através da implementação com uso do *Framework .NET* no capítulo 5.

3.8 Conclusão

Neste capítulo foi possível definir uma visão abrangente das principais características da tecnologia essencial sobre a qual o trabalho é desenvolvido. É possível determinar uma pirâmide de tecnologias sobre a qual a pesquisa é realizada:

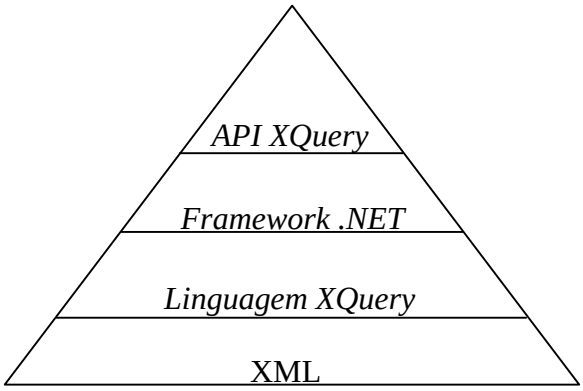


Figura 14 – Pirâmide de tecnologias.

O próximo capítulo traz uma abordagem geral das linguagens de consulta a XML e em especial à linguagem de consulta *XQuery*. Uma analogia às funcionalidades das linguagens de consulta a banco de dados é realizada a fim de comparar as principais características, já que o estudo das linguagens de consulta a banco de dados encontra-se num nível de profundidade bem mais abrangente do que as linguagens de consulta a XML, as quais estão apenas no seu início.

CAPÍTULO IV

LINGUAGENS DE CONSULTA

4.1 Introdução

Este capítulo aborda a importância das linguagens de consulta ao padrão XML, que a cada ano ganha mais e mais importância no ambiente *WEB* e se introduz como tecnologia fundamental nos bancos de dados; faz ainda um estudo das principais características das linguagens mais abordadas nos últimos tempos demonstrando um comparativo de funcionalidades e descreve a linguagem que está em estudo como proposta a ser adotada como um padrão pelo W3C.

A rápida evolução da tecnologia XML de um simples formato para troca de dados para uma sintaxe universal de codificação de informações levou à necessidade de novas linguagens de consulta expressas especificamente para as características da XML. Tais linguagens devem ser capazes de, além de extrair informações de documentos XML, aplicar transformações e operações de reestruturação com base em uma semântica bem definida. Além disso, consultas XML devem ser simples de serem escritas e compreendidas, de tal forma que usuários não-técnicos também tenham o acesso desejado às grandes bases de informação apoiadas pelo mundo da WEB (W3Cd).

A importante aplicação da linguagem XML conhecida como *EDI* (*Interchange of Eletronic Data* – intercâmbio de dados eletronicamente) requer ferramentas que suportem as seguintes tarefas:

- extração de dados de um documento XML;
- conversão de dados (relacionais ou orientados a objeto para o formato XML);
- transformação de dados de um DTD para um outro DTD; e
- integração de múltiplas origens de dados XML.

A extração, conversão, transformação e integração de dados são problemas bem conhecidos na área de banco de dados. As soluções para tais problemas contam com o apoio das linguagens de consulta, ou relacionais (SQL) ou orientadas a objeto (OQL).

Como visto no capítulo anterior, diferentemente dos dados relacionais e orientados a objeto, os dados XML são semi-estruturados, ou seja, eles podem ter uma estrutura irregular e extensível e seus atributos e esquemas são armazenados juntamente com os dados. Neste sentido, o estudo das linguagens de consulta se faz necessário para auxiliar as tarefas acima descritas em relação a XML.

Como destacado por diversos pesquisadores, por exemplo, Maier (1998) e Quass (1998), linguagens de consulta XML são mais do que simples variantes dos já existentes e bem conhecidos paradigmas de processamento de consulta tais como as linguagens de consulta relacionais e orientadas a objeto.

O W3C (W3Cd), em um comitê direcionado para a especificação de linguagem de consulta a XML, resume os requisitos necessários às linguagens de consulta XML na seguinte lista de casos de uso desejáveis:

- *Documentos legíveis por humanos*: as consultas devem poder ser executadas sobre documentos e coleções de documentos, tais como manuais técnicos, para obter documentos individuais, para gerar tabelas de informação, para pesquisar informação em estruturas encontradas num documento ou para gerar novos documentos como resultado de uma consulta;

- *Documentos orientados a dados*: as consultas devem mapear para a representação XML dados de origem de bancos de dados, ou qualquer origem de dados tradicional a fim de extrair dados destas origens, transformar dados em novos documentos XML ou até mesmo integrar dados de múltiplas origens de dados heterogêneas;

- *Documentos de modelos mistos*: ambas as consultas orientadas a documento e orientadas a dados em documentos com dados embutidos, tais como catálogos, registros médicos, registros de empregados ou documentos de análise de negócios devem ser suportadas;

- *Dados administrativos*: as consultas em arquivos de configuração, perfis de usuário ou *logs* administrativos representados em XML devem ser possíveis;

- *Filtragem de fluxo de dados*: as consultas devem mapear fluxo de dados XML para processos, por exemplo, *logs* de mensagens de e-mail, pacotes de rede, dados de bolsas de valores, revistas on-line, *EDI* (intercâmbio de dados eletrônico), ou informações sobre o tempo, a fim de rotear mensagens representadas em XML, extrair dados de fluxos XML ou transformar dados em fluxos de dados XML;

- *Modelo de Objetos de Documento (DOM – Document Object Model)*: as consultas devem ser possíveis em estruturas DOM a fim de retornar conjuntos de nós que satisfaçam um critério especificado;

- *Repositórios XML nativos e servidores WEB*: as consultas devem ser possíveis em coleções de documentos gerenciados por repositórios de dados XML nativos ou servidores WEB;

- *Catálogos de pesquisa*: as consultas devem ser possíveis de serem usadas em catálogos de pesquisa que descrevem servidores de documentos, tipos de documentos, esquemas XML ou documentos propriamente ditos. Tais catálogos devem ser combinados para sustentar pesquisas entre múltiplos servidores;

- *Ambientes sintáticos múltiplos*: as consultas devem ser possíveis em muitos ambientes (por exemplo: em uma URL, em uma página XML, em páginas JSP ou ASP, etc).

É claro que as pesquisas em linguagens que suportem todos estes requisitos estão apenas no início em comparação aos estudos já bem definidos para linguagens de consulta a bancos de dados, porém muitas linguagens para consulta a dados XML já foram propostas na literatura nos últimos anos.

Alguns bons exemplos que podem ser citados são: *XQL* (Robie J., 1998), *XML-QL* (Deutsch et al., 1998), *XSL* (W3Ca), *Lorel* (Abiteboul et al., 1997), *YaTL* (Cluet et al., 1999), *XQuery* (DeRose S. J., 1998), *Quilt* (Chamberlin et al., 2000) e *XMAS* (Ludaescher et al., 2000). Recentemente a linguagem *XQuery* foi proposta pelo W3C para padronização.

Estas linguagens representam tanto uma evolução das linguagens de consulta a bancos de dados (relacionais, orientadas a objeto ou semi-estruturadas) como uma evolução às técnicas de processamento de documentos.

Devido às diferentes origens, as linguagens de consulta XML propostas possuem distinções consideráveis em termos de suas sintaxes. Por exemplo, a linguagem Lorel tem uma sintaxe baseada em OQL (*Object Query Language*; nesta linguagem uma consulta é representada por uma expressão) enquanto a XML-QL é baseada numa sintaxe XML textual; já a XSL e a XQL, as quais foram propostas pela comunidade de processamento de documentos, exploram expressões similares às expressões de caminho em diretórios. A *XQuery* é uma linguagem bastante flexível, originada de uma mistura de ambos os tipos de linguagem.

Não obstante suas origens independentes, as linguagens de consulta XML propostas são comparáveis até certo ponto em relação ao poder de expressão, pois todas elas oferecem ferramentas para:

- especificar diferentes tipos de condições sobre os elementos a serem obtidos;
- selecionar que partes da informação obtida devem ser mantidas no resultado da consulta;
- combinar diferentes partes de documentos;
- e por fim, reestruturar o resultado da consulta, por exemplo, classificando e/ou agrupando elementos.

4.2 A linguagem XML-QL

4.2.1 Introdução

A linguagem XML-QL combina a sintaxe da linguagem XML com técnicas de linguagens de consulta a dados semi-estruturados tais como expressões de caminho.

Expressões de caminho são usadas para extrair informações dos dados XML de entrada; variáveis são expressas a fim de vincular esses dados e modelos (*templates*) mostram como os dados XML de saída devem ser construídos. Quando seu uso é restrito a dados semelhantes aos dados relacionais a XML-QL torna-se tão expressiva quanto o cálculo relacional ou a álgebra relacional, ou seja, a XML-QL é completa em termos relacionais (Deutsch et al., 1998).

Esta linguagem de consulta a documentos XML foi projetada pela *AT&T Labs* e é capaz de realizar as principais tarefas para este fim, tais como: extração de dados de documentos extensos, conversão de dados entre bancos de dados e documentos XML, mapeamento de dados XML entre diferentes *DTDs* e integração entre dados XML de múltiplas fontes, através da realização de junções e outras operações encontradas em SQL. Seu protótipo pode ser encontrado em http://www.research.att.com/sw/tools/xmlql como parte do projeto Strudel.

XML-QL é baseada em uma sintaxe do tipo *where/construct* ao invéz da familiar *select/from/where* de SQL ou OQL. Em um estilo UnQL, a cláusula *construct* corresponde a *select*, enquanto que *where* combina as partes *from* e *where* da consulta, isto é, as faixas de variáveis, bem como alguma filtragem.

A sintaxe básica da linguagem combina, portanto, elementos da sintaxe de XML com elementos de sintaxe tradicionais de linguagens de consulta de sistemas de banco de dados. Sua forma geral é:

where <argumentos de seleção>
construct <resultado>

O trecho "*argumentos de seleção*" representa uma construção, no formato XML, com os dados que devem ser encontrados e variáveis (iniciadas por "*$*"). Nos argumentos da cláusula *where* deve aparecer, em pelo menos um deles, a palavra "*in*", indicando o documento que deve ser consultado. "*Resultado*" é a especificação do documento XML que deve ser construído como saída da consulta.

Um exemplo simples é apresentado abaixo, onde a consulta recupera os títulos e autores dos livros publicados pela editora *Prentice-Hall, Inc*:

```
where
        <livro>
            <editora><nome>Prentice-Hall, Inc</nome></editora>
            <titulo> $T </titulo>
            <autor> $A </autor>
        </livro> in “www.biblio.com/bib.xml”
construct <titulo> $T </titulo>
            <autor> $A </autor>
```

Neste exemplo, *$T* e *$A* são variáveis, enquanto a estrutura abaixo:

```
        <livro>
            <editora><nome>Prentice-Hall, Inc</nome></editora>
            <titulo> $T </titulo>
            <autor> $A </autor>
        </livro>
```

é um padrão (*template*). O processador combina o padrão (*template*) de todas as maneiras possíveis com os dados e instancia as variáveis $T e $A. Para cada instância, produz-se o resultado $A.

4.2.2 Consultas Aninhadas – elementos opcionais

XML-QL lida facilmente com partes opcionais em consultas aninhadas. Por exemplo, supondo a tag *<preço>* em *<livro>* como opcional. Supondo ainda a necessidade de uma consulta de todos os títulos dos livros e, quando disponíveis, seus respectivos preços. Considerando a consulta a seguir a fim de suprir essa necessidade:

```
where
        <livro>
                <titulo> $T </titulo>
                <preço> $P </preço>
        </livro> in “www.biblio.com/bib.xml”
construct
        <resultado>
                <titulo_livro> $T </titulo_livro>
                <preço_livro> $P </preço_livro>
</resultado>
```

Esta consulta apresenta um problema: o padrão define que a tag <*preço*> esteja presente; como consequência disto os livros sem preço não serão apresentados. A fim de contornar este problema é possível usar o poder de expressão da XML-QL, elaborando a seguinte consulta:

```
where
        $B in “www.biblio.com/bib.xml”,
        <titulo> $T </titulo> in $B
construct
        <resultado>
                <titulo_livro> $T </titulo_livro>
                        where <preço> $P </preço> in $B
                      construct <preço_livro> $P </preço_livro>
        </resultado>
```

À direita de *in* é possível ter tanto um URL (indicando um documento em XML) ou uma variável (podendo indicar um fragmento de um documento XML).A segunda consulta *where...construct* é chamada de consulta aninhada. Para cada instância do padrão <*preço*> *$P* </*preço*> é gerado um elemento da forma <*preço_livro*> *$P* </*preço_livro*>. Se nenhum *preço* é encontrado, então nenhum *preço_livro* é gerado, mas o *título* do livro ainda é mantido como parte do resultado.

4.2.3 Consultas Aninhadas - agrupamento

Uma distinção básica entre dados relacionais e XML é com relação ao aninhamento e agrupamento de dados. Por exemplo, considerando que um banco de dados bibliográfico armazenado em um documento XML tem um elemento para cada livro, com todos os autores agrupados neste elemento. Suponha a necessidade de se encontrar todos os autores, e para cada um deles encontrar todos os títulos que este autor publicou. Resumindo, existe a necessidade de reagrupar os dados. Este tipo de operação pode ser feito em XML-QL também através de consultas aninhadas, também denominadas subconsultas, como pode ser visto no exemplo abaixo:

```
where
        <livro><autor> $A </autor></livro>
                in “www.biblio.com/bib.xml”,
construct
```

```
<resultado>
        <autor> $A </autor>
        where
            <livro>
                <autor> $A </autor>
                <titulo> $T </titulo>
            </livro> in “www.biblio.com/bib.xml”,
construct <titulo> $T </titulo>
</resultado>
```

4.2.4 Instanciação

Variáveis em XML-QL são instanciadas em nós no modelo de dados semi-estruturado. Em termos de XML, isto significa que variáveis são instanciadas no conteúdo do elemento e não no elemento em si. A XML-QL possui uma simplificação sintática que permite a instanciação no elemento propriamente dito. Por exemplo, considerando a consulta abaixo:

```
where
    <livro>
        <editora><nome> Prentice Hall </nome></editora>
    </livro>
    element_as $L in “www.biblio.com/bib.xml”
construct <resultado> $L </resultado>
```

A variável *$L* é instanciada no elemento *<livro>...</livro>* através do construtor *element_as*. Isto pode ser considerado como uma simplificação sintática, pois o processador de consultas XML-QL traduzirá a consulta para:

```
where
    <livro> $L </livro> in “www.biblio.com/bib.xml”
    <editora><nome> Prentice Hall </nome></editora> in $L
construct <resultado><livro> $L </livro></resultado>
```

De forma análoga o construtor content_as instancia uma variável no conteúdo de um elemento. Por exemplo:

```
where
    <livro>
        <editora><nome> Prentice Hall </nome></editora>
    </livro>
    content_as $L in “www.biblio.com/bib.xml”
```

```
construct <resultado> $L </resultado>
```

é uma simplificação para:

```
where
        <livro> $L </livro> in "www.biblio.com/bib.xml",
        <editora><nome> Prentice Hall </nome></editora> in $L
construct <resultado> $L </resultado>
```

4.2.5 Consulta de Atributos

A consulta em atributos em XML-QL é realizada de forma direta. Por exemplo, a consulta a seguir encontra todos os títulos de livros em português:

```
where
        <livro língua = "Português">
              <titulo></titulo> element_as $T
        </livro> in "www.biblio.com/bib.xml"
construct <resultado> $T </resultado>
```

É importante notar que o valor de um atributo da entrada se torna o valor de um elemento na saída.

4.2.6 Operação de Junção

As junções na linguagem de consulta XML-QL podem ser expressas através da utilização de uma mesma variável em duas combinações. Supondo a necessidade de uma consulta que recupere todos os autores que publicaram pelo menos dois livros:

```
where
      <livro><autor> $A </autor></livro>
            content_as $B1 in "www.biblio.com/bib.xml",
      <livro><autor> $A </autor></livro>
            content_as $B2 in "www.biblio.com/bib.xml",
      B1 != B2
construct <resultado> $A </resultado>
```

4.2.7 Herança, Superclasse e Subclasse

Documentos XML podem usar marcas distintas para se referir a variantes dos mesmos conceitos. Por exemplo, considerando o caso do exemplo anteriormente citado para o arquivo XML contendo o banco de dados bibliográfico, tanto a *tag* <autor> quanto a *tag* <editor> indicam elementos de pessoa. Num banco de dados orientado a objeto é possível modelar este problema como uma superclasse *pessoa* com duas subclasses, *autor* e *editor*. Porém a XML não possui uma forma para representar estes conceitos. Para contornar este problema, a linguagem XML-QL possibilita o uso de variáveis de marcação. Por exemplo, supondo a necessidade de encontrar todas as publicações editadas em 1995 nas quais João seja tanto um autor quanto um editor. É possível realizar esta consulta da seguinte forma:

```
where
    <$P><titulo> $T </titulo>
        <ano>1995</ano>
        <$E>João</$E>
    </$P> in “www.biblio.com/bib.xml”,
            $E in {autor,editor}
construct
    <$P ><titulo> $T </titulo>
        <$E > João </$E>
    </$P>
```

Duas variáveis de marcação são usadas neste exemplo: a variável *$P* que é usada para instanciar a tag de nível mais alto (que pode ser livro, ou artigo, etc) e *$E* que é usada para instanciar somente as tags *autor* e *editor*.

4.3 A linguagem Lorel

A linguagem *Lorel* (Abiteboul et al., 1997) foi originalmente projetada para realizar consultas em dados semi-estruturados e foi posteriormente estendida para a consulta a dados XML. O seu protótipo pode ser encontrado em http://www-db.stanford.edu/lore.

Esta linguagem apresenta-se no estilo da linguagem SQL/OQL, bem amigável, apresentando um mecanismo que permite poderosas expressões de caminho, extremamente útil quando a estrutura do documento não é bem conhecida.

A seguir são descritas algumas características fundamentais da linguagem *Lorel* para o suporte a XML (Bonifati e Ceri, 2000), estas características, em grande parte, são comparadas às características já bem definidas na linguagem SQL a fim de facilitar a compreensão.

4.3.1 Modelo de dados específico

Os projetores da linguagem Lorel desenvolveram um modelo de dados, onde um elemento XML é um par <*e-id, valor*>, onde *e-id* é um identificador único para cada elemento, e *valor* pode ser uma string atômica ou um valor complexo contendo uma tag valorada, seguida por uma lista (podendo esta lista ser vazia) de pares <*nome_atributo, valor_atômico*> (representando atributos XML), seguidos por uma lista (podendo esta lista ser vazia) ordenada de pares <*e-id, valor*> denominados sub-elementos de crosslink (representando elementos IDREF em XML), seguida por uma lista (podendo esta lista ser vazia) de pares <*e-id, valor*> denominados sub-elementos normais.

4.3.2 Abstrações de consultas básicas

Seleção - uma consulta em Lorel é estruturada da seguinte forma:

'select' {expressão_select}
['from' {expressão_from}]
['where' {expressão_where}]

As expressões select, from e where, assim como em OQL, podem conter outras consultas.

Junção - condições de junção são totalmente suportadas pela linguagem Lorel, tanto num mesmo documento como em vários documentos. Elas são escritas numa

forma semelhante à SQL, por meio da especificação explícita das variáveis envolvidas nas junções.

Semânticas do resultado da consulta - o resultado de uma consulta pode ser definido através do conteúdo atual da base de dados ou através de um novo documento, o qual pode ser consultado e possivelmente atualizado independentemente. Na linguagem Lorel, o resultado de uma consulta é um conjunto de identificadores de objetos apontados por um novo elemento. Portanto, os objetos selecionados da base de dados em determinado instante, são exatamente os objetos da base de dados naquele instante, sendo que acessos subsequentes ao resultado da consulta podem resultar em documentos distintos. Na linguagem Lorel também é possível definir *views* (cláusula *with*), sendo que em tal caso a consulta retorna um documento com todos os nodos especificados pela cláusula *with*. No caso das outras linguagens (XML-QL, XML-GL, XSL, XQL e XQuery) o resultado da consulta é retornado em um novo documento, onde o conteúdo é independente da base de dados.

4.3.3 Expressões de caminho

Ao consultar dados semi-estruturados é conveniente que se use uma forma de consulta de navegação baseada em expressões de caminho. A forma mais poderosa de expressão de caminho não precisa listar todos os elementos do caminho, portanto ela usa caracteres coringa e expressões regulares: esta forma de expressão de caminho é conhecida como expressão de caminho parcialmente especificada. A linguagem Lorel implementa este tipo de expressão de caminho.

As expressões de caminho na linguagem Lorel são poderosas e flexíveis: elas admitem vários caracteres coringa semelhantes ao sistema Unix. Cada expressão de caminho deve ter um contexto (o elemento root do documento).

4.3.4 Quantificação, Negação e Redução

A linguagem Lorel possui a característica de quantificação existencial. A quantificação existencial diz que um predicado existencial sobre um conjunto de instâncias é satisfeito se pelo menos uma das instâncias satisfaz o predicado.

A linguagem ainda possui a quantificação universal. A quantificação universal é regida pela seguinte afirmação: um predicado universal sobre um conjunto de instâncias é satisfeito se todas as instâncias satisfazem o predicado. Em Lorel uma variável pode ser quantificada universalmente com o predicado *for all* (análogo à SQL).

Lorel possui ainda a negação de um predicado - a negação de um predicado sobre um conjunto de instâncias é satisfeita se nenhuma das instâncias satisfaz o predicado. O predicado é negado com a palavra-chave *not*.

Uma das deficiências da linguagem Lorel é a falta de suporte à redução. A redução se resume na seguinte situação: dado um documento e uma consulta sobre este documento, a redução elimina do resultado os elementos especificados na parte de seleção da consulta que satisfazem a condição da consulta.

4.3.5 Abstrações de restruturação

Um novo elemento XML pode ser criado através de um mecanismo de construção de consulta. Na linguagem Lorel um novo elemento é criado através da função *xml()* com três parâmetros: o tipo (não obrigatório), o rótulo (não obrigatório) e o valor(es).

Elementos do resultado de uma consulta podem ser agregados ou reorganizados por meio de funções especiais, tal como *group by*. Na linguagem Lorel, a cláusula *group by* é herdada da linguagem OQL.

4.3.6 Agregação, Aninhamento e Operações de Conjunto

Funções agregadas computam um valor escalar fora do multi-conjunto de valores da origem de dados. Algumas agregações clássicas da linguagem SQL que são destacadas: *min* (mínimo), *max* (máximo), *sum* (soma), *count* (quantidade) e *avg* (média). As funções de agregação na linguagem Lorel estão presentes e completamente implementadas.

A linguagem Lorel também suporta o aninhamento de consultas, tal como em SQL. A linguagem ainda suporta as operações de conjunto *união*, *diferença* e *interseção*.

4.3.7 Gerenciamento da Ordenação

A ordenação do resultado de uma consulta consiste em ordenar as instâncias dos elementos de acordo com os valores, ordem esta que pode vir a ser ascendente ou descendente, tal como a execução da cláusula *order by* da linguagem SQL. Assim como na SQL , a linguagem Lorel pode ordenar o resultado de uma consulta através da cláusula *order by*.

É possível ainda preservar a ordem original dos elementos contidos na origem de dados, de tal forma que o resultado da consulta possua a mesma ordem da origem de dados. A linguagem Lorel possui a cláusula *order by document order*, a qual preserva a ordem original dos elementos.

4.3.8 Tipos e Extensões

Uma das características que se destaca na linguagem Lorel é a de suporte a tipos de dados abstratos, a qual preocupa-se com a necessidade de embutir numa linguagem de consulta XML operações especializadas, por exemplo operações para

selecionar tipos diferentes de conteúdo multimídia. A linguagem Lorel suporta áudio, vídeo, imagens e tipos de dados especializados tais como *jpeg*, *gif* e *ps*.

Outra ferramenta que se destaca na linguagem Lorel é a coerção de tipos. A coerção de tipos permite a habilidade da linguagem de comparar valores representados através de construtores de tipo distintos (por exemplo: valores escalares, conjuntos simples, listas com apenas um elemento). Devido à natureza dos dados semi-estruturados, a coerção de tipos deve ser muito mais flexível numa linguagem de consulta XML do que numa linguagem de consulta a banco de dados.

Em suma, as principais ferramentas e características da linguagem Lorel são:

- Um modelo de dados XML na linguagem Lorel pode ser representado tanto por uma árvore literal onde os atributos IDREFs são strings de texto como por um grafo semântico onde os atributos IDREFs são arestas. A linguagem suporta ambos os modos de representação, permitindo à aplicação escolher uma delas. Esta característica é importante ao levar em consideração o fato de que a aplicação está ou não interessada na apresentação da informação pelos atributos IDREFs;

- A linguagem fornece construtores a fim de "trabalhar" os dados e retornar resultados estruturados. Um exemplo é a cláusula *with* utilizada em conjunto com a estrutura *select-from-where*. A cláusula *with* faz com que o resultado da consulta replique todos os dados selecionados pela cláusula *select* junto com todos os dados alcançáveis pelas expressões de caminho contidas na cláusula *with*;

- A Lorel suporta junções sobre documentos XML pertencentes a origens distintas;

- As expressões de caminho desta linguagem podem incluir caracteres coringa ou operadores de expressões regulares;

- A linguagem fornece a cláusula *order-by*.

4.4 A linguagem XML-GL

4.4.1 Introdução

A linguagem de consulta XML-GL se constitui numa linguagem gráfica para informações armazenadas no formato XML. Segundo Comai (Comai et al., 2001) tal linguagem foi projetada com a característica exclusiva, ao menos na época de sua apresentação, de uma sintaxe baseada em grafos a fim de expressar uma grande variedade de consultas XML, compreendendo desde seleções simples até transformações expressivas dos dados envolvendo agrupamento, agregação e cálculos aritméticos.

Portanto a linguagem XML-GL é constituída de elementos gráficos ao invés de elementos textuais. A idéia central aqui é explorar a representação baseada em grafos dos dados contidos no formato XML a fim de expressar consultas e reestruturação da informação. Esta idéia e a sintaxe da XML-GL, segundo Comai (Comai et al., 2001), é motivada pelas seguintes considerações:

- Os documentos XML possuem uma estrutura hierárquica um tanto quanto intuitiva, podendo ser representados naturalmente como uma *árvore*. Quando referências entre elementos também são consideradas, os grafos tornam-se a representação mais óbvia;

- Árvore e representação baseada em grafo normalmente são representações familiares aos usuários, haja vista o sucesso de

editores visuais para documentos XML tal como o XmlSpy (XMLSpy 2000);

- O processo de selecionar alguma informação dentro de um ou mais documentos XML pode ser visualmente desenhado tal como a tarefa de localizar um sub-grafo dentro de um grande grafo. O usuário poderia facilmente construir uma consulta especificando seu *grafo de interesse*, possivelmente copiando e colando os nodos e arcos diretamente de uma representação gráfica do documento, ou até mesmo de seu DTD;

- O processo de criação de um novo documento baseado em um ou mais documentos XML existentes pode ser visualmente representado como a construção de um novo grafo, correspondendo ao DTD do documento desejado, seguido pela conexão de tal grafo ao grafo usado para localizar a informação de interesse, a fim de expressar o fluxo da informação do documento consultado para o novo documento.

4.4.2 O Surgimento da XML-GL

A linguagem XML-GL é o resultado de várias pesquisas em linguagens lógicas de consulta baseadas em grafos. Segundo Comai (Comai et al., 2001) esta linguagem é o primeiro exemplo de uma linguagem visual que implementa a completa complexidade de consultas a dados XML.

A linguagem XML-GL nasceu da linguagem G-Log (Graph-Based Query Language - Paradaens, 1995), uma linguagem gráfica baseada em lógica, a qual utilizou uma notação semelhante à linguagem Good (Graph-Oriented Object Database) como ponto de partida para representar e consultar objetos complexos. A linguagem Good propôs uma noção abstrata simplificada para descrever modelos alternativos para banco

de dados: nodos representam objetos e arcos representam relacionamentos (sem distinções entre generalização, especialização, agregação, etc.).

A linguagem G-Log evoluiu inicialmente para a linguagem WG-Log (Graph-Based Query Language for WEB), uma linguagem de consulta direcionada para a WEB e dados semi-estruturados e posteriormente foi base para a definição da sintaxe e semântica da linguagem XML-GL.

4.4.3 Composição da XML-GL

Comai (Comai et al., 2001) apresenta, de uma forma geral, duas partes que compõem uma consulta XML-GL:

- um lado esquerdo da consulta, onde um ou mais grafos expressam a seleção da informação de interesse;

- um lado direito da consulta, onde um ou mais grafos expressam o conteúdo desejado no resultado, sendo que este lado direito está interconectado com o lado esquerdo (ou por arcos explícitos no grafo ou por nomes de nodos iguais) a fim de expressar quais os elementos obtidos no lado esquerdo devem ser usados para construir o documento de saída.

Um caso típico proposto por Maier (Maier D., 1998) para medir o poder de expressão das linguagens de consulta XML pode ser considerado para exemplificação. Considerando um documento XML contendo os seguintes elementos:

<!ELEMENT fabricante (fab-nome,ano,modelo+)>
<!ELEMENT fab-nome #PCDATA>
<!ELEMENT ano #PCDATA>
<!ELEMENT modelo (mod-nome,avaliacao-frontal,avaliacao-lateral,nota)>
<!ELEMENT mod-nome #PCDATA>
<!ELEMENT avaliacao-frontal #PCDATA>
<!ELEMENT avaliacao-lateral #PCDATA>
<!ELEMENT nota #PCDATA>

Esta estrutura de elementos XML acima descrita refere-se a documentos XML que determinam o resultado de testes de segurança em automóveis em geral. Considerando agora a seguinte consulta: *Para os elementos <fabricante> é preciso eliminar os sub-elementos <modelo> onde <nota> é maior que 6. É preciso também ocultar os elementos <avaliacao-frontal> e <avaliaçao-lateral> dos modelos resultantes. Ou seja, para os sub-elementos <modelo> com <nota> menor ou igual a 6, a consulta deve manter os sub-elementos <modelo>, porém com apenas um sub-conjunto de seus elementos.* A figura abaixo (Comai et al., 2001) mostra como ficaria esta consulta na linguagem XML-GL:

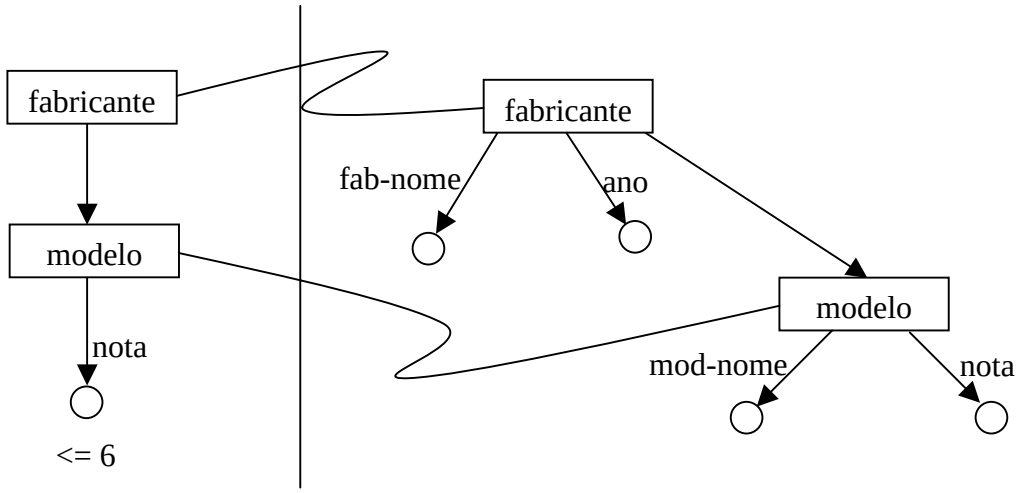


Figura 15 – Um exemplo inicial de XML-GL.

Ceri (Ceri et al., 1999) fornece uma visão mais específica das partes que compõem uma consulta XML-GL. Segundo ele, uma consulta XML-GL é composta por quatro partes:

1. A parte *extract* identifica o escopo da consulta, identificando tanto os documentos de origem quanto os elementos de origem dentro destes documentos. Fazendo uma analogia à linguagem SQL, esta parte pode ser vista como a parte correspondente à cláusula *FROM*, a qual estabelece as relações entre as origens da consulta;

2. A parte *match* (opcional) que especifica condições lógicas que os elementos de origem devem satisfazer a fim de constituírem o resultado da consulta. Em relação à SQL, esta parte de condição corresponde à cláusula *WHERE*, a qual escolhe as tuplas da origem que devem estar no resultado da consulta;

3. A parte *clip* especifica os sub-elementos dos elementos extraídos pela parte *extract* e que satisfazem a parte *match* a serem retidos no resultado da consulta. Continuando com a analogia à SQL, a parte *clip* corresponde à cláusula *SELECT*, a qual permite que o usuário da consulta defina quais as colunas da tuplas resultantes devem ser mantidas na saída final da consulta;

4. A parte *construct* (opcional) especifica os novos elementos a serem incluídos no documento resultante da consulta e seus relacionamentos com os elementos extraídos. Em relação à SQL, a parte *construct* pode ser vista como uma extensão da estrutura *CREATE VIEW*, a qual permite que o usuário projete uma nova relação para o resultado de uma consulta.

Desta forma, a consulta XML-GL é constituída de um par de modelos de dados gráfico XML, dispostos lado a lado e separados por uma linha vertical, conforme pode ser observado na figura anterior. O lado esquerdo representa as partes *extract* e *match* e o lado direito corresponde às partes *clip* e *construct*.

4.4.4 Contribuições da XML-GL

Segundo as pesquisas de Comai (Comai et al., 2001), através da XML-GL é possível expressar visualmente consultas que:

- selecionam partes de um ou mais documentos de entrada, com base em condições existenciais e predicados de comparação;

- expressam condições de junção sobre elementos de um ou mais documentos de entrada;

- usam a informação selecionada para construir novos documentos;

- criam novos elementos ou relacionamentos entre elementos, a fim de serem usados nos documentos resultantes;

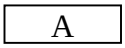
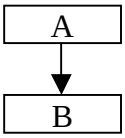
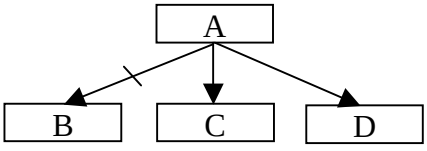
- aplicam funções aritméticas e funções agregadas em elementos de um documento, tanto na fase de seleção como na fase de construção; e

- computam as operações de união, diferença e produto cartesiano.

Diferentemente das linguagens textuais propostas para consulta a documentos XML (por exemplo, a XML-QL), a linguagem XML-GL utiliza uma notação uniforme (grafos XML) a fim de expressar tanto a extração quanto a construção de elementos, graças à notação visual simples.

4.4.5 O Modelo de Dados Gráfico XML e XML-GL

Grafos XML podem ser usados para representar documentos XML, como mostra a figura abaixo (Comai et al., 2001).

CARACTERÍSTICA	DOCUMENTO	REPRESENTAÇÃO GRÁFICA
Elemento	<A></A>	
Hierarquia de elementos	<A> <B></B> </A>	
Ordem de sub-elementos	<A> <B></B> <C></C> <D></D> </A>	

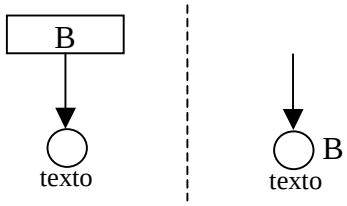
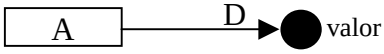
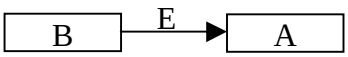
Elemento com conteúdo PCDATA	<B>texto</B>	
PCDATA e atributos ID	<A D="valor"></A>	
Atributos IDREF	<B E="valor"></B>	

Figura 16 – Notação para Grafos XML.

Os elementos XML, conforme pode ser observado na figura acima, são representados por nodos rotulados com o nome do elemento XML, graficamente denotado por um retângulo. O aninhamento de elementos é expresso por meio de arcos de ligação, onde a origem do arco é do elemento "pai", e o destino do arco é para o elemento "filho". A ordem de sub-elementos com um mesmo super-elemento é desenhado através da ordem dos arcos de ligação para os sub-elementos, sendo que o primeiro arco tem uma característica especial para indicar que é o primeiro sub-elemento da ordem.

Conteúdos do tipo PCDATA e CDATA correspondem ao conteúdo do nodo, sendo denotados por um círculo branco contendo um rótulo que corresponde ao conteúdo em si. Os elementos PCDATA e CDATA podem ainda ser abreviados eliminando o nodo e incluindo o conteúdo do nodo num rótulo ao lado do círculo que possui o conteúdo do nodo. Atributos XML são denotados através de nodos representados por círculos pretos com um rótulo expressando o conteúdo do atributo. E por fim, referências entre elementos são representadas através de arcos de referência, rotulados com o nome do atributo IDREF.

Os grafos XML-GL são usados para expressar as consultas em grafos XML. Na figura abaixo (Comai et al., 2001) é possível visualizar as notações utilizadas nas consultas XML-GL.

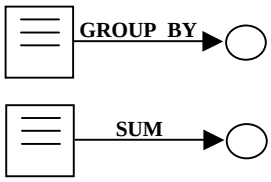
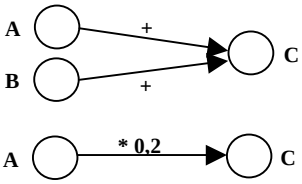
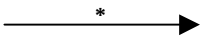
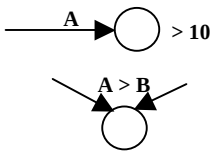
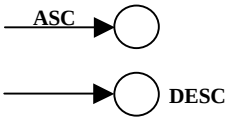
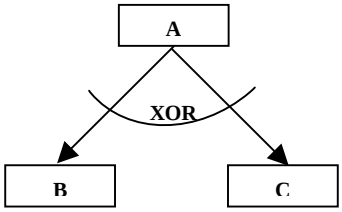
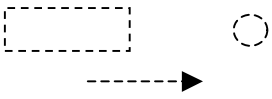
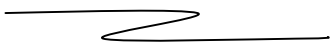
CARACTERÍSTICA	REPRESENTAÇÃO GRÁFICA
Construtor *LIST*	
Construtor *GROUPING LIST*	
Operação *GROUP_BY* e operações de agregação (*SUM,MIN,MAX,COUNT*)	
Operações aritméticas (ex. C=A+B) (ex. C=A*0,2)	
Estrela de Kleene	
Qualquer rótulo	
Rótulos de predicado unário (ex. A > 10) Binário (ex. A > B)	
Rótulos de ordenamento (ASC, DESC)	
Exclusão mútua	
Negação	
Arco de ligação	

Figura 17 – Notações adicionais para consultas XML-GL.

A função do construtor *LIST* é representar todos os elementos extraídos pela parte *extract-match* da consulta XML-GL. O triângulo representa o novo elemento conectado por um relacionamento de posse para os objetos na parte *clip-construct* representando os sub-elementos a serem acoplados. A figura abaixo ilustra o uso deste construtor.

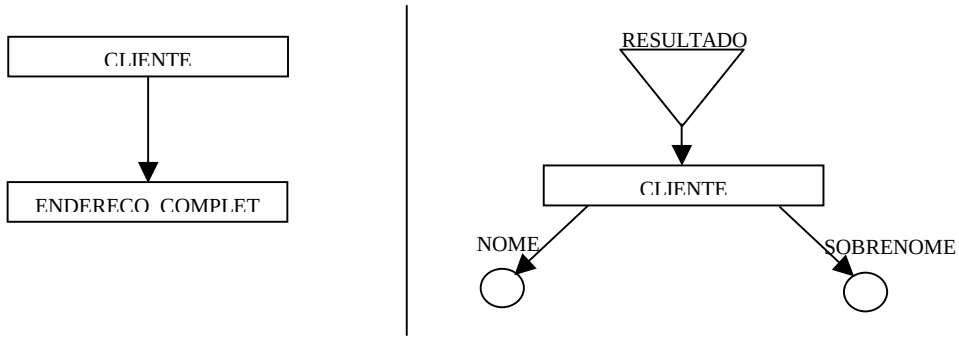


Figura 18 – Exemplo do construtor *LIST*.

Ocorrências do mesmo elemento extraído pela parte *extract-match* da consulta são incluídas em listas múltiplas definidas por um critério de agrupamento. O construtor *GROUPING LIST* é representado por um retângulo com linhas horizontais e representa a nova lista agrupada conectada por um relacionamento aos objetos da parte *clip-construct* representando os sub-elementos a serem incluídos.

O critério de agrupamento é representado por um arco de ligação conectando o construtor *GROUPING LIST* a um ou mais elementos usados para agrupamento. A figura abaixo mostra o uso deste construtor.

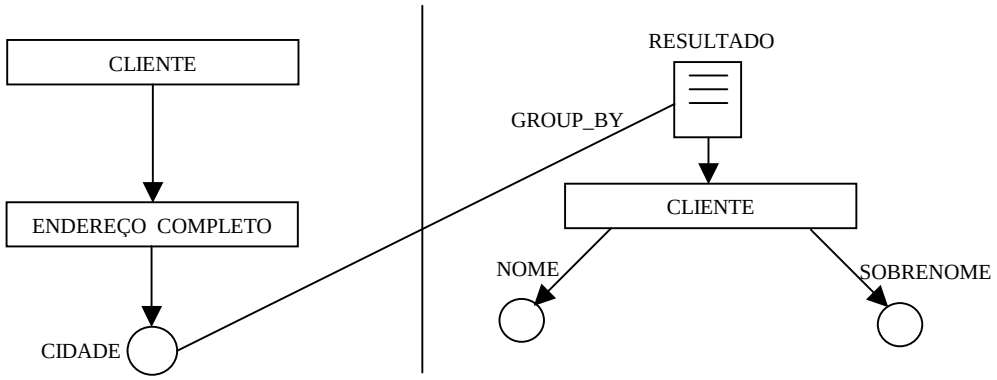


Figura 19 – Exemplo do construtor *GROUPING LIST*.

As demais representações gráficas para uma consulta XML-GL (operações de agregação, operações aritméticas, predicados, exclusão mútua, etc) são equivalentes às operações correspondentes em SQL.

4.4.6 Consultas Complexas em XML-GL

Um poder expressivo na linguagem de consulta XML-GL pode ser obtido através da representação de múltiplos grafos em ambos os lados da consulta, permitindo a representação das operações de união, diferença e produto cartesiano.

A operação de união é representada através de grafos múltiplos no lado esquerdo da consulta, cada grafo destes contendo elementos que possuem uma ligação com o mesmo elemento no lado direito. Um exemplo simples desta operação é ilustrado na figura abaixo.

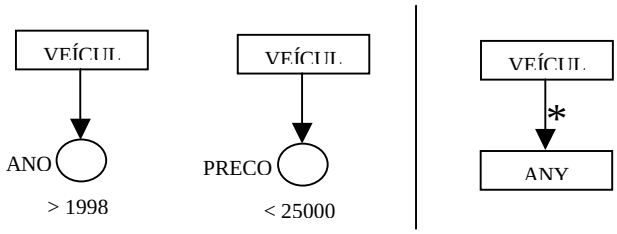


Figura 20 – Exemplo da operação *UNIÃO*.

A tradução para a consulta acima seria: *selecione todos os elementos <VEÍCULO> que foram fabricados após o ano de 1998 ou possuem <PREÇO> menor do que 25000*. Nesta consulta, todos os elementos <*VEÍCULO*> que satisfazem pelo menos uma das condições expressas pelos dois grafos do lado esquerdo da consulta aparecerão no resultado. É preciso salientar que se um elemento <VEÍCULO> satisfaz ambas as condições, ele aparecerá apenas uma única vez no resultado.

Assim como a operação de *UNIÃO*, a operação de *DIFERENÇA* pode ser representada através de múltiplos grafos no lado esquerdo da consulta XML-GL contendo o mesmo elemento raiz ocorrendo tanto positivamente como negativamente, com a restrição de que pelo menos uma ocorrência seja positiva. Esta última condição é

uma restrição de segurança a fim de evitar resultados infinitos. Um exemplo simples da operação de diferença é mostrado na figura abaixo.

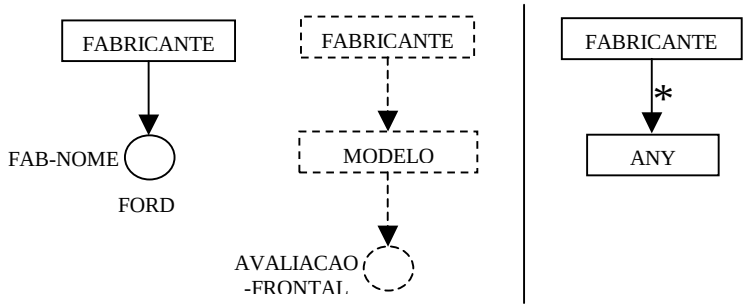


Figura 21 – Exemplo da operação *DIFERENÇA*.

Esta representação se traduz na seguinte consulta: *selecione todos os elementos <FABRICANTE>onde <FAB-NOME> é "FORD" e para o qual o elemento <MODELO> não possui <AVALIACAO-FRONTAL>*.

Nesta consulta, os elementos <*FABRICANTE*> que satisfazem o grafo positivo são selecionados, e então o grafo de negação é considerado a fim de eliminar os elementos sem <*AVALIACAO-FRONTAL*>.

Analogamente à SQL, a operação de diferença pode ser expressa por meio da condição existencial negada (NOT EXISTS). A consulta ilustrada pela figura abaixo fornece o mesmo resultado da consulta exemplificada na operação de *DIFERENÇA*, utilizando uma condição existencial negada.

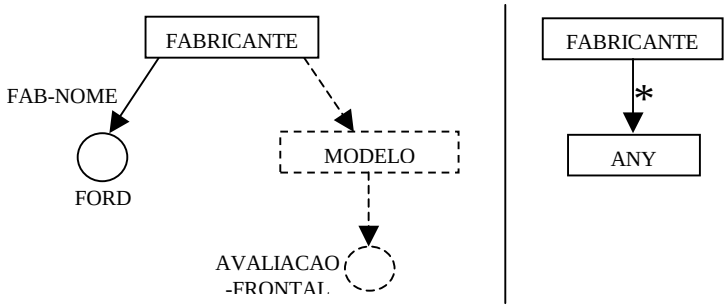


Figura 22 –*DIFERENÇA* através da Condição Existencial Negada.

Assim como nas duas operações anteriores (*UNIÃO* e *DIFERENÇA*) o produto cartesiano pode ser expresso pela representação de múltiplos grafos no lado esquerdo da consulta, tal que cada um deles contêm elementos desvinculados que

fornecem ligações à elementos distintos no lado direito da consulta conectados através de arcos/nodos criados. A figura abaixo ilustra um exemplo simples desta operação.

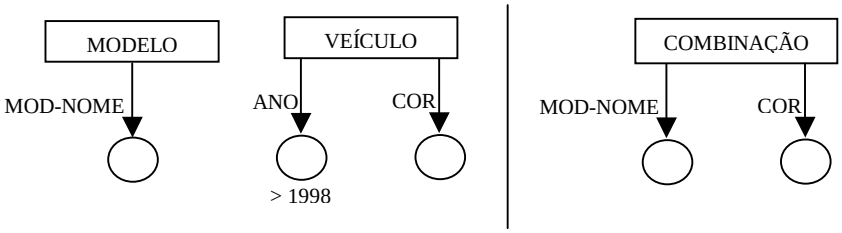


Figura 23 – Exemplo para a operação *PRODUTO CARTESIANO*.

A consulta se traduz na seguinte operação: *combine todos os elementos <COR> dos veículos fabricados a partir de 1999 com todos os elementos <MOD-NOME> possíveis.*

4.4.7 Trabalhos relacionados à XML-GL

A linguagem XML-GL possui vários aspectos semelhantes às linguagens G-Log e WG-Log, pois tanto a sintaxe como a semântica da XML-GL são derivadas destas duas linguagens gráficas de consulta.

Apesar desta derivação, a XML-GL adapta e estende a notação da G-Log/WG-Log para representar algumas ferramentas XML (atributos e elementos, exclusão mútua e ordenação de elementos). Além disso, a XML-GL possibilita as operações de agregação e funções aritméticas, as quais não estão presentes nem na G-Log e nem na WG-Log.

O problema de consultar documentos XML com uma interface visual, interface esta que também possa ser usada por usuários inexperientes, tem sido abordado recentemente com maior atenção pela comunidade de pesquisa XML, e duas propostas estão sendo estudadas: Equix (Cohen et al., 1998) e BBQ (Munroe et al., 2000). Ambas as propostas são baseadas na representação de documentos como uma estrutura de árvore, análogo à representação de diretórios num esquema de árvore, construída diretamente a partir do DTD do documento XML a ser consultado.

4.5 A linguagem XQL

4.5.1 Introdução

A fim de analisar as características da linguagem de consulta XQL, é interessante levar em consideração quatro questões básicas sobre o ambiente onde a consulta ocorre:

1. O que é um banco de dados?

2. O que é a linguagem de consulta?

3. O que vem a ser a entrada de uma consulta?

4. O que é o resultado de uma consulta?

É interessante realizar a análise destas questões em paralelo, tanto para um banco de dados relacional quanto para a XQL.

Num banco de dados tradicional (relacional), pode-se chegar às seguintes respostas simplificadas para tais questões:

1. O banco de dados é um conjunto de tabelas;

2. Consultas são realizadas através da SQL, uma linguagem que utiliza tabelas como modelo básico;

3. A cláusula FROM determina quais tabelas serão examinadas/utilizadas pela consulta;

4. O resultado da consulta é uma tabela contendo um conjunto de linhas.

A XQL possui respostas aproximadas com as respostas acima fornecidas para um banco de dados relacional, porém com algumas diferenças significantes:

1. Um banco de dados é um conjunto contendo um ou mais documentos XML;

2. Consultas são realizadas em XQL, uma linguagem de consulta que utiliza a estrutura da XML como modelo básico;

3. Uma consulta é definida sobre um conjunto de nodos de um ou mais documentos;

4. O resultado de uma consulta é o conjunto de nodos definidos por um documento XML, sendo que este conjunto pode ser encapsulado por um nodo raiz a fim de que seja definido um documento XML bem-formado.

A fim de apresentar estes conceitos de forma concreta, é possível determinar uma consulta XQL simples, analisando a entrada da consulta, a consulta em si e o seu resultado. Neste exemplo (abaixo descrito por um documento XML) a entrada da consulta (conhecida como *contexto da pesquisa*) é um elemento simples <DRF>, o qual é a raiz do documento:

```
<DRF>
    <JURISDICAO>Cascavel</JURISDICAO>
    <LOCAL>
        <CIDADE>Cascavel</CIDADE>
        <FONE>(45)225 1214</FONE>
     </LOCAL>
</DRF>
```

A consulta mais simples possível na linguagem XQL é uma string que representa o nome de um elemento. Por exemplo, *"DRF"* é uma consulta XQL válida e completa para o documento de entrada XML acima descrito.

A consulta descrita simplesmente pelo comando *"DRF"* seleciona todos os elementos *DRF* do contexto de pesquisa, retornando como resultado para o exemplo acima o próprio contexto de pesquisa, ou seja:

```
<DRF>
    <JURISDICAO>Cascavel</JURISDICAO>
    <LOCAL>
        <CIDADE>Cascavel</CIDADE>
        <FONE>(45)225 1214</FONE>
     </LOCAL>
</DRF>
```

4.5.2 Conjunto Resultado versus Documento Resultante

Em diversas situações é interessante que o conjunto resultado de uma consulta seja expressa através de um documento resultante XML bem-formado. Algumas das razões para isto podem ser as seguintes:

- Um documento XML é facilmente analisado por um parser XML, após esta análise ele pode ser transmitido como uma simples cadeia de caracteres ASCII e analisado pela aplicação de recepção;

- Um documento XML pode ser apresentado através de um browser XML padrão;

- Um documento XML pode ser armazenado num repositório de dados XML nativo;

- Um documento XML pode ser utilizado em conjunto com um documento XSL a fim de executar transformações e/ou formatação sobre ele mesmo.

No exemplo do subitem anterior o conjunto resultado da consulta continha apenas um único nodo raiz. No entanto, uma consulta pode retornar mais do que um nodo raiz, o que implicaria numa representação textual sem a característica de ser bem-

formado para uma representação XML, já que um documento XML deve ter apenas um único elemento raiz.

Supondo que o conjunto resultado para uma dada consulta fosse o seguinte:

<CIDADE>Cascavel</CIDADE>
<FONE>(45)225 1214</FONE>

Em virtude deste resultado conter dois nodos raízes, ele não se constitui como um documento XML válido. No entanto, se os nodos deste resultado forem encapsulados por um elemento raiz comum, o documento XML resultante torna-se válido. Portanto o *documento resultante* de uma consulta XQL sempre encapsula os nodos do *conjunto resultado* em um elemento *<XQL:RESULTADO>*:

<XQL:RESULTADO>
<CIDADE>Cascavel</CIDADE>
<FONE>(45)225 1214</FONE>
</XQL:RESULTADO>

Ambientes que não precisam que a consulta retorne documentos XML válidos geralmente trabalham com o *conjunto resultado*. Os outros ambientes normalmente trabalham com o *documento resultante*.

4.5.3 Consultas simples em XQL

Como visto anteriormente, uma string simples com o nome de um elemento contido no documento XML que está sendo consultado pode ser considerada como uma consulta XQL completa.

A fim de permitir navegabilidade à consulta pelo documento XML que está sendo consultado, a linguagem XQL utiliza o operador "/" para indicar hierarquia de elementos. Um exemplo simples é mostrado abaixo:

LOCAL/FONE

A consulta acima descrita seleciona todos os elementos filhos <*FONE*> do seu elemento pai <*LOCAL*>. O elemento raiz deve ser precedido pelo operador "/". Por exemplo, no caso abaixo ilustrado:

/DRF/JURISDICAO

Na linguagem XQL, a raiz de um documento é diferente do elemento raiz. A raiz de um documento se refere à entidade documento, o que é basicamente equivalente ao documento em si. Já o elemento raiz é o elemento que contém o resto dos elementos do documento. A raiz de um documento sempre contém o elemento raiz, mas pode conter também informações relativas ao tipo de documento, instruções de processamento e comentários.

Como a XQL tem por base expressões de caminho, o caminho sempre é descrito do elemento mais externo para o mais interno do documento, ou seja, da raiz para o interior, e a menos que seja especificado, o elemento mais à direita deste caminho é retornado como resultado da consulta. A consulta abaixo é ilustrada como exemplo:

/DRF/LOCAL/CIDADE

Nesta consulta, recebe-se como resultado todos os valores contidos no elemento <*CIDADE*>.

O conteúdo de um elemento ou o valor de um atributo pode ser especificado na consulta utilizando-se o operador de igualdade (“=”). Por exemplo, a consulta abaixo retorna todos os elementos <*CIDADE*> com valor igual a “*CASCAVEL*”.

/DRF/LOCAL/CIDADE= ‘CASCAVEL’

Nomes de atributo devem iniciar-se com o caractere especial “@”. Os atributos são tratados como filhos dos elementos que os contêm. Por exemplo, considerando o atributo *nome* do elemento *LOCAL*, conforme ilustrado abaixo:

```
<DRF>
    <LOCAL nome= “CASCAVEL”>
        <FONE>(45)225 1214</FONE>
     </LOCAL>
</DRF>
```

Uma consulta para selecionar os nomes de todos os locais da DRF que sejam iguais a *CASCAVEL* poderia ser expressa como segue:

/DRF/LOCAL/@nome= ‘CASCAVEL’

O operador de descendentes (“//”) indica qualquer número de níveis (elementos) intermediários, ou caso apareça no começo da consulta indica todos os nodos anteriores ao elemento referenciado. Alguns exemplos:

/DRF//FUNCIONARIO
//LOCAL

O operador de filtragem (“[ ]”) filtra o conjunto de nodos à sua esquerda baseado na condição dentro dos colchetes. Um exemplo da utilização deste operador é descrito a seguir:

/DRF/LOCAL/SETOR/FUNCIONARIO/ENDERECO[@TIPO= “EMAIL”]

É importante denotar a diferença existente entre “*ENDERECO[@TIPO= ‘EMAIL’]*” que retorna endereços e “*ENDERECO/@TIPO= ‘EMAIL’*” retorna os tipos de atributo.

Condições múltiplas podem também ser combinadas através de operadores booleanos.

4.5.4 Operadores XQL Básicos

Os operadores de retorno e de sequência são básicos para o modelo XQL completo, porém não se fazem necessários para todas as implementações XQL. Operadores de retorno são análogos à estrutura *SELECT* da SQL, permitindo melhor controle sobre o que é retornado no resultado de uma consulta. No entanto, estes operadores não são necessários para todas as aplicações, já que muitas aplicações retornam nodos simples em consultas ou possuem requisitos muitos simples a serem retornados.

Operadores de sequência permitem determinar a ordem em que os dados aparecerão num documento a ser utilizado em condições de consulta, sendo extremamente útil em muitos tipos de documentos.

No modelo XQL completo, condições para nodos simples (individuais) podem conter:

- Condições sobre elemento ou nomes de atributo (ex. "*CIDADE*", "*@EMAIL*");
- Condições sobre conteúdo ou valor (ex. "*CIDADE= 'CASCAVEL'*, *@EMAIL= 'LNAKANO@ZIPMAIL.COM'*);
- Condições sobre tipos de nodos (ex. "*ELEMENT()*").

Os relacionamentos básicos entre os elementos são:

- Hierarquia (pai/filho, ancestral, descendente);
- Sequência (antecessor imediato, antecessor);
- Posição (absoluta, relativa, intervalo).

As condições para nodos e as condições para os relacionamentos entre os elementos são combinadas a fim de formar uma consulta de caminhos (expressão de caminho) dentro do contexto de pesquisa. Operadores de retorno são usados para selecionarem nodos específicos das expressões de caminho, estes nodos são então retornados pela consulta.

Basicamente existem dois tipos de operadores de retorno:

- Retorno de superfície ("?");

- Retorno de profundidade (“??”).

Quando o operador de retorno de superfície é aplicado sobre um elemento, ele não retorna nem os atributos deste elemento e nem seus filhos. Já o operador de retorno de profundidade retorna o nodo e todos os seus filhos. Os operadores de retorno podem simplificar consultas para documentos com estruturas mais complexas. Por exemplo, considerando o documento abaixo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="c:\xquery\nakano\estilos\estilo.xsl"?>
<!DOCTYPE biblio SYSTEM "drf.dtd">
<DRF>
    <JURISDICAO>Cascavel</JURISDICAO>
    <LOCAL>
        <CIDADE>Cascavel</CIDADE>
        <FONE>(45)225 1214</FONE>
        <SETOR>
                <NOME>DMA</NOME>
                <RAMAL>292</RAMAL>
                <FUNCIONARIO>
                    <NOME>Antonio Carlos Markewicz</NOME>
                    <MATRICULA>090.1342-1</MATRICULA>
                </FUNCIONARIO>
                <FUNCIONARIO>
                    <NOME>Alice Golenia dos Passos</NOME>
                    <MATRICULA>090.1242-2</MATRICULA>
                </FUNCIONARIO>
        </SETOR>
        <SETOR>
                <NOME>CAC</NOME>
                <RAMAL>223</RAMAL>
                <FUNCIONARIO>
                    <NOME>Sebastiana de Oliveira</NOME>
                    <MATRICULA>090.3323-9</MATRICULA>
                </FUNCIONARIO>
                <FUNCIONARIO>
                    <NOME>Adilson Mazali</NOME>
                    <MATRICULA>090.2343-1</MATRICULA>
                </FUNCIONARIO>
                <FUNCIONARIO>
                    <NOME>Rozeli Silvia de Lima</NOME>
                    <MATRICULA>090.7893-0</MATRICULA>
                </FUNCIONARIO>
        </SETOR>
    </LOCAL>
</DRF>
```

Supondo que nossa consulta seja visualizar todos os funcionários de um setor. É possível realizar esta consulta com a seguinte expressão:

SETOR//NOME

O resultado da consulta acima seria:

```
<XQL:RESULT>
                <NOME>Antonio Carlos Markewicz</NOME>
                <NOME>Alice Golenia dos Passos</NOME>
                <NOME>Sebastiana de Oliveira</NOME>
                <NOME>Adilson Mazali</NOME>
                <NOME>Rozeli Silvia de Lima</NOME>
<XQL:RESULT>
```

Infelizmente o resultado não mostra quais funcionários são do mesmo setor, mas isto pode ser facilmente corrigido com a utilização dos operadores de retorno. Portanto, é possível refazer a consulta da seguinte forma:

SETOR?//NOME??

O resultado da consulta acima seria:

```
<XQL:RESULT>
        <SETOR>
                <NOME>Antonio Carlos Markewicz</NOME>
                <NOME>Alice Golenia dos Passos</NOME>
        </SETOR>
        <SETOR>
                <NOME>Sebastiana de Oliveira</NOME>
                <NOME>Adilson Mazali</NOME>
                <NOME>Rozeli Silvia de Lima</NOME>
        </SETOR>
<XQL:RESULT>
```

Supondo a necessidade de apresentar o nome do setor juntamente com seus funcionários, é possível reformular a consulta para:

SETOR?[NOME??]//NOME??

O resultado então seria:

```
<XQL:RESULT>
        <SETOR>
            <NOME>DMA</NOME>
                <NOME>Antonio Carlos Markewicz</NOME>
                <NOME>Alice Golenia dos Passos</NOME>
        </SETOR>
        <SETOR>
            <NOME>CAC</NOME>
                <NOME>Sebastiana de Oliveira</NOME>
                <NOME>Adilson Mazali</NOME>
                <NOME>Rozeli Silvia de Lima</NOME>
        </SETOR>
```

<XQL:RESULT>

4.5.5 Domínio de Aplicação da Linguagem

Consultas em um documento simples podem ser extremamente úteis em editores ou browsers XML a fim de permitir que o usuário selecione partes de um extenso documento, localizando apenas a informação relevante sem a necessidade de percorrer pelo documento todo.

As consultas podem ser usadas também pelo autor do documento a fim de proporcionar visões distintas para um mesmo documento. Uma implementação sofisticada pode definir índices para um documento e estabelecer métodos para se evitar a necessidade de carregar o documento inteiro na memória. Uma implementação mais simples poderia apenas realizar pesquisas no documento inteiro. Já que muitas aplicações que processam documentos simples são aplicações relativamente simples, é interessante que esta aplicação possua uma linguagem de consulta relativamente simples de ser analisada e implementada. A linguagem XQL é uma linguagem de fácil análise, e consequentemente possui uma implementação simples e pode ser escrita de maneira rápida.

Consultas em conjuntos de documentos também são de extrema importância, por exemplo em repositórios XML, em *web sites* e até mesmo num *data-mining*. Aplicações que gerenciam coleções de documentos devem possuir implementações relativamente sofisticadas a fim de oferecer um desempenho adequado. Para tais aplicações, é importante que ela seja capaz de definir estruturas de indexação adequadas e evitar o carregamento dos documentos inteiramente na memória. A XQL apresenta-se como uma linguagem bastante simples, fornecendo um número pequeno de primitivas, as quais são definidas no contexto da estrutura semântica de documentos XML.

4.6 A linguagem XQuery

4.6.1 Introdução

A linguagem de consulta XQuery foi projetada e está em desenvolvimento a fim de satisfazer os requisitos listados pelo W3C e seus respectivos casos de uso. Ela foi projetada para ser uma linguagem simples e de fácil implementação na qual as consultas são concisas e facilmente entendidas. Ela também foi projetada a fim de ser suficientemente flexível para consultar uma ampla área de origens de informação XML, incluindo tanto bancos de dados como documentos. A XQuery é derivada de uma linguagem de consulta chamada Quilt, a qual utilizou ferramentas de muitas outras linguagens, tais como XPath, XQL, XML-QL, SQL e OQL.

Graficamente, é possível ilustrar esta herança de características da seguinte forma:

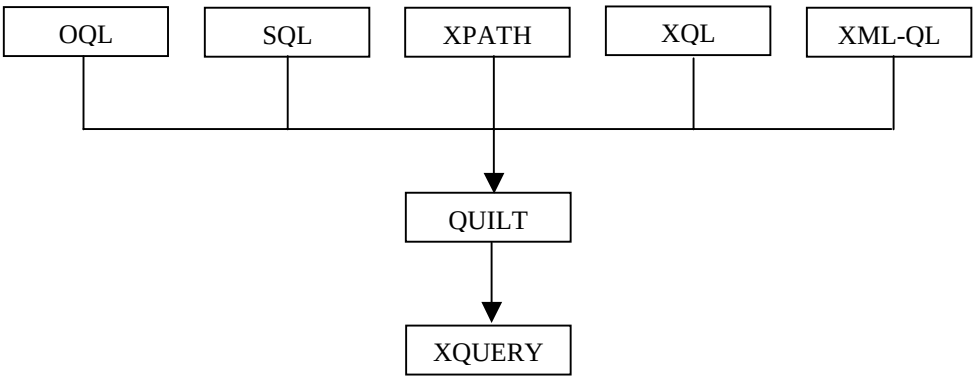


Figura 24 – Relação de herança de características de linguagens.

Assim como a linguagem OQL, a XQuery é uma linguagem funcional (onde a consulta estabelece o resultado desejado e não como atingir o resultado esperado) na qual uma consulta é representada por uma expressão. Esta linguagem suporta muitos tipos de expressões. As várias formas de expressões XQuery podem ser aninhadas e combinadas de uma forma extremamente completa.

4.6.2 O Modelo de Dados da XQuery

O modelo de dados XQuery fornece a entrada e saída de um processador de consultas XQuery. Uma instância do modelo de dados XQuery pode consistir de três itens principais: nodos, valores simples e sequências. Um nodo pode ser um dos seguintes tipos: *Documento, Elemento, Atributo, Namespace*[3], *Comentário, Instrução de Processamento, Texto ou Referência*.

O modelo trata documentos XML e fragmentos do documento como uma árvore de nodos. Um documento é uma árvore com o nodo documento sendo a raiz da árvore. Similarmente, um fragmento de documento é uma árvore com um elemento na sua raiz. Nodos do tipo *documento* e *namespace* não possuem pais. Todos os outros nodos podem ter zero ou um pai. Apenas um nodo *documento* ou um nodo *elemento* podem ter filhos. Um exemplo simples para um documento XML e sua instância do modelo de dados podem ser visualizadas na figura abaixo.

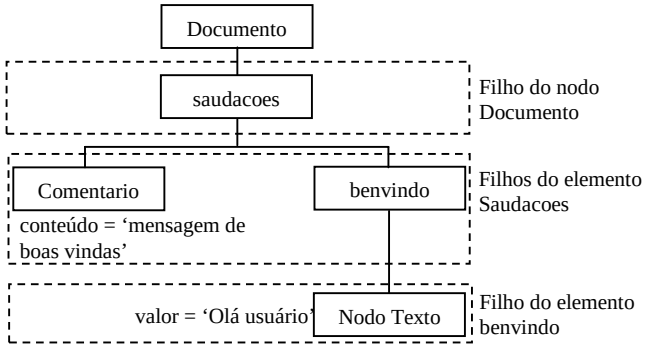
Documento XML	Modelo de dados
<?xml version="1.0"?> <saudacoes> <!—mensagem de boas vindas --> <benvindo>Olá usuário</benvindo> </saudacoes>	

Figura 25 – Instância de um Modelo de dados XML.

O modelo de dados representa o documento XML contendo um elemento raiz <*saudacoes*> e um elemento <*benvindo*> o qual possui um texto.

[3]Namespace é uma coleção de nomes identificada por uma referência URI (Uniform Resource Identifiers - RFC2396) utilizada para diferenciar vocabulários de marcação em XML.

Um valor simples no modelo de dados pode ser um dos dezenove tipos de dados primários definidos no *XML Schema Parte 2*[4]: *string, boolean, decimal, float, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gMonth, gDay, hexbinary, base64Binary, anyURI, QName* e *Notation*.

A sequência da informação para o modelo de dados da XQuery pode ser imaginada como uma lista linear de nodos e/ou valores, o que significa dizer que uma sequência não pode ter uma outra sequência como seu membro. Uma característica importante desta sequência é que o modelo de dados não faz distinção entre um nodo (ou valor) simples e uma sequência contendo este nodo (ou valor) simples.

4.6.3 Estrutura de um módulo XQuery

A linguagem XQuery possui algumas similaridades muito perceptíveis com as linguagens SQL e OQL. O termo módulo XQuery significa uma unidade de consulta. Basicamente um módulo XQuery é composto de três partes: declarações *namespace* e *Schema* (opcional), definição de funções (opcional) e expressões de consulta. O exemplo abaixo ilustra estas três partes.

```
parte 1   namespace xsd = "http://www.w3.org/2000/10/XMLSchema"
parte 2   define function fatorial (xsd:integer $n) returns xsd:integer {
              if ($n eq 0)
              then 1
              else $n * fatorial ($n - 1)
          }
parte 3   <Resultado>
              <descricao>Fatorial de 10</descricao>
              <valor>{fatorial(10)}</valor>
          </Resultado>
```

Figura 26 – Composição de um módulo XQuery.

[4] XML Schema Parte 2: a especificação da XML Schema é composta de 3 partes –XML Schema Parte 0, a qual é direcionada para fornecer um entendimento rápido de como criar esquemas usando a linguagem XML Schema; XML Schema Parte 1, a qual determina a estrutura para se criar esquemas XML e definir a aplicação de esquemas em documentos XML; XML Schema Parte2, define os tipos de dados que podem ser usados na XML Schema.

4.6.4 Um Exemplo Simples sobre XQuery

A fim de demonstrar e entender as características da linguagem XQuery, é ilustrado abaixo um fragmento de um documento usado nos casos de uso apresentado pelo *XQuery Working Group* (W3Cf):

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author>
          <last>Stevens</last>
          <first>W.</first>
      </author>
      <publisher>Addison-Wesley</publisher>
      <price> 65.95</price>
   </book>
   <book year="2000">
      <title>Data on the WEB</title>
      <author>
         <last>Abiteboul</last>
         <first>Serge</first>
      </author>
      <author>
         <last>Buneman</last>
         <first>Peter</first>
      </author>
      <author>
         <last>Suciu</last>
         <first>Dan</first>
      </author>
      <publisher>Morgan Kaufmann Publishers</publisher>
      <price> 39.95</price>
   </book>
</bib>
```

Considerando que o fragmento acima seja um documento denominado "*books.xml*" e, considerando ainda a necessidade de uma consulta para selecionar todos os livros publicados pela editora *Addison-Wesley* após o ano de *1991*, incluindo no resultado da consulta o *ano* e o *título* do livro. Na linguagem XQuery, tal consulta seria expressa da seguinte forma:

```
<bib>
{
 FOR $b IN document("books.xml")/bib/book
 WHERE $b/publisher = "Addison-Wesley" AND $b/@year > 1991
 RETURN
   <book year={ $b/@year }>
    { $b/title }
   </book>
}
```

```
</bib>
```

O resultado gerado pela consulta seria:

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
   </book>
</bib>
```

Esta consulta utiliza algumas expressões XQuery simples, dentre elas as expressões de caminho, construtores de elemento e de atributo e expressões FLWR (*FOR-LET-WHERE-RETURN*).

4.6.5 Expressões de caminho

As expressões de caminho da linguagem XQuery são baseadas nas expressões de caminho da linguagem XPath. Uma expressão de caminho fornece um meio de endereçar partes específicas de um documento XML através de um caminho para o conteúdo de interesse na árvore do documento. Por exemplo, considerando o exemplo anterior, para fazer referência a todos os elementos *book*, os quais são filhos do elemento raiz *bib* do documento *books.xml*, basta utilizar a seguinte expressão:

document("books.xml")/bib/book

4.6.6 Construtores de elemento e atributo

O resultado da consulta ilustrada no item 4.6.4 possui dois elementos: <*bib*> e <*book*>. Para construir estes elementos, os elementos <*bib*>...</*bib*> e <*book*>...</*book*> foram escritos diretamente no corpo da própria consulta. As chaves "{ }" são usadas para separar o conteúdo literal de qualquer sub-expressão dentro de um elemento. As sub-expressões incluídas nas chaves são analisadas pelo processador de consulta XQuery.

De forma similar, os construtores de atributo são especificados pela sua inclusão junto à sub-expressão contida dentro das chaves, como ilustrado abaixo:

<book year={ $b/@year }>

4.6.7 Expressões FLWR

Expressões FLWR (For-Let-Where-Return - pronuncia-se '*flower*') fornecem uma sintaxe semelhante à sintaxe SQL para extração de dados, executando seleção sobre os dados extraídos e retornando o resultado numa estrutura escolhida pelo usuário. Por exemplo, considerando a expressão FLWR contida na consulta abaixo:

```
<bib>
{
 FOR $b IN document("books.xml")/bib/book
 WHERE $b/publisher = "Addison-Wesley" AND $b/@year > 1991
 RETURN
   <book year={ $b/@year }>
    { $b/title }
   </book>
}
</bib>
```

A cláusula *FOR* atua sobre a sequência de nodos retornados pela expressão de caminho, ligando a variável *$b* a cada nodo <*book*> retornado na expressão. Uma vez que a variável é ligada, é possível acessar qualquer sub-parte do nodo ligado à variável através do uso de expressões de caminho. Por exemplo, *$b/@year* retorna o atributo *year* do nodo <*book*> e *$b/title* retorna o elemento filho <*title*> do nodo <*book*>. A cláusula *IN* define o conjunto de nodos ligados à variável definida na cláusula *FOR*.

A cláusula *WHERE* seleciona os nodos <*book*> onde os elementos filhos <*publisher*> contêm a string *'Addison-Wesley'* e o atributo *year* contém *'1991'*. A cláusula *RETURN* constrói elementos <*book*> incluindo o ano de publicação como atributo ('*@year'*) e o elemento <*title*> como filho.

4.6.8 Ferramentas XQuery adicionais

Alguns mecanismos análogos à SQL são também fornecidos pela linguagem XQuery a fim de facilitar algumas operações de estruturação dos dados nas

consultas. Um destes mecanismos é fornecido pela cláusula *SORTBY*, a qual é usada para determinar uma ordem sobre a sequência. Por exemplo, no exemplo anterior, para obter-se todos os nodos <*book*> ordenados pelo nodo filho <*title*>, bastar utilizar a seguinte expressão:

```
FOR $b IN document("books.xml")/bib/book SORTBY (title)
```

Neste exemplo, a expressão de caminho *document (“books.xml”)/bib/book* é avaliada primeiro e a sequência resultante de nodos é então ordenada pela cláusula *SORTBY*.

Declarações *namespace* são usadas para declarar endereços de *namespaces* a serem usados num módulo de consulta XQuery. Ela pode ser usada também para declarar um *namespace* padrão para um módulo XQuery. Por exemplo, a seguinte consulta retorna todos os elementos <*book*> no *namespace* identificado pela URI “*http://www.bibliophile.com*”:

```
Namespace booklovers="http://www.bibliophile.com"
document("books.xml")//booklovers:book
```

A linguagem XQuery possui um conjunto central de biblioteca de funções definida na sua especificação *Funções* e *Operadores*. A função *document* é um exemplo destas funções. Além desta biblioteca central, a XQuery permite que os usuários definam suas próprias funções. O exemplo abaixo demonstra a sintaxe para definição de funções.

```
Namespace xsd = "http://www.w3.org/2001/XMLSchema"
Namespace booklovers="http://www.bibliophile.com"
Schema "http://www.bibliophile.com"
   "http://bibliophile.com/books.xsd"
Define Function getBooksByTitle(xsd:string $title)
   RETURNS book_seq {
   RETURN Document("books.xml")/bib/books[title=$title]
}
```

Este exemplo define uma função que pega uma string como argumento e retorna uma sequência de nodos <*book*> que possuem um determinado título (*title = $title*). O tipo retornado é o *book_seq*, o qual deve ter sido previamente definido no esquema *books.xsd* incluso na cláusula *Schema*.

4.7 Comparativo

Algumas comparações de extrema importância podem ser definidas entre as linguagens estudadas, entre elas a XML-QL, Lorel, XML-GL, XQL, XQuery e XSL; esta última apesar de estar no contexto de estudo da linguagem XML também traz conceitos de linguagem de consulta a dados XML, por isso também pode ser analisada neste contexto, conforme tabela abaixo realizada por Comai (2001).

Ferramenta	**XML-QL**	**Lorel**	**XML-GL**	**XQL**	**XQuery**	**XSL**
Junção	S	S	S	S	S	S
Documentos múltiplos	S	S	S	S	S	S
Quantificação, negação e redução	P	P	P	S	S	P
Modificação de estrutura	S	S	S	N	S	S
Novos elementos/relacionamentos	S	S	S	P	S	S
União, diferença, produto cartesiano	P	S	S	S	S	S
Agrupamento	S	S	S	S	S	P
Funções de agregação	S	S	S	P	S	P
Funções aritméticas	N	S	S	N	S	S
Sub-consultas aninhadas	S	S	N	P	S	S
Ordenamento do resultado	S	S	S	N	S	S

Tabela 11 – Tabela comparativa entre as linguagens.

Legenda:

S = Sim, ferramenta existente
N = Não, ferramenta não existente
P = Parcial, ferramenta parcialmente existente

4.8 Conclusão

Neste capítulo foi possível observar as principais diferenças encontradas nas funcionalidades das linguagens de consulta a XML, realizando uma analogia com as linguagens de consulta a banco de dados. Vários aspectos são semelhantes entre estas linguagens, porém o potencial existente em cada linguagem pode ser melhor observado na sua implementação, que é justamente o que será visto no próximo capítulo, em relação à linguagem XQuery, a qual apresenta-se como a linguagem de consulta XML mais completa em termos de ferramentas embutidas.

É importante frisar que nem todas as ferramentas estão embutidas na implementação do capítulo seguinte, visto que muitas destas ferramentas ainda encontram-se em estudo e pesquisa sobre a melhor maneira de tratamento.

CAPÍTULO V

5. A API XQUERY DO FRAMEWORK .NET

5.1 Introdução

Neste capítulo a linguagem XQuery é apresentada de forma prática através da implementação utilizando-se da API XQuery do Framework .NET com o propósito de demonstrar os recursos disponibilizados pela linguagem fazendo-se uma analogia com as linguagens de consulta a banco de dados a fim de facilitar a compreensão de sua funcionalidade. É definida ainda neste capítulo uma extensão à funcionalidade desta API a fim de demonstrar a possibilidade de junção das linguagens de consulta a XML existentes para proporcionar um ambiente funcional ao usuário final.

A API (Aplication Program Interface – Interface para programação de aplicativos) XQuery do Framework .NET contém uma implementação inicial realizada pela Microsoft para a linguagem XQuery que se encontra em estudo e desenvolvimento pelo W3C XQuery Working Draft. Esta implementação inicial contém as classes necessárias para o desenvolvimento de programas que necessitem utilizar a especificação XQuery.

Estas classes foram projetadas a fim de integrar as classes existentes no Framework .NET SDK (Software Developer Kit). Para utilizar estas classes é necessário a instalação do *release Beta 2* do Framework .NET SDK. As classes distribuídas nesta API estão em fase de testes (é ainda uma versão *beta*) e não são destinadas para o uso num ambiente de produção. Vários testes de implementação têm sido realizados nos últimos meses a fim de desenvolver uma implementação para a linguagem XQuery porém, como o working draft corrente da linguagem ainda está em andamento e em constante mutação, não foi definida ainda uma implementação final para a linguagem.

Esta API XQuery foi desenvolvida pela Microsoft a fim de realizar um estudo e obter alguns resultados junto com a comunidade de pesquisa de desenvolvimento da linguagem XQuery e junto ao W3C XQuery Working Draft.

5.2 O Framework .NET

O Framework .NET é um ambiente para a construção, implantação e execução de serviços WEB e outras aplicações. Ele consiste de duas partes principais: o Common Language Runtime e as bibliotecas de classes.

O Common Language Runtime é o mecanismo de execução para aplicações desenvolvidas no ambiente do Framework. Ele proporciona vários serviços, incluindo os seguintes: gerenciamento de código (carga e execução), isolamento de memória de aplicação, verificação de segurança de tipo, acesso a meta dados (informação de tipo aprimorada), gerenciamento de memória para objetos gerenciados, reforço de segurança de acesso a código, tratamento de exceção, incluindo exceções entre linguagens, inter-operação entre código gerenciado, objetos COM e DLLs pré-existentes (código não gerenciado e dados), automação de layout de objetos, suporte para serviços de desenvolvedor (perfil, debugging e assim por diante).

As bibliotecas de classes básicas provêem as funcionalidades primárias, tais como funções de I/O, manipulações de strings, administração de segurança,

comunicações via rede, gerenciamento de processos, manipulação de texto, ferramentas de projeto de interface com o usuário, entre outras funções. O ponto principal para este trabalho é que o Framework PontoNET possui embutido as classes XML, as quais possibilitam a manipulação de dados XML bem como a pesquisa e tradução sobre estes dados.

O Framework PontoNET suporta uma série de linguagens de programação, ou seja, é possível construir programas PontoNET através de várias linguagens, entre elas a C++, Microsoft Visual Basic.NET, JScript e também a mais nova linguagem de programação da Microsoft, a C# (C Sharp). Várias linguagens de terceiros também são suportadas para a construção de aplicações PontoNET Framework, incluindo as linguagens COBOL, Eiffel, Perl, Pynthon e Smaltalk.

A maior importância deste Framework para este trabalho é que ele traz implementado várias classes que facilitam a manipulação de dados XML através da linguagem XQuery, diminuindo assim o tempo de implementação desta linguagem para testes e conclusões. Outro fator importante é que através da mais recente linguagem de programação C# (C Sharp) é possível visualizar com clareza os resultados obtidos através de tal implementação.

5.3 A tecnologia XML no Framework .NET

Classes XML no Framework .NET fornecem um conjunto integrado de classes, permitindo que dados e documentos XML sejam trabalhados facilmente. Os objetivos da XML no Framework são:

- Compatibilidade com os padrões estabelecidos pelo W3C - esta compatibilidade de padrões determina que as classes obedeçam aos padrões recomendados pelo W3C (XML, Namespaces, XSLT, XPath, Schema e DOM), assegurando interoperabilidade e facilitando o desenvolvimento de aplicações;

- Extensibilidade - a XML no Framework é extensível, pois possibilita o uso de classes abstratas e métodos em novas implementações, por exemplo, a API *XPathNavigator* é uma API que implementa um motor de consulta *XPath* e pode ser implementada/utilizada em qualquer armazém de dados existente, por exemplo sistema de arquivos, registros e bancos de dados relacionais;

- Arquitetura plugável – o termo plugável significa que componentes que são baseados nas classes abstratas do Framework podem ser facilmente substituídos. Arquitetura plugável significa também pode haver fluxo de dados entre os componentes, e novos componentes inseridos neste fluxo podem alterar o processamento;

- Desempenho – as classes XML no Framework representam componentes de processamento XML de baixo nível, utilizadas não apenas como parte do Framework, mas para integrar aplicações XML.

5.4 Principais Classes XML do Framework

Várias classes XML no Framework .NET são fornecidas com o objetivo de facilitar a manipulação de dados XML no ambiente. A seguir são listadas as principais classes, as quais são utilizadas no projeto de aplicação da linguagem XQuery implementada neste trabalho.

5.4.1 A Classe DOM

A classe DOM (Document Object Model) é uma representação de um documento XML na memória do computador. Esta classe permite a leitura, manipulação e modificação de um documento XML de forma programática. Existe também a classe *XmlReader* a qual consegue ler documentos XML, porém através dela não é possível

editar os valores de um atributo ou o conteúdo de um elemento XML, ou até mesmo inserir ou remover nodos XML. A edição é uma função primária da classe DOM.

Um exemplo da estrutura DOM pode ser visualizado através do exemplo abaixo. Considerando o documento XML abaixo:

```
<?xml version="1.0"?>
 <books>
  <book>
  <author>Carson</author>
  <price format="dollar">31.95</price>
  <pubdate>05/01/2001</pubdate>
  <book>
  <pubinfo>
  <publisher>MSPress</publisher>
  <state>WA</state>
  </pubinfo>
 </books>
```

A figura abaixo mostra como a memória é estruturada quando o documento XML é lido para uma estrutura DOM:

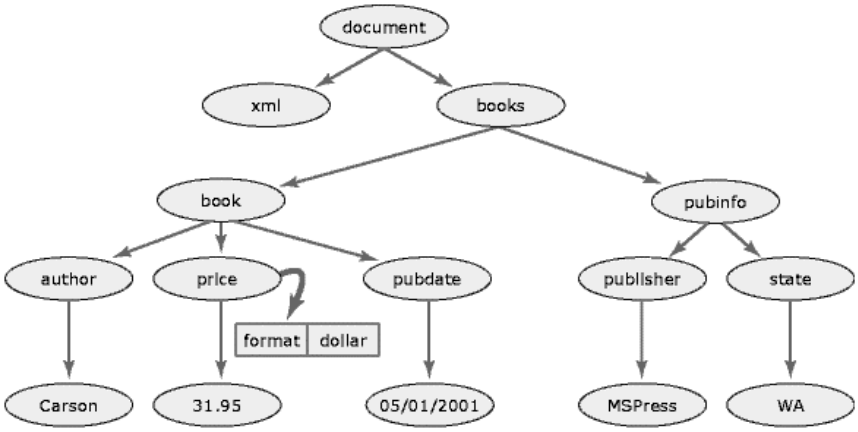


Figura 27 – Estrutura DOM.

A classe DOM é muito útil para carregar dados XML para a memória a fim de modificar sua estrutura, adicionar ou remover nodos, e modificar dados contidos em um nodo. Outras classes estão disponíveis com melhores desempenhos em outros cenários, por exemplo, para um acesso rápido unidirecional sem modificação de informações em dados XML, é possível utilizar as classes *XmlReader* e *XmlWriter*.

5.4.2 A Classe XmlReader

A classe XmlReader é uma classe básica abstrata do Framework que fornece acesso unilateral e apenas para leitura a dados XML. Ela verifica se o documento XML é bem-formado e gera uma exceção se for encontrado um erro no documento.

A classe XmlReader define métodos para:

- Ler o conteúdo XML quando este conteúdo está disponível na sua totalidade, por exemplo, um arquivo texto XML;
- Encontrar a profundidade de uma pilha de elementos XML;
- Determinar se o conteúdo de um elemento é vazio ou não;
- Ler e navegar sob atributos de elementos;
- Ignorar elementos e seu conteúdo.

Esta classe ainda possui propriedades que retornam informações, tais como:

- Nome do nodo corrente;
- Conteúdo do nodo corrente.

5.4.3 A Classe XmlWriter

Esta classe define uma interface para permitir a escrita de documentos XML, fornecendo, assim como a classe XmlReader, uma forma unilateral para a construção de documentos. O exemplo abaixo ilustra a utilização desta classe para a geração de um documento XML:

```
public void WriteDocument(XmlWriter writer)
{
```

```
  writer.WriteStartDocument();
  writer.WriteComment("generated by SampleWriter");
  writer.WriteProcessingInstruction("hack", "on person");
  writer.WriteStartElement("p", "person", "urn:person");
  writer.WriteStartElement("name", "");
  writer.WriteString("joebob");
  writer.WriteEndElement();
  writer.WriteElementInt16("age", "", 28);
  writer.WriteEndElement();
  writer.WriteEndDocument();
}
```

O fragmento de código acima, descrito na linguagem C#, gera o documento XML que pode ser serializado da seguinte forma:

```
<?xml version="1.0"?>
<!--sample person document-->
<?hack on person?>
<p:person xmlns:p="urn:person">
  <name>joebob</name>
  <age unit="year">28</age>
</p:person>
```

XmlTextWriter e XmlNodeWriter são duas implementações existentes da classe XmlWriter. Estas implementações são exatamente análogas às implementações para leitura, trabalhando apenas em direções opostas.

A classe XmlWriter permite a escrita de todos os construtores padrões contidos no Infoset[5] (tais como elementos, atributos e instruções de processamento) utilizando o método correspondente (por exemplo WriteStartDocument, WriteStartElement, WriteProcessingInstruction).

A XmlTextWriter é uma classe concreta derivada da classe XmlWriter que permite escrever cadeias de caracteres.

[5]Infoset (XML Information Set) é um conjunto de dados abstrato para documentos XML.

5.4.4 A Classe XmlNavigator

A classe XmlNode fornece os métodos básicos para navegação na árvore DOM. A navegação em documentos pode ser realizada através de uma classe chamada XmlNavigator, a qual fornece um mecanismo de navegação genérica.

Assim como as classes XmlReader e XmlWriter, esta classe é uma classe básica abstrata que define a funcionalidade comum para a implementação de navegadores. A utilização desta classe é vantajosa sobre a utilização da API DOM visto que implementações típicas desta classe não necessitam carregar a estrutura inteira do documento na memória, ao passo que implementações DOM necessitam deste trabalho.

A classe XmlNavigator também fornece uma interface mais intuitiva quando a aplicação trabalha com expressões de consulta XPath. A classe fornece dois métodos para seleção de nodos: o método *Select* e o método *SelectSingle*, ambos aceitos numa expressão XPath, onde são avaliados para identificar um conjunto de nodos.

Uma classe concreta derivada da classe XmlNavigator é a DocumentNavigator, a qual foi projetada especificamente para navegação em árvores DOM. Quando uma instância desta classe é criada, basta passar para ela uma referência do documento XML pelo qual se deseja navegar. Por exemplo:

```
XmlDocument doc = new XmlDocument();
doc.Load("person.xml");
DocumentNavigator nav = new DocumentNavigator(doc);
nav.Select("/person/name");
```

É possível acessar informações a respeito dos nodos navegados através das propriedades da classe XmlNavigator.

5.5 Classes da API XQuery

A API XQuery do Framework .NET foi utilizada neste trabalho para realizar uma equiparação com as linguagens de consulta a banco de dados relacionais, por

exemplo, a linguagem SQL. As linguagens relacionais possuem um embasamento teórico bem definido e muito bem estudado já há vários anos, enquanto as pesquisas para linguagens de consulta a dados XML estão apenas no começo em comparação às relacionais.

O fato é que muitas funções utilizadas nas linguagens de consulta relacionais se fazem necessárias no ambiente XML tais como extração, conversão, transformação e integração de dados e, por este motivo a API XQuery é usada neste trabalho a fim de proporcionar uma visão prática.

A API XQuery da versão beta do Framework .NET possui quatro classes essenciais:

- *XQueryNavigator* – é a representação navegável de um documento XQuery;

- *XQueryNavigatorCollection* – deve conter um XQueryNavigator para cada documento referenciado pelas palavras-chaves XQuery *FOR/LET*;

- *XQueryDocument* – classe de objeto usada para armazenar de maneira eficiente os dados a serem consultados;

- *XQueryExpression* – encapsula uma expressão XQuery e possibilita a execução da expressão sobre um XQueryNavigatorCollection.

Um exemplo da utilização de duas destas classes da API XQuery pode ser visto na implementação realizada neste trabalho, no seguinte trecho de código:

```
XQueryNavigatorCollection col = new XQueryNavigatorCollection();
col.AddNavigator(this.txtbox2.Text, this.txtbox4.Text);
XQueryExpression expr = new XQueryExpression(this.txtbox1.Text);
this.txtbox3.Text = expr.Execute(col).ToXml();
```

Na primeira linha é criada uma instância para *XQueryNavigatorCollection* denominada *col*, a qual é usada para possibilitar a navegação através do método

AddNavigator invocado na segunda linha, tendo como parâmetros o documento XML (referenciado na caixa de texto *txtbox2*) e o alias para este documento XML (referenciado na caixa de texto *txtbox4*). O alias é o apelido para o arquivo XML a ser usado na expressão de consulta.

Na terceira linha é criada uma instância para a classe *XQueryExpression* chamada de *expr*, tendo como parâmetro o texto contido na caixa de texto *txtbox1*, sendo este parâmetro a expressão de consulta em si. A quarta linha é a execução da expressão de consulta sobre o arquivo XML.

5.6 Extensão à API XQuery

Através do estudo da API XQuery do Framework .NET foi possível visualizar de forma concreta a implementação de uma linguagem de consulta utilizada para o padrão XML. Este estudo possibilitou ainda a equiparação das funcionalidades encontradas na linguagem de consulta relacional padrão SQL e também a constatação de algumas deficiências com relação à referida API no que diz respeito às funcionalidades esperadas numa linguagem de consulta.

Com base neste estudo foi possível verificar que algumas destas deficiências podem ser supridas através da linguagem de programação básica do Framework .NET, a linguagem C#, utilizando-se de recursos disponíveis em outras API's disponíveis e recursos da linguagem XSL.

A intenção maior deste trabalho é demonstrar a possibilidade de acoplar funcionalidades de outras linguagens de consulta, por exemplo a XSL, à funcionalidade da linguagem XQuery, gerando assim uma aplicação final de maior poder através dos recursos disponíveis em ambas as linguagens.

Este trabalho tem como ênfase o motor de consulta XQuery apresentado pela Microsoft através de sua API. Para tal apresentação, foi desenvolvida uma

aplicação com a utilização de um formulário simples, sobre o qual são executadas consultas de exemplo em um documento XML usado como fonte de dados.

O projeto inicial deste trabalho foi realizado sobre uma interface básica de prompt de comando, onde o usuário necessitava digitar arquivos de referência contendo a consulta, a fonte de dados XML e um alias para esta fonte de dados. Posteriormente foi desenvolvida a interface gráfica a fim de proporcionar um melhor entendimento do conjunto do trabalho. Esta interface gráfica pode ser vista na figura abaixo:

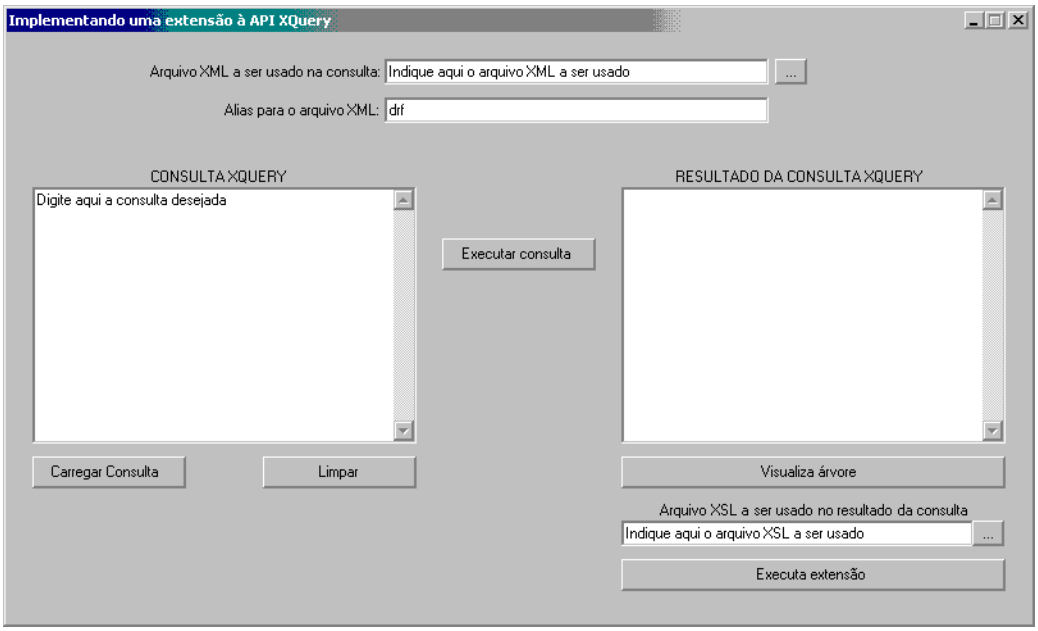


Figura 28 – Interface de implementação.

O parâmetro "*Arquivo XML a ser usado na consulta*" refere-se à indicação de um arquivo XML, incluindo o seu caminho, que servirá de fonte de dados para a implementação da consulta. O "*Alias para o arquivo XML*" indica um apelido para o arquivo XML e seu caminho, o qual será referenciado na expressão de consulta. Na caixa de texto "*CONSULTA XQUERY*" deverá ser digitada a consulta, nos padrões da API XQuery, a qual será avaliada pelo motor de consulta e executada contra o arquivo XML referenciado.

Após a execução da consulta, caso a mesma seja bem sucedida, o resultado desta será apresentado na caixa de texto "*RESULTADO DA CONSULTA XQUERY*". O

botão visualizar árvore serve para uma visualização da estrutura do resultado da consulta em termos do conceito de DOM.

O "*Arquivo XSL a ser usado no resultado da consulta*" será aplicado, em conjunto com as funções estendidas através da programação com a linguagem C#, ao resultado da consulta XQuery, fornecendo uma visão do resultado ao usuário final, possibilitando desta forma o acoplamento de funções não existentes na API XQuery. As linhas de código que implementam as funções desejadas são apresentadas a seguir:

```
//-----------------------------------------------------------------------------------

//Classe padrão definida para aplicar funções extendidas
//à linguagem XQuery com auxílio da XSL
//---------------------------------------------------------
public class Extensao{


  //Função que executa o cálculo de dias trabalhados
  //-------------------------------------------------
  public string CalculaDias(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;

    TimeSpan DiasTrabalhados = DataAtual.Subtract(DataArquivoXML) ;

    return DiasTrabalhados.TotalDays.ToString() ;

  }


  //Função que executa o cálculo de horas trabalhadas
  //-------------------------------------------------
  public string CalculaHoras(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;

    TimeSpan HorasTrabalhadas = DataAtual.Subtract(DataArquivoXML) ;

    return HorasTrabalhadas.TotalHours.ToString() ;

  }


  //Função que executa o cálculo de minutos trabalhados
  //-------------------------------------------------
  public string CalculaMinutos(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;

    TimeSpan MinutosTrabalhados = DataAtual.Subtract(DataArquivoXML) ;
```

```
    return MinutosTrabalhados.TotalMinutes.ToString() ;
  }

  //Função que executa o cálculo de porcentagem
  //sobre o salário do funcionário
  //-------------------------------------------
  public string NovoSalario(string DadoXSL, string Aumento){

    Double SalarioAtual = Double.Parse(DadoXSL) ;

    Double PercentualAumento = Double.Parse(Aumento) ;

    Double SalarioNovo = SalarioAtual * PercentualAumento ;

    return SalarioNovo.ToString("c") ;

  }

}
```

Definidas as funções desejadas, criam-se as instâncias das classes para aplicação do arquivo XSL com acesso as funções criadas anteriormente:

```
// Criação dos objetos a serem utilizados
// ---------------------------------------
XslTransform MeuXSLTrans = new XslTransform();
XsltArgumentList ArgsXSL = new XsltArgumentList();
Extensao ObjExtendido = new Extensao();

// Aplicação do XSL extendido
// --------------------------
MeuXSLTrans.Load(arquivoXSL);
ArgsXSL.AddExtensionObject("http://ObjExtendido", ObjExtendido);
```

Feito isso, basta acoplar ao arquivo XSL a chamada às funções através das instâncias das classes criadas no passo anterior:

```
<xsl:transform
      version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:MinhaExtensao="http://ObjExtendido">
.
.
.
<B><xsl:value-of select="MinhaExtensao:CalculaDias(ADMISSAO)"/></B>
```

O código comentado para a aplicação completa pode ser visto no anexo I deste trabalho. O grande problema deste motor de consulta XQuery desenvolvido pela Microsoft é que ele não suporta a multiplicação de números inteiros por números de

ponto flutuante, impossibilitando cálculos simples em consultas também simples para apresentação dos dados ao usuário final.

As funções acopladas ao código-fonte da aplicação demonstram, além da possibilidade de estender a funcionalidade da API XQuery, o grande benefício do trabalho em complemento com a linguagem XSL, também considerada uma linguagem de consulta, viabilizando resultados ao usuário final.

5.7 Justificativa

Algumas API's, também em estado de desenvolvimento, têm sido apresentadas para uso em outros ambientes operacionais, por exemplo, a API Java XQEngine (www.fatdog.com), porém com funcionalidades totalmente semelhantes. A escolha da API XQuery se deu pelo fato do ambiente de trabalho no qual os testes puderam ser realizados ser na sua totalidade um ambiente operacional Microsoft.

CAPÍTULO VI

6. CONCLUSÕES

Neste trabalho, propôs-se uma extensão à uma implementação da linguagem *XQuery* (*API XQuery*) baseada na especificação W3C *Working Draft XQuery*. Tal extensão teve como fundamentação básica a linguagem XQuery e a linguagem XSL a fim de realizar uma apresentação de dados através da utilização de um *browser* de navegação com um *parser* XML embutido.

Um problema discorrido nesta pesquisa foi a dificuldade de se estabelecer o conceito de esquema em dados semi-estruturados a fim de facilitar a extração e consulta de dados em fontes de dados com esta característica, surgindo aí a alternativa de utilização de fontes de dados XML como requisito básico para um trabalho eficiente quanto às operações de extração e consulta a dados.

O problema de consultar dados XML foi apresentado através dos requisitos necessários em uma linguagem de consulta com esta finalidade, ilustrando-se para isso uma comparação e um estudo das principais linguagens de consulta encontradas na literatura: XML-QL, Lorel, XML-GL, XQL e XQuery.

Por fim, apresentou-se a implementação da *API XQuery* do *Framework .NET* com a demonstração da possibilidade de extensão à funcionalidade final da

linguagem de consulta, utilizando-se para isso a linguagem de transformação XSL e a linguagem de programação C#, bem como as classes XML preexistentes no *framework*.

A partir da implementação realizada neste trabalho foi possível verificar que as diversas pesquisas acerca das linguagens de consulta a XML caminham no sentido de fornecer os mesmos requisitos funcionais atualmente observados nas linguagens de consulta a bancos de dados.

Muitas das funcionalidades ainda encontram-se em fase de pesquisa, pois os estudos acerca destas linguagens são extremamente recentes. É muito provável que algumas mudanças nos padrões definidos até o momento pelo W3C para a linguagem XQuery ainda ocorram com o propósito de implementação e ajustes para algumas destas funcionalidades, o que acarretará numa mudança da API utilizada, determinando assim a necessidade de novos estudos para uma nova extensão às funcionalidades desta possível nova API.

As principais contribuições deste trabalho resumem-se nos seguintes itens:

1. A tecnologia XML juntamente com uma linguagem padrão completamente especificada (XQuery) oferecem grande facilidade na construção de aplicações que forneçam interoperabilidade bem como uma filtragem de dados extremamente simples e poderosa;

2. As diversas pesquisas acerca de linguagens de consulta a XML indicam um futuro promissor tanto para a tecnologia XML quanto para uma especificação padrão de uma linguagem de consulta;

3. Aplicações que se utilizam destes recursos (XML + XQuery) são facilmente estendidas para fornecer outras funcionalidades;

4. A utilização de uma linguagem de consulta pode contribuir com a diminuição do tráfego de rede, já que vai selecionar apenas a parte de interesse do usuário.

Ao longo do desenvolvimento deste trabalho, diversas propostas de linguagens de consulta a XML foram elaboradas e suas evoluções se deram de forma extremamente rápida, porém não existe ainda um padrão bem definido.

É importante destacar que os estudos minuciosos sobre a padronização de uma linguagem de consulta a XML (possivelmente a XQuery) é de extrema importância tanto para a comunidade de pesquisa a banco de dados quanto para a comunidade de pesquisa às tecnologias WEB. A comunidade de pesquisa a banco de dados possui na tecnologia XML uma importante ferramenta de integração de ambientes distribuídos, e uma linguagem de consulta para tal ambiente é o fator principal para extração, conversão, transformação e integração de dados. Já a comunidade de pesquisa às tecnologias WEB têm na XML o recurso essencial para estruturar a informação, de tal forma que esta informação seja qualificada (selecionada e pesquisada) e acessada da maneira mais conveniente possível, e uma linguagem de consulta nesta situação é também a parte essencial para o bom desempenho da qualificação e acesso da informação como um todo.

Em comparação à outro trabalho existente de igual teor nesta área (*XQEngine - www.fatdog.com*) pode-se dizer que o desenvolvimento deste foi beneficiado pelos seguintes fatores:

1. Para implementar a *API XQuery* e a respectiva extensão à sua funcionalidade foram utilizados o *Framework .NET* e a linguagem de programação *C#* (*C-Sharp*) que são componentes de comunicação nativos com o sistema operacional do ambiente de trabalho onde foi desenvolvida esta pesquisa;

2. O *Framework .NET* forneceu grande parte das classes utilizadas na programação da aplicação através da linguagem *C#*. Através do projeto de suas bibliotecas de classes este *Framework* facilitou a inter-operação entre os serviços necessários na aplicação;

3. A linguagem *C#* foi utilizada por permitir o uso eficiente das bibliotecas de classes fornecidas pelo *Framework .NET* bem como por fornecer os recursos necessários e suficientes para o uso extensivo da *API Microsoft.XML.XQuery*.

Além disso, a *API XQEngine* está em conformidade com a especificação da linguagem *XQuery* realizada em dezembro de 2001, a qual está defasada em relação às últimas especificações estabelecidas pelo *Working Draft do W3C*.

Alguns possíveis trabalhos futuros que seriam interessantes e relevantes inerentes às linguagens de consulta e suas implementações seriam:

1. Integração da implementação da linguagem com os *browsers* atualmente disponíveis no mercado;

2. Modelo de integração entre linguagens distintas para origens de dados distintas;

3. Especificação de um analisador prévio de um DTD relacionado a um documento XML a ser consultado.

REFERÊNCIAS BIBLIOGRÁFICAS

Abiteboul, S., Quass, D., McHugh, J., and Wiener, J. *The lorel query language for semistructured data*. Int. J. Dig. Libraries. Ano de publicação: 1997.

Abiteboul S., Buneman P. e Suciu D. *Data on the WEB: From Relations to Semistructured Data and XML*. Ano de publicação 1999.

Abiteboul S., Segoufin L. e Vianu V. *Representing and Quering XML with Incomplete Information*. ACM Press, páginas 150-161, Series-Proceeding-Article. Ano de publicação: 2001.

Alon N. et al. *XML with Data Values: Typecheching Revisited*. ACM Press, páginas 138-149, Series-Proceeding-Article. Ano de publicação: 2001.

AT&T1999 - http://www.research.att.com/sw/tools/xmlql/ - Site da WEB consultado em setembro de 2002.

Babcock C. *XML Databases Offer Greater Search Capabilities*. Interactive Week. Ano de publicação: 2001.

Bonifati A. e Ceri S. *Comparative Analysis of Five XML Query Languages*. Dipartimento di Elettronica e Informazione, Politecnico di Milano. SIGMOD Record. Ano de publicação: 2000.

Buneman P. et al. *Keys for XML*. ACM Press, páginas 201-210, Series-Proceeding-Article. Ano de publicação: 2001.

Ceri S., Comai S., Damiani+ E., Fraternali P., Paraboschi S. e Tanca L. *XML-GL: a Graphical Language for Querying and Restructuring XML Documents*. Dipartimento di Elettronica e Informazione, Politecnico di Milano. SIGMOD Record. Ano de publicação: 1999.

Chamberlin, D., Robie, J., e Florescu, D. *Quilt: A xml query language for heterogeneous data sources*. In Proceedings of WEBdb 2000 – The WEB and Databases Workshop. Ano de publicação: 2000.

Chawathe S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J. e Widom J. *The TSIMMIS project: Integration of heterogeneous information sources*. IPJS, páginas 7 a 18. Ano de publicação 1994.

Cluet, S., Jacqmin, S., and Simeon, J. *The New YaTL: Design and Specifications*. Technical Rep. INRIA. Ano de publicação: 1999.

Cohen S., Kanza Y., Kogan Y., Nutt W., Sagiv Y., e Serebrenik A. *Equix – easy querying in xml databases*. In Proceedings of WEBdb'98 – The WEB and Databases Workshop. Ano de publicação: 1998.

Comai S., Damiani E., e Fraternali P. *Computing Graphical Queries over XML Data*. ACM Transactions on Information Systems, Vol. 19, nro. 4, Outubro 2001, páginas 371-430.

Christophides V., Cluet S. e Siméon J. *On Wrapping Query Languages and Efficient XML Integration*. ACM Press, páginas 141-152, Series-Proceeding-Article. Ano de publicação: 2000.

DeRose S. J. *Xquery: A unified syntax for linking and querying general xml documents*. In Proceedings of Query Languages. Ano de publicação: 1998.

Deutsch A., Fernandez M., Florescu D., Levy A. e Suciu D. *XML-QL: A query language for xml*. In Proceedings of QL'98 – The Languages Workshop, Cambridge – Mass.. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/. Ano de publicação: 1998.

Dragan R. V. *XML your data*. PC Magazine. Ano de publicação: 2001.

Fan W. e Libkin L. *On XML Integrity Constraints in the Presence of DTDs*. ACM Press, páginas 114-125, Series-Proceeding-Article. Ano de publicação: 2001.

Fegaras L. e Elmasri R. *Query Engine for WEB-Accessible XML data*. University of Texas at Arlington. Ano de publicação: 2001.

Fernandez M., Florescu D., Kang J., Levy e Suciu D. *Catching the Boat with Strudel: A WEB-site Management System*. Em Proc. ACM-SIGMOD Int. Conf. On the Management of Data, páginas 549 a 552. Ano de publicação: 1998.

Fontana J. *Database optimized to handle XML aplications*. Network World. Ano de publicação: 2001.

Haas L., Kossmann D., Wimmers E. e Yang J. *Optimizing Queries across Diverse Data Sources*. Em Proc. 23rd Int. Conf. On Very Large Data Bases, páginas 276 a 285. Ano de publicação: 1997.

Holland R. *Breaking the XML data bottleneck*. eWeek. Ano de publicação: 2001.

Hull R. e Zhou G. *A Framework for Supporting Data Integration using the Materialized and Virtual Approaches*. Em Proc. ACM-SIGMOD, Int. Conf. on the Management of Data, páginas 481 a 492. Ano de publicação: 1996.

Jung F. *XML Backgrounder, technology and applications*. Software AG Corporate Headquarters, páginas 4 a 14. Ano de publicação 2000.

Lawrence R. e Barker K. *Integrating Relational Database Schemas using a Standardized Dictionary*. ACM Press, páginas 225-230, Series-Proceeding-Article. Ano de publicação: 2001.

Levy A., Rajaraman A. e Ordille J. *Querying Heterogeneous Information Sources Using Source Descriptions*. Em Proc. 22nd Int. Conf. on Very Large Databases, páginas 251 a 262. Ano de publicação: 1996.

Light R. *Iniciando em XML*. São Paulo: Makron Books. Ano de publicação: 1999.

Liu L., Yan L. L. e Özsu M. T.. Interoperability in Large-Scale Distributed Information Delivery Systems; páginas 246-280.

Ludaescher B., Papakonstantinou Y., e Velikhov P. *Navigation-driven evaluation of virtual mediated views*. In Proceedings of the sixth International Conference on Extending Database Technology (EDBT) (Konstanz, Germany, March), Lecture Notes in Computer Science, vol. 1777, Springer-Verlage, New York. Ano de publicação: 2000.

Maier D. *Database desiderata for an xml query language.* In Proceedings of QL'98 – The Query Language Workshop – Cambridge, Mass. Ano de publicação: 1998.

McGrath S. *XML – Aplicações práticas.* Rio de Janeiro, editora Campus. Ano de publicação: 1999.

McHugh J., Abiteboul S., Goldman R., Quass D. e Widom J. *Lore: A database management system for semistructured data*. Sigmod Record, páginas 54 a 66. Ano de publicação: 1997.

Munroe K. D., Ludaescher B., e Papakonstantinou Y. *Blended browsing and querying of xml in lazy mediator system*. Konstanz, Germany, March. Ano de publicação: 2000.

Murata M. *Extended Path Expressions for XML*. ACM Press, páginas 126-137, Series-Proceeding-Article. Ano de publicação: 2001.

Myllymaki J. *Effective WEB Data Extraction with Standard XML Technologies*. In Proc. ACM POD´s, páginas 689 a 695. Ano de publicação 2001.

Naacke H., Gardarin G. e Tomasic A. *An Extensible Cost Model for Heterogeneous Data Sources*. Em Proc. 14th IEEE Int. Conf. on Data Engineering, páginas 351 a 360. Ano de publicação: 1998.

Oppel K. *The Information Server for Eletronic Business*. Software AG Corporate Headquarters, páginas 5 a 18. Ano de publicação 1999.

Özsu M. T. e Valduriez P. *Principles of Distributed Database Systems*; páginas 620-656.

Paredaens J., Peelman P. e Tanca L. *G-Log: A Graph-Based Query Language*; páginas 436-453, Vol. 7, número 3. IEEE Transactions on Knowledge and Data Eng.

Quass, D. *Ten features necessary for an xml query language*. In Proceedings of QL'98 – The Query Language Workshop – Cambridge, Mass. Ano de publicação: 1998.

Robie, J., e Schach, D. *XML Query Language (xql)*. In Query Languages. Ano de publicação: 1998.

Siméon J., Kuper M. G. e Fan W. *A Unified Constraint Model for XML*. ACM Press, páginas 179-190, Series-Proceeding-Article. Ano de publicação: 2001.

Tatarinov I. et al. *Updating XML*. ACM Press, páginas 413-424, Series-Proceeding-Article. Ano de publicação: 2001.

Tomasic A., Raschid L. e Valduriez P. *Scaling Access to Distributed Heterogeneous Data Sources with Disco*. Em IEEE Trans. Knowl. and Data Eng. Ano de publicação: 1998.

Vianu V. *A WEB Odyssey: from Codd to XML*. In Proc. ACM POD´s, páginas 1 a 13. Ano de publicação 2001.

W3Ca - http://www.w3c.org/Style/xssl. - Site da WEB consultado em novembro de 2001.

W3Cb - http://www.w3.org/XML/ - Site da WEB consultado de outubro de 2001 a julho de 2002.

W3Cc - http://www.w3.org/TR/xmlschema-1/ - Site da WEB consultado em outubro de 2001.

W3Cd - http://www.w3.org/TR/query-datamodel/. Site da WEB consultado em maio de 2002.

W3Ce - *An introdution to xsl*. http://www.w3c.org/Style/xssl. Site da WEB consultado em maio de 2002.

W3Cf - http://www.w3.org/TR/xquery/ site da WEB consultado em abril de 2002.

W3C1999a - http://www.w3c.org/TR/1999/07/REC-MathML-19990707 - Site da WEB consultado em maio de 2001.

W3C1998a - http://www.w3c.org/TR/REC-smil-19980615. - Site da WEB consultado em abril de 2001.

W3C2000a - http://www.w3c.org/TR/2000/REC-xHTML-20000126. - Site da WEB consultado em abril de 2001.

XMLSoftware2001 - http://www.xmlsoftware.com/database/ - Site da WEB consultado em outubro de 2001.

XMLSpy2000 - http://www.xmlspy.com/ - Site da WEB consultado em dezembro de 2001.

XQEngine2002 - http://www.fatdog.com/ - Site da WEB consultado em agosto de 2002.

Zhang C. et al. *On Supporting Containment Queries in Relational Database Management Systems*. ACM Press, páginas 425-436, Series-Proceeding-Article. Ano de publicação: 2001.

ANEXO I

Neste anexo estão contidos os itens referentes à primeira parte do desenvolvimento do trabalho, a qual refere-se à implementação da linguagem de consulta XQuery e respectivos testes com informações predefinidas, as quais são relacionadas abaixo:

- Código-fonte contendo a implementação da API XQuery do Framework .NET bem como a apresentação da extensão à funcionalidade desta API, desenvolvido com a utilização da linguagem C#.

- Consultas implementadas para testes.

- Arquivo de dados XML para execução das consultas.

Código-fonte de Implementação

Para implementar a API XQuery e a respectiva extensão à sua funcionalidade foram utilizados o Framework .NET e a linguagem de programação C# (C-Sharp).

O Framework .NET forneceu grande parte das classes utilizadas na programação da aplicação através da linguagem C#. Através do projeto de suas

bibliotecas de classes este Framework facilitou a inter-operação entre os serviços necessários na aplicação.

A linguagem C# foi utilizada por permitir o uso eficiente das bibliotecas de classes fornecidas pelo Framework .NET bem como por fornecer os recursos necessários e suficientes para o uso extensivo da API Microsoft.XML.XQuery.

Segue abaixo o referido código, fazendo uso de comentários precedidos por // (convenção adotada na linguagem C#):

```
// Declaração das API´s a serem utilizadas pela aplicação
// ------------------------------------------------------
using Microsoft.Xml.XQuery;
using System;
using System.ComponentModel;
using System.Drawing;
using System.IO;
using System.Xml ;
using System.Xml.Xsl ;
using System.Xml.XPath ;
using System.Windows.Forms;


// Classe criada para realizar a interface principal
// -------------------------------------------------
public class meuformulario : Form
{

// Declaração dos objetos a serem utilizados
// -----------------------------------------
  private TextBox txtbox1;// CONSULTA XQUERY
  private TextBox txtbox2;// ARQUIVO XML
  private TextBox txtbox3;// RESULTADO DA CONSULTA XQUERY
  private TextBox txtbox4;// ALIAS DO ARQUIVO XML QUE SERÁ USADO NA CONSULTA
  private TextBox txtbox5;// ARQUIVO XSL

  private Button botao1;   // CARREGA O ARQUIVO DE CONSULTA
  private Button botao2;   // LIMPA A CAIXA DE TEXTO DA CONSULTA
  private Button botao3;   // CAMINHO E NOME DO ARQUIVO XML
  private Button botao4;   // EXECUTA A CONSULTA
  private Button botao5;   // CAMINHO E NOME DO ARQUIVO XSL
  private Button botao6;   // EXECUTA A EXTENSAO XQUERY/XSL
  private Button botao7;   // VISUALIZA A SAÍDA DA CONSULTA EM FORMA DE ÁRVORE

  private Label label1;    // "Arquivo XML a ser usado na consulta:"
  private Label label2;    // "CONSULTA XQUERY"
  private Label label3;    // "RESULTADO DA CONSULTA XQUERY"
  private Label label4;    // "Alias para o arquivo XML:"
  private Label label5;    // "Arquivo XSL a ser usado no resultado da consulta"


  public string txtConsulta
  {
    get{return this.txtbox1.Text;}
  }
```

```
// Objeto criado para concretizar a interface
// ------------------------------------------
  public meuformulario()
  {

// Definição dos valores das variáveis estáticas
// ---------------------------------------------
    int lb = 120;  // largura padrao para um botao
    int ab = 25; // altura padrao para um botao
    int lb3 = 25;  // largura do botao3
    int ab3 = 20; // altura do botao3
    int lfrm = 800; // largura do formulário
    int afrm = 460;  // altura do formulário
    int ltxt = 300;  // largura da caixa de texto
    int atxt = 200;  // altura da caixa de texto
    int llbl = 250;  // largura do label
    int albl = 25; // altura do label
    int x = 10;// coordenada x para o label1
    int y = 10;   // coordenada y para o label1


// Criação  dos objetos a serem usados no formulario
// -------------------------------------------------
    this.botao1 = new Button ();
    this.botao2 = new Button ();
    this.botao3 = new Button ();
    this.botao4 = new Button ();
    this.botao5 = new Button ();
    this.botao6 = new Button ();
    this.botao7 = new Button ();

    this.txtbox1 = new TextBox();
    this.txtbox2 = new TextBox();
    this.txtbox3 = new TextBox();
    this.txtbox4 = new TextBox();
    this.txtbox5 = new TextBox();

    this.label1 = new Label();
    this.label2 = new Label();
    this.label3 = new Label();
    this.label4 = new Label();
    this.label5 = new Label();


// Definição das caracteristicas do formulario
// -------------------------------------------
    this.Text = "Implementando uma extensão à API XQuery";
    this.ClientSize = new System.Drawing.Size(lfrm, afrm);
    //this.WindowState = FormWindowState.Maximized;
    this.FormBorderStyle = FormBorderStyle.FixedDialog;
    this.MaximizeBox = false;
    this.MinimizeBox = true;
    this.StartPosition = FormStartPosition.CenterScreen;


// ----------------------------------------LABELS---------------------------------------

// Label "Arquivo XML a ser usado na consulta:"
// --------------------------------------------
    this.label1.Location = new Point(x + 100, y + 13);
    this.label1.Name = "label1";
```

```
    this.label1.Size = new Size(llbl, albl);
    this.label1.Text = "Arquivo XML a ser usado na consulta:";

// Label "CONSULTA XQUERY"
// ----------------------
    this.label2.Location = new Point(x + 100, y + 95);
    this.label2.Name = "label2";
    this.label2.Size = new Size(llbl, albl);
    this.label2.TabIndex = 1;
    this.label2.Text = "CONSULTA XQUERY";

// Label "RESULTADO DA CONSULTA XQUERY"
// -----------------------------------
    this.label3.Location = new Point(x + 510, y + 95);
    this.label3.Name = "label3";
    this.label3.Size = new Size(llbl, albl);
    this.label3.TabIndex = 1;
    this.label3.Text = "RESULTADO DA CONSULTA XQUERY";

// Label "Alias para o arquivo XML:"
// --------------------------------
    this.label4.Location = new Point(x + 158, y + 43);
    this.label4.Name = "label4";
    this.label4.Size = new Size(llbl, albl);
    this.label4.Text = "Alias para o arquivo XML:";

// Label "Arquivo XSL a ser usado no resultado consulta"
// ----------------------------------------------------
    this.label5.Location = new Point(x + 498, y + 355);
    this.label5.Name = "label5";
    this.label5.Size = new Size(llbl + 100, albl);
    this.label5.Text = "Arquivo XSL a ser usado no resultado da consulta";



// ----------------------------------CAIXAS DE TEXTO---------------------------------

// Caixa de texto para a consulta
// ------------------------------
    this.txtbox1.Multiline = true;
    this.txtbox1.ScrollBars = ScrollBars.Vertical;
    this.txtbox1.AcceptsReturn = true;
    this.txtbox1.AcceptsTab = true;
    this.txtbox1.WordWrap = true;
    this.txtbox1.Text = "Digite aqui a consulta desejada";
    this.txtbox1.Location = new Point (x + 10, y + 110);
    this.txtbox1.Size = new Size (ltxt, atxt);
    this.txtbox1.TabIndex = 3;

// Caixa de texto para indicar o arquivo XML usado
// -----------------------------------------------
    this.txtbox2.Multiline = false;
    this.txtbox2.ScrollBars = ScrollBars.Horizontal;
    this.txtbox2.AcceptsReturn = true;
    this.txtbox2.AcceptsTab = true;
    this.txtbox2.Text = "Indique aqui o arquivo XML a ser usado";
    this.txtbox2.Location = new Point (x + 285, y + 10);
    this.txtbox2.Size = new Size (ltxt, atxt);
    this.txtbox2.TabIndex = 0;
    this.txtbox2.Focus();

// Caixa de texto para o RESULTADO da consulta
// -------------------------------------------
```

```
    this.txtbox3.Multiline = true;
    this.txtbox3.ScrollBars = ScrollBars.Vertical;
    this.txtbox3.AcceptsReturn = true;
    this.txtbox3.AcceptsTab = true;
    this.txtbox3.WordWrap = true;
    this.txtbox3.Text = String.Empty;
    this.txtbox3.Location = new Point (x + 470, y + 110);
    this.txtbox3.Size = new Size (ltxt, atxt);
    this.txtbox3.TabIndex = 7;

// Caixa de texto para indicar o ALIAS para o arquivo XML
// ------------------------------------------------------
    this.txtbox4.Multiline = false;
    this.txtbox4.ScrollBars = ScrollBars.Horizontal;
    this.txtbox4.AcceptsReturn = true;
    this.txtbox4.AcceptsTab = true;
    this.txtbox4.Text = "drf";
    this.txtbox4.Location = new Point (x + 285, y + 40);
    this.txtbox4.Size = new Size (ltxt, atxt);
    this.txtbox4.TabIndex = 2;

// Caixa de texto para indicar o arquivo XSL usado
// -----------------------------------------------
    this.txtbox5.Multiline = false;
    this.txtbox5.ScrollBars = ScrollBars.Horizontal;
    this.txtbox5.AcceptsReturn = true;
    this.txtbox5.AcceptsTab = true;
    this.txtbox5.Text = "Indique aqui o arquivo XSL a ser usado";
    this.txtbox5.Location = new Point (x + 470, y + 370);
    this.txtbox5.Size = new Size (ltxt - 25, atxt);
    this.txtbox5.TabIndex = 8;


// ----------------------------------------BOTOES----------------------------------------

// Botao "Carregar Consulta"
// -------------------------
    this.botao1.Text = "Carregar Consulta";
    this.botao1.Location = new Point (x + 10, y + 320);
    this.botao1.Size = new Size (lb, ab);
      this.botao1.Click += new EventHandler(this.botao1_click);
    this.botao1.TabIndex = 4;

// Botao "Limpar"
// --------------
    this.botao2.Text = "Limpar";
    this.botao2.Location = new Point (x + 190, y + 320);
    this.botao2.Size = new Size (lb, ab);
      this.botao2.Click += new EventHandler(this.botao2_click);
    this.botao2.TabIndex = 5;

// Botao para carregar o arquivo XML a ser usado
// ---------------------------------------------
    this.botao3.Text = "...";
    this.botao3.Location = new Point (x + 590, y + 10);
    this.botao3.Size = new Size (lb3, ab3);
      this.botao3.Click += new EventHandler(this.botao3_click);
    this.botao3.TabIndex = 1;

// Botao "Executar consulta"
// -------------------------
    this.botao4.Text = "Executar consulta";
    this.botao4.Location = new Point (x + 330, y + 150);
```

```
    this.botao4.Size = new Size (lb, ab);
      this.botao4.Click += new EventHandler(this.botao4_click);
    this.botao4.TabIndex = 6;

// Botao para carregar o arquivo XSL a ser usado
// ---------------------------------------------
    this.botao5.Text = "...";
    this.botao5.Location = new Point (x + 744, y + 370);
    this.botao5.Size = new Size (lb3, ab3);
      this.botao5.Click += new EventHandler(this.botao5_click);
    this.botao5.TabIndex = 9;

// Botao "Executa extensão"
// ------------------------
    this.botao6.Text = "Executa extensão";
    this.botao6.Location = new Point (x + 470, y + 400);
    this.botao6.Size = new Size (lb + 180, ab);
      this.botao6.Click += new EventHandler(this.botao6_click);
    this.botao6.TabIndex = 10;

// Botao "Visualiza árvore"
// ------------------------
    this.botao7.Text = "Visualiza árvore";
    this.botao7.Location = new Point (x + 470, y + 320);
    this.botao7.Size = new Size (lb + 180, ab);
      this.botao7.Click += new EventHandler(this.botao7_click);
    this.botao7.TabIndex = 11;


// ------------------------------------------------------------------------------------

// Adicionando os controles no formulario principal
// ------------------------------------------------
    this.Controls.Add(botao1);
    this.Controls.Add(botao2);
    this.Controls.Add(botao3);
    this.Controls.Add(botao4);
    this.Controls.Add(botao5);
    this.Controls.Add(botao6);
    this.Controls.Add(botao7);

    this.Controls.Add(txtbox1);
    this.Controls.Add(txtbox2);
    this.Controls.Add(txtbox3);
    this.Controls.Add(txtbox4);
    this.Controls.Add(txtbox5);

    this.Controls.Add(label1);
    this.Controls.Add(label2);
    this.Controls.Add(label3);
    this.Controls.Add(label4);
    this.Controls.Add(label5);

  }


// ----------------------EVENTOS RELACIONADOS AOS CLICKS DOS BOTÕES-----------------------

// Evento do botao para carregar o arquivo de consulta
// ---------------------------------------------------
  private void botao1_click(object sender, System.EventArgs e)
  {
    Stream myStream;
```

```
    String consulta = String.Empty, q = String.Empty, str = String.Empty;

    OpenFileDialog ofd1 = new OpenFileDialog();
    ofd1.InitialDirectory = "c:\\xquery\\nakano\\consultas" ;
    ofd1.Title = "Selecione a consulta a ser carregada" ;
    ofd1.Filter = "All files (*.*)|*.*|XQU files (*.xqu)|*.xqu" ;
    ofd1.FilterIndex = 2 ;
    ofd1.RestoreDirectory = true ;
    ofd1.ShowHelp = false ;

    if(ofd1.ShowDialog() == DialogResult.OK)
    {
      if((myStream = ofd1.OpenFile())!= null)
      {
            StreamReader sr = new StreamReader( myStream );
            q = sr.ReadToEnd();
            sr.Close();
        myStream.Close();
        this.txtbox1.Text = q;
      }
    }
  }


// Evento do botao que limpa a caixa de texto de consulta
// ------------------------------------------------------
  private void botao2_click(object sender, System.EventArgs e)
  {
    this.txtbox1.Text = String.Empty;
  }


// Evento do botao que indica o caminho e o nome do arquivo XML a ser usado
// ------------------------------------------------------------------------
  private void botao3_click(object sender, System.EventArgs e)
  {
    Stream myStream;
    OpenFileDialog ofd1 = new OpenFileDialog();
    ofd1.InitialDirectory = "c:\\xquery\\nakano\\dadosXML" ;
    ofd1.Title = "Selecione o arquivo XML a ser utilizado" ;
    ofd1.Filter = "All files (*.*)|*.*|XML files (*.xml)|*.xml" ;
    ofd1.FilterIndex = 2 ;
    ofd1.RestoreDirectory = true ;
    ofd1.ShowHelp = false ;

    if(ofd1.ShowDialog() == DialogResult.OK)
    {
      if((myStream = ofd1.OpenFile())!= null)
      {
        this.txtbox2.Text = ofd1.FileName.ToString();
      }
    }
  }


// Evento do botao que executa a consulta XQuery
// ---------------------------------------------
  private void botao4_click(object sender, System.EventArgs e)
  {

    Cursor.Current = new Cursor("c:\\xquery\\testes\\cursor\\busy.cur");

    // Verifica se o arquivo XML indicado existe
```

```
    // ------------------------------------------
    if (this.txtbox2.Text == String.Empty || File.Exists(this.txtbox2.Text) == false)
    {
      MessageBox.Show("ARQUIVO XML NÃO ENCONTRADO!", "Localização do arquivo XML",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else
    {
      // Verifica se o alias foi digitado
      // --------------------------------
      if (this.txtbox4.Text.Trim() == String.Empty)
      {
        MessageBox.Show("ALIAS NÃO FOI DIGITADO!", "Alias para o arquivo XML",
          MessageBoxButtons.OK, MessageBoxIcon.Error);
      }

      else
      {
        // Verifica se alguma consulta foi carregada/digitada
        // --------------------------------------------------
        if (this.txtbox1.Text.Trim() == String.Empty)
        {
          MessageBox.Show("NENHUMA CONSULTA A SER PROCESSADA!", "Informação..."
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
          try
          {
            // Execução da consulta através da API XQuery
            // -------------------------------------------
              XQueryNavigatorCollection col = new XQueryNavigatorCollection();
              col.AddNavigator(this.txtbox2.Text, this.txtbox4.Text);
              XQueryExpression expr = new XQueryExpression(this.txtbox1.Text);
            this.txtbox3.Text = expr.Execute(col).ToXml();

          }
          catch (XQueryException erro)
          {
            MessageBox.Show(erro.Message,
            "Erro na sintaxe/semântica da consulta XQUERY",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
          }
        }
      }
    }

    Cursor.Current = Cursors.Default;

  }


// Evento do botao que indica o caminho e o nome do arquivo XSL a ser usado
// ------------------------------------------------------------------------
  private void botao5_click(object sender, System.EventArgs e)
  {
    Stream myStream;
    OpenFileDialog ofd1 = new OpenFileDialog();
    ofd1.InitialDirectory = "c:\\xquery\\nakano\\estilos" ;
    ofd1.Title = "Selecione o arquivo XML a ser utilizado" ;
    ofd1.Filter = "All files (*.*)|*.*|XSL files (*.xsl)|*.xsl" ;
    ofd1.FilterIndex = 2 ;
    ofd1.RestoreDirectory = true ;
```

```
    ofd1.ShowHelp = false ;

    if(ofd1.ShowDialog() == DialogResult.OK)
    {
      if((myStream = ofd1.OpenFile())!= null)
      {
        this.txtbox5.Text = ofd1.FileName.ToString();
      }
    }
  }


// Evento do botao que executa a XSL extendida
// -------------------------------------------
  private void botao6_click(object sender, System.EventArgs e)
  {

    // Verifica se o arquivo XSL indicado existe
    // -----------------------------------------
    if (this.txtbox5.Text == String.Empty || File.Exists(this.txtbox5.Text) == false)
    {
      MessageBox.Show("ARQUIVO XSL NÃO ENCONTRADO!", "Localização do arquivo XSL",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else
    {
      // Verifica se existe um resultado de consulta XQUERY
      // --------------------------------------------------
      if (this.txtbox3.Text.Trim() == String.Empty)
      {
        MessageBox.Show("NENHUM RESULTADO A SER PROCESSADO!", "Informação...",
          MessageBoxButtons.OK, MessageBoxIcon.Information);
      }

      else
      {
        // Execução da XSL extendida
        // -------------------------
        try
        {
          // Declaração dos objetos a serem usados
          // -------------------------------------
          String arquivoXML = "c:\\xquery\\testes\\tmp.xml";
          String arquivoXSL = this.txtbox5.Text;
          String arquivoHTML = "c:\\xquery\\testes\\xquery_xsl.htm";

          // Cria um arquivo temporário com o resultado da consulta XQUERY
          // -------------------------------------------------------------
          if (File.Exists(arquivoXML) == true)
          {
            File.Delete(arquivoXML);
          }
          TextWriter temp = File.AppendText(arquivoXML);
          temp.WriteLine(txtbox3.Text);
          temp.Close();

          // Criação dos objetos a serem utilizados
          // --------------------------------------
          XPathDocument MeuDocumentoXML = new XPathDocument(arquivoXML);
          XslTransform MeuXSLTrans = new XslTransform();
          XsltArgumentList ArgsXSL = new XsltArgumentList();
          Extensao ObjExtendido = new Extensao();
          XmlTextWriter MeuEscritor = new XmlTextWriter(arquivoHTML, null);
```

```
            // Aplicação do XSL extendido
            // -------------------------
            MeuXSLTrans.Load(arquivoXSL);
            ArgsXSL.AddExtensionObject("http://ObjExtendido", ObjExtendido);
            MeuXSLTrans.Transform(MeuDocumentoXML,ArgsXSL, MeuEscritor) ;
            MeuEscritor.Close() ;

            // Abrir o browser com o html extendido
            // ------------------------------------
            System.Diagnostics.Process.Start("iexplore.exe",
              "c:\\xquery\\testes\\xquery_xsl.htm");


          }
          catch (Exception erro)
          {
            MessageBox.Show(erro.Message,
              "Erro na execução da extensão XSL/XQUERY!",
              MessageBoxButtons.OK, MessageBoxIcon.Error);
          }
        }
      }

    }


// Evento do botao que visualiza o resultado da consulta em forma de árvore
// ------------------------------------------------------------------------
    private void botao7_click(object sender, System.EventArgs e)
    {

      Cursor.Current = new Cursor("c:\\xquery\\testes\\cursor\\busy.cur");

      // Verifica se existe um resultado de consulta XQUERY
      // --------------------------------------------------
      if (this.txtbox3.Text.Trim() == String.Empty)
      {
        MessageBox.Show("NENHUM RESULTADO A SER VISUALIZADO!", "Informação...",
          MessageBoxButtons.OK, MessageBoxIcon.Information);
      }

      else
      {
        try
        {
          // Início da conversão para apresentação em árvore
          // -----------------------------------------------
          String arquivoXML = "c:\\xquery\\testes\\tmp.xml";
          if (File.Exists(arquivoXML) == true)
          {
            File.Delete(arquivoXML);
          }
          TextWriter temp = File.AppendText(arquivoXML);
          temp.WriteLine(txtbox3.Text);
          temp.Close();

          form_arvore f_arvore = new form_arvore();
          }

        catch (Exception erro)
        {
          MessageBox.Show(erro.Message,
```

```
            "Erro na tentativa de visualizar em árvore!",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
      }

      Cursor.Current = Cursors.Default;

    }


//------------------------------CHAMADA PRINCIPAL DA APLICAÇÃO----------------------------
    [STAThread]
    static void Main()
    {
      Application.Run(new meuformulario());
    }
}


//-------------------------------------------------------------------------------------
// Classe criada para apresentar um formulário secundário (árvore)
// ----------------------------------------------------------------

public class form_arvore : Form
{

    private Button botao1;
    private TreeView treeview1;  // TREEVIEW

    public form_arvore()
    {
      int lb = 120;  // largura padrao para um botao
      int ab = 25;// altura padrao para um botao
      int ltxt = 300;  // largura da caixa de texto
      int atxt = 200;  // altura da caixa de texto
      int lfrm = 360; // largura do formulário
      int afrm = 280;  // altura do formulário
      int x = 100; // coordenada x para o label1
      int y = 10;   // coordenada y para o label1

      this.treeview1 = new TreeView();
      this.botao1 = new Button();

      this.botao1.DialogResult = DialogResult.OK;
      this.botao1.Location = new Point(x + 120, y + 220);
      this.botao1.Name = "botao1";
      this.botao1.Size = new System.Drawing.Size(lb, ab);
      this.botao1.TabIndex = 1;
      this.botao1.Text = "OK";
      this.botao1.Click += new System.EventHandler(this.botao1_click);

// Caixa de texto para visualizar o resultado da consulta em forma de árvore
// ----------------------------------------------------------------------
      this.treeview1.Text = "RESULTADO TREEVIEW";
      this.treeview1.Location = new Point (x - 70, y);
      this.treeview1.Size = new Size (ltxt, atxt);

      this.ClientSize = new System.Drawing.Size(lfrm, afrm);
      this.Controls.AddRange(new Control[] {this.botao1, this.treeview1});
      this.Name = "form_arvore";
      this.Text = "Árvore de representação";
      this.Load += new System.EventHandler(this.form_arvore_load);
      this.FormBorderStyle = FormBorderStyle.FixedDialog;
```

```
    this.MaximizeBox = false;
    this.MinimizeBox = false;
    this.StartPosition = FormStartPosition.CenterScreen;
    this.ShowDialog();

  }


  private void botao1_click(object sender, System.EventArgs e)
  {
    Close();
  }


  private void form_arvore_load(object sender, System.EventArgs e)
  {

    try
    {
      // Objetos usados para povoar o TreeView com auxílio do DOM
      // ------------------------------------------------------
      String arquivoXML = "c:\\xquery\\testes\\tmp.xml";
      XmlDocument dom = new XmlDocument();
      dom.Load(arquivoXML);
      treeview1.Nodes.Clear();
      treeview1.Nodes.Add(new TreeNode(dom.DocumentElement.Name));
      TreeNode tNode = new TreeNode();
      tNode = treeview1.Nodes[0];

      // Povoamento do controle TreeView com os nodos DOM
      // ------------------------------------------------
      AddNode(dom.DocumentElement, tNode);
      treeview1.ExpandAll();
    }

    catch (Exception erro)
    {
      MessageBox.Show(erro.Message,
        "Erro na tentativa de visualizar em árvore!",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
  }



// Procedimento usado no evento acima para percorrer recursivamente toda a árvore DOM
// ---------------------------------------------------------------------------------
  private void AddNode(XmlNode inXmlNode, TreeNode inTreeNode)
  {
    XmlNode xNode;
    TreeNode tNode;
    XmlNodeList nodeList;
    int i;

    // Loop através dos nodos XML até atingir as folhas
    // Adiciona os nodos ao TreeView durante o processo de loop
    // --------------------------------------------------------
    if (inXmlNode.HasChildNodes)
    {
      nodeList = inXmlNode.ChildNodes;
      for(i = 0; i < nodeList.Count; i++)
      {
        xNode = inXmlNode.ChildNodes[i];
```

```
        inTreeNode.Nodes.Add(new TreeNode(xNode.Name));
        tNode = inTreeNode.Nodes[i];
        AddNode(xNode, tNode);
      }
    }
    else
    {
      // É necessário puxar os dados do XmlNode baseado no tipo de dados
      // (se são elementos, atributos, valores de atributos, etc)
      // -------------------------------------------------------------
      inTreeNode.Text = (inXmlNode.OuterXml).Trim();
    }
  }

}


//---------------------------------------------------------------------------------------

//Classe padrão definida para aplicar funções extendidas
//à linguagem XQuery com auxílio da XSL
//-------------------------------------------------------
public class Extensao{


  //Função que executa o cálculo de dias trabalhados
  //------------------------------------------------
  public string CalculaDias(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;

    TimeSpan DiasTrabalhados = DataAtual.Subtract(DataArquivoXML) ;

    return DiasTrabalhados.TotalDays.ToString() ;

  }


  //Função que executa o cálculo de horas trabalhadas
  //-------------------------------------------------
  public string CalculaHoras(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;

    TimeSpan HorasTrabalhadas = DataAtual.Subtract(DataArquivoXML) ;

    return HorasTrabalhadas.TotalHours.ToString() ;

  }


  //Função que executa o cálculo de minutos trabalhados
  //---------------------------------------------------
  public string CalculaMinutos(string DadoXSL){

    DateTime DataArquivoXML = DateTime.Parse(DadoXSL) ;

    DateTime DataAtual = DateTime.Today ;
```

```
    TimeSpan MinutosTrabalhados = DataAtual.Subtract(DataArquivoXML) ;

    return MinutosTrabalhados.TotalMinutes.ToString() ;

  }


  //Função que executa o cálculo de porcentagem
  //sobre o salário do funcionário
  //-------------------------------------------
  public string NovoSalario(string DadoXSL, string Aumento){

    Double SalarioAtual = Double.Parse(DadoXSL) ;

    Double PercentualAumento = Double.Parse(Aumento) ;

    Double SalarioNovo = SalarioAtual * PercentualAumento ;

    return SalarioNovo.ToString("c") ;

  }


}
```

Consultas implementadas para testes

A seguir são apresentadas as consultas usadas para realização dos testes de implementação da linguagem XQuery:

Arquivo ***Tudo.xqu***:

```
FOR $b IN document("drf")
RETURN
  $b
```

Arquivo ***TodosFones.xqu***:

```
<todosFONES>
{
FOR $a IN document("drf")
   FOR $b IN $a/DRF
      FOR $c in $b/LOCAL
        RETURN
        $c/FONE
}
</todosFONES>
```

Arquivo ***SetoresCVL.xqu***:

```
<setoresCVL>
{
  FOR  $a in document("drf")/DRF/LOCAL
    FOR $b in $a/SETOR
      WHERE $a/CIDADE="Cascavel"
  return
    <setor>{$b/NOME}</setor>
```

```
}
</setoresCVL>
```

Arquivo ***FuncionariosCVL.xqu***:

```
FOR $b IN document("drf")/DRF/LOCAL
WHERE $b/CIDADE="Cascavel"
RETURN
  <funcionariosCVL>{$b/SETOR/FUNCIONARIO/NOME}
  </funcionariosCVL>
```

Arquivo ***erro.xqu***:

```
FOR $a IN document("drf")
          FOR $b IN $a/DRF/qualquer erro
                    FOR $c in $b/LOCAL
                    RETURN
     $c/FONE
```

Arquivo ***AfrfCVL.xqu***:

```
<afrfCVL>
{
FOR $a IN document("drf")/DRF/LOCAL
  FOR $b IN $a/SETOR/FUNCIONARIO
  WHERE $a/CIDADE = "Cascavel" AND $b/CARGO = "AFRF"
  RETURN
    $b/NOME
}
</afrfCVL>
```

Arquivo de Dados XML

Abaixo é apresentado o arquivo de dados XML (DRF.XML) sob o qual as consultas acima são realizadas:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="c:\xquery\nakano\estilos\estilo1.xsl"?>
<!DOCTYPE biblio SYSTEM "drf.dtd">
<DRF jurisdicao="Cascavel">
  <LOCAL cidade="Cascavel">
    <FONE>(45)225 1214</FONE>
    <SETOR nome="DMA">
      <RAMAL>292</RAMAL>
      <FUNCIONARIO>
        <NOME>Antonio Carlos Markewicz</NOME>
        <MATRICULA>090.1342-1</MATRICULA>
        <CARGO>TRF</CARGO>
        <ADMISSAO>12/05/1980</ADMISSAO>
        <SALARIO>115,00</SALARIO>
      </FUNCIONARIO>
      <FUNCIONARIO>
        <NOME>Alice Golenia dos Passos</NOME>
        <MATRICULA>090.1242-2</MATRICULA>
        <CARGO>TRF</CARGO>
        <ADMISSAO>22/03/1985</ADMISSAO>
        <SALARIO>100,00</SALARIO>
```

```
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Isaura da Silva Madureira</NOME>
    <MATRICULA>090.1133-3</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>15/03/1972</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="CAC">
  <RAMAL>223</RAMAL>
  <FUNCIONARIO>
    <NOME>Selma do Rocio Xavier</NOME>
    <MATRICULA>090.1144-3</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>01/07/1984</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Sebastiana de Oliveira</NOME>
    <MATRICULA>090.3323-9</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>21/03/1977</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Adilson Mazali</NOME>
    <MATRICULA>090.2343-1</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>13/07/1994</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Rozeli Silvia de Lima</NOME>
    <MATRICULA>090.7893-0</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>18/04/1996</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Walderez Otto</NOME>
    <MATRICULA>090.5321-2</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>13/07/1981</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="SATEC">
  <RAMAL>238</RAMAL>
  <FUNCIONARIO>
    <NOME>Selvina Lemes de Campos</NOME>
    <MATRICULA>090.7854-4</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>15/03/1970</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Maria Isabel Reginato</NOME>
    <MATRICULA>090.5123-7</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>09/02/1977</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
```

```
  <FUNCIONARIO>
    <NOME>Marcelo Marcio Oliveira</NOME>
    <MATRICULA>090.3388-4</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>13/09/1989</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Maria de Lourdes Reque</NOME>
    <MATRICULA>090.4412-0</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>25/07/1982</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="SACAT">
  <RAMAL>250</RAMAL>
  <FUNCIONARIO>
    <NOME>Urandir Rodrigues de Lima</NOME>
    <MATRICULA>090.4821-0</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>19/03/1979</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Jarbas de Oliveira Santana</NOME>
    <MATRICULA>090.4587-5</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>04/03/1982</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Albino Antonio Sobrinho</NOME>
    <MATRICULA>090.9578-1</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>30/05/1980</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Fatima Radaelli</NOME>
    <MATRICULA>090.9854-6</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>07/03/1977</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="FIANA">
  <RAMAL>241</RAMAL>
  <FUNCIONARIO>
    <NOME>Hilson Ribeiro Novo</NOME>
    <MATRICULA>090.1245-1</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>03/03/1971</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Adilson Betoni</NOME>
    <MATRICULA>090.5432-8</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>01/11/1978</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
```

```
    <NOME>Jorge Fernandes</NOME>
    <MATRICULA>090.1199-0</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>12/09/1982</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Ivoni Rodrigues</NOME>
    <MATRICULA>090.5500-1</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>12/07/1974</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Filisberto Mioto</NOME>
    <MATRICULA>090.4785-0</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>13/05/1991</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="SAORT">
  <RAMAL>231</RAMAL>
  <FUNCIONARIO>
    <NOME>Miriam Akemi Kubo</NOME>
    <MATRICULA>090.9658-5</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>12/07/1993</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Neuza Paranhos</NOME>
    <MATRICULA>090.1178-8</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>15/03/1977</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Marcio Bento</NOME>
    <MATRICULA>090.9587-9</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>13/07/1993</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Nelson Takeshi</NOME>
    <MATRICULA>090.9999-0</MATRICULA>
    <CARGO>AFRF</CARGO>
    <ADMISSAO>17/08/1970</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
</SETOR>
<SETOR nome="SAPOL">
  <RAMAL>261</RAMAL>
  <FUNCIONARIO>
    <NOME>Jose Maria Andrade</NOME>
    <MATRICULA>090.1178-3</MATRICULA>
    <CARGO>TRF</CARGO>
    <ADMISSAO>12/05/1988</ADMISSAO>
    <SALARIO>100,00</SALARIO>
  </FUNCIONARIO>
  <FUNCIONARIO>
    <NOME>Jandira Cordeiro</NOME>
```

```
      <MATRICULA>090.1893-5</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>07/03/1978</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
  <SETOR nome="GABINETE">
    <RAMAL>220</RAMAL>
    <FUNCIONARIO>
      <NOME>Silene Bohn</NOME>
      <MATRICULA>090.1845-6</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>22/06/1977</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Amilton Zaroni</NOME>
      <MATRICULA>090.4725-1</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>17/12/1971</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Mario Bartos</NOME>
      <MATRICULA>090.5498-2</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>13/05/1969</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Francisco Beltrao">
  <FONE>(46)523 3341</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Oneide Parizotto</NOME>
      <MATRICULA>090.1132-7</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>28/03/1989</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Eliete Sgarbi</NOME>
      <MATRICULA>090.1134-8</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>13/05/1988</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Niceia Maria de Oliveira</NOME>
      <MATRICULA>090.1135-0</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>17/04/1983</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Elois Rodrigues</NOME>
      <MATRICULA>090.1135-1</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>28/10/1975</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
```

```
  </SETOR>
</LOCAL>
<LOCAL cidade="Laranjeiras do Sul">
  <FONE>(42)635 1212</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Neli Maria Bonatto</NOME>
      <MATRICULA>090.2215-1</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>13/12/1982</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Toledo">
  <FONE>(45)378 1404</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Telma Fernandes da Silva</NOME>
      <MATRICULA>090.2216-0</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>25/11/1973</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Silvio Henkemeier</NOME>
      <MATRICULA>090.3316-7</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>13/10/1988</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Berta Nilza Wessel</NOME>
      <MATRICULA>090.3318-4</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>17/03/1979</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Francisco Dinarte</NOME>
      <MATRICULA>090.6645-0</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>15/04/1971</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Ipora">
  <FONE>(44)652 1621</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Jamerson Lucio da Silva</NOME>
      <MATRICULA>090.6672-5</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>13/02/1981</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Pato Branco">
```

```
  <FONE>(46)225 1139</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Sirlei Tomasini</NOME>
      <MATRICULA>090.4895-7</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>26/02/1982</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Zenir Tonial</NOME>
      <MATRICULA>090.4897-0</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>15/09/1976</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Lauri Evaristo Beber</NOME>
      <MATRICULA>090.2039-6</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>12/12/1970</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Santo Antonio do Sudoeste">
  <FONE>(46)563 1190</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Edilson Tadeu Bandeira</NOME>
      <MATRICULA>090.0012-7</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>22/05/1970</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Lourdes Aparecida Gil</NOME>
      <MATRICULA>090.0019-2</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>30/04/1974</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
  </SETOR>
</LOCAL>
<LOCAL cidade="Capanema">
  <FONE>(46)552 1006</FONE>
  <SETOR nome="UNICO">
    <RAMAL>---</RAMAL>
    <FUNCIONARIO>
      <NOME>Antonio Augusto Roos</NOME>
      <MATRICULA>090.0090-0</MATRICULA>
      <CARGO>TRF</CARGO>
      <ADMISSAO>13/04/1978</ADMISSAO>
      <SALARIO>100,00</SALARIO>
    </FUNCIONARIO>
    <FUNCIONARIO>
      <NOME>Marcelo Dadam Nau</NOME>
      <MATRICULA>090.0097-5</MATRICULA>
      <CARGO>AFRF</CARGO>
      <ADMISSAO>11/09/1998</ADMISSAO>
      <SALARIO>100,00</SALARIO>
```

```
      </FUNCIONARIO>
    </SETOR>
  </LOCAL>
</DRF>
```

ANEXO II

Neste anexo estão contidos os itens referentes à segunda parte do desenvolvimento do trabalho, a qual refere-se aos respectivos testes com informações predefinidas, as quais são relacionadas abaixo:

- Arquivo XSL utilizado para exibição final dos dados.
- Arquivo de saída gerado pela aplicação.

Arquivo XSL Usado para Extensão

A seguir é apresentado o arquivo XSL usado para realização dos testes de implementação da extensão a linguagem XQuery:

Arquivo ***trab_dias.xsl***:

```
<xsl:transform
      version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:MinhaExtensao="http://ObjExtendido">

<xsl:output method="html" omit-xml-declaration="yes" />

  <xsl:template match="/">
    <H2 ALIGN="CENTER">Extensao XQuery com uso de XML e C#</H2>
    <H4 ALIGN="CENTER">Mestrado em Ciencia da Computacao - UFSC</H4>
      <xsl:for-each select="DRF">
        <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
          <TD COLSPAN="6">
            <B><I>JURISDICAO:</I></B>
            <I><xsl:value-of select="@jurisdicao"/></I>
          </TD>
```

```
</TABLE>
<xsl:for-each select="LOCAL">
  <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
  <FONT SIZE="2">
    <TD WIDTH="760">
      <B>----------</B>
    </TD>
  </FONT>
  </TABLE>
  <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
      <TD WIDTH="280">
        CIDADE: <B><xsl:value-of select="@cidade"/></B>
      </TD>
      <TD WIDTH="480">
        FONE: <B><xsl:value-of select="FONE"/></B>
      </TD>
  </TABLE>
 <xsl:for-each select="SETOR">
  <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
  <FONT SIZE="2">
    <TD WIDTH="400">
      <B>----------</B>
    </TD>
  </FONT>
  </TABLE>
  <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
  <FONT SIZE="2">
    <TD WIDTH="200">
      SETOR: <B><xsl:value-of select="@nome"/></B>
    </TD>
    <TD WIDTH="560">
      RAMAL: <B><xsl:value-of select="RAMAL"/></B>
    </TD>
  </FONT>
  </TABLE>
  <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0" CELLPADDING="0">
  <FONT SIZE="2">
    <TR><TD WIDTH="200">
      <I>Nome</I>
       </TD>
       <TD WIDTH="80">
        <I>Matricula</I>
       </TD>
       <TD WIDTH="70">
        <I>Cargo</I>
       </TD>
       <TD WIDTH="80">
        <I>Admissao</I>
       </TD>
       <TD WIDTH="160">
        <I>Dias trabalhados</I>
       </TD>
    </TR>
  </FONT>
  </TABLE>
  <xsl:for-each select="FUNCIONARIO">
    <TABLE BORDER="0" WIDTH="760" ALIGN="CENTER" CELLSPACING="0"
CELLPADDING="0">
      <FONT SIZE="1">
        <TR><TD WIDTH="200">
          <B><xsl:value-of select="NOME"/></B>
           </TD>
           <TD WIDTH="80">
```

```
                <B><xsl:value-of select="MATRICULA"/></B>
                 </TD>
                 <TD WIDTH="70">
                <B><xsl:value-of select="CARGO"/></B>
                 </TD>
                 <TD WIDTH="80">
                <B><xsl:value-of select="ADMISSAO"/></B>
                 </TD>
                 <TD WIDTH="160">
                <B><xsl:value-of select="MinhaExtensao:CalculaDias(ADMISSAO)"/></B>
                 </TD>
               </TR>
             </FONT>
             </TABLE>
           </xsl:for-each>
           <TR></TR>
           <TR></TR>
           <TR></TR>
         </xsl:for-each>
       </xsl:for-each>
     </xsl:for-each>
  </xsl:template>
</xsl:transform>
```

Arquivo de Saída Gerado pela Aplicação

A figura abaixo apresenta um pequeno trecho do arquivo de saída HTML gerado pela utilização do aplicativo de extensão deste anexo com o arquivo XSL *trab_dias.xsl* apresentado anteriormente neste mesmo anexo. A coluna "*Dias trabalhados*" refere-se à aplicação de uma das funções de extensão desenvolvidas através do aplicativo:

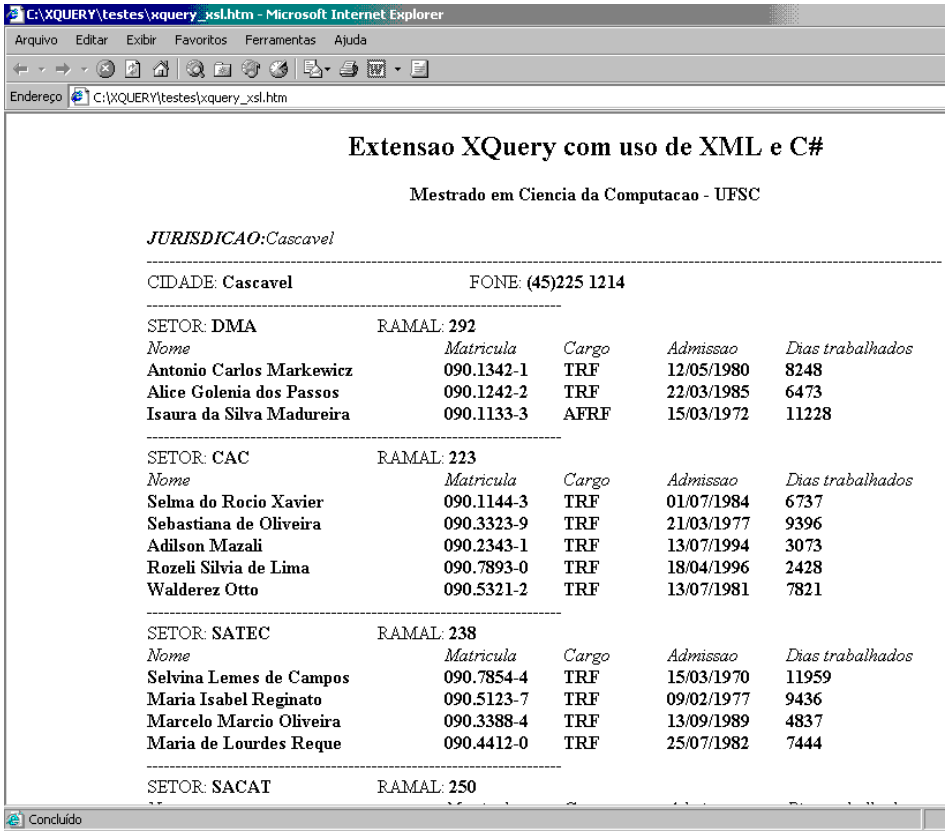


Figura 29 – Arquivo de saída da consulta extendida.