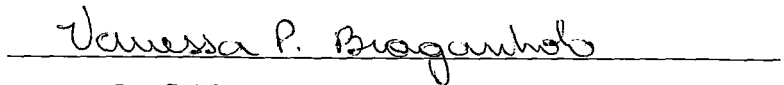
ARAXA: UMA ABORDAGEM OBJETO-RELACIONAL PARA O ARMAZENAMENTO DE DOCUMENTOS XML ATIVOS

Cláudio Ananias Ferraz

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

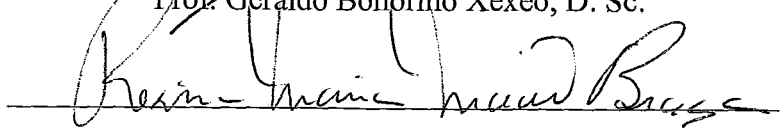
Aprovada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.



Profª. Vanessa de Paula Braganholo, D.Sc.

Prof. Geraldo Bonorino Xexéo, D. Sc.



Profª. Regina Maria Maciel Braga, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

DEZEMBRO DE 2007

FERRAZ, CLÁUDIO ANANIAS

ARAXA: Uma abordagem Objeto-Relacional para o armazenamento de documentos XML Ativos [Rio de Janeiro] 2007

XII, 96 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Documentos XML ativos
2. XML
3. Serviços Web

I. COPPE/UFRJ II. Título (série)

À minha mãe

Agradecimentos

Primeiramente a Deus, pela minha vida. Em especial por esses dez últimos sofridos, apaixonantes, aventurosos, inesquecíveis, libertadores e gratificantes anos em que resolvi extrapolar a minha zona de conforto e me aventurar por algo que o destino reservara para mim, e que infelizmente não estava, em longitude e latitude, próximo a minha família.

À minha mãe, a qual poderia me valer de todos os elogios e agradecimentos dignos a maternidade, que mesmo assim seriam insuficientes para dizer o que eu sinto por ela. Às minhas irmãs Adriana, Valéria e Maria Júlia, por serem exemplos de completude da fraternidade. E além de tudo, me renderam os capetinhas mais anjinhos do mundo: Filipe, Marcos, João e Laurinha. Ao meu pai, por compreender que minhas virtudes não estavam associadas à vida bucólica.

À minha namorada Milene, que esteve comigo durante o ano menos propício a relacionamentos da minha vida.

Às minhas orientadoras Marta e Vanessa, que me orientaram com serenidade e sabedoria, sempre de forma oportuna e adequada. Sabendo conduzir um mestrando desesperado e ansioso como eu.

À Gabriela Ruberg, que sempre apresentou sugestões valiosas a esta dissertação. Às colegas Hildilene e Vanessa pela colaboração significante que tiveram nesta dissertação.

Aos meus antigos amigos Bruno, Patrícia, Victor, Tato, Ariadne, Jairo, Cafete, Viviane, Melissa, Stainam e Ricardo, que me acompanharam durante esta trajetória. Aos meus novos amigos, que me acolheram nessa nova vida: VonHeld, Amanda, Vinicios, Juliano, Jonice, Daniel, Camille, Mateus, Rógea, Rafael Monclair, Amanda Matos, Talita, Nelson, Elder, Margarete, Vinagre, Luciana... graças a Deus são muitos.

À toda a Linha de Banco de Dados, pela oportunidade e pelo conhecimento. Ao Departamento de Ciência da Computação da UFJF, pela excelente formação que me proporcionaram. À Fundação Coppetec, pela oportunidade de desenvolvimento profissional.

A Profª Regina e ao Prof Xexéo por participarem da minha banca.

A todos que de alguma forma pensaram positivo ao meu favor e hoje fazem parte dessa conquista!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ARAXA: UMA ABORDAGEM OBJETO-RELACIONAL PARA O ARMAZENAMENTO DE DOCUMENTOS XML ATIVOS

Cláudio Ananias Ferraz

12 / 2007

Orientadoras: Marta Lima de Queirós Mattoso
Vanessa de Paula Braganholo

Programa: Engenharia de Sistemas e Computação

Os documentos XML ativos (AXML) combinam dados XML tradicionais com dados intencionais definidos através de chamadas a Serviços Web. A propriedade dinâmica desses documentos apresenta desafios tanto para a materialização de seu conteúdo intencional quanto para o seu armazenamento. Nesta dissertação, apresentamos a ARAXA, uma abordagem não intrusiva para o armazenamento de documentos AXML. Nossa abordagem faz uso de objetos complexos dos SGBDs Objeto-Relacionais para representar essa nova estrutura de dados, sendo assim, beneficia-se de ferramentas eficientes de armazenamento e de processamento de consultas. Nós definimos um mecanismo de armazenamento de documentos AXML associado a uma metodologia para materialização em tempo de consulta. Apresentamos uma arquitetura extensível para a ARAXA, independente de produtos ou tecnologias específicas. Implementamos também um protótipo para a ARAXA. Os resultados experimentais mostram que nossa abordagem é escalável, apresentando-se como uma solução viável para o problema abordado.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ARAXA: AN OBJECT-RELATIONAL APPROACH TO STORE ACTIVE XML DOCUMENTS

Cláudio Ananias Ferraz

12 / 2007

Advisors: Marta Lima de Queirós Mattoso
Vanessa de Paula Braganholo

Department: Computer and Systems Engineering

Active XML (AXML) documents combine extensional XML data with intentional data defined through Web service calls. The dynamic properties of these documents pose challenges to both storage and data materialization techniques. This dissertation presents ARAXA, a non-intrusive approach to store AXML documents. It takes advantage of complex objects from object-relational DBMS to represent both extensional and intentional data. By using a DBMS we benefit from efficient storage tools and query engine. We have defined a storage mechanism with a methodology to materialize AXML documents at query time. We present an extensible architecture for ARAXA, which is independent of specific products or technologies. Additionally, a prototype was implemented. Our experimental results show that ARAXA is scalable; therefore it is a viable solution to the problem addressed.

Lista de Abreviações e Siglas

AGM – Agente Monitor

AXML – Active XML

BDA – Banco de Dados Ativos

CCS – Catálogo de Chamadas de Serviços

DOM – Document Object Model

DTD – Document Type Definition

EA – Define Esquema de Armazenamento XML - Relacional

EAC – Evento-Condição-Ação

GCS – Gerente de Chamadas de Serviços

GR – Gerente de Resultados

INRIA – l'Institut National de Recherche en Informatique et en Automatique

JDBC – Java Database Connectivity

LPQ – Linear Path Queries

REE – Representação Específica para os Elementos

RGE – Representação Genérica para os Elementos

RUE – Representação Única para os Elementos

SGBD – Sistema de Gerência de Banco de Dados

SGBDOR – Sistema de Gerência de Banco de Dados Objeto-Relacional

SLS-MC – Stochastic Local Search with Multiple stop Conditions

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

TC – Suporta Tradução de Consultas XQuery e XPath para SQL

URI – Uniform Resource Identifier

WSDL – Web Services Description Language

XML – Extensible Markup Language

XyQL –Xyleme Query Language

Índice:

Índice de Figuras

Índice de Tabelas

Capítulo 1 Introdução

1.1 Motivação

Serviços Web (W3C, 2007a) e documentos XML(W3C, 2007b) têm sido intensamente usados para intercâmbio de dados e comunicação entre aplicações. Documentos XML tornaram-se senso comum para publicação e troca de dados. A tecnologia de serviços Web, por outro lado, provê uma forma de acesso simples e não acoplada a provedores de serviço distribuídos sobre a Internet. O sucesso dessas duas tecnologias e o fato de elas terem ampla aceitação pela indústria fez com que surgisse uma nova classe de documentos XML: os documentos XML Ativos (AXML, para abreviar).

Os documentos XML Ativos são, basicamente, documentos XML que são compostos por dados XML (o chamado conteúdo extensional, por estarem efetivamente explícitos no documento) e chamadas a Serviços Web (intituladas de conteúdo intencional, por representarem dados que em determinado momento, também comporão o conteúdo do documento). Ao serem executadas as chamadas de serviço, os seus resultados são embutidos no conteúdo do documento AXML.

Da mesma forma que documentos XML, documentos AXML podem ser grandes e, levando em consideração que a estrutura de dados de documentos XML é uma estrutura cara de se trabalhar em memória, isto pode resultar em problemas em gerenciar esses documentos em memória principal. Outra questão é que, por vários motivos, esses documentos devem ser passíveis de consulta. Sendo assim, são necessárias alternativas para a gerência e armazenamento desses documentos. É neste contexto que se encontra a motivação deste trabalho, o armazenamento e gerência de documentos XML Ativos.

O conteúdo intencional deve ser gerenciado em tempo de processamento de consulta, isto é, as chamadas de serviços devem ser coordenadas. Em outras palavras, as chamadas de serviço, cujos resultados farão parte do resultado da consulta, devem ser executadas. Um outro comportamento dos documentos AXML que deve ser observado é que chamadas de serviços podem ser disparadas totalmente desassociadas do processamento de consultas, ou seja, chamadas de serviços podem obedecer a uma

periodicidade de execução, definida em tempo de projeto do documento, ficando a cargo do projetista definir esta periodicidade.

A gerência dos documentos AXML envolve principalmente dois fatores: (1) armazenamento e (2) manipulação/gerência de materialização (processo onde ocorre a incorporação do resultado das chamadas de serviço ao documento AXML). Esta manipulação envolve a invocação e a execução remota e distribuída dos serviços Web ao longo do processamento de consultas sobre estes documentos. Dessa forma, podemos perceber que a existência das chamadas de serviço implica em uma complexidade adicional ao problema de armazenamento e gerência de documentos XML. Atualmente, não é possível gerenciar documentos AXML diretamente por ferramentas convencionais de gerenciamento de dados XML, como os SGBDs XML Nativos.

Neste contexto Abiteboul et al.(2005) desenvolveram uma plataforma para trabalhar diretamente com o processamento de documentos AXML. Nessa plataforma, o ambiente de armazenamento dos documentos AXML é desassociado do ambiente de manipulação dos mesmos. Existe nessa plataforma um componente que representa um repositório onde são persistidos os documentos AXML. Porém, no ambiente operacional disponibilizado, os documentos são manipulados em memória principal.

A primeira alternativa adotada para o repositório de documentos AXML em tal ambiente foi a utilização do sistema de arquivos do próprio sistema operacional. Entretanto, essa alternativa apresenta sérios empecilhos como falta de segurança, indexação, controle de concorrência, recuperação, processadores de consulta, escalabilidade, entre outros. Deste modo, a utilização de sistemas de arquivos para armazenamento de documentos AXML não é uma alternativa quando se deseja funcionalidades de armazenamento e consulta na mesma solução.

Uma alternativa mais adequada seria a utilização de Sistemas de Gerência de Banco de Dados (SGBDs). Existem várias alternativas para o armazenamento de documentos XML em SGBDs, tanto utilizando o paradigma relacional quanto usando as abordagens nativas XML. Contudo, o conteúdo intencional dos documentos AXML representa novos desafios para ambas as abordagens.

Abiteboul et al.(2005) também exploram em sua plataforma a alternativa de utilização de um SGBD XML Nativo. Porém, nesta abordagem, o SGBD trata os documentos AXML como documentos XML tradicionais, delegando o tratamento do conteúdo intencional à aplicação que faz uso desses documentos. Isto implica em que as aplicações tenham que gerenciar esses documentos e manipulá-los em memória

principal. Esta manipulação pode acarretar sérios problemas, principalmente ao se trabalhar com documentos grandes. Devemos levar em consideração que a estrutura de dados XML é uma estrutura de dados complexa e muito custosa de se trabalhar em memória principal. Portanto, as aplicações teriam que, por fim, implementar serviços que tradicionalmente são delegados a SGBDs e que já se encontram amadurecidos nestes. Outra alternativa seria disponibilizar uma camada para fazer essa gerência de memória e manipulação da estrutura de dados AXML.

Ao partirmos do princípio que uma solução para o armazenamento de documentos AXML baseada em um SGBD deve considerar também a gerência das propriedades ativas desses documentos, verificamos que SGBDs XML Nativos e SGBDs Relacionais não têm suporte para a execução e gerência de chamadas de serviço Web. Os SGBDs Relacionais podem oferecer uma certa dinamicidade através de gatilhos SQL, o que poderia ser uma alternativa. Entretanto, as chamadas de serviço devem ser executadas em tempo de consulta enquanto que gatilhos sobre o evento de seleção (SQL SELECT) não são implementados na maioria dos SGBDs Relacionais. O tratamento de chamadas de serviços com execução definida a partir de determinada periodicidade também não seria possível em SGBD Relacionais.

Uma outra alternativa que deve ser analisada é a utilização de SGBDs Objeto-Relacionais (SGBDORs). Também não pode ser encontrada uma modelagem explícita do comportamento ativo em SGBDORs. Contudo, SGBDORs são capazes de lidar com objetos complexos e métodos associados a estes objetos. Esta característica nos permite criar uma classe de objetos ativos que são responsáveis por gerenciar as chamadas de serviço e suas execuções. É esta alternativa que será explorada nesta dissertação.

1.2 Objetivo

O principal objetivo dessa dissertação é apresentar uma solução para o armazenamento e a gerência de documentos XML Ativos. A estratégia adotada nessa dissertação foi tratar o problema de armazenamento juntamente com problema da gerência e manipulação destes documentos, combinando em um mesmo ambiente, uma solução para estes dois problemas. Optou-se pela utilização de SGBDORs e seus recursos oferecidos para definição e manipulação de objetos complexos definidos pelo usuário em linguagem de alto nível. Além disso, aproveitou-se a possibilidade de vinculação de procedimentos e funções, definidos em SQL, a métodos desses objetos complexos. Como utilizar esses recursos de maneira eficiente e fazer com que a

utilização de um SGBDOR se torne transparente para o usuário final constitui o problema a ser resolvido nessa dissertação.

A alternativa explorada foi, através da utilização desses recursos de vinculação de funções e procedimentos em SQL a comportamentos de objetos, criar funções e procedimentos em SQL que apontem para métodos de objetos. Deste modo, novos comportamentos, necessários para a manipulação de chamadas de serviços, foram incorporados ao processamento de consultas SQL. Dessa forma, chamadas de serviços podem ser realizadas juntamente ao processamento de consultas SQL. Do mesmo modo, pudemos definir uma função que aponta para um objeto que atua como um monitor das atividades do sistema, gerenciando as chamadas de serviços que têm sua execução definida sobre uma periodicidade. Esta alternativa foi inicialmente apresentada por FERRAZ *et al* (2006).

Utilizando esta estratégia, definimos a ARAXA, uma abordagem não intrusiva para o armazenamento e gerência de documentos AXML. As principais contribuições da ARAXA são: (i) o mecanismo de armazenamento de documentos XML em um SGBDOR, (ii) a representação das chamadas de serviços Web neste SGBD, (iii) o processamento de documentos AXML(materialização) e (iv) uma metodologia para materialização seletiva do documento AXML em tempo de consulta.

Apresentamos uma arquitetura extensível para a ARAXA, independente de produtos ou tecnologias específicas. Implementamos também um protótipo para a ARAXA. Os primeiros resultados obtidos através de um protótipo podem sem encontrados em FERRAZ *et al* (2007a) (2007b). Outros experimentos foram realizados e os seus resultados mostram que nossa abordagem é escalável, apresentado-se como uma solução viável para o problema abordado.

1.3 Organização do Texto

Nesta dissertação analisaremos o modelo de dados AXML, discutiremos a proposta de Abiteboul et al.(2005), aboradaremos os novos conceitos em torno dessa nova classe de documentos XML e as alternativas para o seu armazenamento e gerência. Com base nesse estudo, apresentaremos nossa solução pare este problema, a ARAXA. Para tal, o texto foi organizado da seguinte forma:

No **Capítulo 2** revisitamos alguns conceitos sobre Bancos de Dados Ativos. Apresentamos também uma visão geral sobre o modelo de dados AXML, abordando a utilização de regras ativas no modelo de dados XML. Analisamos o projeto Active

XML, que tem se tornado a principal referência em documentos XML ativos. Alguns pontos referentes ao armazenamento de documentos XML Ativos também são discutidos também neste capítulo.

Já no **Capítulo 3** abordamos o problema de armazenamento e consulta de documentos XML utilizando as propostas que se baseiam nos paradigmas relacional e objeto-relacional. Analisamos os principais esquemas de mapeamento encontrados na literatura. São apresentados os tratamentos realizados para possibilitar mecanismos de consulta e atualizações sobre essas bases XML.

No **Capítulo 4** apresentamos a abordagem ARAXA. Definimos os principais conceitos relacionados à abordagem, como o esquema de mapeamento de documentos XML, representação e gerência de chamadas de serviço e processamento de consultas.

Apresentamos, no **Capítulo 5,** a arquitetura da ARAXA. Detalhamos os seus principais módulos, os quais viabilizam o processo de gerência e armazenamento de documentos AXML. Apresentamos também o protótipo desenvolvido e discutimos as tecnologias utilizadas para a construção deste protótipo.

No **Capítulo 6** apresentamos um estudo experimental, onde verificamos o comportamento da ARAXA sobre diferentes cenários. Discutimos também os resultados obtidos a partir da execução desse estudo experimental.

Finalmente, no **Capítulo 7** concluímos a nossa dissertação, falando sobre os principais objetivos alcançados e as perspectivas de crescimento e expansão da nossa arquitetura.

Capítulo 2 Documentos XML Ativos

A utilização de documentos XML ativos (AXML, para simplificar) aos poucos tem se tornado uma realidade e tem recebido a atenção da comunidade científica. Isso ocorre porque estes documentos podem suprir as necessidades de dinamicidade, distribuição, capacidade de adaptação e de reação às variações externas ao ambiente computacional, típicas dos sistemas computacionais atuais. Do mesmo modo, os documentos AXML se apresentam como uma alternativa para prover integração e interoperabilidade entre aplicações de maneira simples, utilizando padrões bem estabelecidos como XML e Serviços Web (ABITEBOUL *et al.*, 2005). Para o melhor entendimento desta nova classe de documentos XML, bem como dos comportamentos desejados para estes documentos, faz-se necessário um estudo de aplicações que fazem uso desta nova tecnologia e do aparato ferramental e conceitual utilizado.

Este capítulo busca explorar as características da estrutura de dados AXML e que devem ser preservadas em um modelo para armazenamento desta classe de documentos XML. Para tal, o capítulo está organizado da seguinte forma: na Seção 2.1 é apresentado o projeto Active XML, que tem se tornado a principal referência em documentos XML ativos. Já na Seção 2.2 são discutidos alguns pontos referentes ao armazenamento de documentos XML ativos e na Seção 2.2.1 discutimos este armazenamento em SGBDs. Por fim, na Seção 2.3 completamos com uma breve conclusão sobre o que foi explorado neste capítulo.

Antes de prosseguir, no entanto, alguns conceitos de Sistemas de Banco de Dados Ativos também são relevantes para a definição de um arcabouço que se propõe a abordar o comportamento ativo dos documentos AXML, uma vez que os Bancos de Dados Ativos representam a primeira iniciativa para o tratamento de propriedades dinâmicas em relação ao armazenamento de dados. Sistemas de Banco de Dados Ativos (BDA) não apenas armazenam dados, mas também executam ações em respostas a eventos, como por exemplo, as alterações desses dados. Regras ativas especificam quando e quais ações deverão ser executadas (DITTRICH *et al.*, 1995).

Um BDA geralmente é implementado sob a forma de gatilhos e possibilita a utilização de várias funcionalidades, tais como a garantia de restrições de integridade, manutenção de dados derivados, uso de mecanismos de alertas, proteção de acesso e mecanismos de versões (DITTRICH *et al.*, 1995). Este tipo de implementação segue a

semântica de evento–condição–ação (EAC), onde o evento é uma operação do banco de dados, a condição é um predicado e a ação é uma seqüência de procedimentos a serem executados.

A aplicação de regras ativas em documentos XML foi discutida por Paraboschi (2001). As regras ativas foram definidas sobre o modelo EAC e visavam à transformação automática do documento XML. De acordo com Paraboschi (2001), a partir da definição das regras ativas seria possível invocar serviços automaticamente, dado a execução de algum evento relevante sobre o documento XML, além de uma série de características vantajosas para o modelo XML. No trabalho é discutida a implementação de um protótipo sobre um gerenciador de dados XML, o Lorel. Foram encontradas algumas dificuldades, como a definição de regras conflitantes, problemas relacionados à ordenação dos documentos XML e à granularidade das regras.

A adoção de um modelo EAC para atingir os requisitos do modelo AXML pode ser necessária, contudo não suficiente para o modelo AXML. A seguir, na Seção 2.1 será explorado a estrutura de dados AXML, na qual esta dissertação está fundamentada, e serão discutidos os comportamentos necessários para atender essa nova classe de documentos XML.

2.1 Plataforma Active XML

O projeto Active AXML é um projeto de pesquisa do grupo GEMO, INRIA (*Institut National de Recherche en Informatique et Automatique*)[1], onde foi desenvolvido um *framework* para integração e gerência de dados. O framework de integração AXML surgiu com o desafio de permitir o desenvolvimento de aplicações baseados em dados distribuídos e dinâmicos fornecidos através de documentos ativos XML. O tema central é a utilização de documentos XML Ativos para a integração de dados, combinando mediação e armazenamento, buscando uma forma de integração usando serviços Web e XML (MARINOIU, 2004). O projeto é baseado em dois componentes básicos: Documentos AXML e Serviços AXML (ABITEBOUL *et al.*, 2005).

[1] http://www.inria.fr/

2.1.1 Documentos AXML

Documentos AXML consistem em documentos que possuem parte de seu conteúdo composto por dados XML tradicionais, de forma explícita e possuem também partes intencionais, que simbolizam os dados resultantes das chamadas de serviços WEB.

No projeto GEMO, os documentos AXML foram modelados como árvores rotuladas não ordenadas com nós de dados e nós de funções. Os nós de funções correspondem a chamadas de serviços e seus nós filhos correspondem a parâmetros a serem passados para estas chamadas de serviços. Para uma definição formal de um documento AXML assume-se a seguinte disjunção de domínios: $\mathcal{N}$, o domínio de nomes dos nós, $\mathcal{L}$ o domínio de nomes dos rótulos, $\mathcal{F}$ o domínio de nomes das funções e $\mathcal{V}$ os valores atômicos.

Um documento (não ordenado) AXML (documento para simplificar) é uma expressão (T, λ), onde T = (N,E) é uma árvore finita não ordenada. $N \subset \mathcal{N}$ *é um conjunto finito de nós,* $E \subset N \times N$ *são vértices direcionados, e* $\lambda : N \rightarrow \mathcal{L} \cup \mathcal{U} \cup \mathcal{V}$ *é uma função sobre os nós, tal que (i) somente nós folhas podem ser associados a valores atômicos (ii) a raiz é associada a um nome ou a um valor atômico* (ABITEBOUL *et al.*, 2004b).

Na Figura 1 é apresentado um exemplo da estrutura de um documento AXML, onde podemos identificar a principal característica dos documentos AXML, as chamadas a serviços embutidas no documento XML. O documento exemplo trata de informações sobre a coleção de livros e revistas de uma livraria. Ele possui informações na forma convencional dos documentos XML, além de chamadas a procedimentos que recuperam e retornam as informações relevantes ao documento, como o preço atual dos livros, definido como uma chamada de serviço que acessa um serviço Web disponibilizado pela livraria matriz.

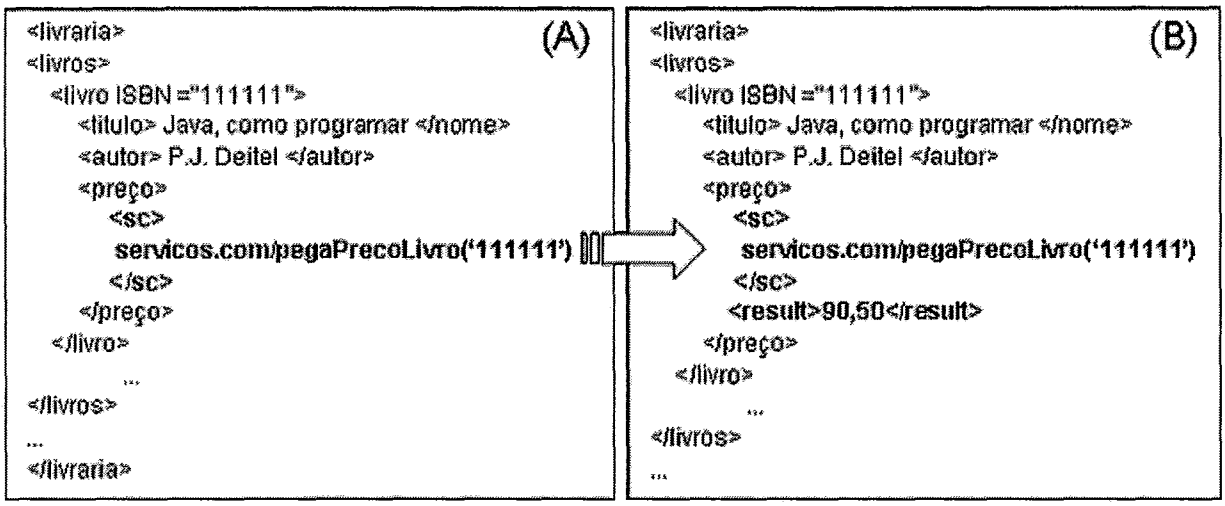


Figura 1 - Exemplo de documento AXML, em (A) o documento antes de sua materialização e em (B) temos o documento já materializado

Estas chamadas de procedimentos (Serviços *Web*) estão delimitadas por uma tag especial **<sc></sc>**, criada unicamente para definir estas chamadas. O seu conteúdo muda de acordo com o resultado obtido pela invocação do Serviço Web. Quando é realizada uma chamada de serviço, o conteúdo referente ao resultado desta invocação é utilizado para "enriquecer" o documento. Este processo é chamado **materialização** (BENJELLOUN, 2004). O processo de materialização deve respeitar as dependências de execução entre as chamadas de serviços, decorrentes do aninhamento dos elementos intencionais (RUBERG *et al.*, 2004). Caso não ocorra nenhum erro na execução da chamada de serviço, o resultado é inserido como um elemento vizinho direito imediato ao elemento que representa a chamada de serviço.

Neste contexto, Serviços Web também podem trocar dados intencionais, uma vez que os documentos AXML obedecem à estrutura definida para documentos XML e dessa forma conservam as características dos mesmos. Um exemplo de troca de documentos AXML pode ser vislumbrado, entre outros cenários, em um ambiente hierárquico de informações, como o discutido por Ruberg et al. (2004). O cenário abordado neste exemplo é o de um programa governamental de empréstimos a famílias que exercem atividades agropecuárias. Existem vários níveis hierárquicos envolvidos, compostos por diversos agentes financeiros, institucionalmente diferentes. Existem várias informações dinâmicas que afetam vários agentes financeiros, que estão distribuídas entre os vários níveis de agentes financeiros e estes são responsáveis por provê-las. Sendo assim, um documento que formaliza um empréstimo em sua situação atual pode ser definido como uma conjunção de informações que estão distribuídas e ser compartilhado através de um serviço Web, definido para tal.

Três aspectos podem ser considerados principais ao se trabalhar com documentos AXML (ABITEBOUL *et al.*, 2004b):

- **Confluência**: Em geral, o estado do sistema pode depender da ordem de invocação dos serviços, ou seja, a determinação da seqüência de invocação de serviços a ser executada, analogamente à definição de um *workflow.*
- **Recursividade e término**: Uma chamada de um serviço pode ativar uma outra chamada a outro serviço e assim por diante, possivelmente de forma recursiva. Ou seja, um serviço pode ter como resposta alguma nova chamada de serviço embutida em seu conteúdo. Assim, a atividade de computação da requisição pode ser uma atividade infinita. Não foi incorporado no modelo AXML nenhum protocolo de segurança específico para o controle de computação infinita executada por códigos maliciosos retornados em resultados de chamadas de serviços. Tal controle poderia limitar o nível de recursão para uma determinada requisição a um serviço ou mesmo adotar *"templates"* a serem seguidos e verificados no momento do recebimento dos resultados. Ambas as abordagens são restritivas e foge ao escopo deste trabalho o tratamento minucioso dos resultados obtidos a partir das chamadas de serviços. Deste modo, a recursividade do modelo AXML é desconsiderada no domínio deste trabalho.
- **Avaliação seletiva**: Quando uma consulta é submetida a um documento AXML, pode ser desnecessária a invocação de todas as chamadas de serviço e a materialização de todo o documento para se conseguir a resposta a esta consulta (ABITEBOUL *et al.*, 2004a). Deve-se focar em chamadas relevantes. O tratamento destinado à avaliação seletiva é definido sobre três atributos contidos na chamada de serviço: *Callable, LastCalled e Frequency.* Estes atributos são definidos e detalhados na Seção 2.1.2 .

O processo de avaliação seletiva de chamadas de serviço deve receber uma atenção especial, uma vez que definir o menor conjunto de chamadas de serviço que devem ser executadas é essencial para aumentar o desempenho no que tange a execução de consultas. O tempo de execução de uma chamada de serviço é relativamente alto quando confrontado com o tempo de execução de consultas, devido à necessidade de se esperar pela resposta do serviço remoto. Esse tempo de espera possui diversas variáveis, como a largura de banda do canal de comunicação e seu tráfego no instante da chamada do serviço, a disponibilidade do provedor de serviço em responder a requisição, a quantidade de dados a serem repassados, etc.

Abiteboul et al. (2004a) apresentam um algoritmo dinâmico para identificar as chamadas de serviço que são necessárias para materializar o resultado de uma determinada consulta. O algoritmo usa alguns conceitos básicos também definidos em Abiteboul et al. (2004a), como a poda de serviços baseada em seus parâmetros de saída, ou seja, a contribuição do resultado da execução da chamada de serviço para o documento (o que pode ser analisado a partir da definição WSDL do serviço), a seqüência em que as chamadas de serviço devem ser executadas, além da criação de um catálogo de chamadas de serviço para detecção rápida dessas chamadas.

Ainda em ABITEBOUL *et al.* (2004a), os autores apresentam uma abordagem para minimizar o conjunto de chamadas de serviços a serem executadas seguindo o princípio das Consultas de Caminho Linear (*Linear Path Queries*- LPQ). A LPQ tem como base o princípio de que dada uma consulta q definida por uma expressão de caminho p, um nó n representa uma chamada de serviço relevante para q somente se ele está num caminho atravessado por p. Dessa forma é possível gerar um conjunto de nós de chamadas de serviço que atendam a esta propriedade para uma determinada consulta q. Contudo, como será discutido nos próximos capítulos, este conjunto ainda pode possuir chamadas de serviços irrelevantes para a consulta q.

Na Figura 2 apresentamos um exemplo da aplicação de LPQ. No topo encontra-se uma consulta que contém uma expressão de caminho que retorna o preço dos livros de uma determinada coleção de livros. O conjunto S é gerado usando cada passo da expressão de caminho concatenada com "*()" (o que representa chamadas de serviços). A partir da execução das consultas geradas, são obtidas as chamadas de serviços a serem executadas.

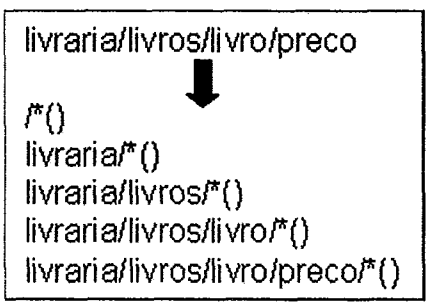


Figura 2- Exemplo de LPQ

A partir desses três aspectos, confluência, avaliação seletiva e recursividade/término, de acordo com a arquitetura definida no projeto GEMO, a invocação dos serviços contidos dentro dos documentos AXML pode se dar das seguintes maneiras (ABITEBOUL *et al.*, 2003):

Chamada Explícita: A forma padrão de se realizar uma chamada explicita é através da definição de um determinado intervalo de tempo. Contudo, a chamada pode ser realizada após o disparo de algum evento ou quando a execução de outra chamada tiver sido realizada.

Chamada Implícita: a chamada ao serviço deve ser executada somente quando os dados de resposta são realmente necessários.

Chamada de Inserção (*Push*): a decisão sobre a realização da chamada fica a critério do servidor, ou seja, do provedor do serviço requisitado. Assim, o retorno da execução do Serviço Web é enviado ao cliente (o sistema proprietário do documento solicitante) de forma assíncrona. Parâmetros adicionais como a freqüência de mensagens a serem trocadas e escolha de representação de diferenças de documentos devem ser fornecidos para a execução do serviço. Contudo, detalhes deste tipo de serviço não serão abordados neste trabalho, na medida em que se busca uma abordagem generalista para serviços, onde todos os tipos de serviços devem ser moldados sobre o padrão definido para Serviços Web.

2.1.2 Serviços AXML

Deve-se lembrar que os documentos AXML são portáveis, pois constituem arquivos XML sintaticamente válidos. Além disso, as chamadas a Serviços Web que estão neles contidas podem ser materializadas simplesmente seguindo a especificação dos protocolos SOAP e WSDL. Como conseqüência dessa portabilidade, documentos AXML podem ser trocados por aplicações. Isso introduz a noção de serviços AXML, ou seja, Serviços Web que aceitam documentos AXML como parâmetro de entrada e/ou retornam documentos AXML como resultado. Os serviços AXML ainda se caracterizam como Serviços Web, na medida em que usam o protocolo SOAP para a troca de mensagens e são descritos usando WSDL. Contudo, para possibilitar algumas características dos documentos AXML, como a definição de que partes do documento serão atualizadas, é necessário o fornecimento de algumas informações, além dos parâmetros a serem utilizados na chamada. Estas informações são definidas como atributos do elemento **<sc>**. Estes atributos são (ABITEBOUL *et al.*, 2005):

- ***ServiceURL***: Especifica a URL do Serviço Web.
- ***ServiceNameSpace***: Este atributo especifica o nome a ser usado no elemento *body* da mensagem SOAP. De forma mais simples é o nome URI do serviço.
- ***MethodName***: Define o nome da operação a ser chamada no Serviço Web.

- ***Signature***: Este atributo marca a URL do arquivo WSDL que define o Serviço Web.
- ***UseWSDLDefinition***: Este atributo especifica se o arquivo WSDL definido pela assinatura deve ser usado na validação de tipos.
- ***Id***: Identifica unicamente uma chamada de serviço e não deve ser definida manualmente.
- ***Name***: Especifica o nome da chamada de serviço. Não possui nenhum significado e é opcional.
- ***Frequency***: O momento no qual a chamada de um serviço deve ser ativada é controlado por um atributo especial do elemento **<sc>** chamado *frequency*. A freqüência (*frequency*) pode ser definida como um intervalo de tempo ou disparada por uma mudança no conteúdo do documento. Uma chamada de serviço é tida como expirada quando, de acordo com o valor definido em *frequency,* ela tiver que ser chamada novamente. Os valores possíveis para este parâmetro são: ***Once*** – indica que o serviço só será executado uma única vez, quando o sistema for iniciado; ***Lazy*** – a chamada do serviço só será ativada quando seus resultados forem úteis; ***On Date*** – especifica exatamente quando um serviço será executado, por exemplo, frequency="25/12/05 14:36"; ***Every X*** – define de quantos em quantos milisegundos o Serviço Web será chamado, por exemplo, frequency= "every 60000".
- ***Callable***: Através deste atributo é possível configurar o serviço para ele não ser ativado, mesmo que ele possua uma configuração de freqüência adequada.
- ***LastCalled***: Este atributo é usado para monitorar a última vez que o serviço foi chamado de modo a cumprir a freqüência sempre que o sistema for iniciado ou recarregado após alguma modificação.
- ***FollowedBy***: Este atributo permite encadear a execução de vários serviços web dentro de um mesmo documento (uma forma simples de *workflow*).
- ***Mode***: Este atributo especifica o que fazer com o resultado obtido pela chamada de serviço AXML e pode assumir dois valores: ***merge*** – significa que os resultados serão adicionados logo após o elemento **<sc>** e que os resultados anteriores serão mantidos; ***replace*** – significa que os resultados anteriores serão substituídos pelos resultados obtidos pela nova chamada ao serviço.
- ***doNesting***: Usado quando se deseja manter um rastro dos nós de texto inseridos no arquivo AXML.

2.1.3 Gerência dos resultados obtidos pelas chamadas de serviço

Nesta seção, abordaremos o tratamento dos resultados obtidos através da execução da chamada de serviço. A fusão dos resultados pode ocorrer localmente ou remotamente (ABITEBOUL *et al.*, 2005).

Fusão Local: quando uma chamada de serviço é executada, os resultados são embutidos dentro da *tag* responsável pela chamada **<sc>**. Quando essa chamada for realizada novamente, é necessário saber o que fazer com os resultados já materializados anteriormente. Diferentes comportamentos podem ser desejados. Três são providos pelo sistema (ABITEBOUL *et al.*, 2005):

- *Apend*: este é o comportamento padrão. Simplesmente adiciona o resultado mais recente anteriormente ao resultado mais antigo;
- *Replace*: simplesmente substitui o resultado antigo pelo novo resultado;
- *Fusion*: é baseado em algum critério de fusão, como a definição de algum esquema onde somente alguns novos elementos são inseridos na árvore do resultado anterior;

Fusão global: Os mesmos mecanismos da fusão local são usados, exceto que a árvore de resultado será fundida com o documento original inteiro e não com os subelementos da *tag* **<sc>.** É importante observar que, na fusão global, o processo de fusão é muito mais complexo.

Uma outra questão importante é a definição do intervalo de tempo em que o resultado obtido pela execução de uma chamada de serviço ainda se mantém válido. Esta característica é definida pelo atributo *valid* contido na chamada do serviço, ou seja, dentro da *tag* **<sc>**. Este atributo pode receber vários valores definidos em termos de tempo. Por exemplo, pode ser definido como zero, assim, a informação só é valida durante o momento de execução da consulta. Pode também ser definido que a informação nunca será apagada, a não ser que isso seja feito explicitamente. Este comportamento pode ser desejado para questões de arquivamento de dados e manutenção de uma massa de dados histórica.

2.1.4 Planos de Materialização

Um plano de materialização de um documento AXML consiste na definição das chamadas de serviço a serem executadas, na ordem em que elas serão executadas, e caso haja delegação de execuções, por quem serão executadas. As chamadas de serviços contidas em um documento AXML podem ter relacionamentos entre si, gerando

dependências em seu processo de materialização. Existem dois tipos de dependências entre chamadas de serviços em documentos AXML (RUBERG e MATTOSO, 2006):

- Dependências causadas por chamadas aninhadas: uma chamada de serviço pode estar aninhada dentro de uma outra chamada de serviço, fornecendo o resultado de sua materialização como parâmetro para a chamada de serviço imediatamente externa. Esta dependência pode ocorrer em vários níveis, o que implica que chamadas de serviços que se encontrem mais internas à estrutura do documento devem ser executadas primeiramente.
- Dependências causadas pelo atributo *FollowedBy*: o atributo *FollowedBy* permite ao projetista do documento AXML definir a seqüência de execução de serviços. Através deste mecanismo pode-se definir que imediatamente após a execução de uma chamada de serviço, caso esta possua a atributo *FollowedBy* atribuído, deve ser executada a chamada de serviço indicada como valor do atributo *FollowedBy*. Dessa forma, o processo de materialização deve respeitar essas decisões do projetista.

Ao tratar o processo de geração de planos de materialização, ou seja, a definição de como se dará a execução do conjunto de operações que resultará na materialização das chamadas de serviço contidas em um documento AXML, Ruberg et al (2006) apresentam um formalismo para representação destas restrições entre as execuções de chamadas de serviço, chamado grafo de dependências. No grafo de dependências, cada nó representa uma chamada de serviço. Dois nós n_1 e n_2 são conectados se o resultado de n_1 interfere em n_2. A ordem de execução é dada pelo direcionamento da aresta.

Um exemplo de grafo de dependência é mostrado na Figura 3. Em (0) é apresentado o documento AXML de forma resumida, explicitando somente as chamadas de serviço; em (1) é apresentado o grafo de dependências do documento (0). As restrições caudadas por chamadas de serviço aninhadas são representadas por linhas completas. As restrições causadas por atributos *followedBy* são representadas por linhas pontilhadas.

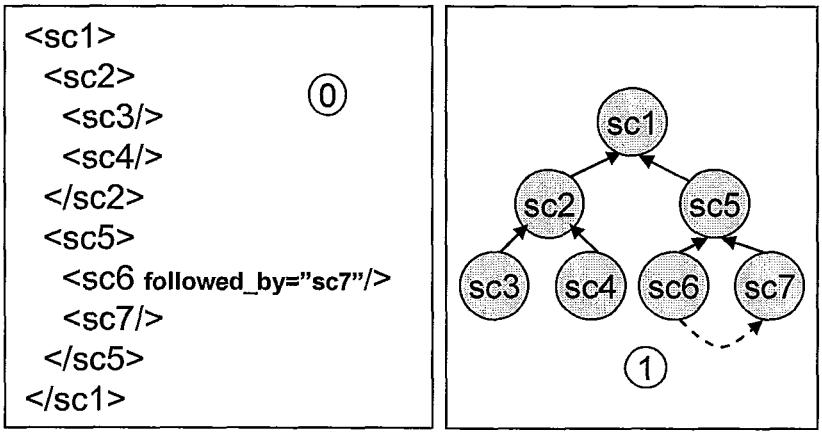


Figura 3 - Exemplo de grafo de dependências (PEREIRA *et al.*, 2006)

O problema relacionado aos custos de planos de materialização e suas otimizações foi inicialmente tratado em Ruberg et al (2004). Em Ruberg et al (2006) é apresentado o XCraft, um otimizador baseado em um modelo de custos, construído sobre a plataforma Active XML. O XCraft executa um processo de otimização de forma dinâmica e descentralizada para documentos AXML. No trabalho de Pereira et al (2006) é realizada uma extensão ao XCraft, apresentando a SLS-MC, uma estratégia eficiente de geração de planos de materialização, baseado em busca local e heurísticas estocásticas. Esta estratégia permite ao XCraft evitar métodos de buscas exaustivas.

2.1.5 Arquitetura Active XML

A arquitetura adotada pelo projeto Active XML (exibida na Figura 4) é uma arquitetura baseada em redes ponto a ponto (P2P). A adoção de arquiteturas centralizadas para integração de dados é de alguma forma contraditória à essência da WEB, que é baseada em um baixo acoplamento de sistemas autônomos (ABITEBOUL *et al.*, 2005). Arquiteturas ponto a ponto capturam a natureza autônoma dos sistemas que participam dela, e possibilitam a esses sistemas atuarem tanto como produtores de informação quanto consumidores, ou seja, como clientes e servidores, de acordo com a necessidade momentânea.

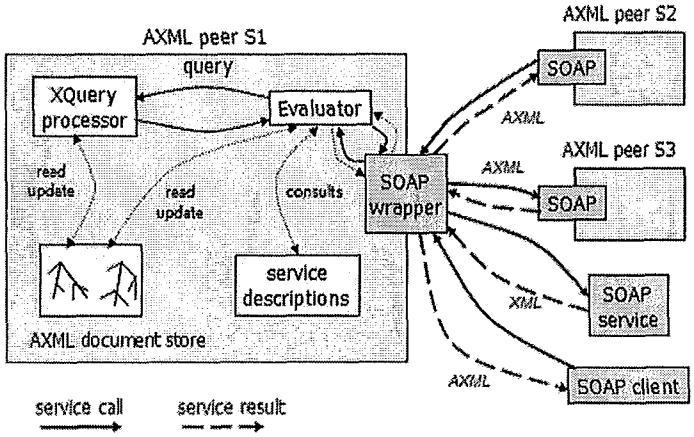


Figura 4– Arquitetura Ponto a Ponto AXML (ABITEBOUL *et al.*, 2003)

A infra-estrutura AXML é composta por *Peers* AXML, ou seja, sistemas que armazenam e gerenciam documentos AXML e trabalham de forma descentralizada em um ambiente ponto a ponto. Estes sistemas são responsáveis pela ativação automática de chamadas de serviços e atualizações nos documentos AXML. A estrutura interna de um *Peer* AXML concentra-se em três módulos principais (ABITEBOUL *et al.*, 2003).

- **Um repositório de documentos:** que provê o armazenamento persistente dos documentos AXML
- **Um avaliador:** que tem o papel de disparar as chamadas dos Serviços Web embutidas dentro do documento AXML e, se necessário, realizar a sua atualização a *posteriori*.
- **Um processador XQuery:** Responsável por lidar com as requisições dos serviços e avaliar as consultas correspondentes a estas requisições.

Dessa forma, um *Peer* AXML atua em três diferentes papeis: (1) como um repositório de documentos, que possui uma máquina de consultas para processar as consultas sobre os documentos armazenados; (2) como cliente, pois há uma série de chamadas de serviços nos documentos do repositório que precisam ser ativadas; (3) como servidor, na medida em que uma série de serviços são providos sob a forma de consultas nos documentos presentes no repositório.

Os pontos se comunicam entre si somente por meio de chamadas de serviços através de seus módulos *wrappers* SOAP. Eles trocam dados XML com qualquer outro provedor de Serviços Web e dados AXML com *Peers* AXML.

Um documento AXML pode realizar chamadas a Serviços Web convencionais, neste caso os parâmetros e resultados são arquivos XML convencionais. Mas, quando

há comunicação com outros sistemas AXML, dados AXML podem ser trocados através de documentos AXML. Note que isso é, de certa forma, um tipo de computação distribuída. Dados contendo chamadas de serviços (como parâmetro de algum outro serviço) podem ser enviados a outros pontos, e ao se fazer isso, um sistema está delegando ao outro parte do trabalho a ser realizado. Da mesma forma que ao receber dados contendo chamadas de serviços, o sistema passa a ter controle sobre essas chamadas e possivelmente pode receber as informações diretamente de seu fornecedor (ABITEBOUL *et al.*, 2003).

A segurança no Active XML atualmente é gerenciada pela validação da estrutura dos dados recebidos utilizando um XML *Schema* representando os dados emitidos (incluindo chamadas de serviços). Além de contar também, é claro, com os aspectos de segurança já implementados em *Serviços Web*. Esta solução não satisfaz completamente o caso de ambiente aberto onde provedores de serviços (os *Peers*) freqüentemente não se conhecem ou não confiam uns nos outros.

Canaud et al.(2004) propõem a criação de um gerente de nomes dos provedores de serviços. Este gerente seria usado para estabelecer uma relação de confiança entre os provedores de serviços. Deste modo, os provedores que tivessem certificado de confiança teriam maior liberdade para a execução de serviços. Isto permitiria que estes serviços pudessem oferecer resultados diferentes de um *template* previamente definido. Tanto o usuário do serviço obtém vantagem desta abordagem, quanto os provedores de serviço que por algum motivo ainda não possuem a certificação de confiança.

2.1.6 Implementação

A implementação de *peers* ActiveXML está principalmente vinculada à noção de documentos AXML e padrões de Serviços Web. A implementação foi definida para usar de forma completa e eficiente todas as vantagens do modelo AXML, suportando tanto as funcionalidades do lado cliente quanto do lado servidor, além de gerenciar os dados AXML de maneira persistente. A implementação foi projetada para ser executada em estações de trabalho fixas e com conexão permanente com a Internet. Contudo, novas extensões podem ser desenvolvidas para outros tipos de plataformas.

As tecnologias utilizadas na implementação do protótipo foram: o Sistema ponto a ponto Active XML, que é a aplicação de gerência de documentos XML Ativos propriamente dita; o Processador AXIS, como ferramenta de infra-estrutura para Serviços Web; o processador X-OQL, responsável pelo tratamento de consultas em X-

OQL, apesar de ser previsto na arquitetura dos *Peers* AXML um processador XQuery; o TomCat Servlet como servidor de aplicação; Ferramentas XML (Xerces, Xalam) para tratamento e manipulação de dados XML; além de contar com serviços básicos de infra-estrutura como a Máquina Virtual Java, e um sistema operacional que no caso pode ser Windows ou Linux.

As camadas da arquitetura definida podem ser visualizadas na Figura 5.

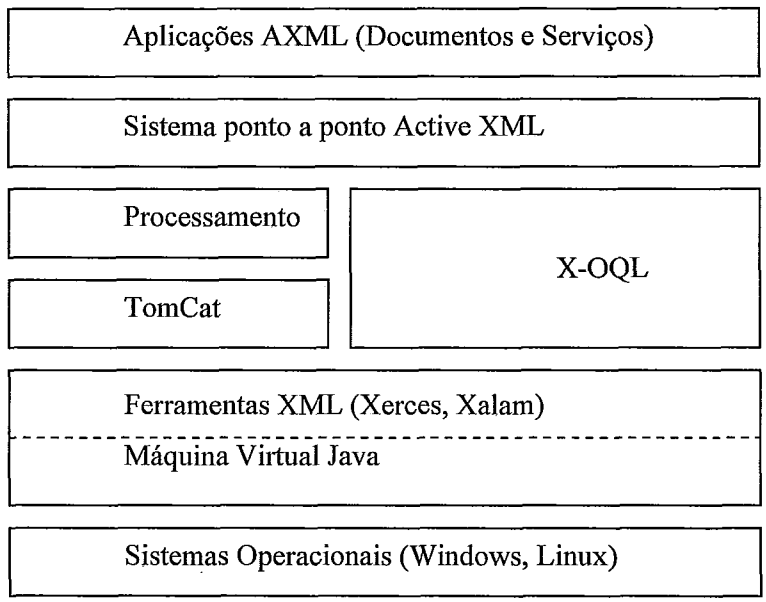


Figura 5– Tecnologias utilizadas no Peer AXML, adaptada de Abiteboul et al (2005)

O Processador X-OQL é responsável por executar consultas complexas em documentos XML. O processador opera sobre uma representação em DOM, que é gerado pelo analisador Xerces quando o documento é recuperado do sistema de arquivos e então armazenado na memória principal.

2.2 Armazenamento de Dados AXML

Um documento AXML é antes de tudo, um documento XML e contém dados. Desta forma, pode ser consultado através de uma linguagem de consulta XML, tal como XQuery (W3C, 2007c) e X-OQL (ABITEBOUL *et al.*, 2005). Quando os documentos AXML se tornam maiores, uma estrutura de índice é necessária para acelerar consultas. É possível, por exemplo, armazenar documentos AXML em um repositório XML nativo. Uma outra solução consiste em criar um índice simples dentro do próprio documento AXML (usando um Serviço Web externo) (ABITEBOUL *et al.*, 2002).

O armazenamento de dados AXML na implementação do Projeto Gemo é delegado ao Repositório de Documentos. Tal repositório consiste em um diretório do

sistema de arquivos, definido pela aplicação, onde se encontram os documentos AXML. Desse modo, existem problemas no gerenciamento de grandes massas de documentos, o que interfere diretamente na escalabilidade desta implementação. Para contornar este problema algumas extensões foram propostas, como o caso do Xyleme-AXML, uma implementação de *Peer* AXML integrada com Xyleme Server, um repositório de dados XML nativo. Esta extensão será melhor detalhada na próxima seção.

2.2.1 Dados AXML em SGBDs

A utilização de SGBDs para o armazenamento e gerência de dados AXML é um tema pouco discutido na literatura revisada. Maiores esforços estão sendo direcionados para a consolidação da tecnologia e definição de padrões a serem utilizados. Contudo, a necessidade de um esquema de gerência de dados sofisticado já foi identificada por Abiteboul et al.(2005), que discutem a utilização de uma extensão dos *Peers AXML* implementados no projeto GEMO. Nesta extensão, o armazenamento dos documentos AXML é delegado ao SGBD XML nativo Xyleme. Ao invés do carregamento como um objeto DOM na memória principal, o documento permanece persistente no Xyleme. As informações sobre as chamadas de serviços embutidas nos documentos são mantidas em memória.

Consultas Xyleme (XyQL) são usadas para acessar seletivamente parte dos dados. Sendo assim, o documento não precisa ser totalmente carregado em memória para que possa ser consultado e atualizado. A Figura 6 representa como a arquitetura do *Peer AXML* seria modificada para a integração com o Xyleme. Do lado esquerdo **(a)**, a arquitetura do Peer AXML originalmente concebida, ao lado direito **(b)**, a arquitetura que representa a integração.

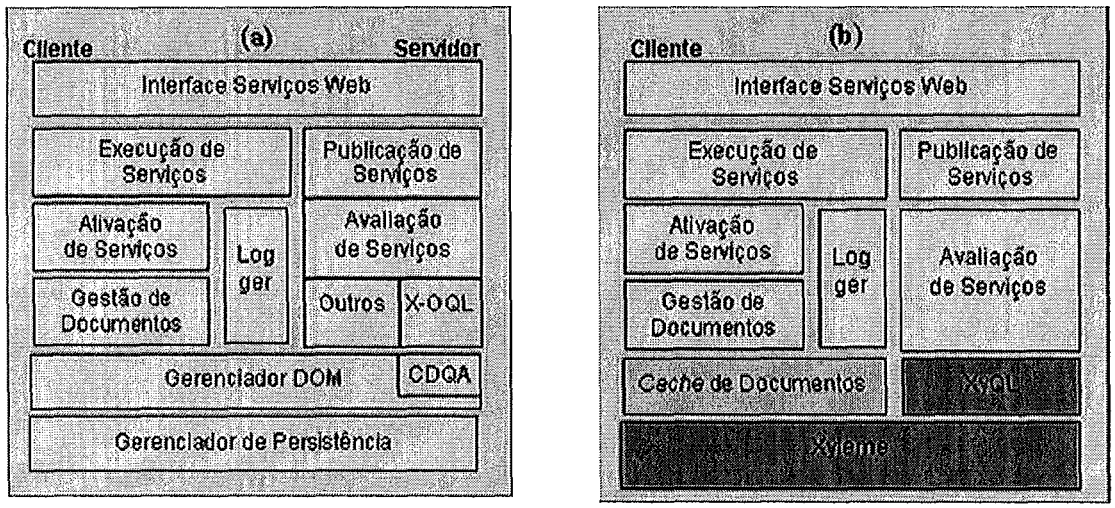


Figura 6 – Integração *Peer* AXML + Xyleme

Na arquitetura de integração, os processadores de consultas são substituídos pelo módulo XyQL, uma linguagem definida e implementada no Xyleme. A XyQL permite a consulta tanto de estrutura quanto de dados em documentos XML. Todavia, a linguagem não representa um padrão para consultas em XML e é de caráter proprietário, o que dificulta a adoção desta solução para integração com outros sistemas que também fazem uso dos dados AXML, ou mesmo com sistemas legados.

Em relação à SDBDs Relacionais e Objeto-relacionais, não foi identificada na literatura nenhuma abordagem para o armazenamento de dados AXML, apesar do grande volume de trabalhos direcionados ao armazenamento de dados XML em tais sistemas de gerência de dados (DEUTSCH *et al.*, 1999; SHANMUGASUNDARAM *et al.*, 1999; MANOLESCU *et al.*, 2000; BOHANNON *et al.*, 2002; NAMBIAR *et al.*, 2002; TATARINOV *et al.*, 2002b; TATARINOV *et al.*, 2002a; DEHAAN *et al.*, 2003; KRISHNAMURTHY *et al.*, 2003; LU *et al.*, 2003; VIEIRA *et al.*, 2003; PLOUGMAN *et al.*, 2004; LEONARD, 2006).

Em sistemas de bancos de dados relacionais, pode-se fazer uma analogia à ativação de chamadas de serviços com o uso *triggers* (ABITEBOUL *et al.*, 2005). Procedimentos disparados automaticamente, baseados no paradigma de Evento-Condição-Ação e em aspectos temporais, representam uma poderosa ferramenta para a manipulação de propriedades dinâmicas sobre bases de dados. Estes comportamentos dinâmicos não se restringem aos requisitos atuais de gerência de dados, sendo amplamente discutidos na literatura de Bancos de Dados Ativos e implementados em grande parte dos SGBDs Relacionais e Objeto-Relacionais comerciais.

Contudo, uma possível integração de um sistema AXML com sistemas de banco de dados relacionais utilizando o conceito de Bancos de Dados Ativos deve considerar uma classe de *triggers* que não se encontra implementada na grande maioria dos produtos SGBDs relacionais comerciais. A implementação desta classe de *triggers* envolve os eventos de seleção de entidades do modelo relacional, ou seja, gatilhos disparados por expressões SQL da forma "*select* <colunas> *from* <tabelas> *where* <predicados>". Esta classe se faz necessária para a ativação automática das chamadas de serviço, uma vez que o usuário não deve se preocupar com a existência ou não dessas chamadas ao submeter suas consultas ao banco de dados.

2.3 Considerações finais

Neste capítulo podemos verificar que a inclusão de comportamentos ativos em documentos XML já tem sido alvo de discussão pela comunidade científica e que a utilização dos padrões XML e de Serviços Web pode representar uma ferramenta poderosa de integração de dados e sistemas de forma simples e clara. O modelo de documentos XML Ativos apresentado pelo projeto Active XML (ACTIVEXML, 2007) corresponde à melhor referência sobre a estrutura desses documentos que pode ser encontrada na literatura e que possivelmente representará um padrão para a definição de documentos AXML.

Evidenciamos também a necessidade de um armazenamento de dados AXML mais sofisticado, que possibilite a escalabilidade de sistemas baseados em documentos AXML. Esperamos que esta abordagem de armazenamento não se distancie dos paradigmas e tecnologias já amplamente difundidos e validados pelas organizações que trabalham com sistemas de informação, além da coexistência harmoniosa com outras estruturas e unidades de informação de dados. A utilização do arcabouço teórico-conceitual para o armazenamento de dados XML sob a perspectiva relacional e objeto-relacional, amplamente validado, mostra-se como uma alternativa para a gerência de dados AXML, na medida em que um documento AXML é, antes de tudo, um documento XML bem formado.

Enfim, pode-se vislumbrar um cenário onde todos os tipos de informação utilizados comercialmente, sejam elas estruturadas ou não, possam ser persistidas, dada uma função de mapeamento, para o paradigma Objeto-Relacional. Este repositório único torna-se uma alternativa interessante para os ambientes empresariais, não representando custos adicionais e não requerendo mudanças bruscas na gerência de armazenamento de dados na qual estão atrelados o conhecimento e o histórico organizacional.

No próximo capítulo discutiremos as principais abordagens de mapeamento e armazenamento de documentos XML no modelo Objeto-Relacional e analisaremos os critérios que também são aplicáveis aos documentos XML ativos.

Capítulo 3 Mapeamentos XML-Relacional

O armazenamento de documentos XML já foi amplamente discutido na literatura. Várias abordagens já foram propostas (DEUTSCH *et al.*, 1999; SHANMUGASUNDARAM *et al.*, 1999; MANOLESCU *et al.*, 2000; BOHANNON *et al.*, 2002; NAMBIAR *et al.*, 2002; TATARINOV *et al.*, 2002b; TATARINOV *et al.*, 2002a; DEHAAN *et al.*, 2003; KRISHNAMURTHY *et al.*, 2003; LU *et al.*, 2003; VIEIRA *et al.*, 2003; PLOUGMAN *et al.*, 2004; LEONARD, 2006), utilizando tanto SGBDs Relacionais, quanto Objeto-Relacionais, além dos SGBDs XML Nativos, que surgiram especificamente para este objetivo.

O armazenamento de documentos XML Ativos pode ser considerado uma variante do estudo de armazenamento de documentos XML convencionais. Contudo, a inserção de chamadas de Serviços Web nos documentos XML representa um novo desafio para as abordagens convencionais já discutidas na literatura. Todavia, as abordagens para armazenamento de documentos XML podem representar um arcabouço conceitual inicial para a definição de uma nova abordagem onde o comportamento ativo dos documentos AXML esteja contemplado.

Este capítulo está organizado da seguinte forma: na Seção 3.1 será abordado o problema de armazenamento e consulta de documentos XML utilizando as propostas que se baseiam nos paradigmas relacional e objeto-relacional encontradas na literatura. Na Seção 3.1.1 são definidos os principais esquemas de mapeamento encontrados na literatura. Na Seção 3.1.2 são apresentados os tratamentos realizados para possibilitar mecanismos de consulta e atualizações sobre essas bases XML. Finalmente, na seção 3.2 é feita uma breve discussão sobre o que foi apresentado no capítulo.

3.1 Armazenamento e Consulta de dados XML em SGBD´s Relacionais

O ímpeto inicial para a utilização de XML foi primeiramente aumentar a capacidade de aplicações remotas de interpretar e manipular documentos disponibilizados na Internet. Entretanto, do ponto de vista de Banco de Dados, XML traz uma outra possibilidade instigante: documentos XML representam fonte rica de dados e informação, que são passíveis de consulta (SHANMUGASUNDARAM *et al.*, 1999).

A estratégia mais simples para o armazenamento de documentos XML consiste na geração e manutenção de arquivos de texto no sistema operacional, respeitando o formato original de cada documento. Porém, esta alternativa apresenta inúmeras desvantagens quanto à flexibilidade e desempenho, além da dificuldade de realização de consultas sobre a base de dados. Dessa forma, é recomendável que bases de documentos XML estejam sob jurisdição de algum tipo de SBGD, e dessa forma possam ser beneficiadas pelas funcionalidades disponíveis em tais sistemas de gerência.

A seguir, na Seção 3.1.1 , serão apresentadas as principais abordagens encontradas na literatura sobre o mapeamento de documentos XML para SGBDs Relacionais e Objeto-Relacionais.

3.1.1 Esquemas de Mapeamento XML – Relacional

Existem na literatura várias propostas para solucionar o problema do mapeamento XML-Relacional. Existem também outras abordagens que foram implementadas em produtos comerciais. Em Vieira Jr (VIEIRA *et al.*, 2003) foi apresentado uma classificação para estas abordagens, que é descrita da seguinte forma:

(i) caixa preta;

(ii) representação genérica para os elementos;

(iii) representação única para os elementos;

(iv) representação específica para os elementos.

A abordagem caixa preta é a alternativa mais simples, quase análoga à abordagem onde o armazenamento fica a critério do sistema de arquivos do sistema operacional. Consiste no armazenamento do documento XML sem nenhum tipo de mapeamento. Normalmente é utilizada uma estrutura de dados correspondente a um arquivo texto no SGBD.

As abordagens que usam a representação genérica para os elementos e a representação única para os elementos utilizam esquemas de dados genéricos para a representação dos elementos do documento XML, independente da estrutura do documento original. São definidos objetos ou relações que contemplam todo o universo de documentos XML. O ponto de divisão entre essas duas categorias está na rigidez imposta por uma estrutura única (um único objeto no esquema de dados), que é utilizada na representação única. Ou seja, na representação única temos uma única relação alvo em nosso esquema definido no SGBD, enquanto na representação genérica para os elementos existem várias relações que compõem o esquema de mapeamento. Estas

abordagens utilizam esquemas indiretos de mapeamento, uma vez que cada elemento definido no documento XML não representa uma relação ou uma classe no esquema da base de dados no SGBD.

As propostas encontradas na literatura que utilizam a representação genérica normalmente fazem uso de um esquema proprietário, definido pelos autores, para a decomposição dos documentos XML em relações. Um dos trabalhos pioneiros neste sentido foi o apresentado por Florescu et al (1999). Neste, foi realizada uma simplificação acerca dos documentos XML. Assumiu-se que documentos XML podem ser representados como grafos direcionados e rotulados. Este simplificação faz com que ocorra alguma perda de informação acerca da estrutura do documento e sua semântica, tal como a diferenciação entre elementos e atributos. A partir desta simplificação foram propostos os seguintes esquemas de mapeamento de arestas:

- Abordagem *Edge*: o esquema definido nesta abordagem é representado pela relação *Edge(source, ordinal, name, flag, target)*. *Source* e *target* armazenam os OIDs dos nós envolvidos, *Ordinal* representa a ordenação dentro do documento XML, *Name* representa o rótulo e *flag* indica a referência entre objetos ou aponta para um valor, no caso de uma folha.
- Abordagem Tabela Universal: a tabela Universal armazena todas as arestas, e conceitualmente corresponde à junção externa de todas as tabelas Binárias da abordagem Binária, que será vista em breve. No caso de $n_1 \ldots n_K$ rótulos, teremos o seguinte esquema: *Universal(source,* $ordinal_{n1}$, $flag_{n1}$, $target_{n1}$, $ordinal_{n2}$, $flag_{n2}$, $target_{n2}$, ..., $ordinal_{nk}$, $flag_{nk}$, $target_{nk}$*)*.

Para o mapeamento de valores, foram apresentadas duas possibilidades, o armazenamento de valores junto com as arestas e o armazenamento de valores em tabelas separadas. As duas estratégias para o armazenamento de valores podem ser aplicadas às duas abordagens de mapeamentos de arestas, totalizando quatro alternativas de esquemas de armazenamento.

Além das simplificações impostas aos documentos XML nestes esquemas, tem-se outros problemas, como por exemplo, a manutenção do número ordinal seqüencial definido para cada elemento. Isso acarreta um tempo de execução extra no momento em que ocorre uma atualização, uma vez que ao se inserir ou excluir um elemento, toda a numeração deve ser revista.

Outro esquema de mapeamento que faz uso de uma estrutura única é o método de Codificação por Intervalos e sua extensão, a Codificação por Intervalos Dinâmicos

(DEHAAN *et al.*, 2003). O método de codificação por intervalos atribui a cada nó um valor L e um valor R, de acordo com as seguintes regras:

(v) L < R para cada nó

(vi) Se o nó s1 é um ancestral do nó s2 então Ls1 < Ls2 e Rs2 < Rs1

(vii) Se o nó s1 é um irmão esquerdo do nó s2 então Rs1 < Rs2

Uma maneira de se gerar uma codificação por intervalos válida é realizar um percurso transversal, em profundidade, na árvore do documento XML, usando um contador que atribui o valor L quando o nó é encontrado pela primeira vez e o valor R quando for encontrado pela última vez (DEHAAN *et al.*, 2003). Na Figura 7 tal mapeamento é exemplificado.

```
<site>
<pessoas>
        <pessoa id = "person0">
                <nome> Cláudio Ferraz </nome>
                <email> cferraz@cos.ufrj.br </email>
                <tel>8147-8407</tel>
                <hp>meusite.com</hp>
        </pessoa>
        ...
</pessoas>
...
</site>
```

S	L	R
<site>	0	85
<pessoas>	1	46
<pessoa>	2	23
@id	3	6
person0	4	5
<nome>	7	10
Cláudio Ferraz	8	9
<email>	11	14
cferraz@cos.ufrj.br	12	13
<tel>	15	18
8147-8407	16	17
<hp>	19	22
meusite.com	20	21

Figura 7- Exemplo de Codificação por Intervalos, adaptada de (DEHAAN *et al.*, 2003)

Na proposta definida em Manolescu et al. (2001), foi adotada uma estratégia onde o número de tabelas se mantinha fixo, mesmo variando o número de rótulos existentes no documento XML. Com isso é feito um controle do número de relações existentes no esquema, mesmo para documentos com grande variedade de rótulos. Tal abordagem é um exemplo de representação genérica para os elementos. A Figura 8 apresenta o esquema proprietário desta proposta.

Document	(docID, docURL)	Element	(elID, tagID)
Value	(valid, value)	ElementContent	(parented, childID, valID)
Tag	(tagID, valID)	ElemAttribute	(elID, attID, valID)
Attribute	(attID, valID, type, isRequired)	ElemDoc	(elemID, docID)
Word	(wordeID,word)	Contais	(elID,worded, deph, tag)

Figura 8 – Esquema Genérico proposto por Manolescu et al (2001)

Um outro esquema genérico pode ser encontrado em Tatarinov (2002b). Destaca-se a estratégia que faz uso de um esquema de ordenação baseado na ordenação Dewey (OCLC, 2002). Esta ordenação minimiza o custo de reordenação no caso de atualizações no documento (inserções e exclusões), uma vez que devem ser renumerados somente os nós irmãos do nó que sofre a alteração e suas sub-árvores. A Figura 9 apresenta o modelo de ordenação Dewey utilizado para documentos XML.

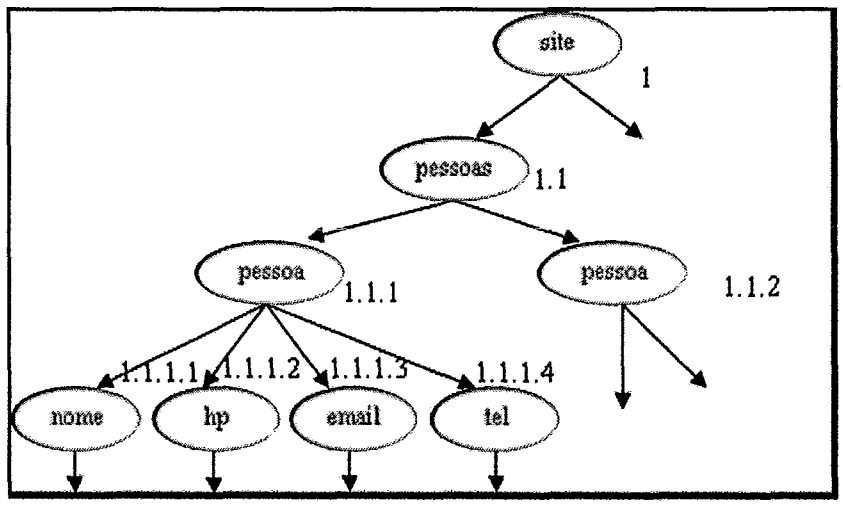


Figura 9– Estratégia de ordenação Dewey em uma árvore XML

Utilizando a ordenação Dewey, o esquema proposto por Tatarinov (2002b) fica com a seguinte estrutura: relação *edge* (*dewey, path_id, value*) onde o valor *dewey* guarda a informação sobre a ordenação do nó e sobre os ancestrais *e* relação *path* (*id, path*). A relação *path* representa uma compactação das informações sobre a expressão de caminho dos elementos armazenados, uma vez que, geralmente, um mesmo caminho se repete para vários nós armazenados em um mesmo documento.

A abordagem Binária também encontrada em Manolescu *et al* (2001) é um exemplo de representação específica para os elementos. Um esquema genérico é definido por relações B_{name}(source, ordinal, flag, target). Neste mapeamento são definidas relações B_{name} distintas para cada elemento diferente existente dentro da estrutura do documento XML, seja ele um elemento ou um atributo. Dessa forma, ocorre um agrupamento de todos os nós do grafo que possuem o mesmo rótulo. Esta abordagem apresenta alguns problemas, tais como a definição de um grande número de relações, dentre as quais algumas podem possuir um número muito pequeno de instâncias, além de problemas na reconstrução do documento, devido à perda da ordenação global.

Outras abordagens que utilizam representação específica utilizam DTDs e XML Schemas, uma vez que estes regem a estrutura do documento XML que está sendo armazenado. A abordagem apresentada em Shanmugasundaram (1999) faz uso de DTDs para direcionar o mapeamento a ser criado para cada documento XML, ou seja, são criadas tabelas de acordo com o relacionamento entre os elementos. Cada tupla em uma tabela possui um identificador e uma coluna para identificar o seu elemento pai. Um elemento XML que só pode aparecer uma vez no elemento pai deve ser adicionado como uma coluna na tabela que representa o seu pai. Já os elementos que aparecem mais de uma vez no elemento pai são armazenados em uma outra tabela criada para este fim. Caso o DTD contenha ciclos, uma tabela separada é criada para quebrar este ciclo. A Figura 10 apresenta um exemplo deste mapeamento. Outras estratégias que utilizam DTDs podem ser encontradas em (KHAN e RAO, 2001; 2002; LU *et al.*, 2003).

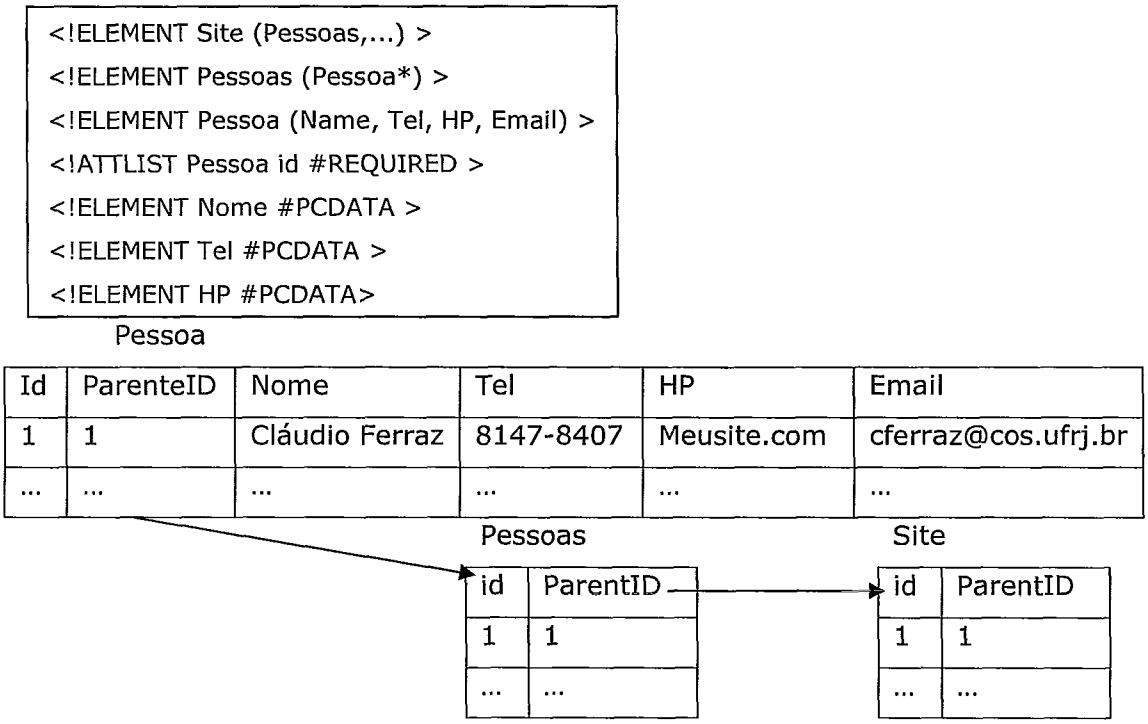


Figura 10 – Exemplo de mapeamento utilizando DTD

A utilização de DTDs para direcionar o mapeamento também pode apresentar algumas falhas como (1) a presença de expressões regulares, elementos aninhados e tipos recursivos não são adequadas ao modelo de relações planas, (2) DTDs não fazem diferenciação entre elementos que representam as entidades e elementos que representam algum atributo desta entidade. Dessa forma, não é muito claro o que deve ser mapeado como uma relação e o que deve ser apenas um atributo de uma relação (BOHANNON *et al.*, 2002). No caso da utilização de XML *Schemas*, uma estratégia a partir de uma visão simplista seria o mapeamento direto de cada estrutura de dado

definida no *schema* para uma relação, uma vez que o XML *Schema* provê tipos de dados mais explícitos, o que conduziria a um mapeamento mais natural. A Figura 11 apresenta um exemplo deste mapeamento.

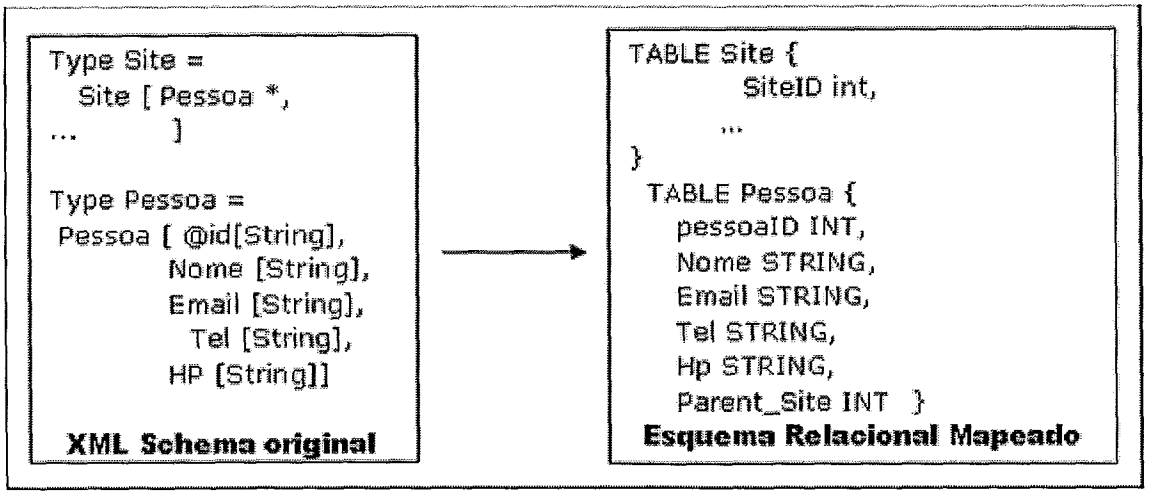


Figura 11– Exemplo de mapeamento usando XML *Schema (BOHANNON et al., 2002)*

Contudo, também há uma inadequação entre tipos definidos e relações lineares. Isto ocorre devido à existência de informações aninhadas em documentos XML, além da falta de informação sobre a cardinalidade entre as coleções e o numero de valores distintos que um atributo pode receber. Tais informações são necessárias para a definição de um projeto eficiente de armazenamento (BOHANNON *et al.*, 2002). Em Bohannon et al. (2002) também é apresentado um esquema de mapeamento utilizando XML *Schemas.* A estratégia adotada neste trabalho inclui a realização de um mapeamento intermediário, que os autores definem como "*P-shema*" (de *Physical Schema*). A Figura 12 mostra um exemplo desta abordagem.

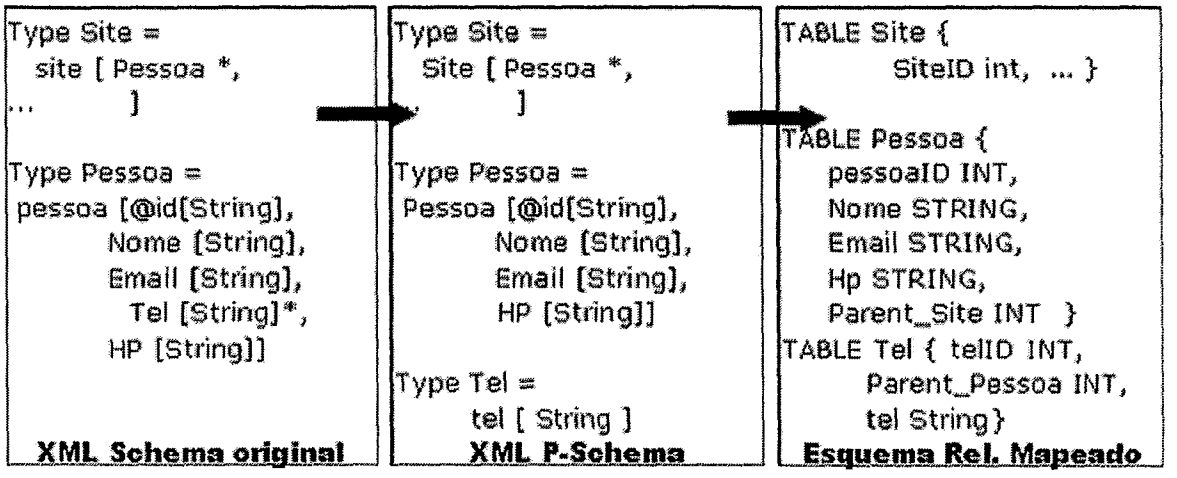


Figura 12 – Esquema intermediário definido na abordagem (BOHANNON *et al.*, 2002)

O processo utilizado consiste basicamente em um processo análogo ao de normalização no modelo relacional, caracterizado pela extração dos tipos que serão mapeados para relações.

3.1.2 Tratamentos de consultas e atualizações

Devido à grande extensão de propostas e esquemas proprietários para o armazenamento de dados XML em sistemas relacionais e objeto-relacionais, existem na literatura várias abordagens e algoritmos para a tradução de consultas XPath e XQuery para SQL, bem como algoritmos para a reconstrução dos documentos, cada qual direcionado para o esquema de mapeamento definido. Desse modo, a descrição de detalhes sobre cada mecanismo de tradução de consultas e reconstrução de documentos está fora do escopo desta dissertação. Limitar-nos-emos a fazer uma breve descrição dessas abordagens.

A estratégia de execução de consultas e atualizações sobre a massa de dados XML armazenada é definida sobre dois aspectos principais (NAUGHTON, 2003):

- O projeto do esquema relacional utilizado: qual esquema relacional será utilizado para o armazenamento XML.
- Algoritmos de tradução de consultas: uma vez escolhido o esquema a ser utilizado, definir como será feita a tradução de consultas XML em consultas SQL.

Existem três etapas distintas para o tratamento de consultas e atualizações: a tradução da consulta em linguagem de consulta XML para SQL; a execução em si da consulta SQL pelo SGBD; algum processamento posterior para traduzir o resultado obtido em uma resposta XML, como o esperado pela consulta inicial (FLORESCU e KOSSMAN, 1999).

Nas abordagens onde é utilizado um esquema de mapeamento definido como caixa preta, normalmente a avaliação de consultas é realizada de forma semelhante aos SGBDs XML nativos, pois não há a necessidade de tradução XML-SQL.

Para as abordagens onde é utilizado um esquema genérico para a representação de documentos XML, o trabalho Florescu et al (1999) faz uma discussão sobre como traduzir operações básicas em consultas por expressões de caminho em consultas SQL. As operações descritas são (1) retorno de um elemento que possui um filho, (2) seleção de valores, (3) combinação de padrões, (4) predicados opcionais, (5) predicados sobre

atributos e (6) consultas de expressões regulares de caminho que podem ser traduzidas em consultas SQL recursivas (NAUGHTON, 2003).

O método de associação binária descrito em Schmidt et al (2001) trabalha com a tradução de consultas semelhantes a OQL para consultas SQL, ou seja, consultas que correspondem à ramificação de consultas de expressão de caminho em XQuery (NAUGHTON, 2003). Em Özsu (DEHAAN *et al.*, 2003) é apresentada a abordagem de intervalos dinâmicos que trabalha com um universo maior de fragmentos da XQuery, como expressões FLWR aninhadas, construções de elementos e funções internas incluindo comparações estruturais.

Esta estratégia também pode ser encontrada em Leonard et al (2006). São propostos novos operadores para que possa ser implementada a geração eficiente de SQL. Esses operadores são parecidos com os existentes em SGBDs XML nativos.

Em Tatarinov (2002b) é apresentado um algoritmo em alto nível para a implementação de consultas onde a ordenação do documento XML é considerada. São tratados predicados baseados em posicionamento, por exemplo, /play/act/scene[2], e ordenação com conceitos de *following* e *following-sibling*.

Em Naughton (2003) também são apresentadas soluções gerais para a tradução de consultas envolvendo expressões de caminho simples, expressões de caminho com predicados e consultas XQuery um pouco mais complexas.

Para as abordagens onde é utilizado um esquema de mapeamento específico para os elementos, geralmente utilizando DTDs e XML Schema, Shanmugasundaram et al (1999) apresenta, sem detalhes algorítmicos, mecanismos para a tradução de consultas semi-estruturadas para SQL. Neste trabalho foram contempladas consultas de expressões de caminho simples, consultas de expressão de caminho simples recursivas, a transformação de expressões de caminhos arbitrárias para expressões de caminho simples, além da reconstrução de resultados para formato XML.

A Tabela 1 traz um resumo das técnicas de mapeamento de dados XML para SGBDs Relacionais e Objeto-Relacionais. É levado em consideração o esquema de mapeamento em si, a abstração acerca do modelo XML utilizado e o suporte a consultas sobre essa massa de dados. Nenhuma delas suporta documentos XML Ativos.

Tabela 1 - Técnicas de mapeamento XML – Relacional e Objeto Relacional, adaptada de Naughton (2003)

Técnica	Problemas abordados	Esquema XML considerado	Tipo de Mapeamento	Consultas suportadas	Suporte a XML Ativo

XPeranto (SHANMUGASUNDARAM *et al.*, 2001)	TC	Árvore	-	XQuery	-
SilkRoute (FERNANDEZ *et al.*, 2001)	TC	Árvore	-	XML-QL	-
Rolex (LI *et al.*, 2003)	TC	Árvore	-	XSLT	-
(BENEDIKT *et al.*, 2002)	TC	Recursivo	-	-	-
Oracle XML DB	EA,TC	Recursivo	CP	SQL/XML XPath com restrições 1	-
SQL Server 2000 SQLXML	EA,TC	Recursivo com profundidade limitada	CP	XPath com restrições 2	-
DB2 XML Extender (IBM, 2006)	TC, EA	Não recursivo	CP	Extensões SQL através de UDFs	-
Agora (MANOLESCU *et al.*, 2000)	TC, EA	Não recursivo	RGE	XQuery	-
MARS (DEUTSCH e TANNEN, 2003)	TC	Não recursivo		XQuery	-
STORED (DEUTSCH *et al.*, 1999)	EA,TC	Todas	REE	STORED	-
Edge (KOSSMANN, 1999)	EA,TC	Todas	RGE	Expressões de caminho	-
XRel (YOSHIKAWA *et al.*, 2001)	EA, TC	Todas	RGE	Expressões de caminho	-
(TATARINOV *et al.*, 2002a)	EA,TC	Todas	RGE	Consultas com ordenação	-
Intervalos Dinâmicos (DEHAAN *et al.*, 2003)	EA	Todas	RUE	XQuery	-
(SHANMUGASUNDARAM *et al.*, 1999)	EA	Recursivo	REE	-	-
(BOHANNON *et al.*, 2002)	EA	Árvore	REE	-	-

EA – Define Esquema de Armazenamento XML - Relacional
TC – Suporta Tradução de Consultas XQuery e XPath para SQL
XPath com restrições 1 = consultas com predicados de relacionamento com filhos e atributos
XPath com restrições 2 = consultas com predicados de relacionamento com filhos, atributos e pais
REE - representação específica para os elementos
RGE - representação genérica para os elementos
RUE - representação única para os elementos
CP – Caixa Preta

Vale ressaltar que as abordagens de mapeamento que, de acordo com a classificação apresentada, são de representação especifica para os elementos, possuem por si só uma restrição ao seu uso, que está relacionada à necessidade de existência de esquemas (DTDs ou XML Schemas) dos documentos a serem armazenados. Nem sempre documentos XML possuem tais esquemas ou possuem tal rigidez de estrutura.

3.2 Considerações Finais

Neste capítulo foram exploradas as principais estratégias de mapeamento XML para o modelo relacional encontradas na literatura. A grande maioria dos outros mapeamentos também encontrados na literatura e que não são abordados neste capítulo,

são variações ou extensões dos esquemas de mapeamento discutidos aqui. Foram analisados aspectos referentes à utilização desses esquemas de mapeamento e suas propriedades, como o suporte à execução, tradução de consultas e apresentação de algum suporte ao tratamento de propriedades dinâmicas em documentos XML.

Podemos observar que a utilização de SGBDR e SGBDORs para armazenamento de documentos XML tem sido aplamente discutida pela comunidade científica e tem sido bem aceita em cenários comerciais. Contudo, não foi encontrada nenhuma abordagem de mapeamento que tivesse algum tipo de preocupação com a adaptação de propriedades dinâmicas de documentos XML.

Entretanto, apesar dos esquemas de mapeamento apresentados não possuírem nenhuma estratégia associada ao armazenamento de documentos XML ativos, estes esquemas de mapeamento fornecem um suporte inicial. Deste modo, é necessário definir qual destes esquemas melhor se aplica ao problema de armazenamento de documentos XML ativos.

No próximo capítulo esta decisão é tomada e justificada, sendo apresentada a ARAXA, uma abordagem para armazenamento e gerência de documentos XML ativos, que tem como base o estudo realizado neste capítulo.

Capítulo 4 A abordagem ARAXA para armazenamento de documentos XML Ativos

A ausência de soluções que realizassem tanto o armazenamento quanto a manipulação de documentos XML ativos em um mesmo ambiente nos levou a analisar possíveis alternativas para a realização de tal tarefa. Ao considerarmos a utilização de SGBDOR ponderamos a possibilidade de definição de tipos,complexos, inerente ao modelo objeto-relacional. A vinculação de métodos definidos nesses objetos a funções/procedimentos definidos em SQL propicia uma alternativa para gerência das propriedades ativas dos documentos AXML de forma não intrusiva. Podemos assim, adicionar esses novos comportamentos ao processamento de consultas SQL, necessários para a gerência das chamadas de serviço contidas nos documentos AXML.

A quebra de paradigma entre a estrutura de dados XML e o modelo relacional pode ser abstraída através de processos de mapeamento. Até mesmo o processo de tradução de consultas, como analisado por Krishnamurthy et al (2003), pode ser realizado de forma automatizada, transparecendo ao usuário a utilização de consultas verborrágicas, comuns aos processos de mapeamento. O custo das junções, como avaliado por Shanmugasundaram et al (1999) e Khan (2001), não representa fator de impacto devido à maturidade da tecnologia envolvendo SGBDR e SGBDOR.

No capítulo anterior avaliamos os principais mapeamentos XML – Objeto-Relacional encontrados na literatura. Estes fornecem o ponto inicial para a definição da nossa solução. Neste capítulo, escolhemos o esquema de mapeamento a ser utilizado e apresentamos a ARAXA, nossa abordagem para armazenamento e gerência de documentos XML Ativos. ARAXA representa um acrônimo para **A**bordagem objeto-**R**elacional para **A**rmazenamento de documentos **X**ML **A**tivos. A abordagem ARAXA mantém o formalismo definido para os documentos XML ativos e ao mesmo tempo oferece recursos mais sofisticados de consulta e armazenamento através de tecnologias já consolidadas.

Devemos destacar que na ARAXA os documentos não são mapeados para uma estrutura de objetos, mas sim para relações. No entanto, a utilização de um SGBD Objeto-Relacional é justificada pelo fato de que somente esta classe de SGBDs oferece o recurso para a definição de objetos complexos em linguagem de alto nível (Linguagens de programação de 3ª geração, como Java, C++). Estes objetos, que podem

ter seus métodos associados a procedimentos e funções SQL, é que possibilitam a gerência e a execução das chamadas de serviço contidas nos documentos AXML. Devido ao fato de a ARAXA estar fortemente embasada sobre esta característica, a definimos como uma abordagem Objeto-Relacional.

Este capítulo está organizado como segue. Na Seção 4.1 apresentamos os principais conceitos relacionados à ARAXA, como o esquema de mapeamento de documentos XML, representação e gerência de chamadas de serviço e processamento de consultas. Por fim, na Seção 4.2 é apresentada uma breve conclusão acerca da abordagem ARAXA.

4.1 Armazenamento de documentos XML ativos na abordagem ARAXA

A abordagem ARAXA tem como ponto principal a utilização de SGBDs Objeto-Relacionais para o armazenamento de documentos AXML. Através de tipos definidos pelo usuário (usuário do SGBD) e métodos incorporados a estes tipos criamos uma estrutura para gerenciar as chamadas de serviços contidas nos documentos AXML.

O uso de SGBDs Objeto-Relacionais mantém coerência com os ambientes organizacionais e suas necessidades, uma vez que SGBDs Objeto-Relacionais são robustos tanto para o armazenamento quanto para consulta sobre os dados armazenados. Em tal cenário, a utilização de um repositório único é a abordagem ideal para armazenar os dados organizacionais. Desta forma, é possível auxiliar a integração de aplicações que fazem uso destes dados.

O ponto inicial para mapear documentos AXML para bancos de dados Objeto-Relacionais na ARAXA é o trabalho existente na literatura sobre o mapeamento XML-Relacional e a tradução de consultas XML para SQL (FLORESCU e KOSSMAN, 1999; TATARINOV *et al.*, 2002a). Dentre os esquemas de mapeamentos encontrados na literatura, o mapeamento utilizado na ARAXA foi o proposto por Tatarinov *et. al.* (2002a). Esta abordagem define um esquema genérico para armazenar documentos XML preservando a ordem dos elementos nos documentos, fazendo uso da técnica de numeração dos nós.

A ordenação é um ponto importante, uma vez que o uso de documentos da XML depende não apenas da representação dos dados expressa através de marcações, mas também da ordenação dos componentes dos dados (NAMBIAR *et al.*, 2002). Um dos esquemas de ordenação definido por Tatarinov foi baseado no modelo de codificação

Dewey (OCLC, 2002). Este tipo de codificação minimiza os custos de reordenação nos casos de atualizações sobre o documento (inserções e exclusões), na medida em que somente vizinhos dos nós que sofreram alterações e suas subárvores devem ser renumerados. Este esquema de ordenação foi o adotado na ARAXA.

Outro aspecto importante que foi considerado na escolha do modelo de mapeamento é que a estrutura escolhida fosse uma estrutura genérica capaz de representar qualquer documento XML, seja qual for o esquema e estrutura do documento XML a ser armazenado. Esta é uma característica importante ao se trabalhar com documentos que possuem estruturas dinâmicas. O resultado de uma chamada de serviço Web pode ser heterogêneo, ou seja, pode possuir uma subárvore com estrutura não previamente definida. Caso o mapeamento escolhido estivesse definido sobre um modelo que se prendesse à rigidez da estrutura dos documentos, tal fato acarretaria em alterações dinâmicas no esquema Objeto-Relacional. Estas alterações dinâmicas no esquema não são desejáveis para a administração da infra-estrutura de informação e aplicações que fazem uso desta infra-estrutura. Deste modo, apresentamos na próxima seção detalhes sobre o mapeamento dos documentos para o modelo Objeto-Relacional.

4.1.1 Mapeamento de documentos XML Ativos para SGBDORs

No mapeamento definido por Tatarinov (2002a) e adotado na ARAXA, duas relações são definidas como alvo do esquema de mapeamento XML-Relacional:

Edge (dewey, path_id, value);

Path (id, path);

Na relação *Edge*, o atributo *dewey* armazena o código *dewey* do nó, o atributo *value* armazena o valor do nó e o atributo *path_id* representa uma chave estrangeira para a relação *Path*. A relação *Path* armazena todas as expressões de caminho possíveis contidas nos documentos XML armazenados. Frequentemente expressões de caminho se repetem para diferentes nós dentro de um mesmo documento. Dessa forma, a utilização da relação *Path* representa uma compactação da informação a ser armazenada.

Contudo, alguns pontos devem ser observados neste esquema de mapeamento. Primeiro, ele não distingue elementos de atributos, ou seja, todos os nós são armazenados como se fossem elementos. Entretanto, é necessário distinguir este tipo de informação quando é preciso reconstruir o documento armazenado. Além deste

problema, outro ponto a ser considerado no mapeamento proposto por Tatarinov é que não há suporte ao armazenamento de coleções de documentos, pois sua proposta aborda um documento único.

Para superar estas limitações, propomos duas extensões ao modelo de Tatarinov e as aplicamos à ARAXA. A primeira extensão está direcionada ao armazenamento de atributos. Aqui nos beneficiamos do fato de que a ordem entre atributos de um mesmo elemento não é importante para o modelo XML. Assim, o armazenamento de atributos é feito da seguinte forma: é atribuído ao atributo o mesmo código dewey de seu elemento pai. Porém, para diferenciar um atributo de seu nó pai, é adicionado o símbolo "@" ao seu código dewey. Por exemplo, se um elemento tem código dewey "1.4.5", os seus atributos receberiam como código dewey "1.4.5.@". Dessa maneira é possível reconstruir o documento armazenado exatamente da mesma forma que ele se encontrava antes do mapeamento, sem que haja perdas de sua estrutura durante o processo.

A segunda extensão se refere ao armazenamento de múltiplos documentos. Para tal foi proposto a adição de uma nova relação no esquema de mapeamento, a relação *Document*.

Document (id, doc_name);

Também foi adicionada a coluna *doc_id* na relação *Edge*. Esta nova coluna representa uma chave estrangeira para a relação *Document*. A partir desta extensão, o esquema de mapeamento torna-se capaz de armazenar documentos distintos, na medida em que é possível identificar a qual documento cada nó armazenado pertence.

As extensões propostas na ARAXA para o modelo de Tatarinov podem ser visualizadas na Figura 13. Do lado esquerdo é exibida uma subárvore AXML onde é mostrado o esquema de ordenação dewey. Sobre ela, aplicamos a primeira extensão. O elemento *book* possui código dewey 1.1.1, e possui o atributo *id*, que possui código dewey 1.1.1.@. Do lado direito encontramos o esquema de relações utilizado no mapeamento de documentos AXML na ARAXA.

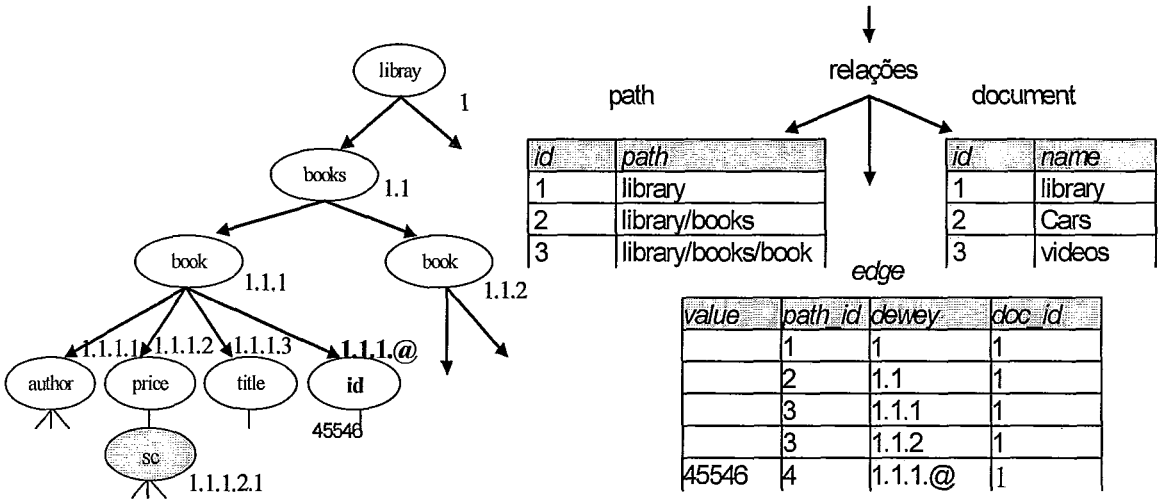


Figura 13- Extensões para proposta de Tatarinov

Nosso esquema de mapeamento também beneficia a avaliação seletiva de serviços. Quando há filtros na expressão de consulta é possível, através da junção da relação *Edge* com a relação *Path*, procurar por combinações nó-valor que satisfaçam à condição de filtro. Uma vez identificada a existência de combinações nó-valor que se encaixem na condição de seleção, é possível isolar sub-árvores que contribuem para o resultado da consulta. Desta forma podemos executar somente as chamadas de serviço que estão contidas nestas sub-árvores. Note que a junção entre as relações *Edge* e *Path* é usualmente mais rápida do que realizar todo o processo de execução de chamadas de serviços que serão irrelevantes para a solução da consulta.

Este mapeamento também favorece a pré-analise da estratégia LPQ, uma vez que a relação *Path* representa a estrutura compactada do documento, pois contém todas as expressões de caminho existentes nos documentos armazenados. A relação *Path* é utilizada para auxiliar a combinação entre expressões de caminho definidas pela estratégia LPQ com expressões de caminho que contêm chamadas de serviços dentro do documento.

Na Figura 14 é mostrado como a estratégia LPQ tira proveito do esquema de mapeamento definido. Temos como exemplo uma consulta que faz a busca por todos os preços de livros em um determinado documento AXML. A partir da definição das consultas lineares da LPQ verificamos a existência de chamadas de serviços (verificando o catálogo de serviços) que possuem expressão de caminho correspondente aos definidos nas consultas lineares da LPQ. No nosso exemplo, a consulta linear "livraria/livros/livro/preco/*()", que verifica a existência de chamadas de serviço ao fim desta expressão de caminho, pode ser aplicada à relação *path*, não havendo necessidade

de verificação de todo o documento. No caso de existirem tuplas que satisfaçam a esta condição, como a tupla de identificador 6 (seis) no exemplo, é realizada uma busca no Catálogo de Chamadas de Serviços por chamadas que possuam esta expressão de caminho. No exemplo, mostramos minimamente o Catálogo de Chamadas de Serviços, que será explorado na próxima seção.

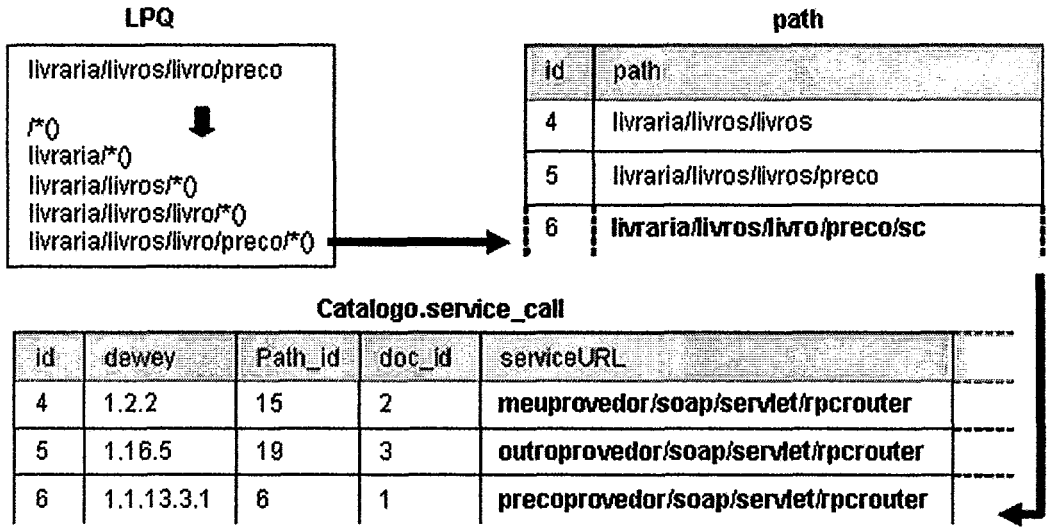


Figura 14 – Utilização da tabela *path*, associada ao catálogo de chamadas de serviços para beneficiar a estratégia LPQ

As características relacionadas à avaliação seletiva de chamadas de serviço com base na consulta submetida pelo usuário serão melhores exploradas na definição da arquitetura da ARAXA, no próximo capítulo.

4.1.2 Representação das chamadas de serviços em SGBDORs

Uma vez definido o esquema de mapeamento utilizado, definimos uma estratégia para gerenciar as propriedades dinâmicas dos documentos. Para gerenciar o comportamento ativo dos documentos de forma efetiva, criamos uma estrutura auxiliar para armazenar as chamadas de serviço, o Catálogo de Chamadas Serviços (CCS).

Inicialmente é necessário identificar e armazenar no CCS as chamadas de serviço. Este processo de identificação é relativamente simples devido ao fato de que elementos que contêm chamadas de serviços possuem padronização relativa ao seu nome, o elemento <sc>. Na abordagem ARAXA o CCS é representado por um esquema adicional no SGBD, possuindo a seguinte estrutura:

Service_call (id, path_id, dewey, doc_id, serviceURL, methodName, serviceNameSpace, useWSDLDefinition, signature, callalbe, frequency, lastcalled, followed, mode, doNesting)

Parameter (id, service_id, path_id, type, name)

A relação *Service_call* armazena todas as chamadas de serviços contidas em todos os documentos AXML. Também são armazenadas as informações de onde as chamadas de serviço aparecem (doc_id) e qual a sua localização dentro do documento (dewey). A relação também armazena os atributos das chamadas de serviço de acordo com o modelo AXML. A relação *Parameter* armazena os parâmetros que serão necessários ao se ativar a chamada de serviço.

A existência do CCS é muito importante para a gerência das chamadas de serviços, pois fornece acesso rápido a informações sobre as chamadas de serviço, que se encontram de forma compactada no catálogo. O CCS exerce, de certa forma, o papel de índice para as chamadas de serviço contidas nos documentos.

4.1.3 Gerenciamento de chamadas de serviços em SGBDORs

A partir do momento em que os documentos foram mapeados e armazenados, suas chamadas de serviço identificadas e armazenadas no CCS, é necessário uma infra-estrutura que seja capaz de efetivamente gerenciar estas chamadas de serviços. Estas chamadas de serviço podem ser realizadas em tempo de consulta ou em um tempo especifico, no caso de chamadas de serviços que têm periodicidade de execução definida. Nesta seção explicamos como essa infra-estrutura foi definida na ARAXA.

Na ARAXA, fazemos uso de tipos complexos definidos em linguagem de alto nível para montar uma infra-estrutura capaz de gerenciar as execuções das chamadas de serviço. Esta característica é um ponto comum em SGBDORs, e em parte, justifica nossa escolha por tais SGBDs, pois SGBDRs não fornecem tal serviço. A utilização destes objetos traz uma nova gama de possibilidades para a manipulação de dados e execuções de tarefas complexas, que normalmente fogem às características convencionais dos SGBDs, como a comunicação com provedores de serviços remotos.

Deste modo, na ARAXA, serviços são chamados através de consultas SQL do tipo *select execute_service(doc_id, service_id, dewey).* Na cláusula *select* descrita anteriormente, *execute_service* é uma função que está associada a um método, que pertence a um objeto que representa um cliente genérico para Serviços Web, também definido na estratégia da ARAXA.

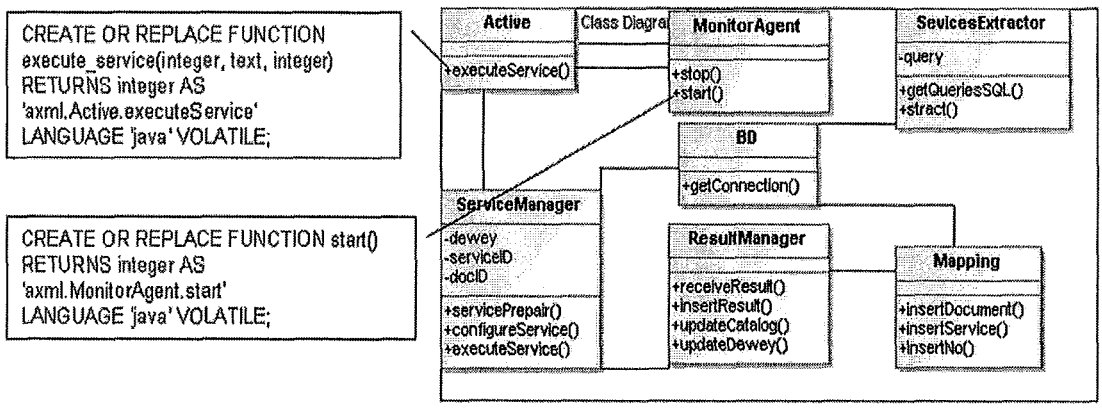


Figura 15 - Diagrama de classes dos componentes de armazenamento da ARAXA

Este cliente genérico para Serviços Web foi desenvolvido independentemente do SGBD hospedeiro, e posteriormente associado ao mesmo. Esta associação é feita de uma forma simples, através da definição de uma função no esquema do SGBD Objeto-Relacional hospedeiro. Esta função, por fim, aponta para o método do objeto cliente de Serviços Web. Este processo de associação está disponível na grande maioria dos SGBDs Objeto-Relacionais, desde que eles tenham suporte a linguagens de programação de alto nível.

Na Figura 15 apresentamos um diagrama simplificado da nossa infra-estrutura de gerência de serviços onde é exposto um exemplo de como esta infra-estrutura está acoplada ao SGBD. O diagrama de classes mostrado na figura foi implementado em Java, fora do SGBD. No SGBD, definimos funções que foram associadas a métodos implementados em nossa infra-estrutura. A definição da função *execute_service()* é um exemplo de utilização deste mecanismo. Ela está associada ao método *executeService*() da classe *Active*, que também é mostrada na Figura 15. A classe *Active* representa o cliente genérico de serviços Web anteriormente mencionado. Um objeto desta classe é capaz de encapsular mensagens e trocá-las com os provedores de serviços através de mensagens no protocolo SOAP.

Na Figura 15 também é mostrado a definição de uma função chamada *start(),* na parte inferior esquerda da figura. Em sua definição, esta função está associada ao método *start* da classe AgenteMonitor. Ao se executar uma chamada a esta função, a classe AgenteMonitor é chamada e realiza todo o processamento. Esta classe é responsável por monitorar de forma contínua o relógio do sistema, verificando a necessidade de execução de alguma chamada de serviço que tem seu disparo associado a alguma periodicidade, como definido no modelo AXML.

Ao definir as funções *start()* e *execute_service()* e acoplar a infraestrutura com a qual elas estão associadas no SGBD, fornecemos ao mesmo um mecanismo capaz de gerenciar as execuções de chamadas de serviços, sejam elas disparadas em momento de consulta ao documento ou a partir de intervalos de tempo pré-definidos. Dessa forma, abrangemos os dois comportamentos básicos para execuções de chamadas de serviço em documentos XML ativos.

A sintaxe apresentada na definição das funções *start() e execute_service()* é a sintaxe de definição de funções utilizada pelo PostegreSQL para realizar este tipo de associação. Todavia, mecanismos similares a este estão disponíveis em outros SGBDORs, como Oracle 9i (ORACLE, 2007), IBM DB2 (ALMEIDA *et al.*, 2000), entre outros. Na Figura 16 exemplificamos como é feita associação entre funções e métodos nos SGBDs Oracle 9i e IBM DB2.

```
-- exemplo de associação função – método no ORACLE
CREATE OR REPLACE PROCEDURE start ()
 AS LANGUAGE JAVA
 NAME 'axml.MonitorAgent.start()';

-- exemplo de associação função – método no DB2
CREATE PROCEDURE execute_service
        (
        IN serviceId INTEGER,
        IN dewey VARCHAR,
        IN docId INTEGER,
        )
        EXTERNAL NAME 'axml.Active.executeService'
```

Figura 16 – Exemplo de associação entre funções e métodos no Oracle 9i e IBM DB2

É importante destacar que a implementação pode ser desenvolvida externamente ao SGBD e posteriormente acoplada ao mesmo. Claramente este mecanismo não interfere na estrutura interna do SGBD, ou seja, não é necessário um recompilação do sistema. Sendo assim, nossa implementação do cliente de Serviços Web pode ser reutilizada em SGBDs diferentes, bem como toda a infra-estrutura fornecida pela ARAXA.

4.1.4 Metodologia para processamento de consultas e materialização

A execução de uma consulta sobre documentos AXML pode envolver a materialização de um conjunto de chamadas de serviços, cujos resultados serão necessários para formar o conjunto solução da resposta da consulta. Existem várias

alternativas para o processo de materialização de chamadas de serviços, envolvendo tanto a seleção do conjunto de chamadas de serviço a serem executadas quanto o plano de materialização para estas chamadas de serviço. Estratégias de otimização buscando a definição do conjunto mínimo de chamadas foram propostas em Abiteboul et al (2004a). Processos de definição e otimização de planos de materialização foram propostos em Ruberg et al (2006) e discutidos no capítulo 2.

Na ARAXA, o processamento de consultas sobre documentos AXML tem como base a análise da consulta, sua tradução para o paradigma relacional e sua reescrita para o tratamento do comportamento dinâmico dos documentos AXML. Deste modo nos beneficiamos de técnicas de tradução de consultas XQuery/XPath-SQL descritas na literatura, como as técnicas descritas em Krishnamurthy et al (2003), e também do processamento de consultas robusto encontrado nos SGBDOR.

Nossa metodologia envolve os seguintes passos a serem executados a partir da submissão da consulta:

1. Identificar quais chamadas de serviço devem ser executadas para responder à consulta;
2. Traduzir a consulta XQuery para SQL;
3. Identificar quais são as dependências entre as chamadas de serviços a serem executadas
4. Definir a ordem de execução das chamadas de serviços e executá-las;
5. Armazenar os resultados das chamadas de serviço;
6. Executar a consulta traduzida e reescrita;
7. Mapear o resultado em tuplas, do modelo relacional, para XML e retornar o resultado da consulta ao usuário

No passo 1 é feita a análise de quais serviços são relevantes para responder a consulta submetida. Esta etapa é necessária devido ao fato de que o resultado retornado por algumas chamadas de serviço não contribuem para responder a consulta (vide processo de avaliação seletiva de chamadas de serviços discutido no Capítulo 2). Para essa tarefa utilizamos algumas diretrizes apresentadas pelo processo de avaliação seletiva de consultas apresentado em Abiteboul et al (2004a).

No Passo 2 ocorre a tradução de consultas XQuery/XPath para SQL, baseada em algoritmos presentes na literatura. Durante o passo 3 é utilizado o gerador do grafo de dependências definido em Ruberg et al (2006) para elaborar o plano de materialização a ser utilizado, respeitando as dependências entre as chamadas de serviços. Feito isso, a

consulta SQL é reescrita, incluindo as consultas que efetivamente farão a execução das chamadas de serviço, usando a função *execute_service()*. Em Ruberg et al (2006) a avaliação dos planos de materialização ocorre com controle de delegação em uma rede ponto a ponto, com um sítio mestre que somente inicia a execução.

No passo 4 da nossa abordagem, utilizamos esse algoritmo somente para definir uma ordem de execução otimizada para as chamadas de serviço. Não há delegação no controle da execução. Deste modo, assumimos que o SGBD representa o sitio mestre e o orquestrador, e todas as execuções são controladas pelo mesmo sítio.

Na Figura 17 apresentamos um exemplo onde podemos visualizar melhor estes quatro passos iniciais. Em (A) mostramos uma sub-árvore de um documento AXML que tem uma chamada de serviço. Em (B) mostramos o processo de tradução de XPath → SQL. Em (C) adicionamos as consultas que são necessárias para a ativação dos serviços.

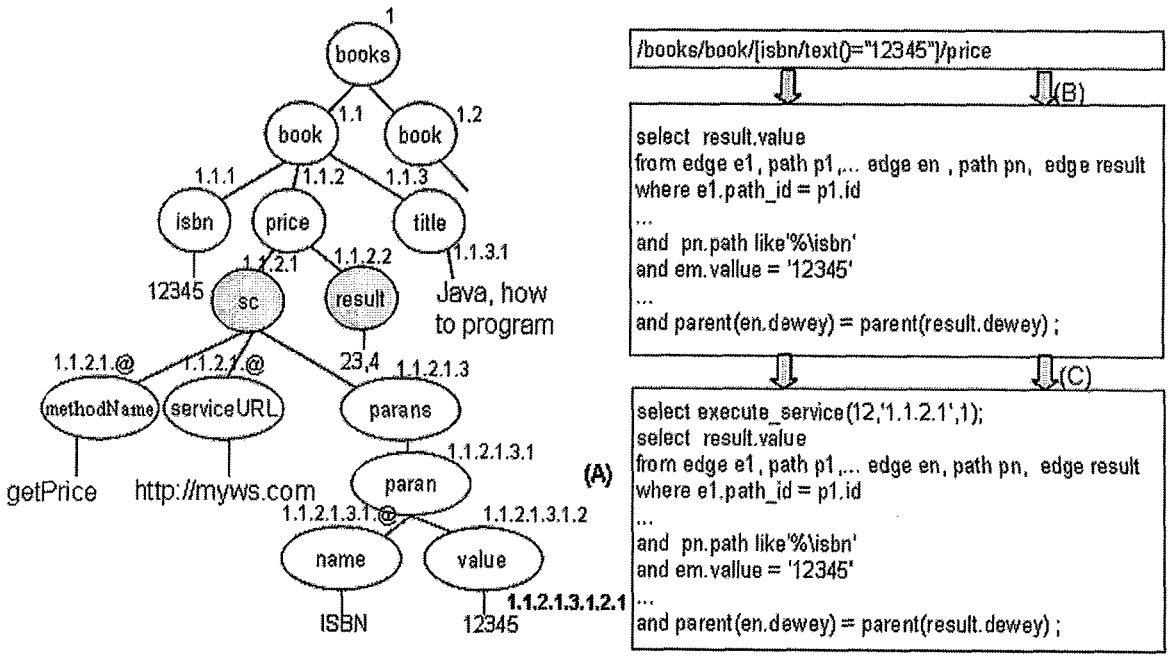


Figura 17 – Exemplo do mecanismo de tradução de consultas

No exemplo, o documento AXML possui informações sobre livros. Cada livro possui um preço, autores, identificação ISBN, etc. O preço do livro é uma informação dinâmica que é provida por uma chamada de serviço. O número ISBN é passado como parâmetro ao se recuperar o preço atual do livro. Quando o usuário submete uma consulta que busca o preço de um livro, a informação sobre o preço do livro necessita ser materializada, ou seja, é necessário executar a chamada de serviço que retorna o preço do livro buscado.

Após a tradução da consulta, adicionamos as chamadas da função *execute_service()* à consulta SQL, como mostrado em (C). Depois disso, a consulta final é executada pelo SGBD.

Na abordagem ARAXA a ordem de execução das chamadas de serviços é imposta no momento da inclusão das funções *execute_service()*. Chamadas de serviços que possuem alguma restrição na ordem de execução são definidas em consultas distintas, respeitando a ordem em que elas devem ser executadas.

select execute_service(...);

select execute_service(...);

Por outro lado, chamadas de serviço que podem ser executadas em paralelo são definidas na mesma consulta.

select execute_service(...), execute_service(...);

Os parâmetros necessários para a execução da chamada de serviço são retornados a partir do Catalogo de Chamadas de Serviços. É interessante notar que, como resultado do passo 4, temos um conjunto de consultas SQL.

No passo 5 os resultados de cada chamada de serviço são inseridos no documento AXML armazenado usando as mesmas regras de mapeamento utilizadas no mapeamento do documento AXML. O resultado da chamada de serviço é definido pelo elemento <result>, que é inserido como vizinho direito imediato do elemento <sc> que contém a chamada de serviço. O Catálogo de chamadas de serviços é então atualizado com informações relevantes como a hora da ultima execução de cada chamada de serviço executada, entre outros.

Assim, no passo 6 ocorre a execução da consulta SQL que se beneficia do processador de consultas do SGBD hospedeiro. Depois da execução da consulta, o resultado obtido (relacional), precisa ser mapeado para XML, como é esperado do resultado de uma consulta XPath/XQuery. Este processo é realizado no passo 7. A construção desse resultado é baseada na estrutura da consulta XQuery/XPath, ou seja, no que elas esperam como estrutura de resultado. Este pós-processamento ocorre devido ao fato de que a ARAXA tem como propósito a total transparência para o usuário final.

4.2 Considerações Finais

Foi apresentada neste capítulo a abordagem ARAXA para o armazenamento de documentos XML ativos. A ARAXA foi desenvolvida com base na realidade atual de armazenamento de dados em ambientes organizacionais, fornecendo aos atuais sistemas

Objeto-Relacionais de gerência de dados a possibilidade de armazenar essa nova estrutura de dados dinâmica, bem como processar consultas sobre essa nova massa de dados. Todavia, a ARAXA não só fornece suporte aos documentos XML ativos, mas também a documentos XML tradicionais. Outro ponto a ser destacado é a possibilidade de utilização da abordagem ARAXA em diversos SGBDORs, não se limitando a um único SGBDOR específico.

Podemos observar também que a utilização da função *execute_service()* fornece ao SGBD um novo mecanismo de integração de aplicações através de bases de dados, uma vez que pode ser utilizada fora do âmbito de documentos AXML. A infra-estrutura da *execute_service* fornece ao SGBD uma poderosa ferramenta de comunicação e troca de dados com provedores de serviços remotos, fornecendo interoperabilidade com baixo acoplamento.

No próximo capítulo, definimos uma arquitetura também proposta neste trabalho, moldada sobre os preceitos da ARAXA. Também abordaremos um protótipo que implementa a arquitetura definida.

Capítulo 5 ARAXA – Arquitetura e Protótipo

Neste capítulo apresentamos uma arquitetura proposta como parte desta dissertação sobre a abordagem ARAXA, apresentada no capitulo anterior. Procuramos definir uma arquitetura independente de produtos e tecnologias específicas, a qual pode ser aplicada sobre qualquer SGBDOR. Apresentamos também um protótipo onde foi aplicada e implementada a arquitetura proposta.

Este capítulo está organizado da seguinte forma: na Seção 5.1 apresentamos a arquitetura da ARAXA, onde são vistos os principais módulos nas subseções 5.1.1 e 5.1.2 . Já na Seção 5.2 apresentamos o protótipo desenvolvido. Seguindo a estrutura da arquitetura apresentada, o protótipo também é constituído de módulos, os quais apresentamos nas seções 5.2.1 e 5.2.2 . Por fim, na Seção 5.3 apresentamos algumas considerações sobre este capítulo, concluindo dessa forma.

5.1 Arquitetura da abordagem ARAXA

A arquitetura da ARAXA, como mostrada na Figura 18, é divida em dois módulos: o Módulo de Integração, responsável pelas atividades que ocorrem externamente ao SGBD, e o Módulo de Controle, responsável pelas atividades que ocorrem internamente ao SGBD.

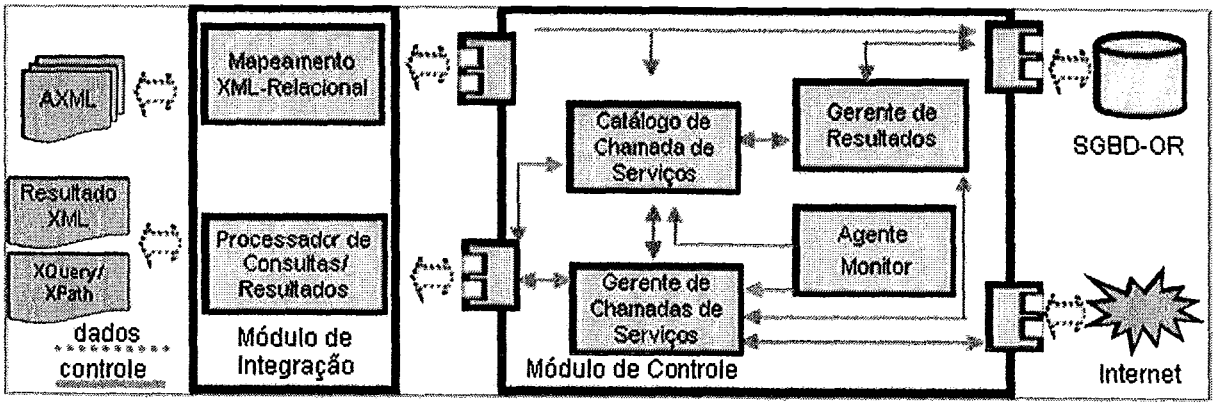


Figura 18-Arquitetura ARAXA

Como podemos observar na Figura 18, os documentos AXML/XML a serem armazenados são submetidos ao módulo de Integração, onde sofrem o processo de mapeamento para o modelo Objeto-Relacional. Durante este processo, o documento também é analisado, verificando e capturando as chamadas de serviços, caso o documento as contenha.

As informações sobre as chamadas de serviço são armazenadas no componente de Catálogo de Chamadas de Serviços (CCS). Encerra-se então o processo de armazenamento dos documentos. O documento AXML/XML pode ser recuperado aplicando-se o inverso do modelo de mapeamento utilizado. No nosso caso, utilizamos o modelo apresentado no capítulo anterior, que possui a possibilidade de reconstrução do documento AXML/XML sem nenhuma perda de estrutura.

Uma segunda forma de interação é provida através do processamento de consultas sobre os documentos armazenados. Através de uma interface de consulta, o usuário submete uma consulta XQuery/XPath à ARAXA. A consulta então é traduzida para uma consulta SQL. Ao mesmo tempo, é feita a análise sobre o impacto da consulta sobre as chamadas de serviços contidas nos documentos, onde são aplicados os algoritmos de avaliação seletiva de chamadas de serviço. Após a realização dessas duas atividades, as consultas responsáveis pela ativação das chamadas de serviço a serem executadas são incorporadas à consulta SQL traduzida. As consultas SQL resultantes são submetidas e resolvidas pelo SGBD.

Ao receber as consultas, obedecendo à ordem em que foram submetidas, o SGBD executa as funções responsáveis por ativar as chamadas de serviço, acionando o Módulo de Controle. O Módulo de Controle se responsabiliza pela execução das chamadas de serviço, solicitando a execução dos serviços aos provedores remotos e recebendo os resultados por eles retornados. É aplicado sobre os resultados obtidos o mesmo mapeamento que foi realizado inicialmente sobre todo o documento, contudo é levado em consideração que o resultado da execução da chamada de serviço é uma subárvore do documento. O resultado mapeado é então inserido no SGBD, mantendo a abstração que ele representa uma subárvore do documento armazenado. De acordo com a ordem imposta para a execução das consultas, a última representa a consulta SQL oriunda da tradução XQuery/XPath-SQL. Sendo assim, quando esta consulta é avaliada, o documento terá o seu conteúdo dinâmico, necessário para responder a consulta, já materializado, ou seja, todas as chamadas de serviços já estarão com os seus resultados disponíveis e armazenados no SGBD.

Após a avaliação da consulta, o resultado oferecido pelo SGBD é um conjunto de tuplas. Este conjunto sofre um novo processo de mapeamento, onde é transformado para a uma estrutura XML, formato no qual o usuário espera pelo resultado de sua consulta.

Detalhes sobre todo esse processo descrito anteriormente, além de algumas informações adicionais, como o papel do Agente Monitor, são apresentadas na seções 5.1.1 e 5.1.2 , onde detalhamos respectivamente o Módulo de Integração e o Módulo de controle.

5.1.1 Módulo de Integração

O Módulo de Integração é composto pelo Processador de Consultas e Resultados e pelo Mapeador XML-Relacional (MXR, para simplificar). O Módulo de Integração não está relacionado com o SGBD, sendo ele executado externamente ao SGBD, atuando como uma aplicação cliente do mesmo.

O MXR recebe um documento AXML/XML e o armazena em relações definidas pelo esquema de mapeamento já apresentado no capítulo anterior. Durante este processo, ele identifica as chamadas contidas no documento e armazena as informações sobre estas chamadas no Catálogo de Chamadas de Serviço (CCS). O CCS faz parte da abordagem ARAXA, e em nossa arquitetura encontra-se no Módulo de Controle, que será descrito na Seção 5.1.2 .

Na Figura 19 apresentamos um exemplo do mapeamento de um documento AXML para o esquema relacional. É apresentado o documento e parte de seus dados mapeados para as relações *edge, document* e *path*. Vale ressaltar que as informações do CCS não foram representadas, mas elas também são recolhidas neste momento.

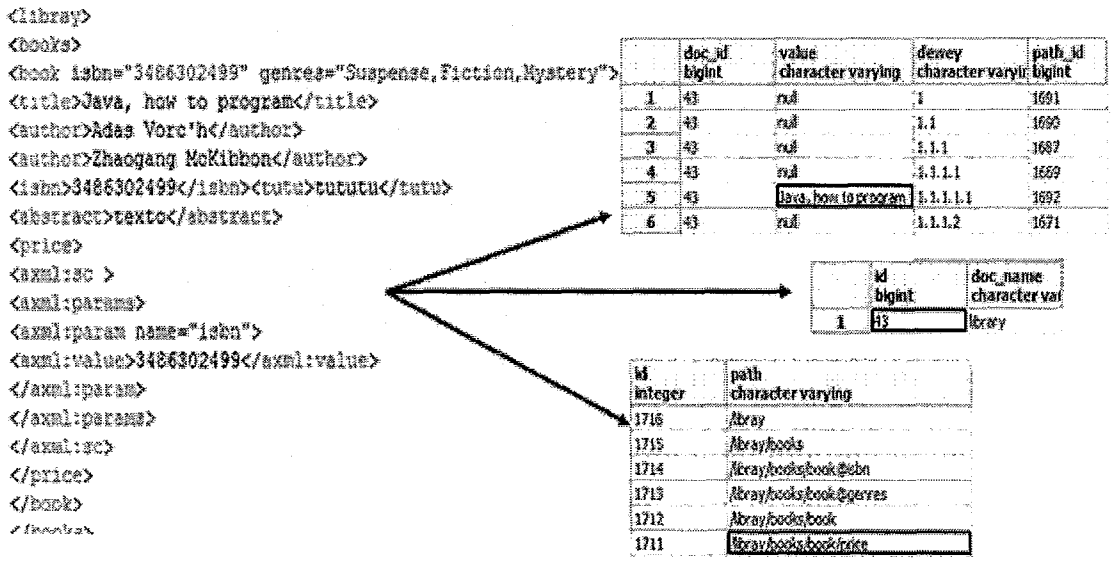


Figura 19 – Mapeamento do documento AXML para o esquema relacional

O componente Processador de Consultas e Resultado, PCR, é responsável pelas seguintes atividades:

1. Tradução da consulta XQuery/XPath, submetida pelo usuário, para um consulta SQL.
2. Análise da consulta submetida e aplicação de algoritmos de avaliação seletiva de chamadas de serviço, tendo como base a consulta submetida.
3. Definição de um plano de materialização para o conjunto de chamadas de serviço definido na atividade (2).
4. Integração da consulta SQL traduzida com as consultas que representam a execução de chamadas de serviço e submissão da consulta final ao SGBD.
5. Tradução do resultado retornado pelo SGBD para o formato XML, a ser exibido ao usuário.

Na atividade (1) o algoritmo usado para mapeamento de consultas foi baseado no algoritmo definido em alto nível por Tatarinov et al (2002b). Na Figura 20 é apresentado um exemplo desta tradução. A partir da consulta XPath apresentada, geramos a sua correspondente em SQL.


```
select * from (
select A.* from (
select * from (
select * from (
select A.* from(
select * from
document d inner join edge e on d.id = e.doc_id inner join path p on p.id = e.path_id
 where
( p.path similar to '/library[/|@]%' or p.path like '/library')A ) A
where
A.path similar to '/library/books[/|@]%' or A.path similar to'/library/books') A
where
A.path similar to '/library/books/book[/|@]%' or A.path similar to'/library/books/book') A inner join ( select * from (select * from (select * from (select
A.* from( select * from document d inner join edge e on d.id = e.doc_id inner join path p on p.id = e.path_id
where
( p.path similar to '/library[/|@]%' or p.path like '/library')A ) A
where
 A.path similar to '/library/books[/|@]%' or A.path similar to'/library/books') A where A.path similar to '/library/books/book[/|@]%' or A.path similar
to'/library/books/book') A
where
 (A.path similar to  '%/library/books/book/title/%' and A.value = 'Java, how to program')  ) B on substring(A.dewey,'[0-9]+[.][0-9]+[.][0-9]+') =
substring(B.dewey,'[0-9]+[.][0-9]+[.][0-9]+')) A
where
A.path similar to '/library/books/book[/|@]price[/|@]%' or A.path like '/library/books/book/price'
order by
CASE WHEN (split_part(A.dewey, '.',1)='') THEN 0 WHEN (split_part(A.dewey, '.',1)='@') THEN 0 else cast(split_part(A.dewey, '.',1) as int) end,
CASE WHEN (split_part(A.dewey, '.',2)='') THEN 0 WHEN (split_part(A.dewey, '.',2)='@') THEN 0 else cast(split_part(A.dewey, '.',2) as int) end,
CASE WHEN (split_part(A.dewey, '.',3)='') THEN 0 WHEN (split_part(A.dewey, '.',3)='@') THEN 0 else cast(split_part(A.dewey, '.',3) as int) end,
CASE WHEN (split_part(A.dewey, '.',4)='') THEN 0 WHEN (split_part(A.dewey, '.',4)='@') THEN 0 else cast(split_part(A.dewey, '.',4) as int) end,
CASE WHEN (split_part(A.dewey, '.',5)='') THEN 0 WHEN (split_part(A.dewey, '.',5)='@') THEN 0 else cast(split_part(A.dewey, '.',5) as int) end,
CASE WHEN (split_part(A.dewey, '.',6)='') THEN 0 WHEN (split_part(A.dewey, '.',6)='@') THEN 0 else cast(split_part(A.dewey, '.',6) as int) end,
CASE WHEN (split_part(A.dewey, '.',7)='') THEN 0 WHEN (split_part(A.dewey, '.',7)='@') THEN 0 else cast(split_part(A.dewey, '.',7) as int) end,
CASE WHEN (split_part(A.dewey, '.',8)='') THEN 0 WHEN (split_part(A.dewey, '.',8)='@') THEN 0 else cast(split_part(A.dewey, '.',8) as int) end
```

Figura 20 – Exemplo de tradução de consulta XPath para SQL

Para a atividade (2) existem várias heurísticas que podem ser aplicadas no processo de avaliação seletiva de serviços, como definido por Abiteboul *et al* (2004). Nesta dissertação, além da utilização do CCS, foram aplicadas duas estratégias, a LPQ

(ABITEBOUL *et al.*, 2004a) e a estratégia baseada da utilização de filtros, caso a consulta submetida os contenha.

Na Figura 21 temos um exemplo de utilização dessas duas estratégias. Em (A) temos uma consulta XPath que retorna o preço de um livro intitulado "Java, how to program". Em (B) temos uma subárvore do documento AXML que está sendo consultado. Nesta subárvore, podemos notar que a informação sobre preço é uma informação intensional, e que deve ser obtida através de uma chamada de serviço. Em (C) são destacadas as chamadas de serviços que são irrelevantes e que são excluídas usando o princípio da LPQ. Em (D) são destacadas as chamadas de serviço que podem ser ignoradas usando a seleção baseada em critérios de filtro, associadas à LPQ.

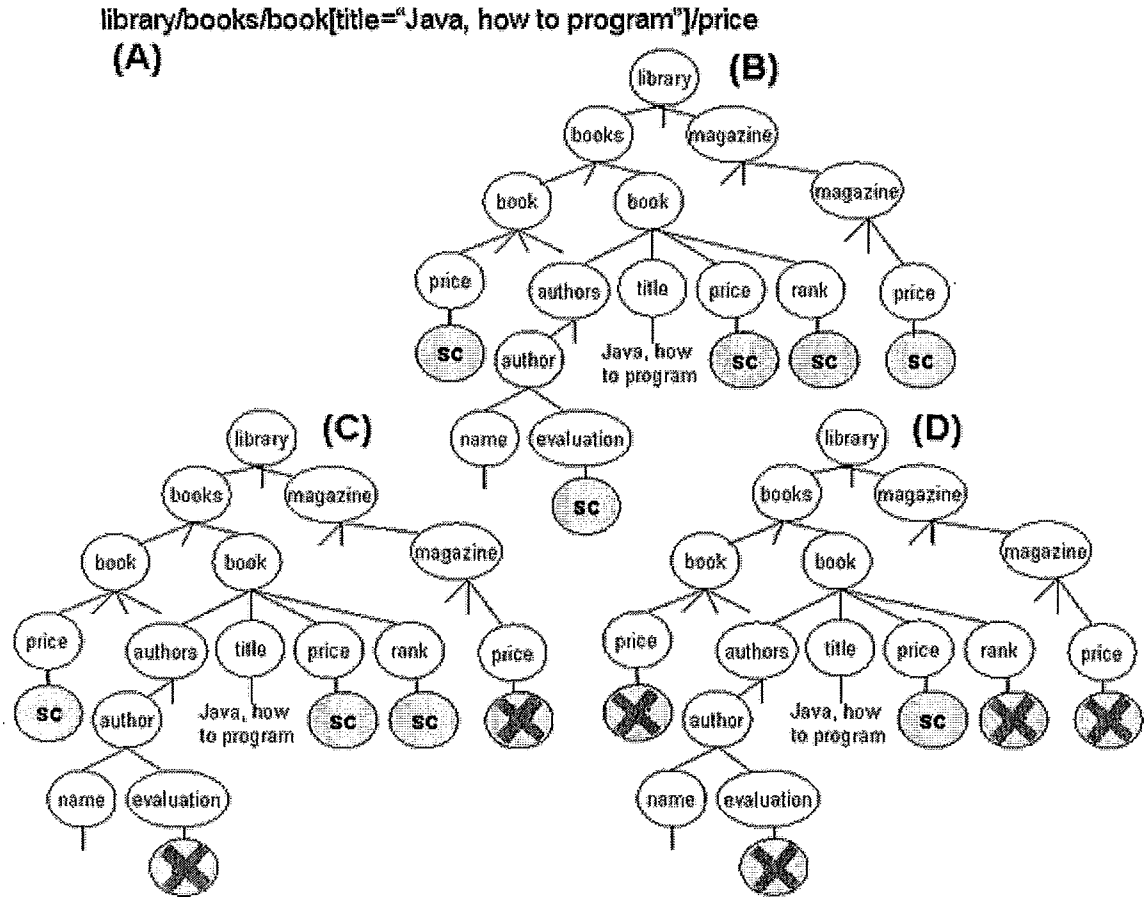


Figura 21- Exemplo de utilização das estratégias de LPQs e Filtros

No processo (3), é minimamente necessário definir um plano de materialização que respeite os dois tipos de restrições existentes entre as chamadas de serviço a serem executadas. Contudo, mecanismos de otimização do plano de materialização podem ser empregados nesse momento. Vale ressaltar que o problema de otimização de planos de materialização de documentos AXML é um problema NP – difícil (PEREIRA *et al.*, 2006). O XCraft, proposto por Ruberg *et al.* (2006) consiste em um otimizador para a

materialização de documentos AXML. O XCraft abrange um conjunto de técnicas que lidam principalmente com a complexidade e o dinamismo da materialização de documentos AXML.

No XCraft, uma álgebra de operadores baseados em serviços Web permite uma avaliação incremental de planos de materialização, reduzindo o espaço de busca ao mesmo tempo em que considera aspectos dinâmicos dos sítios envolvidos. Pereira et al (2006) propôs a SLS-MC (de "Stochastic Local Search with Multiple stop Conditions"), uma estratégia baseada em busca local estocástica com múltiplas condições de parada. A SLS-MC avalia múltiplas condições de parada para a busca por bons planos de materialização. Como o espaço de busca é muito grande neste problema, a busca não pode ser limitada a uma única condição de parada, como um número fixo de iterações, por exemplo. Conseqüentemente, ela reduz a sensibilidade do processo de otimização quanto à altura dos sub-planos gerados pelo XCraft.

Nossa arquitetura é capaz de incorporar otimizadores para geração de planos de materialização de documentos. No caso do XCraft, apesar de seu contexto ser um ambiente de redes ponto a ponto, ele possui arquitetura orientada a serviços, facilitando a integração com nossa arquitetura. O conjunto de informações necessárias ao XCraft para tomada de decisão podem ser facilmente incorporadas ao CCS.

Na atividade (4), as consultas SQL referentes a chamadas de serviços são encadeadas de acordo com o plano de materialização definido, e por fim, são concatenadas à consulta SQL traduzida. Este processo é relativamente simples, pois representa apenas a criação das consultas SQL com as funções que caracterizam as chamadas de serviço (*execute_service()*), obedecendo a ordem imposta na atividade (3). Após isso, é necessário concatenar essas consultas com a consulta SQL traduzida. Finalmente, a consulta é submetida ao SGBD. Na Figura 22 apresentamos um exemplo do encadeamento de consultas responsáveis pela execução das chamadas de serviço. Ilustramos a consulta traduzida concatenada a uma outra consulta que contém a chamada da função *execute_service(),* responsável pela invocação do serviço Web.

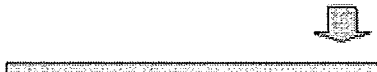
```
select execute_service(32,'1.4.5.7',334);
select * from (
select A.* from (
select * from (
select * from (
select A.* from(
select * from
document d inner join edge e on d.id = e.doc_id inner join path p on p.id = e.path_id
 where
...
```

Figura 22 – Exemplo de encadeamento de consultas para a execução das chamadas de serviço

Já na atividade (5), o Módulo de Controle intercepta o resultado relacional retornado pelo SGBD e, baseado na consulta XQuery/XPath originalmente submetida, o PCR traduz o conjunto de tuplas em um resultado em formato XML. O PCR segue nesta atividade os mesmos princípios de reconstrução de documentos XML que foram mapeados para o modelo relacional. Na Figura 23 temos um exemplo deste mapeamento, onde apresentamos o resultado da consulta traduzida, mostrada na Figura 20. Na parte superior da figura temos o resultado relacional, e na parte inferior temos o resultado em XML, que é exibido ao usuário.

Saída de Dados | Explicação | Mensagens | História

	id bigint	doc_id bigint	value character	dewey character varying	path_id bigint	id integer	path character varying
1	42	42	null	1.1.1.7	1686	1686	/library/books/book/price
2	43	43	null	1.1.1.7	1686	1686	/library/books/book/price
3	43	43	null	1.1.1.7.1	1683	1683	/library/books/book/price/axml:sc
4	42	42	null	1.1.1.7.1	1683	1683	/library/books/book/price/axml:sc
5	42	42	null	1.1.1.7.1.1	1682	1682	/library/books/book/price/axml:sc/axml:params
6	43	43	null	1.1.1.7.1.1	1682	1682	/library/books/book/price/axml:sc/axml:params
7	43	43	isbn	1.1.1.7.1.1.1.@	1681	1681	/library/books/book/price/axml:sc/axml:params/axml:param/@name
8	42	42	isbn	1.1.1.7.1.1.1.@	1681	1681	/library/books/book/price/axml:sc/axml:params/axml:param/@name
9	43	43	null	1.1.1.7.1.1.1	1680	1680	/library/books/book/price/axml:sc/axml:params/axml:param
10	42	42	null	1.1.1.7.1.1.1	1680	1680	/library/books/book/price/axml:sc/axml:params/axml:param
11	43	43	null	1.1.1.7.1.1.1.1	1679	1679	/library/books/book/price/axml:sc/axml:params/axml:param/axml:value
12	42	42	null	1.1.1.7.1.1.1.1	1679	1679	/library/books/book/price/axml:sc/axml:params/axml:param/axml:value
13	43	43	348630	1.1.1.7.1.1.1.1.1	1678	1678	/library/books/book/price/axml:sc/axml:params/axml:param/axml:value/#text
14	42	42	348630	1.1.1.7.1.1.1.1.1	1678	1678	/library/books/book/price/axml:sc/axml:params/axml:param/axml:value/#text
15	43	43	null	1.1.1.7.2	1685	1685	/library/books/book/price/result
16	42	42	null	1.1.1.7.2	1685	1685	/library/books/book/price/result
17	43	43	105	1.1.1.7.2.1	1684	1684	/library/books/book/price/result/#text
18	42	42	105	1.1.1.7.2.1	1684	1684	/library/books/book/price/result/#text



```
<price>
    105
</price>
```

Figura 23 – Exemplo de mapeamento do resultado relacional da consulta para o resultado em XML

5.1.2 Módulo de Controle

O Módulo de Controle é responsável por gerenciar a execução das chamadas de serviço, fazendo requisições aos provedores remotos de serviço e incorporando os

resultados obtidos nos documentos, aplicando o mapeamento necessário. Sua definição é feita como um conjunto de objetos que exercem tarefas bem definidas e que estão acoplados ao SBGD. Este módulo é composto pelo Catálogo de Chamadas de Serviços (CCS), pelo Gerente de Chamadas de Serviço (GCS), pelo Gerente de Resultados (GR) e pelo Agente Monitor (AGM).

Como discutido anteriormente em outras seções, o CCS é responsável por armazenar as informações contidas nas subárvores que representam chamadas de serviços. Representa, dessa forma, um mecanismo de rápida detecção de chamadas de serviços. Além das informações contidas nas subárvores de chamada de serviço, são armazenadas também as informações referentes à localização da chamada de serviço dentro do documento; a qual documento a chamada de serviço pertence; estatísticas sobre o histórico de chamadas disparadas e seus provedores (informações necessárias para o processo de otmização de planos de materialização, como disponibilidade do provedor, tempo de resposta, etc); e grafos de dependências entre as chamadas de serviço de um documento. Inicialmente o CCS é instanciado com as informações extraídas durante o processo de mapeamento do documento AXML.

O CCS provê informação para todos os outros componentes da arquitetura, exercendo o papel de guia em atividades de tomada de decisão pelos outros componentes. Da mesma forma que o CCS é consultado, ele também é atualizado pelos outros componentes. Entretanto, o CCS não executa nenhuma atividade no sistema. Ele é definido como um esquema adicional ao esquema de dados. Devido à sua simplicidade, o CCS pode ser facilmente estendido para incorporar novas necessidades de informação dos componentes do sistema.

O GCS representa um cliente genérico para Serviços Web. Ele é acionado pela função *execute_service* por meio do mecanismo de associação de funções a objetos, já abordado no capítulo anterior. Ao ser requisitado para execução de determinada chamada de serviço, o GCS define e parametriza a chamada de serviço, além de verificar se a chamada de serviço deve mesmo ser disparada. Uma vez parametrizada, a chamada de serviço é encapsulada em uma mensagem SOAP e enviada ao provedor remoto do serviço. O GCS aguarda até o momento em que o provedor remoto de serviço envia o resultado de sua requisição. Ao receber o resultado, o GCS instancia o GR, repassando como parâmetro a chamada de serviço que foi disparada e o seu resultado.

O GR realiza o desempacotamento do resultado da chamada de serviço, que se encontra originalmente em uma mensagem SOAP, e aplica sobre ele o mesmo

mapeamento imposto ao documento original. Por fim, o GR insere o resultado como uma subárvore do documento. Ou seja, o GR é responsável pela materialização efetiva do conteúdo intensional. Ao fazer isto, provavelmente será necessário realizar alterações na numeração do código dewey dos nós do documento armazenado. O GR realiza também esta tarefa.

Entretanto, antes de realizar este processo, o GR consulta o CCS para definir o comportamento de materialização, que de acordo com o modelo AXML, pode apresentar duas possibilidades distintas: *replace* – substitui a subárvore de resultado de execuções anteriores da mesma chamada pela nova subárvore, ou *append* – adiciona a nova subárvore como vizinho esquerdo imediato das subárvores de resultados anteriores. Ao final deste processo, o GR insere e atualiza no CCS algumas informações sobre a chamada de serviço executada, como a data-hora em que a chamada de serviço foi ativada e o tempo gasto para tal.

O AGM é o responsável por monitorar o relógio do sistema, verificando a necessidade de execução de chamadas de serviço que possuem periodicidade de execução definida. Este comportamento é configurado para cada chamada de serviço no momento do projeto do documento AXML, usando o atributo *frequency* do elemento <sc>. Consequentemente, essa informação encontra-se disponível no CCS. O AGM representa um processo cujo estado final não é determinado, ou seja, é um processo residente no SGBD hospedeiro, monitorando de forma contínua o CCS. Quando é identificada uma chamada de serviço que deve ser disparada, o agente instancia o GCS que cuidará de todo o resto do processo de materialização, conforme descrito anteriormente.

A inicialização do AGM é realizada através da execução de uma função que está associada a um método de uma classe da mesma forma que é ativado o objeto que instancia o GCS. O GR não necessita deste mecanismo devido ao fato de que o próprio GCS o instancia.

5.2 Protótipo

Nesta seção apresentamos um protótipo que foi desenvolvido para confirmarmos nossos argumentos acerca da ARAXA. Tivemos a preocupação de utilizar somente tecnologias abertas e softwares livres, que não representam nenhum custo adicional para a adoção de nossa solução. Nós utilizamos o PostgreSQL 8.2.4 (2007) como nosso SGBDOR hospedeiro. O PostgreSQL é um sistema de gerência de dados poderoso, que

possui código fonte aberto. Possui mais de 15 anos do desenvolvimento ativo, possuindo arquitetura comprovadamente de qualidade, que lhe garante confiabilidade, integridade de dados, e exatidão.

De forma geral, as tecnologias utilizadas no desenvolvimento do protótipo foram a linguagem Java e APIs Java para manipulação de documentos XML como o XERCES, JDOM e para aplicações com Serviços Web como as APIs HTTP, SOAP, WSDL4J, Activation, Mail, e Qname. Na Figura 24 apresentamos as camadas tecnológicas utilizadas na construção do protótipo.

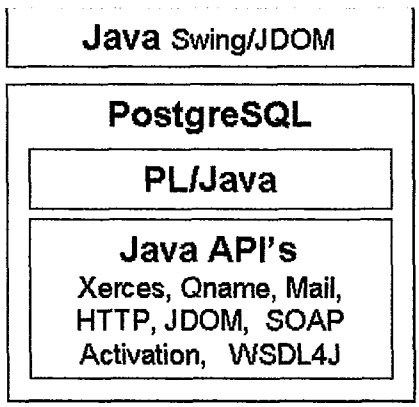


Figura 24 - Tecnologias utilizadas em nosso protótipo

Os diagramas de classes tanto do Módulo de Controle quanto do Módulo de Integração encontram-se disponibilizados no Anexo B.

5.2.1 Modulo de Integração

O Módulo de Integração foi desenvolvido como uma aplicação cliente na linguagem JAVA, tendo interface gráfica *swing*. A Aplicação se conecta ao SGBD e se comunica com ele através de *driver* JDBC. É através deste módulo que o usuário interage com os outros componentes do protótipo.

Através da interface gráfica, o usuário seleciona o documento AXML que deve ser armazenado. Então, o Mapeador AXML-Relacional é instanciado, mapeando o documento e recolhendo as informações sobre as chamadas de serviço. Através da interface gráfica o usuário submete consultas aos documentos armazenados. É nesse momento é que instanciado o Tradutor de Consultas.

O Tradutor de Consultas foi desenvolvido por Medeiros e Taok (2007) como Trabalho de Conclusão de Curso de graduação. Teve como base o algoritmo em alto nível proposto em Tatarinov et al (2002b) para a tradução de consultas XPath para SQL.

Para a avaliação seletiva de consultas foram implementadas duas estratégias, a aplicação de critérios de filtros e a estratégia LPQ. As duas estratégias foram implementadas apenas realizando a análise da consulta, não sendo executado nenhum outro processo iterativo durante o processo de materialização dos resultados.

Para a construção de planos de materialização ainda não foi implementada nenhuma otimização, contudo as dependências entre as chamadas de serviço são respeitadas. Nas próximas versões, está prevista a integração com o otimizador XCraft (RUBERG e MATTOSO, 2006).

Algumas telas capturadas da aplicação que corresponde ao Módulo de Integração podem ser vistas no Anexo A.

5.2.2 Módulo de Controle

O Módulo de Controle foi implementado em JAVA, que é a linguagem que a maioria dos SGBDOR oferece como linguagem de definição de objetos complexos, e acoplado ao PostgreSQL utilizando a PL/Java (PLJAVA, 2006). A PL/Java é um módulo acoplável, de código aberto, que permite a definição de procedimentos, gatilhos e funções em Java no PostgreSQL. O seu desenvolvimento começou no final de 2003 e a primeira versão da PL/Java foi apresentada em janeiro 2005. Hoje encontra-se como componente opcional na instalação do PostgreSQL. A PL/Java permite fazer uso de APIs para programação em Java, além da fornecida pela própria linguagem. Deste modo, podemos fazer uso de APIs para desenvolvimento utilizando XML e Serviços Web.

A PL/Java fornece ao PostgreSQL a possibilidade de definição de tipos complexos em uma linguagem de alto nível, como é possivel em outros SGBDORs amplamente difundidos no mercado como o Oracle9i (2007) e o DB2 (ALMEIDA *et al.*, 2000) entre outros. O uso da PL/Java permitiu um baixo acoplamento entre a implementação do Módulo de Controle e o SGBD hospedeiro. Este baixo acoplamento ocorre devido ao fato de que a implementação pôde ser desenvolvida independentemente, e posteriormente associada ao SGBD através do mecanismo de associação entre as funções definidas no esquema do banco de dados e os métodos dos objetos desenvolvidos.

Os componentes do Módulo de Controle, com exceção do CCS, são classes Java que podem fazer uso de todo o poder desta linguagem, inclusive se comunicar com

provedores remotos de serviço. Estas classes são empacotadas e instaladas no SGBD. A instalação destes pacotes usando PL/Java é feita da seguinte forma:

```
SELECT sqlj.install_jar('file:///claudio/bibliotecas/axml.jar', 'axml', false);
SELECT sqlj.set_classpath('public','xercesImpl:xmlrel:http:soap:activation:mail:axml');
```

A primeira consulta instala o pacote no SGBD, tendo como parâmetros a localização do pacote no sistema de arquivos, o nome associado a este pacote e por último um valor booleano definindo a existência de um descritor. A segunda consulta define a dependência em relação a outras APIs que foram previamente instaladas. Uma vez instalados os pacotes desenvolvidos, pode-se então definir funções e procedimentos que apontam para métodos de classes destes pacotes. Em nosso protótipo definimos duas funções, a função *execute_service()* e a *start()*.

A função *execute_service()* está associada ao método *executeService* da classe *Active*. Esta função recebe como parâmetros de entrada o identificador do documento ao qual pertence a chamada de serviço; o identificador do serviço, que é um identificador artificial do serviço, utilizado no CCS; e o código dewey do elemento <sc> que representa essa chamada de serviço. A classe *Active,* por sua vez, instancia um objeto GCS. O GCS faz uso das APIs WSDL4J, Activation, Mail, SOAP e HTTP para realizar as execuções das chamadas de serviço. Ao receber o resultado da chamada de serviço, o GCS instancia um objeto GR.

O GR, ao ser instanciado, recebe como parâmetro o identificador da chamada de serviço, o identificador do documento ao qual pertence a chamada de serviço, o código dewey dessa chamada e o objeto que corresponde à resposta recebida.

O GR por sua vez instancia um objeto Mapeamento, que corresponde ao mesmo definido no Módulo de Integração. Aplica então o mapeamento ao resultado e insere o resultado no banco de dados, materializando a chamada de serviço. Depois disso, atualiza o CCS, mantendo atualizadas as informações referentes à ultima execução da chamada de serviço. Estas informações são utilizadas para o processamento de chamadas de serviços definidas sobre uma periodicidade, além de constituírem dados estatísticos para possíveis processos de otimização nas próximas execuções. Após a execução de cada instância do GR, todo o processo de materialização de uma chamada de serviço é finalizado.

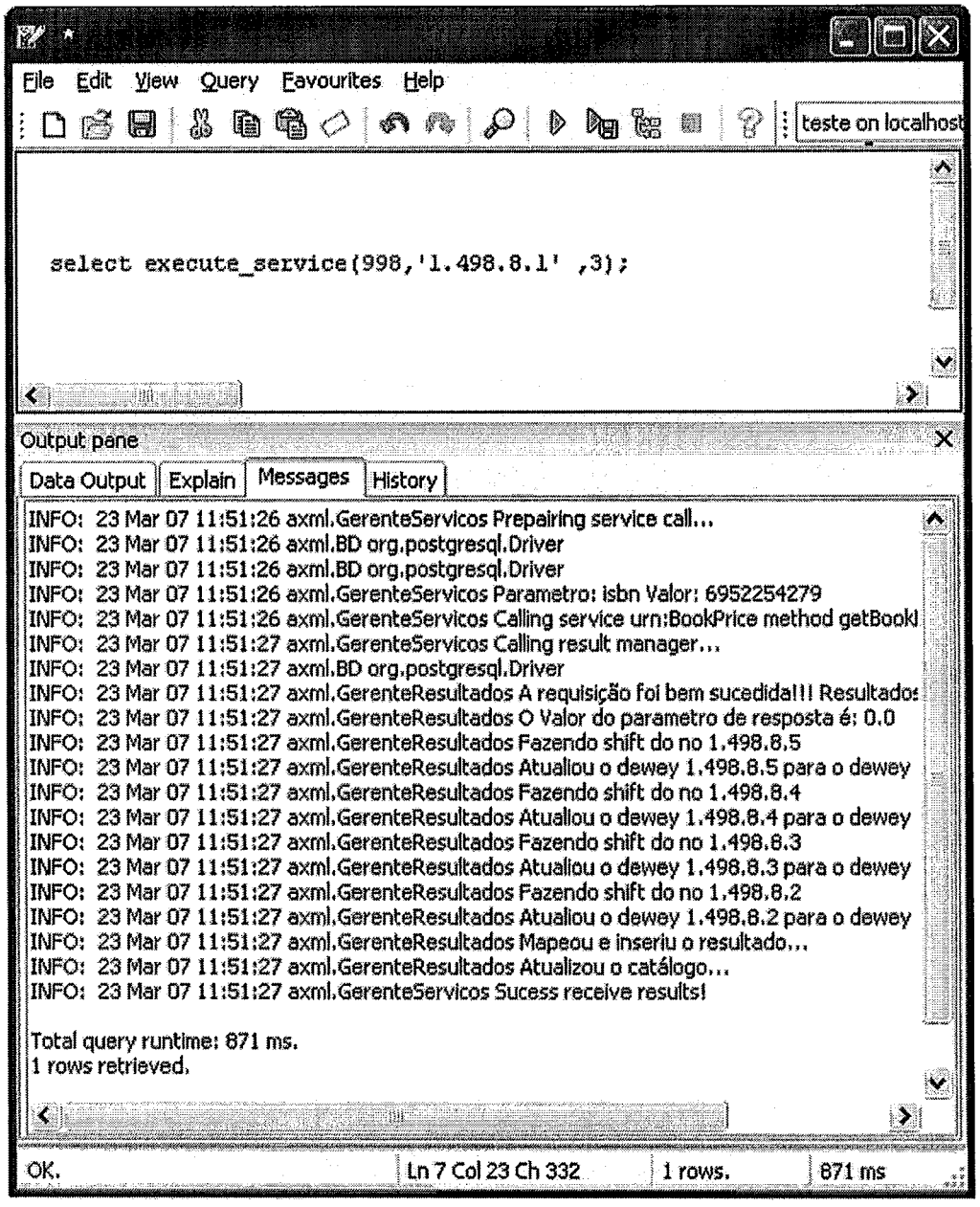


Figura 25 – Exemplo de execução da função *execute_service*

Na Figura 25 apresentamos um exemplo da execução da função *execute_service*. Neste exemplo foi escolhida uma chamada de serviço contida em um documento AXML mapeado. A partir da submissão da consulta ao SGBD foram rastreadas todas as etapas de sua execução, indicando o processo descrito para a materialização de uma chamada de serviço.

A função *start()* realiza a inicialização do AGM. Ela está associada ao método *start* da classe AgenteMonitor, que representa o componente AGM. O AGM faz uso de *threads* para monitorar o relógio sistema, de forma a não comprometer o desempenho do mesmo. Para não penalizar o sistema com o processamento de uma tarefa contínua, é possível configurar o tempo entre uma varredura e outra.

5.3 Considerações finais

Apresentamos neste capítulo uma arquitetura para a nossa abordagem ARAXA. Mostramos que essa arquitetura não se restringe a um único SGBD, mas a toda classe de SGBDs Objeto-Relacional, além de não interferir no ambiente organizacional, preservando as tecnologias vigentes. Portanto, nossa arquitetura apresenta uma solução para o problema de armazenamento de documentos AXML de uma forma não intrusiva.

Fornecemos, através desta arquitetura, os serviços de armazenamento e consulta de documentos XML estáticos (XML convencional), e como principal objetivo, fornecemos também, de forma transparente ao usuário, o armazenamento e consulta de documentos AXML. Todo o aspecto dinâmico constituído pela parte ativa dos documentos AXML é gerenciado de forma integrada ao SGBD hospedeiro, sobretudo, de forma não intrusiva. Vale ressaltar que, a partir desta arquitetura, também conseguimos atingir um dos propósitos iniciais desta dissertação, pois dados nos modelos XML, AXML, Relacional e Objeto-Relacional convivem de maneira harmoniosa em um único SGBD.

Mostramos também a viabilidade de nossa arquitetura através da implementação de um protótipo. Em nosso protótipo utilizamos tecnologias e ferramentas "livres", que não representam investimento adicional para o suporte de documentos AXML.

No protótipo foram contemplados todos os requisitos básicos de gerência de execução de chamadas de serviço típicos dos documentos AXML. Critérios de otimização de planos de materialização não foram contemplados em nossa primeira versão do protótipo, apesar de serem previstos em nossa arquitetura. Isto se deve ao fato de não estarmos inicialmente preocupados com a obtenção de resultados ótimos, mas sim com a validade e aplicabilidade de nossa solução.

No próximo capítulo apresentamos alguns resultados experimentais utilizando configurações variadas de testes em nosso protótipo, a partir dos quais podemos fazer uma melhor avaliação da nossa solução.

Capítulo 6 Avaliação Experimental

Apresentamos neste trabalho uma abordagem para armazenamento de documentos XML Ativos utilizando SGBD Objeto-Relacional, a ARAXA. Para a validação desta abordagem implementamos um protótipo. A partir da construção do protótipo, buscamos realizar a validação dos conceitos e metodologias abordados na ARAXA, além de demonstrar a aplicabilidade da alternativa adotada. Neste capítulo definimos um estudo experimental usando como ferramental o protótipo implementado e diferentes cenários de experimentação, a fim de avaliar de uma forma geral a ARAXA. Este capítulo está organizado da seguinte forma: na Seção 6.1 definimos o estudo experimental e suas configurações; na Seção 6.2 falamos sobre o planejamento da execução desse estudo; na Seção 6.3 discutimos a execução das avaliações propriamente ditas; na Seção 6.4 analisamos os resultados obtidos e na Seção 6.5 concluímos este capítulo fazendo algumas considerações finais.

6.1 Definição do Estudo Experimental

Nesta seção definimos o estudo experimental e seu objetivo.

Objeto de Estudo: Realizar a validação dos conceitos e metodologia apresentada na abordagem ARAXA, bem como a viabilidade e aplicabilidade desta alternativa para o armazenamento de documentos XML Ativos. Através deste estudo procuramos analisar o acoplamento da gerência de dados XML Ativos a SGBDs Objeto-Relacionais. Procuramos também analisar o impacto dos mapeamentos entre os paradigmas XML e Objeto-Relacional no contexto de processamento de consultas e materialização de documentos AXML.

Perspectiva: o estudo foi desenvolvido sob a lógica do processamento de consultas e materialização apresentada na abordagem ARAXA. Avaliamos o tempo gasto em cada etapa definida em nossa metodologia de execução de consultas. Avaliamos também o impacto da nossa abordagem sobre o comportamento e desempenho do SGBDOR utilizado. Não realizamos comparações das nossas execuções com o tempo de execução do ActiveXML (ACTIVEXML, 2007) uma vez que esta abordagem é baseada em sistemas de arquivos e não oferece funcionalidade similar à apresentada em nossa abordagem.

Também não avaliamos o tempo necessário para o mapeamento do documento AXML para o SGBD Objeto-Relacional, bem como o caminho oposto devido ao fato de que estas operações ocorrem em poucos momentos e não representam um fator crítico de acordo com nossa abordagem.

Utilizando a notação baseada em VAN SOLINGEN *et al* (1999), podemos direcionar o nosso estudo da seguinte forma:

> **Analisar o** emprego e a aplicabilidade da abordagem ARAXA para o armazenamento de documentos XML Ativos, além do impacto da mesma em SGBDs Objeto-Relacionais **com o propósito de** validar a metodologia utilizada na ARAXA **referente** aos ganhos obtidos por sua utilização e funcionalidades disponibilizadas **do ponto de vista** do tempo de processamento de consultas e materialização e sobrecarga devido a aplicação da nossa estratégia **no contexto** de armazenamento e gerência de documentos XML Ativos.

6.1.1 Definição dos cenários para avaliação

Nesta seção apresentaremos os cenários definidos para a avaliação experimental. Serão discutidas características dos documentos avaliados, das consultas submetidas e do ambiente de *software* e *hardware* utilizado.

Definição de documentos: Para a avaliação experimental foram utilizados documentos XML Ativos com configurações que possuem variações tanto em seu tamanho quanto em sua estrutura.

A variação sobre o nível de estruturação do documento justifica-se pela necessidade de se avaliar as alternativas implementadas para a avaliação seletiva em momento de consulta, as estratégias LPQ e Filtros. A diversidade na estruturação dos documentos também torna a avaliação mais próxima da diversidade de documentos existentes em ambientes organizacionais. Dessa forma, temos duas classes de documentos, de acordo com o nível de estruturação: documentos fortemente estruturados e documentos pouco estruturados.

A variação sobre o tamanho é necessária para se avaliar a escalabilidade da solução e para realizar a verificação de algum gargalo em relação ao tamanho de documentos, analisando o comportamento da ARAXA na medida em que se varia o tamanho dos documentos. Vale ressaltar que o tamanho físico de um documento XML, ou seja, o tamanho que ele ocupa em disco, não representa o seu tamanho quando

manipulado em memória. Isto porque a estrutura criada em memória para a manipulação dos documentos é muito mais complexa do que a estrutura armazenada em disco. Sendo assim, apesar desta relação tamanho em disco × tamanho em memória variar de acordo com a estrutura do documento, o espaço a ser endereçado por memória é muito maior do que o espaço em disco para o mesmo documento. Este problema é contornado na ARAXA, uma vez que para processar consultas não é feita a manipulação da estrutura XML em memória. Os únicos momentos onde isto é necessário são no armazenamento e na recuperação do documento AXML por inteiro.

O conteúdo dos documentos fortemente estruturados é composto por um catálogo de livros e revistas (elementos *book* e *magazine*). O conteúdo dos documentos pouco estruturados é composto por um catálogo de itens em geral. Sendo que, dentre esses itens, existem livros (elementos *book*). Tanto elementos *item* quanto *book* possuem um elemento filho que define seu preço, o elemento *price*. Como o preço de determinado livro ou revista constitui uma informação dinâmica, ele é definido como uma chamada de serviço web, que faz uma requisição ao fornecedor do item ou do livro, retornando o atual preço de determinado livro ou revista.

Para os documentos pouco estruturados, ou seja, os documentos que possuem um catálogo de itens, os outros elementos (*item*) não possuem informação sobre o preço, mas possuem um elemento que representa as informações de entrega, que são definidas de acordo com um determinado CEP. Estas informações incluem, por exemplo, se haverá taxa de entrega ou não, de acordo com a promoção do mês. Portanto, trata-se uma propriedade dinâmica que também foi definida com uma chamada de serviço. O elemento referente ao livro não tem chamada de serviço para informações de entrega, pois não se cobra frete de livros.

Para a geração dos documentos foi utilizada a ferramenta de geração ToXgene (BARBOSA *et al.*, 2002). Toxgene é um gerador de coleções de documentos XML grandes e consistentes, baseado em *templates*. Através de uma linguagem de especificação de templates, a TSL, a ferramenta oferece a possibilidade de que os usuários possam controlar a estrutura e conteúdo dos documentos a serem gerados.

Dessa forma, definimos nossos próprios *templates* tomando como base os *templates* disponibilizados na ferramenta Toxgene. Como base, foram utilizados os *templates XMark* e *Catalog*. Foram necessárias algumas customizações para a construção de novos *templates* devido à necessidade de incorporação de chamadas de

serviço no conteúdo dos documentos e também para que fosse obtida a variação desejada no tamanho e na estrutura dos documentos.

Observando então as características a serem analisadas, foram gerados os seguintes documentos:

- **Documento 1 fortemente estruturado:** documento de pequeno porte com cerca de 100 Kb de tamanho ocupado em disco. O seu conteúdo é composto por um catálogo com 10 elementos, divididos entre revistas e livros, e suas respectivas subárvores. Destes 10 itens, a proporção é de 5 livros e de 5 revistas. Deste modo, o documento possui 10 chamadas de serviços embutidas em seu conteúdo.
- **Documento 1 pouco estruturado:** documento de pequeno porte com aproximadamente 100 Kb de tamanho ocupado em disco. O seu conteúdo é composto por um catálogo com 10 itens e suas respectivas subárvores. Destes 10 itens, apenas 1 elemento representa um elemento livro. Este livro possui em sua subárvore um elemento filho que representa um capítulo do livro que é oferecido ao comprador como forma de incentivo à compra. Contudo, o capítulo oferecido varia de acordo com o tempo. Deste modo, esta informação está definida de forma intencional, sob a forma de uma chamada de serviço. Diante disso, o documento possui ao todo, 11 chamadas de serviço, 9 sobre informações de entrega, 1 para o preço do livro e 1 para o capítulo promocional.
- **Documento 2 fortemente estruturado:** documento de grande porte com cerca de 10Mb de tamanho ocupado em disco. O seu conteúdo é composto por um catálogo com 70 elementos, também divididos entre revistas e livros, e suas respectivas informações. Destes 70 elementos, a proporção é de 35 livros e 35 revistas. Deste modo, o documento possui 70 chamadas de serviços embutidas em seu conteúdo.
- **Documento 2 pouco estruturado:** documento de grande porte com cerca de 10 Mb de tamanho ocupado em disco. O seu conteúdo é composto por um catálogo de itens e suas respectivas informações. Destes itens, apenas 10 elementos representam livros, sendo que cada livro possui apenas uma informação intencional, que constitui seu preço. Existem distribuídas pelo conteúdo do documento mais 60 chamadas de serviços que representam instruções de entrega

de seus respectivos itens. Dessa forma, no total, existem 70 chamadas de serviço em todo o documento.

- **Documento 3 fortemente estruturado:** documento de grande porte com cerca de 100 Mb de tamanho ocupado em disco. Com este tamanho, o documento apresenta uma grande dificuldade de ser gerenciado em memória por aplicações específicas de manipulação de documentos XML. O seu conteúdo é composto por um catálogo com 140 elementos, entre revistas e livros, e suas respectivas informações. Destes 140 itens, a proporção é de 70 livros e 70 revistas. Portanto o documento possui 140 chamadas de serviços embutidas em seu conteúdo.
- **Documento 3 pouco estruturado:** documento de grande porte com cerca de 100 Mb de tamanho em disco, o mesmo tamanho do documento anterior, o que representa a mesma dificuldade de manipulação anteriormente mencionada. O seu conteúdo também é composto por um catálogo de itens. Destes itens, apenas 10 elementos representam livros. Existem distribuídas pelo conteúdo do documento mais 130 chamadas de serviços que representam instruções de entrega de seus respectivos itens. Dessa forma, no total, existem 140 chamadas de serviço em todo o documento.

Ao se tratar de documentos XML, os documentos de tipo 3 são documentos muito grandes, e que na maioria das aplicações comerciais de manipulação de documentos XML seria dificilmente gerenciado. Como exemplo desta dificuldade podemos destacar a ferramenta *XMLSpy* (ALTOVA, 2007), uma das ferramentas para manipulação e consulta de documentos XML mais comercializadas, que em testes, não consegue ao menos abrir um documento deste tamanho (em um computador com 2 Gb de memória RAM). Isto demonstra a fragilidade de ferramentas que mesclam tarefas tipicamente relacionadas a SGBDs às suas funcionalidades.

Definição das consultas a serem executadas: como nesta avaliação estamos interessados em analisar o processamento de consultas que envolvam a gerência de conteúdo dinâmico, é preciso definir consultas que incidam sobre a parte intencional do documento AXML a ser consultado. Também iremos ponderar as duas estratégias de avaliação seletiva de chamadas de serviço em tempo de execução de consulta. Portanto, definimos dois tipos de consultas. Apesar destas consultas serem simples, seu processamento abrange todo o processo de resposta a consultas definido na abordagem ARAXA. Os dois tipos de consulta são:

1. **Consulta com expressão de caminho simples:** buscamos com essa consulta avaliar o uso da estratégia de avaliação seletiva LPQ, apresentada no capítulo anterior. Neste tipo de consulta não são definidos critérios de seleção especifico sobre determinado grupo de elementos. As consultas utilizadas neste caso foram:

   Consulta Q1: */catalog/book/price*, executada nos documentos bem estruturados;

   Consulta Q2: */item_list/book/price*, executada nos documentos pouco estruturados;

2. **Consulta com expressão de caminho com filtro em elemento:** buscamos com essa consulta avaliar o uso da estratégia baseada em critérios de filtro para a avaliação seletiva de chamadas de serviços, apresentada no capítulo anterior. Neste tipo de consulta, são definidos critérios de seleção específicos sobre determinado elemento ou grupo de elementos. Esta estratégia é bem mais eficiente, contudo só pode ser aplicada em consultas que possuam critérios de filtro definidos. Para esta avaliação foram executadas as seguintes consultas:

   Consulta Q3: */catalog/book[isbn="5950193442"]*, executada nos documentos bem estruturados;

   Consulta Q4: */item_list/book[isbn="7813071809"]*, executada nos documentos pouco estruturados;

   Consulta Q5: */item_list/book[isbn="7813071809"]/freeChapter*, executada no documento 1 pouco estruturado;

A identificação de qual estratégia será utilizada para a avaliação seletiva de chamadas de serviço ocorre de forma automática a partir da detecção de critérios de filtro presentes na consulta submetida. Utilizamos o princípio de que, se existem critérios de filtro na consulta, a abordagem a ser utilizada para a avaliação seletiva será sempre a abordagem baseada em filtros.

Infra-estrutura e ambiente computacional: definimos aqui a plataforma computacional utilizada para a realização desta avaliação experimental:

- **Software:**
  - Sistema Operacional: Windows Vista Home Premium Edition
  - SGBDOR hospedeiro: PostgreSQL 8.2
  - Compilador Java: Java 6.0
  - Máquina Virtual Java: JRE1.6.0_02
- **Hardware:**

- Processador: Intel(R) Core(TM)2 CPU T5300 1.73 GHz 1.73 GHz
- Memória RAM: 2038 MB
- Tecnologia Toshiba Mobile

- **Rede:** o sítio provedor de serviços web utilizado na avaliação experimental esta conectado ao SGBD hospedeiro através de um enlace Internet.

6.2 Planejamento do Estudo Experimental

Para cada documento gerado foram realizadas duas consultas (com exceção do documento 1 pouco estruturado, onde foram executadas 3 consultas): uma consulta com expressão de caminho com filtro em elemento e uma outra consulta com expressão de caminho simples, sendo que, para cada consulta foram realizadas 10 (dez) repetições. A partir dessas repetições foi retirada a média em cada etapa executada, constituindo os valores a serem apresentados em nossos resultados.

Para a coleta de dados foi utilizada uma ferramenta desenvolvida em Java para a captura do tempo gasto na execução de cada etapa definida. Os comandos de coleta de dados foram embutidos no código dos componentes de protótipo e não interferem na avaliação.

Seguindo nossa metodologia para o processamento de consultas, definida no Capítulo 4, estabelecemos as etapas que ocorrem durante o processo de execução de consultas e serão mensuradas em relação à variável tempo. Estas etapas são:

1. Tradução da consulta XPath para SQL: etapa onde ocorre o processo de tradução da consulta submetida pelo usuário no formato XPath para o formato SQL. O produto desta etapa é a consulta SQL.

2. Identificação das chamadas de serviços a serem executadas: etapa onde são aplicadas estratégias de avaliação seletiva de chamadas de serviço. Nesta etapa também é realizada a definição das dependências entre as chamadas de serviço e a definição da ordem de execução do conjunto de chamadas de serviço a serem executadas. O produto desta etapa é a definição do conjunto de chamadas de serviço a serem executadas cujos resultados possivelmente compõem o resultado da consulta submetida, com uma ordem de execução definida. Contudo, no protótipo não foi implementado nenhum mecanismo para otimização da seqüência de execuções.

3. Parametrização das chamadas de serviço: esta etapa corresponde basicamente à coleta de todas informações necessárias para a execução das chamadas de serviço, tais como parâmetros a serem utilizados em cada requisição e recuperação de seus valores na base de dados. As mensagens SOAP de requisição são criadas e enviadas aos provedores remotos de serviços.
4. Execução dos serviços nos sítios remotos: esta etapa não representa um passo da nossa metodologia, mas por ser uma etapa importante, também foi mensurada. Ela representa o tempo gasto na troca de mensagens, tempo de comunicação e tempo de execução do serviço no sítio remoto.
5. Materialização: nesta etapa, as respostas obtidas a partir das chamadas de serviço são mapeadas seguindo o mesmo processo de mapeamento definido para o documento e incorporadas ao documento mapeado, completando o processo de materialização.
6. Execução da consulta SQL: etapa onde a consulta SQL correspondente à consulta XPath submetida é avaliada pelo SGBD.
7. Tradução do resultado relacional para formato XML: corresponde à etapa onde ocorre a tradução do resultado da consulta da etapa anterior, que é constituído por um conjunto de tuplas. Este conjunto de tuplas é mapeado para um resultado no formato XML.

Dentre as etapas descritas anteriormente podemos destacar três que representam uma sobrecarga causada pela adoção da abordagem ARAXA: a etapa 1, a etapa 5 e a etapa 7. As etapas 1 e 7 representam mapeamentos e traduções que representam as mudanças de paradigmas relacionais e XML. Já a etapa 5, que corresponde ao processo de materialização dos resultados das chamadas de serviço no documento AXML, é uma parte inerente do processamento de chamadas de serviço dos documentos AXML. Contudo, devido à necessidade de se empregar novamente o algoritmo de mapeamento sobre o resultado da chamada de serviço, assumimos esta etapa como uma sobrecarga, mas vale a pena lembrar que parte desta etapa, a atualização do documento AXML, é parte do processo de materialização canônico de documentos AXML, que sempre será executado, independente da abordagem de armazenamento utilizada.

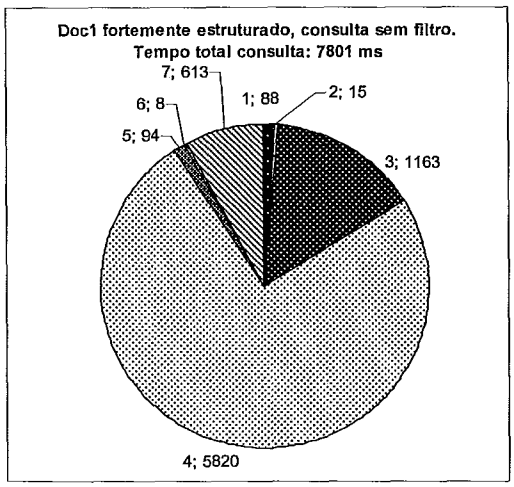
Portanto, a partir dessas três etapas é que definimos a sobrecarga da adoção da ARAXA, uma vez que as outras etapas devem possuir etapas equivalentes em qualquer outra abordagem, pois executam tarefas que fazem parte do processamento de consultas sobre documentos AXML.

6.3 Execução do Estudo Experimental

Mostramos a seguir, nas próximas seções, a execução deste estudo experimental, levando em consideração cada documento e consultas executadas sobre ele:

6.3.1 Avaliação 1 - documento 1 fortemente estruturado

No documento 1 fortemente estruturado foram submetidas duas consultas, uma consulta com critério de filtro e outra sem critério de filtro sobre elemento. Na Figura 26 apresentamos o gráfico do processamento da consulta Q1 (sem critérios de filtro) em relação ao tempo, dado em milissegundos.



Legenda:
X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:
1-Tradução da consulta XPath para SQL
2-Identificação das chamadas de serviços a serem executadas
3-Parametrização das chamadas de serviço
4-Execução dos serviços nos sites remotos
5-Materialização
6- Execução da consulta SQL
7-Tradução do resultado relacional para formato XML

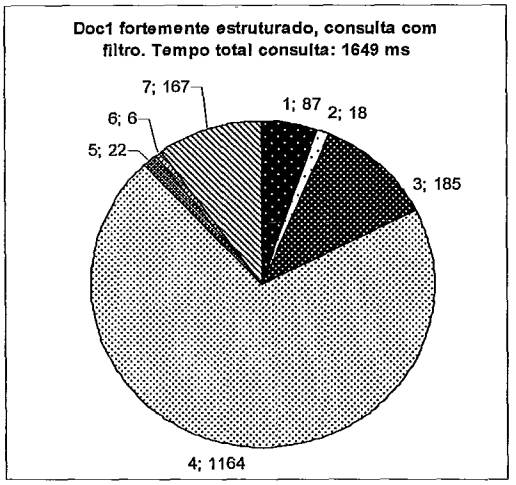
Figura 26 – Processamento de consulta sem filtro sobre o documento 1 fortemente estruturado

No processamento desta consulta, em grande parte do tempo total, o sistema fica "ocioso" na etapa 4, esperando pelos resultados da execução dos serviços nos sítios remotos. Contudo, esta parcela de tempo seria a mesma para qualquer abordagem utilizada para armazenamento de documentos AXML. Isto decorre do fato de que este tempo é dependente de fatores externos como disponibilidade do provedor remoto para executar a requisição, ambiente de rede, e troca de mensagens através dos protocolos de serviços Web.

No processamento desta consulta não foram executadas todas as chamadas de serviço contidas no documento, uma vez que através da utilização da estratégia LPQ conseguimos reduzir o número de chamadas a serem executadas (de 10 para 5). Ou seja, como as informações referentes a revistas não eram relevantes para a resultado da consulta, não foram executadas chamadas de serviços referentes a estas subárvores.

Na execução desta consulta, as etapas 1, 5 e 7 representam, em seu total, cerca de 10% do todo o tempo de processamento da consulta, ou seja, temos 10% de sobrecarga na execução desta consulta.

Na Figura 27 apresentamos o gráfico do processamento da consulta Q3 (com critério de filtro) em relação ao tempo, dado em milissegundos. Como o valor do atributo ISBN é único dentro da coleção de elementos *book*, o número de chamadas de serviço a serem executadas foi reduzido a uma única chamada de serviço. Deste modo, apesar do documento possuir 10 chamadas de serviço em seu conteúdo, apenas uma é relevante para o resultado da consulta.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

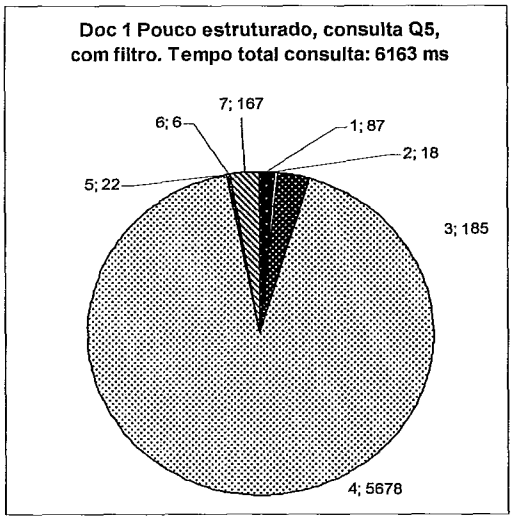
Figura 27 – Processamento de consulta sem filtro sobre o documento 1 fortemente estruturado

Nesta execução, a sobrecarga é de 17% em relação ao tempo total de execução da consulta.

Na Figura 28 temos a execução da consulta Q5, que retorna ao usuário o conteúdo referente ao capítulo do livro disponibilizado gratuitamente (elemento *freeChapter*) de um determinado elemento *book*. Neste caso, a sobrecarga é de 4.5% sobre o tempo total do processamento da consulta. Essa queda deve-se ao fato de que a resposta do serviço invocado possui tamanho significantemente superior ao do serviço de verificação de preço, o que implica em um tempo maior para transferência de dados (que depende das características de enlace ao qual estão conectados o consumidor e o provedor do serviço Web).

Através desta execução podemos verificar que tanto características dos serviços a serem executados (podem exigir um poder de processamento maior, acarretando maior tempo de processamento) quanto dos enlaces de rede dos envolvidos interferem na taxa de sobrecarga, vista em relação ao tempo. Deste modo, à medida em que o tempo da etapa 4 tende a ser maior, a taxa de sobrecarga diminui.

Esta característica foi explorada apenas neste documento. Nos demais seguiremos o padrão de execução de apenas duas consultas.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

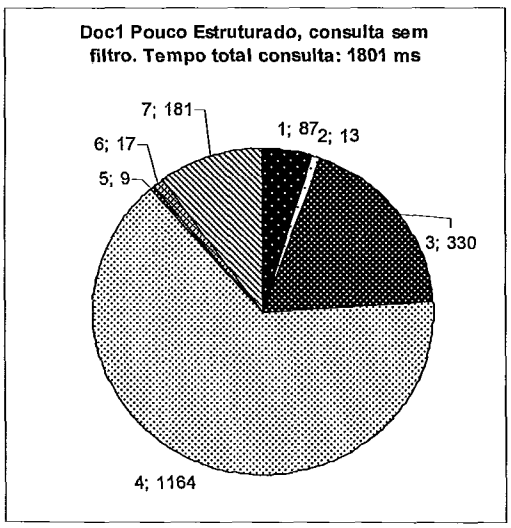
6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

Figura 28 - Processamento de consulta Q5, com filtro sobre o documento 1 pouco estruturado

6.3.2 Avaliação 2 - documento 1 pouco estruturado

No documento 1 pouco estruturado também estão presentes 10 chamadas de serviços. Contudo, há somente um elemento *book*. Portanto, com a utilização da LPQ podemos reduzir o número de chamadas de serviço a serem executadas a apenas uma única chamada de serviço. Na Figura 29 encontra-se o gráfico referente à execução da consulta Q2 (sem critério de filtro).



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

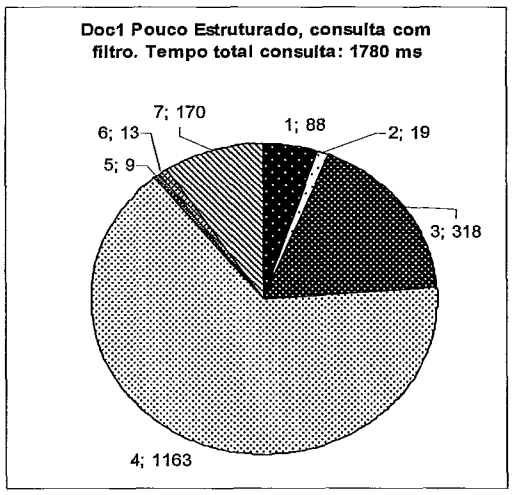
6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

Figura 29 – Processamento de consulta sem filtro sobre o documento 1 pouco estruturado

Podemos observar que a eficiência da estratégia é maior em documentos com maior variação em sua estrutura, uma vez que possuem menos elementos com a mesma expressão de caminho, característica principal utilizada na estratégia LPQ. Nesta execução a sobrecarga é de 15.3%, levando em consideração os critérios já definidos.

Na Figura 30 é mostrado o gráfico referente à execução da consulta Q4 (com critério de filtro).



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

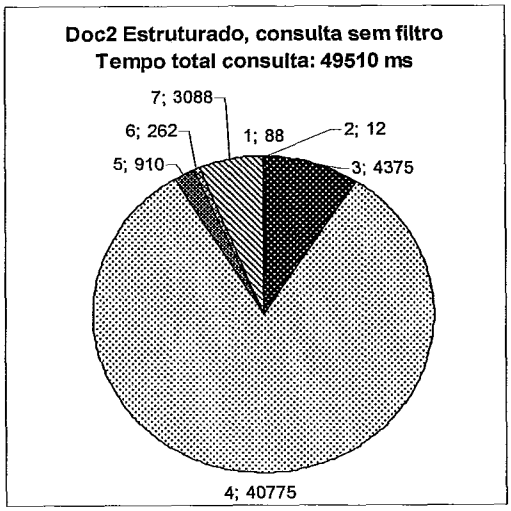
Figura 30 – Processamento de consulta sem filtro sobre o documento 1 pouco estruturado

Neste caso, a execução da consulta utilizando critério de filtro não difere muito da execução da consulta utilizando LPQ, uma vez que a utilização da LPQ também resultou em apenas uma chamada de serviço. O gráfico da Figura 30 segue a mesmas proporções de tempo em cada etapa das do gráfico da Figura 29, e estes também possuem valores próximos de tempo total de processamento de consulta. Nesta avaliação, a sobrecarga é de 15% sobre o tempo total de processamento da consulta.

6.3.3 Avaliação 3 - documento 2 fortemente estruturado

Avaliamos nesta seção o documento 2 fortemente estruturado. Na Figura 31 encontra-se o gráfico referente à execução da consulta Q1 (sem critérios de filtro).

Como neste documento existem 70 chamadas de serviço, mas apenas 35 elementos são elementos *book*, a utilização da LPQ faz com que sejam executadas somente as 35 chamadas de serviço necessárias para a construção do resultado da consulta.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

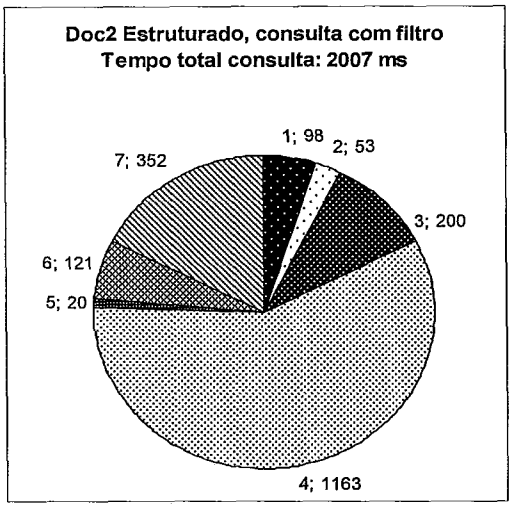
6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

Figura 31 - Processamento de consulta sem filtro sobre o documento 2 fortemente estruturado

Podemos notar que a sobrecarga acarretada pela ARAXA nesta execução é de 8.3%. Este percentual diminuiu em relação aos outros devido ao fato da execução de um número maior de chamadas de serviço.

Já na Figura 32 é apresentado o gráfico da execução da consulta Q3 sobre este documento.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

6- Execução da consulta SQL

7-Tradução do resultado relacional para formato XML

Figura 32 - Processamento de consulta com filtro sobre o documento 2 fortemente estruturado

A sobrecarga de mapeamentos nesta execução foi de 23.4%. Contudo o tempo total da consulta é de cerca de 2 segundos, ou seja, o impacto da realização dos mapeamentos da ARAXA não representa um gargalo para o tempo total de execução da consulta, pois o tempo total da consulta permanece aceitável.

6.3.4 Avaliação 4 - documento 2 pouco estruturado

O documento 2 pouco estruturado possui o mesmo tamanho em disco do documento anterior, mas difere em sua estrutura. A primeira consulta executada sobre

esse documento foi a consulta Q2 (sem critérios de filtro). Na Figura 33 podemos visualizar o gráfico referente ao processamento desta consulta.

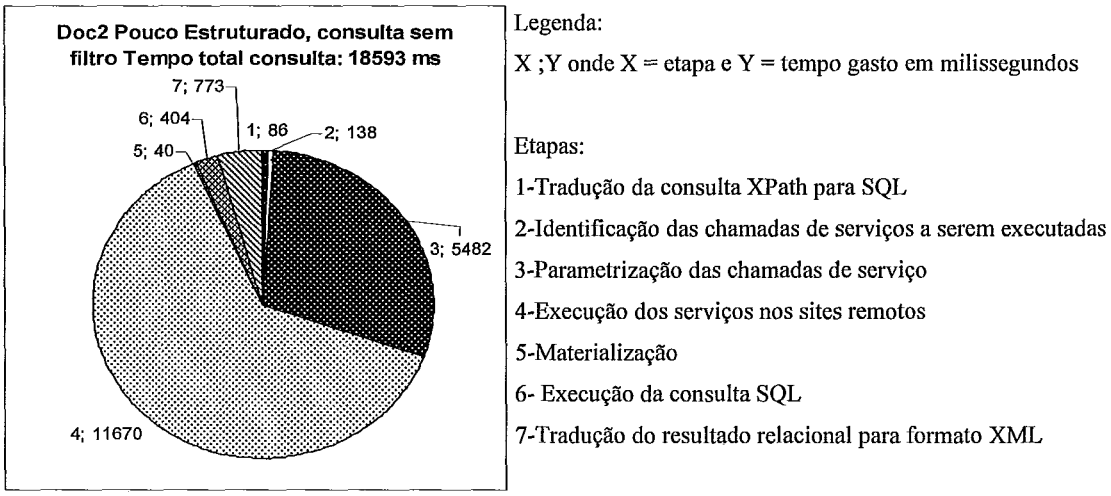
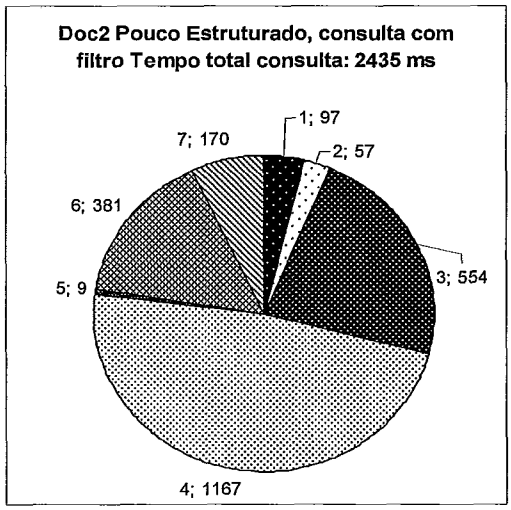


Figura 33 - Processamento de consulta com filtro sobre o documento 2 fortemente estruturado

Apesar do documento conter 70 chamadas de serviço, apenas 10 são referentes a preço de livro, uma vez que neste documento existem 10 elementos *book*. Sendo assim, fazendo uso da estratégia LPQ, somente essas 10 chamadas de serviço foram executadas.

Nesta execução a sobrecarga devido à utilização da ARAXA é de 4,8%. Bem menor se comparada com a execução da consulta de mesmo tipo sobre o documento estruturado. Contudo, isso se explica devido à variação no tempo de reconstrução do resultado, ou seja, o mapeamento das tuplas de resultado para um resultado XML. A coleção de elementos *book* retornada na consulta sobre o documento pouco estruturado é bem menor do que no documento fortemente estruturado, e o tempo de mapeamento deste resultado é diretamente proporcional ao tamanho da coleção retornada.

A consulta Q4, com critério de filtro, também foi submetida a este documento e na Figura 34 é apresentado o gráfico das etapas de seu processamento. Podemos verificar que a sobrecarga nesta execução é de 11,3%.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

6- Execução da consulta SQL

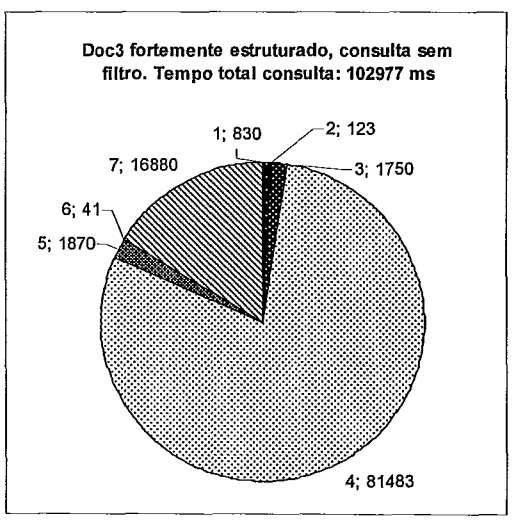
7-Tradução do resultado relacional para formato XML

Figura 34 - Processamento de consulta com filtro sobre o documento 2 pouco estruturado

Podemos verificar que, mesmo com variações de tamanho e estrutura dos documentos, o tempo total de execução de consultas com critérios de filtro não variou em mais de 1 segundo. Podemos perceber então que a aplicação apresenta uma variação de desempenho bem comportada, principalmente ao se tratar da variação do tamanho dos documentos, que entre os documentos 1 e os documentos 2 é de duas ordens de grandeza.

6.3.5 Avaliação 5 - documento 3 fortemente estruturado

A classe de documentos 3 é a que possui maior tamanho em disco, 100 Mb. Na Figura 35 temos o gráfico referente ao tempo de processamento da consulta Q1(sem critério de filtro). Estão presentes neste documento 140 chamadas de serviços, sendo que 70 destas são relevantes para construção do resultado da consulta.



Legenda:

X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:

1-Tradução da consulta XPath para SQL

2-Identificação das chamadas de serviços a serem executadas

3-Parametrização das chamadas de serviço

4-Execução dos serviços nos sites remotos

5-Materialização

6- Execução da consulta SQL

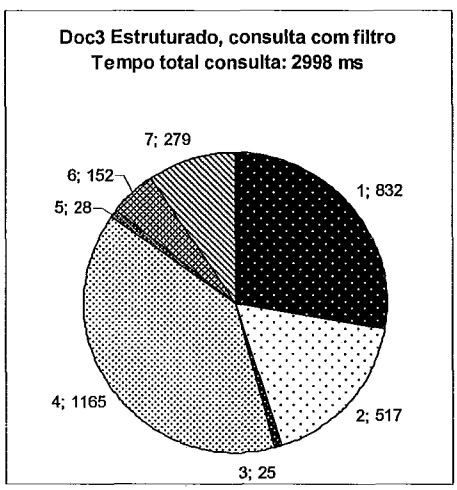
7-Tradução do resultado relacional para formato XML

Figura 35 - Processamento de consulta sem filtro sobre o documento 3 pouco estruturado

A sobrecarga nesse caso é de 18,3%. Destacamos novamente que grande parte do tempo é despendido na execução das chamadas de serviço, na etapa 4, devido ao grande numero de chamadas de serviço executadas.

Na Figura 36 apresentamos a execução da consulta Q3, com critério de filtro. Nesta execução podemos notar a sobrecarga dos mapeamentos foi de 19%.

Podemos verificar que houve um aumento expressivo no tempo gasto na etapa 1, que representa a tradução de consulta XPath para SQL. Como durante este processo de tradução ocorre algumas consultas sobre o documento, por exemplo, para definir a profundidade máxima do documento consultado e definir critérios de ordenação dos elementos do resultado, esse processo sofre impacto direto com o volume de dados a ser consultado, ou seja, com o tamanho do documento.



Legenda:
X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:
1-Tradução da consulta XPath para SQL
2-Identificação das chamadas de serviços a serem executadas
3-Parametrização das chamadas de serviço
4-Execução dos serviços nos sites remotos
5-Materialização
6- Execução da consulta SQL
7-Tradução do resultado relacional para formato XML

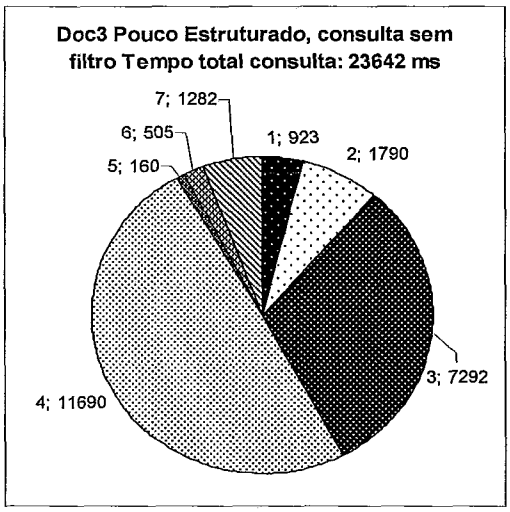
Figura 36 - Processamento de consulta com filtro sobre o documento 3 fortemente estruturado

Porém, mesmo com este tempo adicional, o tempo total da consulta não ultrapassa mais de 1,5 segundos em relação ao processamento da mesma consulta sobre os documentos bem menores, como no caso das execuções nos documentos da classe 1. De certo modo, isto confirma o bom comportamento do sistema no contexto global das execuções de consultas. Vale ressaltar, no entanto, que esse impacto seria muito maior em uma abordagem em que o gerenciamento dos documentos é feito pelas aplicações. O tempo de carga do documento em memória e o consumo de memória sofreriam uma alteração muito mais significativa do que a vista em nossa abordagem.

Podemos ver nesse caso uma das vantagens da ARAXA, que constitui no fato de explorar processadores de consulta altamente desenvolvidos em SGBDORs.

6.3.6 Avaliação 6 - documento 3 pouco estruturado

Avaliamos nesta seção o documento 3 pouco estruturado. Na Figura 37 apresentamos o gráfico do processamento da consulta Q2, que não possui critérios de filtro. Lembramos que este documento possui 140 chamadas de serviço, sendo que possui somente 10 elementos *book*, ou seja, somente 10 chamadas de serviço serão disparadas por essa consulta. Podemos verificar que a sobrecarga dos mapeamentos nesta execução é de 10%.



Legenda:
X ;Y onde X = etapa e Y = tempo gasto em milissegundos

Etapas:
1-Tradução da consulta XPath para SQL
2-Identificação das chamadas de serviços a serem executadas
3-Parametrização das chamadas de serviço
4-Execução dos serviços nos sites remotos
5-Materialização
6- Execução da consulta SQL
7-Tradução do resultado relacional para formato XML

Figura 37 - Processamento de consulta com filtro sobre o documento 3 pouco estruturado

Na Figura 38 temos o gráfico do processamento da consulta Q4, com critério de filtro. A sobrecarga nesta execução foi de 27,9%. Neste caso podemos observar que o percentual de sobrecarga é o mais alto. Isto se deve em grande parte à etapa 1, a tradução de consulta XPath para SQL, pelos mesmos motivos mencionados na seção anterior.

Contudo, o tempo do processamento total da consulta ainda permanece não discrepante, visto que seu valor total fica pouco acima de 2 segundos.

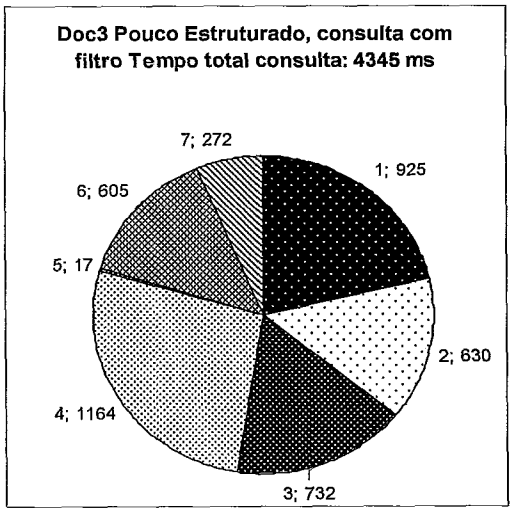


Figura 38 - Processamento de consulta sem filtro sobre o documento 3 pouco estruturado

6.3.7 Avaliação do impacto da ARAXA sobre o SGBD hospedeiro

Além da avaliação referente ao processamento de consultas em diferentes cenários, ou seja, com diferentes configurações de documentos AXML, foi necessário também avaliar o impacto da ARAXA sobre o desempenho e funcionamento do SGBD hospedeiro. Principalmente no que se refere ao Agente Monitor, uma vez que este processo se torna um processo residente no SGBD.

Para realizar a verificação desse impacto, foi realizado o monitoramento do histórico de uso de memória por parte do SGBD durante um intervalo de 240 segundos. No instante 90 desse intervalo foi iniciado o Agente Monitor. O gráfico referente a esta aferição pode ser visto na Figura 39.

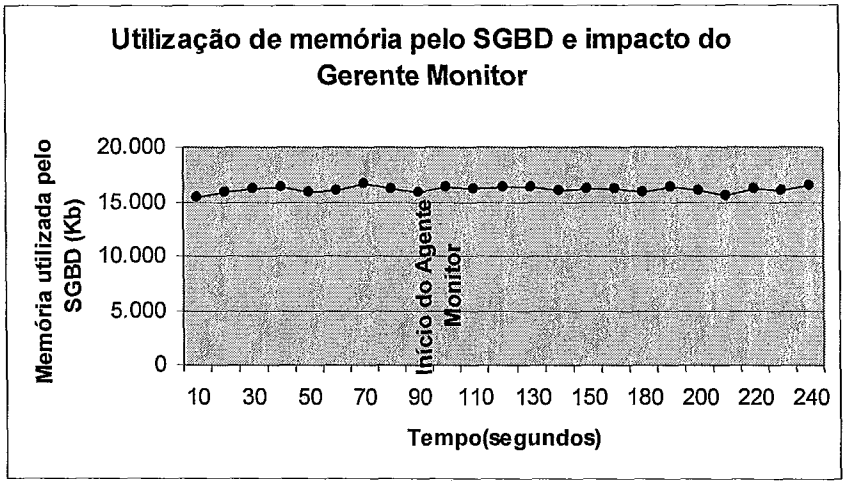


Figura 39 – Impacto do Agente Monitor no SGBD hospedeiro (como critério o uso de memória)

Podemos verificar que a inicialização do Agente Monitor não representou nenhuma alteração sensível no consumo de memória do SGBD hospedeiro. Da mesma

forma foi verificado o tempo de CPU destinado ao SGBD e não foram verificadas alterações perceptíveis. Deste modo, o acoplamento da ARAXA ao SGBD hospedeiro não representa nenhum comprometimento ao bom funcionamento do mesmo.

6.4 Análise dos resultados

Os valores apresentados nas avaliações das seções anteriores representam a média dos valores de um conjunto de 10 execuções para cada avaliação. Este conjunto de valores pode ser visto na Tabela 2. Como o objetivo da avaliação experimental foi a validação dos conceitos e da metodologia apresentada na abordagem ARAXA, principalmente no que se refe à gerência das propriedades ativas, não houve a preocupação com a realização de um número maior de repetição para cada execução.

Um maior número de repetições, neste caso, não implica em uma melhor validação das características que eram foco de avaliação. Isso ocorre principalmente porque lidamos com variáveis externas ao ambiente do SGBD, como a dependência de fatores como a disponibilidade de enlace de rede e o tempo de resposta do provedor de serviços remotos.

Para esta avaliação experimental, também não focamos na execução da consulta SQL, já que avaliações referentes a execuções de consulta envolvendo mapeamento XML – Relacional já foram discutidas pela literatura (KHAN e RAO, 2001).

Tabela 2 – Valores (em MS) em cada rodada de execução das avaliações

Avaliação	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9	Exec 10
Doc1 estrut.LPQ	7801	7802	7809	7799	7779	7811	7801	7798	7800	7810
Doc1 estrut. Filtros	1641	1643	1652	1649	1666	1649	1638	1642	1649	1661
Doc1 pouco estrut. LPQ	1795	1809	1799	1812	1798	1786	1797	1792	1813	1809
Doc1 pouco estrut. Filtros	1789	1781	1794	1770	1781	1780	1769	1789	1775	1782
Doc2 estrut. LPQ	49506	49492	49517	49530	49528	49481	49525	49523	49482	49516
Doc2 estrut. Filtros	1996	1999	2018	2020	1985	2014	2018	2004	2007	2009
Doc2 pouco estrut. LPQ	18590	18621	18608	18611	18571	18589	18571	18605	18569	18595
Doc2 pouco estrut. Filtros	2444	2439	2420	2448	2426	2422	2441	2439	2446	2425
Doc3 estrut. LPQ	102977	102903	102971	103001	102979	102984	103002	102991	102975	102987
Doc3 estrut. Filtros	2988	2989	3009	3011	2984	3007	3004	2987	3011	2990
Doc3 pouco estrut. LPQ	23642	23629	23651	23617	23622	23624	23661	23654	23701	23619
Doc3 pouco estrut. Filtros	4344	4359	4348	4346	4352	4354	4351	4323	4341	4332

A partir dos valores obtidos em cada avaliação foram geradas as informações de média, mediana, desvio padrão e valores máximo e mínimo para cada avaliação. Estes valores são apresentados na Tabela 3. Para fins desta avaliação decidimos utilizar o valor da média devido ao fato de haver uma variação pequena nos valores apresentados

para cada execução. Esta variação pode ser verificada pelos valores de desvio padrão de cada avaliação.

Tabela 3 - Média, mediana, desvio padrão, máximo e mínimo referentes às execuções das avaliações.

Avallacão	média	mediana	desv. padrão	max	min
Doc1 estrut.LPQ	7801	7801	9,0921	7811	7779
Doc1 estrut. Filtros	1649	1649	8,8944	1666	1638
Doc1 pouco estrut. LPQ	1801	1798,5	9,2135	1813	1786
Doc1 pouco estrut. Filtros	1781	1781	8,1650	1794	1769
Doc2 estrut. LPQ	49510	49516,5	18,7617	49530	49481
Doc2 estrut. Filtros	2007	2008	11,2645	2020	1985
Doc2 pouco estrut. LPQ	18593	18592,5	18,4089	18621	18569
Doc2 pouco estrut. Filtros	2435	2439	10,6145	2448	2420
Doc3 estrut. LPQ	102977	102981,5	28,0198	103002	102903
Doc3 estrut. Filtros	2998	2997	11,2448	3011	2984
Doc3 pouco estrut. LPQ	23642	23635,5	26,0640	23701	23617
Doc3 pouco estrut. Filtros	4345	4347	10,7600	4359	4323

A Tabela 4 apresenta a sumarização de todos os percentuais de sobrecarga dos mapeamentos da abordagem ARAXA em todas as avaliações realizadas. Na tabela, o X representa a não aplicação da consulta sobre o documento. Podemos notar que apesar de sofrer variações, a sobrecarga não atingiu valores que comprometessem nossa abordagem, principalmente quando se considera o tempo total de processamento da consulta.

Tabela 4 - Sumarização das sobrecargas da utilização da ARAXA nos cenários avaliados

Documentos avaliados

Consultas executadas / estratégia utilizada	Doc1 fort. estruturado	Doc1 pouco estruturado	Doc2 fort. estruturado	Doc2 pouco estruturado	Doc3 fort. estruturado	Doc3 pouco estruturado
Q1/LPQ	10.1%	X	8.3%	X	19%	X
Q2/LPQ	X	15%	X	4.8%	X	10%
Q3/Filtros	16.7%	X	23.4 %	X	38%	X
Q4/Filtros	X	15.3%	X	11.3%	X	27.9%
Q5/Filtros	X	4.5%	X	X	X	X

Os maiores valores de sobrecarga são referentes a consultas que possuem critérios de filtro, onde a estratégia de filtros é aplicável. Isto ocorre porque a partir da utilização de critérios de filtro, pode-se chegar a um conjunto menor de chamadas de serviço a serem disparadas (geralmente, a execução dos serviços nos sítios remotos é o processo mais caro em tempo). Isso ocorre principalmente nos cenários abordados onde foi possível reduzir o conjunto a uma única chamada de serviço.

Podemos verificar que nos cenários definidos, o provedor de serviços comporta-se de forma regular, ou seja, responde a requisição em tempos semelhantes. Isto se deve ao fato de que o provedor de serviços está servindo apenas a requisições feitas pelo SGBD hospedeiro da ARAXA. Dessa forma não há concorrência pela execução dos serviços, o que seria normal em uma ambiente comercial, o que aumentaria ainda mais o tempo de execução dos serviços nos sites remotos, e consequentemente, diminuiria a sobrecarga da nossa abordagem.

Outro ponto que podemos observar é que, apesar do SGBD se conectar ao provedor remoto de serviços através de um enlace Internet, isto não traz limitação de largura de banda. Isso ocorre porque o volume de dados trafegado é pequeno na grande parte das chamadas de serviços executadas e não há nenhum gargalo no enlace entre o provedor de serviços e o SGBD. Essa nuance pode ser identificada na execução da consulta Q5, onde um volume de dados maior é retornado pela chamada de serviço. Basta comparar a sobrecarga da execução da consulta Q3 (15.3%) com a da consulta Q5 (4.5%), ambas executadas sobre o mesmo documento. Dessa forma, consultas com comportamentos semelhantes, que basicamente se diferem em qual chamada de serviço é executada, têm taxas de sobrecarga bem diferenciadas, devido justamente às características dessas chamadas de serviço.

Cenários onde requisições de diferentes origens concorrem pelo provedor de serviços, além de múltiplas variações nas propriedades do enlace entre o SGBD e o provedor remoto de serviços, são mais próximos dos ambientes organizacionais. Dessa forma, a sobrecarga da ARAXA tende a ser menor nesses cenários. Possivelmente, a sobrecarga pode se tornar até mesmo desprezível, tendo em vista a realização de chamadas a serviços mais complexos, implicando em um custo maior de processamento de mensagens SOAP. Esta sobrecarga de empacotamento, desempacotamento e processamento de mensagens é inerente ao protocolo SOAP. Em Ruberg et al. (2004) foi realizada uma analise sobre os custos envolvendo a ativação de chamadas de

serviços contidas em um documento AXML, onde verificou-se que a utilização do protocolo SOAP representa uma sobrecarga de desempenho significativa. Dessa forma o percentual da sobrecarga dos mapeamentos empregados na ARAXA tende a decrescer.

Observamos também que apesar de haver uma grande variação no tamanho dos documentos avaliados (4 ordens de grandeza), a ARAXA como um todo demonstrou-se bem comportada, ou seja, não houve nenhum resultado suficientemente discrepante a ponto de invalidar a abordagem.

Outro resultado importante é referente ao impacto da ARAXA no SGBD hospedeiro. Através desta avaliação podemos verificar que o emprego da ARAXA não traz nenhum prejuíso ao desempenho do SGBD utilizado.

6.5 Considerações finais

Neste capítulo apresentamos um estudo experimental definido para avaliar a abordagem ARAXA. Para a execução deste estudo foi utilizado o protótipo implementado no contexto desta dissertação.

Foi possível avaliar todas as etapas definidas na abordagem ARAXA, sendo observadas características referentes ao impacto e sobrecarga da utilização da ARAXA. A partir deste estudo experimental foi possível constatar a viabilidade da mesma, ou seja, foi possível aplicar a prova dos conceitos e metodologia envolvida na ARAXA.

Foi possível também atestar a aplicabilidade da ARAXA, uma vez que foram observados diferentes cenários com variações de documentos AXML e consultas aplicadas sobre eles. A ARAXA comportou-se de maneira satisfatória em relação à sobrecarga imposta devido a sua adoção, principalmente demonstrando bons resultados em um ambiente menos favorável a sua utilização. Contudo, ao se pensar em um cenário organizacional e suas propriedades, este demonstra-se mais favorável a utilização da ARAXA, onde, possivelmente, os resultados serão ainda melhores.

Podemos constatar também que a ARAXA demonstrou um bom comportamento, não sofrendo alterações bruscas em seu desempenho, mesmo quando houve grandes alterações nos documentos avaliados.

No próximo capitulo teremos algumas conclusões acerca do trabalho realizado nesta dissertação bem como a definição de pontos que merecem mais algum nível de aprofundamento, passíveis de serem explorados em trabalhos futuros.

Capítulo 7 Conclusão

A busca por soluções que possibilitem tanto a interoperação entre aplicações de uma forma simples quanto o intercâmbio de informações de maneira padronizada fez com que surgissem tecnologias para publicação e acesso a informações e serviços na Web, tais como XML e Serviços Web. O sucesso dessas tecnologias proporcionou o desenvolvimento de uma nova classe de documentos XML, os documentos AXML.

A gerência e o armazenamento de documentos AXML apresentam particularidades referentes ao aspecto ativo desta classe de documentos XML. As invocações das chamadas de serviços contidas nos documentos AXML devem ser coordenadas em tempo de processamento de consulta. Uma outra característica dos documentos AXML é que chamadas de serviços podem ser disparadas totalmente desassociadas do processamento de consultas, ou seja, chamadas de serviços podem obedecer a uma periodicidade de execução.

A existência das chamadas de serviço reflete então em uma complexidade adicional ao problema de armazenamento e gerência de documentos XML. Atualmente, não é possível gerenciar documentos AXML diretamente por ferramentas convencionais de gerenciamento de dados XML, como os SGBDs XML Nativos.

A detecção das chamadas de serviço contidas em um documento AXML, suas ativações e a gerência dos resultados obtidos através dessas chamadas são tarefas que devem ocorrer de forma transparente ao usuário. Essas particularidades devem ser levadas em consideração para um sistema efetivo de armazenamento de documentos AXML.

Esta dissertação propôs uma abordagem para o armazenamento e a gerência de documentos XML Ativos, a ARAXA. Analisamos as características desse novo modelo de dados e construímos uma solução que tem como base o uso de SGBD Objeto-Relacionais. Os SGBDOR, da mesma forma que SGBDR e XML Nativos, não fornecem mecanismos explícitos para gerência das propriedades dinâmicas dos documentos AXML. Contudo, conseguimos visualizar na vinculação de funções e procedimentos em SQL a métodos de objetos definidos pelo usuário, uma alternativa para que possamos incorporar os comportamentos necessários para a realização da gerência dos elementos ativos.

Além disso, a utilização de SGBDOR se mostra interessante, pois podemos oferecer um cenário onde todas as estruturas de dados utilizadas em uma organização, sejam eles bem estruturados ou não, possam ser persistidas, dada uma função de mapeamento, para o paradigma Objeto-Relacional. Este repositório único apresenta-se como uma alternativa adequada para os ambientes empresariais, não representando custos adicionais e não requerendo mudanças bruscas na gerência de armazenamento de dados, na qual estão atrelados o conhecimento e o histórico organizacional.

A ARAXA representa uma alternativa interessante para o armazenamento e gerência de documentos XML Ativos, na medida em que faz uso de ferramentas poderosas tanto de armazenamento quanto de processamento de consultas, extremamente evoluídas em SGBDORs e, por outro lado, mantém essa utilização encapsulada, não impondo ao usuário final nenhuma tarefa adicional devido ao fato de estar fazendo uso da ARAXA. Todos os processos de mapeamento, incluindo o armazenamento dos documentos, o processo de tradução de consulta XPath para SQL o processo de reconstrução do resultado relacional para um resultado XML acontecem sem participação do usuário, ficando totalmente transparente ao usuário final a utilização de um SGBDOR para as tarefas de armazenamento e consulta dos documentos AXML.

O ponto de partida da ARAXA constituiu a definição do esquema de mapeamento XML-R. Para definir o esquema de mapeamento de documentos AXML para o contexto relacional exploramos as principais estratégias de mapeamento XML para o modelo relacional encontradas na literatura. Contudo, não foi encontrada nenhuma abordagem de mapeamento que tivesse algum tipo de preocupação com a adaptação de propriedades dinâmicas a documentos XML. Apesar de os esquemas de mapeamento analisados não possuírem nenhuma estratégia associada ao armazenamento de documentos XML ativos, estes esquemas de mapeamento fornecem uma base inicial para tal tarefa. Sendo assim, escolhemos dentre os esquemas de mapeamento pesquisados, o esquema Edge com ordenação através da codificação Dewey (TATARINOV *et al.*, 2002a). Foram necessárias duas extensões a este esquema de mapeamento, uma para que ele suportasse o armazenamento de múltiplos documentos e outra para que fosse possível a diferenciação entre elementos e atributos.

Apresentamos uma arquitetura que abrange e satisfaz a abordagem ARAXA. Propomos também uma metodologia para processamento de consultas sobre documentos AXML em SGBDORs, onde combinamos características do processamento

de consultas sobre documentos AXML com os mapeamentos necessários entre o paradigma XML-OR.

Mostramos que essa arquitetura não se restringe a um único SGBD, mas a toda classe de SGBDs Objeto-Relacional, além de não haver grandes desafios tecnológicos para a sua adoção, já que maioria dos cenários organizacionais já fazem uso de SGBDOR para gerência de dados. Portanto, nossa arquitetura apresenta uma solução não intrusiva para o problema abordado no que se refere ao impacto tecnológico de sua adoção. Todavia, a ARAXA não só fornece serviços de armazenamento e gerência aos documentos AXML, mas também representa uma alternativa para armazenamento e consulta para documentos XML tradicionais.

Mostramos também a viabilidade de nossa arquitetura através da implementação de um protótipo. Em nosso protótipo utilizamos tecnologias e ferramentas "livres". No protótipo foram contemplados os requisitos básicos de gerência de documentos AXML, como uma metodologia para execução de chamadas de serviço em tempo de consulta e o tratamento de chamadas de serviços com ativação baseada em uma periodicidade.

O protótipo apresenta uma interface gráfica que, apesar de trivial, fornece um mecanismo simples de interação com o usuário. Através desta interface o usuário pode instanciar o esquema de dados utilizado para o armazenamento dos documentos AXML na ARAXA, armazenar e recuperar documentos XML e XML Ativos, executar consultas sobre os documentos armazenados e iniciar o Agente Monitor .

A partir da construção do protótipo foi possível avaliar a arquitetura definida através de um estudo experimental. Este estudo teve como objetivo validar os conceitos e a metodologia apresentada na abordagem ARAXA, além de verificar a viabilidade e aplicabilidade da nossa abordagem. Através deste estudo procuramos analisar o acoplamento da gerência de dados XML Ativos a SGBDs Objeto-Relacionais. Procuramos também analisar o impacto dos mapeamentos entre os paradigmas XML e Objeto-Relacional no contexto de processamento de consultas e materialização de documentos AXML.

Foram observadas características referentes ao impacto e sobrecarga da utilização da nossa abordagem, a partir das quais podemos constatar também a viabilidade da mesma, ou seja, foi possível aplicar a prova dos conceitos e metodologia envolvida na ARAXA. Outro ponto observado é que a ARAXA mostrou um bom comportamento, não sofrendo alterações bruscas em seu desempenho, mesmo quando houve grandes alterações nos documentos avaliados.

7.1 - Trabalhos futuros

Como trabalhos futuros, podemos ressaltar a necessidade de evolução do componente de tradução de consultas definido em nossa arquitetura. Hoje ele está preparado apenas para consultas XPath simples, com opção de filtros sobre elementos e atributos. Da mesma forma o componente responsável pela tradução do resultado relacional (em tuplas) para o resultado XML também precisa ser aprimorado.

Outro ponto de nosso protótipo que necessita de uma maior atenção é a construção de planos de materialização e otimização desses planos. Apesar de essa questão ter sido abordada em nossa arquitetura, em nosso protótipo não foi implementado nenhum processo de otimização nos planos de materialização. Apenas nos certificamos que o plano de execução utilizado não infringisse nenhuma das restrições existentes entre as chamadas de serviço a serem executadas. Isto se deve ao fato de não estarmos inicialmente preocupados com a obtenção de resultados ótimos, mas sim com a validade e aplicabilidade de nossa solução. Contudo, a otimização desse processo, bem como a dos demais componentes do protótipo pode ser realizada, agregando mais valor ainda a ARAXA.

O processo de avaliação seletiva de chamadas de serviço a partir da consulta submetida também é um ponto extremante interessante, e de impacto direto em toda abordagem, e sua evolução é algo altamente relevante. Em nosso protótipo empregamos algumas técnicas como a LPQ, critérios de filtro e catálogo de chamadas de serviços. Contudo, outros algoritmos podem ser aplicados e novas heurísticas podem ser desenvolvidas.

Um ponto que não foi explorado nessa dissertação, mas que de certa forma surge como produto da ARAXA é a possibilidade dada aos SGBDOR de realizar chamadas de serviços, uma vez que na ARAXA foi implementado um cliente genérico para serviços Web. Novas arquiteturas que explorassem essa nova oportunidade de integração tanto de dados quanto aplicações representam um trabalho futuro interessante.

Um outro trabalho futuro a esta dissertação, mas que já encontra-se em andamento, surgiu devido ao interesse por parte do Grupo Gemo. Constitui na construção de uma API, que será disponibilizada junto com a plataforma ActiveXML, utilizando a ARAXA como alternativa de armazenamento dos documentos XML Ativos.

Referências Bibliográficas

ABITEBOUL, S., BENJELLOUN, O., CAUTIS, B., *et al.*, 2004a, "Lazy Query Evaluation for Active XML". In: *SIGMOD*, Paris, França.

ABITEBOUL, S., BENJELLOUN, O., MILO, T., 2003, *Active XML Primer* Relatório Técnico 275, INRIA Futurs.

ABITEBOUL, S., BENJELLOUN, O., MILO, T., 2004b, "Positive Active XML". In: *PODS* pp. 35-45, France.

ABITEBOUL, S., BENJELLOUN, O., MILO, T., 2005, *The Active XML project: an overview*,Relatório Técnico 331, GEMO INRIA.

ABITEBOUL, S., COBENA, G., NGUYEN, B., *et al.*, 2002, "Construction and Maintenance of a Set Of Pages of Interest (SPIN) using ActiveXML". In: *Conference on Bases de Donnees Avancees*, Montpellier, França.

ACTIVEXML, "ACTIVE XML WEBSITE". Disponível em: http://activexml.net/, Acessado em 03/11/2007.

ALMEIDA, M.S., CONDON, K., FISCHER, M., *et al.*, 2000, *DB2 Java Stored Procedures*,Relatório Técnico SG24-5945-00, IBM International Technical Support Organization.

ALTOVA, 2007, "XMLSpy". Disponível em: http://www.altova.com/products/xmlspy/xml_editor.html, Acessado em 02/12/2007.

BARBOSA, D., MENDELZON, A.O., KEENLEYSIDE, J., *et al.*, 2002, "ToXgene: An extensible template-based data generator for XML". In: *SIGMOD*, Madison, Wisconsin, EUA.

BENEDIKT, M., CHAN, C.Y., FAN, W., *et al.*, 2002, "DTD-Directed Publishing with Attribute Translation Grammars". In: *VLDB*, pp. 838--849, Hong Kong, China.

BENJELLOUN, O., 2004, *Active XML : A data-centric perspective on Web Services*, Tese de Doutorado. I'Université Paris XI, IÚniversité Paris XI (10/2004).

BOHANNON, P., FREIRE, J., ROY, P., *et al.*, 2002, "From XML Schema to Relations: A Cost-Based Approach to XML Storage". In: *Conference on Data Engineering, ICDE*, pp. 64--75, San Jose, California, USA.

CANAUD, E., BENBERNOU, S., HACID, M., 2004, "Managing trust in Active XML, Services Computing". In: *IEEE International Conference on Publication*, pp. 41 - 48, 09/2004.

DEHAAN, D., TOMAN, D., CONSENS, M., *et al.*, 2003, "A Comprehensive XQuery to SQL Translation Using Dynamic Interval Encoding". In: *SIGMOD*, pp. 623-634, San Diego, California, USA.

DEUTSCH, A., FERNANDEZ, M., SUCIU, D., 1999, "Storing Semistructured Data with STORED". In: *International Conference on Management of Data, SIGMOD*, pp. 431--442, Philadelphia, Pennsylvania, USA.

DEUTSCH, A., TANNEN, V., 2003, "MARS: A System for Publishing XML from Mixed and Redundant Storage". In: *International Conference on Very Large Databases, VLDB*, pp. 201--212, Berlin, Germany.

DITTRICH, K., GATZIU, S., GEPPERT, A., 1995, "The Active Database Management System Manifesto: A Rulebase of ADBMS Features". In: *2nd International Workshop on Rules in Database Systems*, v. 985, pp. 3–20, Atenas , Grécia.

FERNANDEZ, M., MORISHIMA, A., SUCIU, D., 2001, "Efficient Evaluation of XML Middle-ware Queries". In: *International Conference on Management of Data, SIGMOD*, pp. 103-114, Santa Barbara, California, USA.

FERRAZ, C., BRAGANHOLO, V., MATTOSO, M., 2006, "Uma Abordagem para o Armazenamento de Documentos XML Ativos". In: *WTDBD*, pp. 38-42, Florianópolis, SC, Brasil, 20/10/2006.

FERRAZ, C., BRAGANHOLO, V., MATTOSO, M., 2007a, *ARAXA: an object-relational approach to store active XML documents*,Relatório Técnico ES-708/07, COPPE/UFRJ.

FERRAZ, C., BRAGANHOLO, V., MATTOSO, M., 2007b, "Storing AXML documents with ARAXA". In: *Simpósio Brasileiro de Banco de Dados (SBBD)*, João Pessoa, Paraíba, 10/2007.

FLORESCU, D., KOSSMAN, D., 1999, "Storing and Querying XML Data using an RDBMS", *IEEE Data Engineering Bulletin*, v. 22, pp. 27-34.

IBM, 2006, "DB2 XML Extender". Disponível em: http://www-306.ibm.com/software/data/db2/extenders/xmlext/index.html, Acessado em 09/09/2006.

KHAN, L., RAO, Y., 2001, "A Performance Evaluation of Storing XML Data in Relational Database Management Systems". In: *WIDM*, pp. 31 - 38 Atlanta, Georgia, USA.

KOSSMANN, D.F.D., 1999, "Storing and Querying XML Data using an RDBMS", *IEEE Data Engineering Bulletin*, v. 22, pp. 27-34.

KRISHNAMURTHY, R., KAUSHIK, R., NAUGHTON, J.F., 2003, "XML-to-SQL Qery Translation Literature: The State of the Art and Open Problemsu". In: *XSym*, pp. 1--18, Germany, 2003.

LEONARD, J.L., 2006, *Strategies for Encoding XML Documents in Relational Databases: Comparisons and Contrasts*, Department of Computer and Information Sciences, Dissertação de Mestrado, East Tennessee State University.

LI, C., BOHANNON, P., KORTH, H.F., *et al.*, 2003, "Composing XSL Transformations with XML Publishing Views". In: *International Conference on Management of data, SIGMOD*, pp. 515--526, San Diego, California, USA.

LU, S., SUN, Y., ATAY, M., *et al.*, 2003, "A New Inlining Algorithm for Mapping XML DTDs to Relational Schemas". In: *International Workshop on XML Schema and Data Management*, Chicago, Illinois, USA, 10/2003.

MANOLESCU, I., FLORESCU, D., KOSSMANN, D., 2001, "Answering XML Queries Over Heterogeneous Data Sources". In: *nternational Conference on Very Large Databases. VLDB*, pp. 241-250, Roma, Italy.

MANOLESCU, I., KOSSMANN, D., XHUMARI, F., *et al.*, 2000, "Agora: Living with XML and Relational", *The VLDB Journal*, pp. 623-626.

MARINOIU, B.-E., 2004, *Development environment for dynamical Web data*,Relatório Técnico X2001, INRIA Futurs, Parc Club Universitaire Orsay.

MEDEIROS, H., TAOUK, V., 2007, "Tradução de Consultas XPath para SQL(título provisório)", *Departamento de Ciência da Computação/UFRJ (Projeto de Conclusão de Curso de Graduação, em andamento)*, Rio de Janeiro, RJ , Brasil.

NAMBIAR, U.L., BRESSAN, Z., LI, S., 2002, "Current approaches to XML management", *Internet Computing, IEEE*, v. 6, n. 4, pp. 43-51.

NAUGHTON, R.K.R.K.J.F., 2003, "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems". In: *1st Int'l XML Database Symposium, XSym*, pp. 1 - 18, Berlin, Germany.

OCLC, "Introduction to the Dewey Decimal Classification". Disponível em: www.oclc.org/dewey/versions/abridgededition14/intro.pdf Acessado em 01/03/2006.

ORACLE, 2007, "Oracle 9i". Disponível em: http://www.oracle.com/technology/sample_code/tech/java/jsp/oracle9ijsp.html, Acessado em 01/04/2007.

PARABOSCHI, A.B.S.C.S., 2001, "Active rules for XML: A new paradigm for E-services", *The VLDB Journal — The International Journal on Very Large Data Bases*, v. 10 (Agosto, 2001), pp. 39 - 47

PEREIRA, D., RUBERG, G., MATTOSO, M., 2006, "Geração Eficiente de Planos de Materialização para Documentos XML Ativos". In: *Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 236-250, Brasília, DF, Brasil, Outubro.

PLJAVA, 2007, "The pljava Project ". Disponível em: http://gborg.postgresql.org/project/pljava/projdisplay.php, Acessado em 01/01/2007.

PLOUGMAN, D., LAURITSEN, T., OLESEN, J., "Shredding and Querying XML Data Using an RDBMS". Disponível em:

http://www.cs.aau.dk/library/files/rapbibfiles1/1085738327.ps, Acessado em 05/03/2006.

POSTGRESQL, 2007, "Postgresql". Disponível em: http://www.postgresql.org/, Acessado em 01/01/2007.

RUBERG, G., MATTOSO, M., 2006, *XCraft: A Dynamic Optimizer for the Materialization of Active XML Documents*,Relatório Técnico ES-709/07, COPPE/UFRJ, Rio de Janeiro.

RUBERG, N., RUBERG, G., MONALESCU, I., 2004, "Towards cost-based optimization for data-intensive Web service computations". In: *Simpósio Brasileiro de Banco de Dados(SBBD)*, pp. 283-297.

SCHMIDT, A., KERSTEN, M., WINDHOUWER, M., *et al.*, 2001, "Efficient Relational Storage and Retrieval of XML Documents". In: *International Workshop on the Web and Databases*, pp. 47--52, Dallas, Texas, USA, May 2000.

SHANMUGASUNDARAM, J., KIERNAN, J., SHEKITA, E., *et al.*, 2001, "Querying XML views of relational data". In: *Conference on Very Large Databases, VLDB*, pp. 261-270, Rome, Italy, September 2001.

SHANMUGASUNDARAM, J., TUFTE, K., ZHANG, C., *et al.*, 1999, "Relational Databases for Querying XML Documents: Limitations and Opportunities". In: *International Conference on Very Large Data Bases, VLDB*, pp. 302-304, Edinburgh, Scotland, UK, September 1999.

TATARINOV, I., VIGLAS, S.D., BEYER, K., *et al.*, 2002a, "Storing and Querying Ordered XML Using a Relational Database System". In: *SIGMOD*, pp. 204-215, Madison, Wisconsin, USA.

TATARINOV, I., VIGLAS, S.D., BEYER, K., *et al.*, 2002b, *Storing and Querying Ordered XML using a Relational DBMS*,Relatório, Tech Report Univ. of Washington, Washington.

VAN SOLINGEN, R., BERGHOUT, N., 1999, *The Goal / Question / Metric Method: A Practical Guide for Quality Improvement of Software Development.*

VIEIRA, H., RUBERG, G., MATTOSO, M., 2003, "XVerter: querying XML data with OR-DBMS". In: *WIDM*, pp. 37-44, New Orleans, USA.

W3C, 2007, "Web Services, W3C Recommendation". Disponível em: http://www.w3.org/2002/ws/, Acessado em 22/11/2007.

W3C, 2007, "XML, W3C Recommendation". Disponível em: http://www.w3.org/XML/, Acessado em 26/07/2007.

W3C, 2007, "XQuery(W3C Recommendation)". Disponível em: http://www.w3.org/XML/Query/, Acessado em 07/07/2007.

YOSHIKAWA, M., AMAGASA, T., SHIMURA, T., *et al.*, 2001, "XRel: a path-based approach to storage and retrieval of XML documents using relational databases". In: *ACM Transactions on Internet Technology ,TOIT*, v. 1, pp. 110--141, New York, NY, USA, Agosto de 2001.

Interface da ARAXA para interação com o usuário

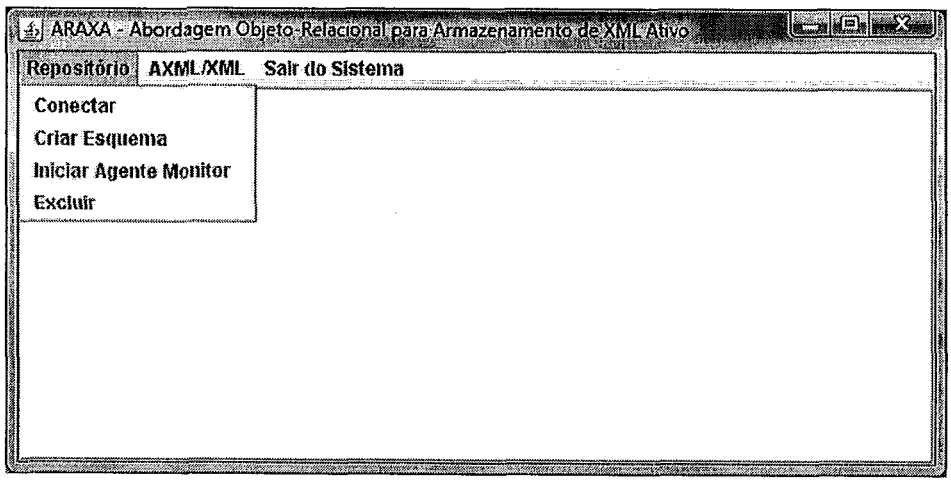


Figura 40 – Menu de conexão com o SGBD e gerência do esquema de dados criado pela ARAXA

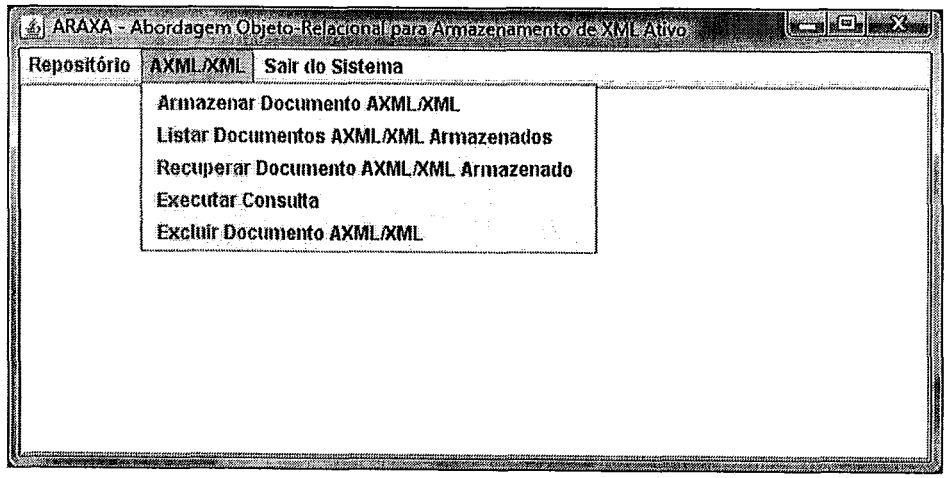


Figura 41 – Menu de armazenamento e consulta de documentos AXML

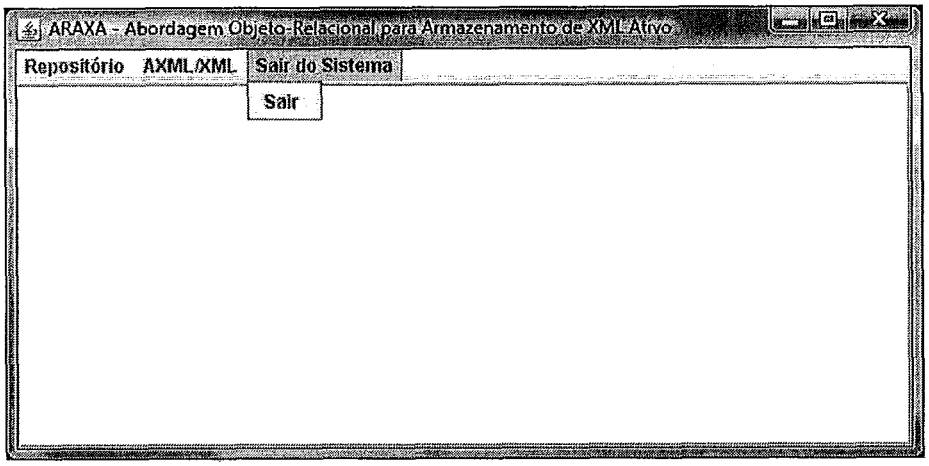


Figura 42 – Menu para encerramento de sessão do usuário

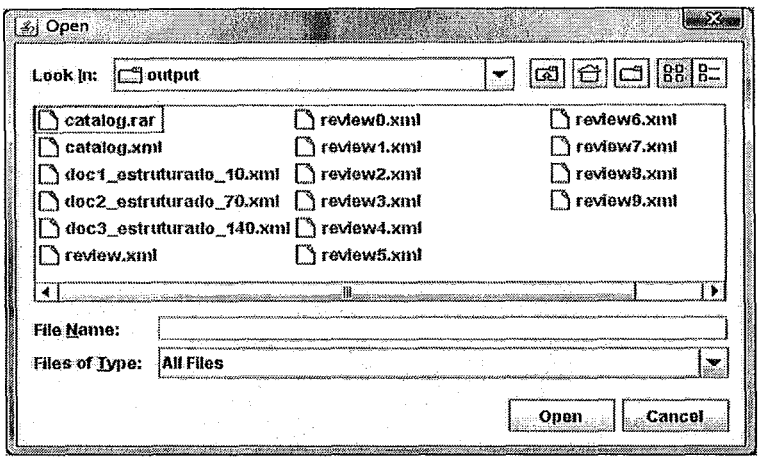


Figura 43 – Tela apresentada ao usuário para seleção do documento a ser armazenado

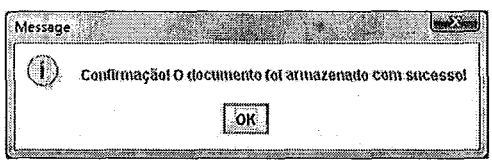


Figura 44 – Tela de confirmação de armazenamento de documento AXML

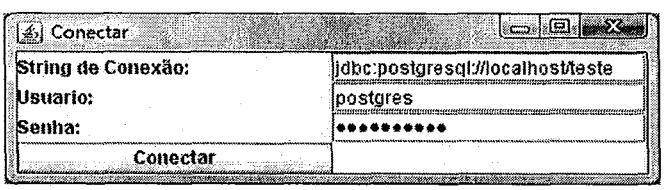


Figura 45 – Tela de definição de Base de Dados a ser utilizada e autenticação de usuário e senha

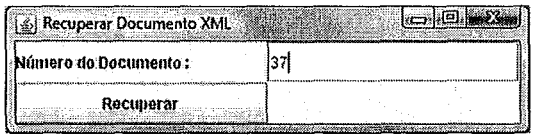


Figura 46 – Tela para recuperação de documento da base de dados

Figura 47 – Tela onde é apresentado o documento XML/AXML recuperado

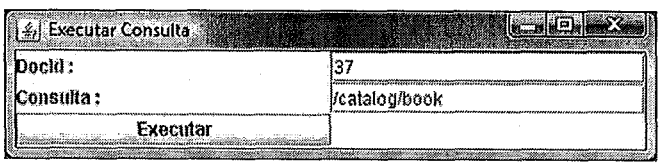


Figura 48 – Tela para realização de consultas sobre determinado documento da base de dados

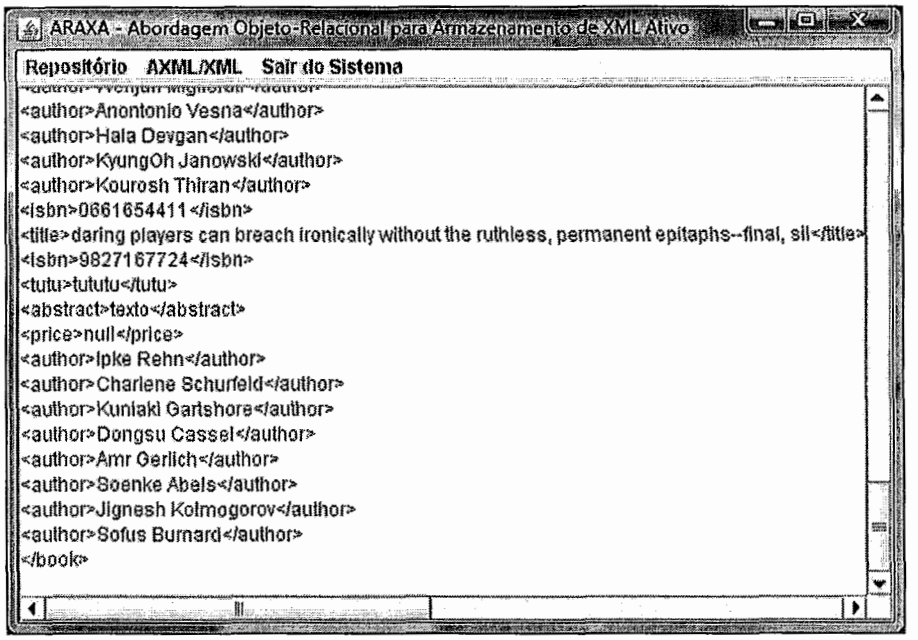


Figura 49 – Tela de resposta da consulta submetida

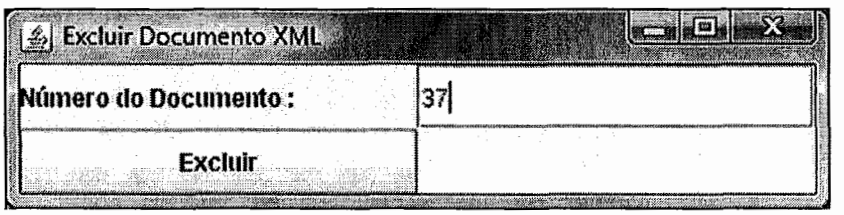


Figura 50 - Tela para exclusão de documento da base de dados

Anexo B

Diagramas de classes da ARAXA