UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

ALEX MARIANO COSTA DE OLIVEIRA

SAVE:
SISTEMA DE ATENDIMENTO AO VESTIBULANDO UTILIZANDO BANCO DE DADOS NATIVO XML

FLORIANÓPOLIS
2003

ALEX MARIANO COSTA DE OLIVEIRA

SAVE:
SISTEMA DE ATENDIMENTO AO VESTIBULANDO UTILIZANDO BANCO DE DADOS NATIVO XML

Monografia apresentada à Universidade Federal de Santa Catarina, como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação, sob Orientação do Prof. Leandro José Komosinski.

FLORIANÓPOLIS
2003

ALEX MARIANO COSTA DE OLIVEIRA

SAVE:
SISTEMA DE ATENDIMENTO AO VESTIBULANDO UTILIZANDO BANCO DE DADOS NATIVO XML

Monografia aprovada em, / /2003, como requisito para obtenção do Grau de Bacharel em Ciências da Computação.

BANCA EXAMINADORA

_____
Professor Leandro José Komosinski
Orientador

_____
Professor José Mazzuco Jr.
Co-orientador

_____
Professor José Eduardo De Lucca
Membro

Aos meus queridos pais, Neubens e Sandra
e aos meus irmãos Aline, Hebert e Nissens.

AGRADECIMENTOS

A Deus, Senhor e Criador dos homens e de todas as coisas em que vivemos, que me concedeu forças para a realização deste trabalho, iluminando o meu caminho e ajudando-me a vencer as barreiras encontradas.

Em especial, ao mestre e orientador Leandro José Komosinski, pela profunda dedicação em cada etapa deste trabalho e também pelos grandes conselhos e idéias.

Aos professores José Mazzuco Júnior e José Eduardo De Lucca pelo interesse e participação nesse projeto.

Aos grandes companheiros e amigos da turma 98.2 Alexandre Honório, Handerson Koerich, Júlio César, Marcelo de Souza e Vinícius Fagundes que de alguma forma contribuíram para a realização deste trabalho.

À família Soares, pela sua generosidade e constante auxílio desde que nos conhecemos.

À grande equipe de desenvolvedores da Apache Software Foundation pelo suporte prestado nos momentos de maior dúvida, principalmente aos senhores Kimbro Staken, Gianugo Rabellino e Vladimir Bossicard.

“*Não podemos dirigir o vento...*

...Mas podemos ajustar as velas”.

SUMÁRIO

LISTA DE ABREVIATURAS

HTML – Hypertext Markup Language (Linguagem de Marcação de Hipertexto)

JDK – Java Development Kit (Kit de Desenvolvimento Java)

JDBC – Java Database Connectivity

JVM – Java Virtual Machine (Máquina Virtual Java)

JSP – Java Server Pages

NXDs – Native XML Databases (Bancos de Dados Nativos XML)

RMI – Remote Method Invocation (Invocação Remota de Métodos)

RPC – Remote Procedure Call (Chamada Remota de Procedimentos)

ODBC – Open Database Connectivity

SGDB – Sistema de Gerenciamento de Banco de Dados

TI - Tecnologia da Informação

XML – Extensible Markup Language

W3C – World Wide Web Consortium

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho descreve a implementação de um sistema de atendimento aos vestibulandos de uma Universidade sob a ótica dos Bancos de Dados Nativos XML, um novo conceito de armazenamento de documentos XML em pleno desenvolvimento. O sistema proposto permite que os vestibulandos possam tirar suas dúvidas em relação às datas e locais das provas do concurso Vestibular.O software – que consiste em uma arquitetura cliente/servidor – gerencia e automatiza esse atendimento, com o objetivo de criar mais um canal de comunicação entre vestibulandos e a instituição. Sua principal característica é a portabilidade, já que a linguagem de programação utilizada para desenvolvê-lo é Java™ e seus dados são armazenados em um banco de dados nativo XML, o tornando independente de plataforma.

Palavras-chave: Atendimento – Banco de Dados – Xindice – XML

ABSTRACT

This work describes the implementation of an online an attendance system to vestibular contest takers of a university, focusing native XML databases, a new storage concept for XML documents still under development. The system allows vestibular contest takers to solve doubts in relation to tests dates can be solved. The software – which consists of a client/server architecture – manages and automatizes this attendance in order to create one more communication channel between vestibular contest takers and the institution. Its main feature is portability, since the programming language used to develop it is Java™ and its data is stored into a native XML database system, making it platform-independent.

Key-words: Attendance – Databases – Xindice – XML

1 INTRODUÇÃO

Nesse capítulo, são apresentados o contexto no qual se insere este trabalho e as justificativas que motivam sua realização, assim como o objetivo proposto, os métodos de pesquisa utilizados e por fim, a forma de organização deste texto.

1.1 CONTEXTO

As instituições e universidades de hoje estão tradicionalmente organizadas em departamentos e comissões responsáveis por atividades específicas. Uma dessas comissões tem o papel de organizar e divulgar o concurso vestibular para a sociedade, especificamente para aqueles que serão avaliados no concurso, os vestibulandos. Um dos objetivos de uma universidade ao oferecer vagas para os estudantes, além de preenchê-las com os melhores deles, é ter o maior número de vestibulandos inscritos no concurso a fim de dar créditos à instituição e ainda arrecadar e investir na infra-estrutura, na pesquisa e no ensino. Com isso, a preocupação com uma boa plataforma de atendimento é evidente. Porém, poucas são as instituições de ensino que oferecem atendimento ao vestibulando através de outro canal de comunicação além da tradicional linha telefônica.

Para que uma central de atendimento de uma universidade tenha bons resultados com pouco capital, toma-se como solução o desenvolvimento de um sistema de atendimento via internet, que é um sistema onde o vestibulando interage com atendentes da instituição através de uma Sala de Chat. Com este tipo de atendimento, a instituição reduz drasticamente seus gastos [ECK00], já que a infra-estrutura necessária é mínima. Além disso, a quantidade de atendentes também se torna menor, visto que vários vestibulandos podem ser atendidos ao mesmo tempo por apenas um operador.

Por trás desta solução, está a XML, uma meta-linguagem criada há poucos anos e que está se tornando um padrão para troca de informações entre sistemas. Neste trabalho, documentos XML serão a fonte dos dados necessários à aplicação. Para tanto, faz-se uso de um novo conceito em armazenamento e recuperação de dados XML, os banco de dados nativos XML ou NXDs.

1.2 MOTIVAÇÃO

XML está rapidamente se tornando o formato de dados escolhido por uma grande parte das soluções existentes na área de tecnologia da informação, emergindo como um padrão para troca de dados na grande rede mundial. De acordo com o IDC[1], estima-se que os gastos em Bancos de Dados Nativos XML ultrapassem 280 milhões de dólares em 2006 representando uma taxa de crescimento anual superior a quarenta por cento.

Fazem parte das aplicações que utilizam XML o envio de documentos em sistemas B2B, a troca de mensagens para integração de sistemas legados, armazenamento de dados e várias outras atividades de manipulação de dados. Os benefícios geralmente associados a essa meta-linguagem são a independência de plataforma, capacidade de extensão e o baixo custo de implantação. Neste trabalho, a aplicação da XML está no armazenamento, manipulação e apresentação dos dados.

Para manter os dados de uma aplicação em um sistema de banco de dados, eles devem ser armazenados e recuperados de forma consistente, confiável e eficiente. Dentre os modelos de banco de dados existentes, o modelo relacional é o dominante e também o mais estudado. Por isso, utilizar essa estrutura parece ser uma boa maneira de organizar e gerenciar dados em formato XML. Porém, o mapeamento de documentos XML em uma estrutura rígida como a dos bancos de dados relacionais não é elementar e pode ainda resultar em esquemas pouco elegantes [LIO02], desperdiçando espaço em disco e trazendo ineficiência nas consultas. Essa dificuldade ocorre porque o modelo relacional criado por E.F Codd é consideravelmente diferente do modelo de dados XML. Os bancos de dados relacionais organizam os dados em estrutura tabular e utilizam ligações relacionais para representar a hierarquia dos dados [XIN02], enquanto a XML é intrinsecamente uma estrutura de árvore hierárquica.

1.3 OBJETIVO

O objetivo principal deste trabalho é avaliar o uso de um banco de dados nativo XML em uma plataforma de atendimento ao vestibulando através da Internet.

1.3.1 Objetivos Específicos

Conhecer diferentes técnicas de armazenamento e recuperação de documentos XML.

Comparar características de um Sistema de Gerenciamento de Banco de Dados Relacional e um Banco de Dados XML Nativo.

Gerar mais um canal de comunicação entre vestibulandos e universidade.

1.4 MÉTODOS DE PESQUISA

A fim de atingir o objetivo proposto por este trabalho, faz-se uso de uma abordagem de pesquisa documental, que consiste no estudo de qualquer base de conhecimento acessível que esteja focada em metodologias de implementação de software, comparação de tecnologias e exemplificações. A maior fonte de pesquisa deste trabalho foram artigos encontrados na Internet.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está estruturado da seguinte maneira:

O capítulo 2 faz uma abordagem da linguagem XML e sua relação com banco de dados, partindo desde seu conceito até o resumo das técnicas de armazenamento de documentos nesse formato. No capítulo 3, é feita uma análise do funcionamento e das características do Xindice, o banco de dados nativo XML escolhido para este trabalho. O quarto capítulo é composto pela implementação do sistema proposto. Enquanto que o quinto e último capítulo é dedicado à conclusão deste trabalho.

---

[1] Internet Data Corporation (IDC) é líder mundial como provedor de inteligência em tecnologia e dados de mercado. É também um guia estratégico para usuários da tecnologia da informação.

2 XML E BANCO DE DADOS

2.1 BREVE HISTÓRICO DA XML

Há alguns anos, a publicação de dados eletrônicos estava limitada a poucas áreas científicas e técnicas, mas atualmente, trata-se de uma atividade universal. O uso da HTML na Internet possibilitou que os dados fossem apresentados em uma estrutura simples e de fácil leitura. Entretanto, a HTML apresenta limitações fundamentadas em sua própria concepção, baseada em marcações fixas. A emergência da XML como um padrão para a representação de dados na Internet pode facilitar a publicação em meios eletrônicos, por prover uma sintaxe simples, legível para computadores e seres humanos.

A XML (Linguagem de Marcação Estendida) é um subconjunto de uma linguagem de marcação criada em 1986 pelo W3C[2] chamada SGML, que permitia que uma marcação específica fosse criada para especificar idéias e compartilhá-las. Apesar de ser uma linguagem de marcação poderosa, a SGML era demasiadamente complexa para ser utilizada. Em 1996, o engenheiro da Sun Microsystems, Jon Bosak, convenceu o W3C a formar uma equipe com o objetivo de utilizar a SGML na internet. Em Novembro daquele ano, foi criado a XML, inicialmente como uma versão simplificada da SGML, e em fevereiro de 1998, a XML tornou-se uma especificação formal, reconhecida pelo W3C.

2.2 CONCEITO

A XML é uma linguagem de marcação compacta, enxuta e flexível que provê meios para descrever, armazenar, intercambiar e manipular dados estruturados [HEI01]. Faz parte também de uma iniciativa para estabelecer um padrão mundial para troca de dados.

[2] World Wide Internet Consortium: grupo baseado no MIT – Massachussets Institute of Technology – responsável pelo projeto de padrões para a Internet.

De acordo com o W3C os objetivos estabelecidos na especificação e modelagem da linguagem XML são os seguintes:

- A XML deve ser diretamente utilizável na Internet;
- A XML deve suportar uma grande variedade de aplicações;
- Deve ser fácil implementar programas que processam documentos XML;
- Documentos XML devem ser claros e legíveis por humanos;
- Possibilitar um meio independente para publicação eletrônica;
- Facilitar o processamento de dados pelo uso de sistemas de baixo custo;
- Facilitar a utilização de meta-dados que auxiliam na busca de informações.

Observa-se com os objetivos acima descritos, a clara preocupação do W3C em tornar a XML um padrão mundial para troca de informações na grande rede ou até mesmo dentro de uma instituição.

2.3 APLICAÇÕES DA XML

A XML pode ser usada tanto no transporte quanto na representação e armazenamento de dados e meta-dados, especialmente em aplicações que utilizam o ambiente da Internet como a principal plataforma de execução e comunicação. Entre suas principais vantagens, pode-se citar a independência de plataforma, tanto de hardware quanto de software, e a capacidade da separação clara entre as três principais camadas do desenvolvimento de software atual: armazenamento, negócios e apresentação.

Em relação à XML pura, as aplicações incluem basicamente páginas na Internet com conteúdo dinâmico, mas que dependam em grande parte de dados está ticos. Como por exemplo, jornais, revistas, documentações de produtos, manuais e etc. Pode-se citar também o uso da XML como a entrada de dados em aplicativos com a finalidade de configurá-los.

Há ainda o campo do transporte de dados, no qual incluem-se principalmente a troca de mensagens entre sistemas semelhantes que são executados em plataformas incompatíveis, e a troca de dados nas corporações que realizam transações com outras organizações. Um exemplo claro disto são os sistemas bancários, que envolvem grandes quantidades de troca de dados entre empresas e aplicações diferentes.

No campo de armazenamento e recuperação de dados, o escopo deste trabalho, existem vários sistemas construídos com o intuito de acelerar e tornar mais eficiente o processo de armazenamento e persistência dos dados ou documentos XML. Esses sistemas serão abordados na próxima seção deste capítulo, assim como os métodos e técnicas de armazenamento e recuperação de documentos XML.

2.4 TÉCNICAS DE ARMAZENAMENTO DE DOCUMENTOS XML

As aplicações que utilizam XML como formato de dados se classificam de acordo com o tipo de documentos que armazenam, que podem ser documentos centrados em dados e documentos centrados em documentos [BAL01].

Os documentos centrados em dados utilizam a XML para transporte de dados e são modelados para o consumo de uma aplicação [BOU02]. Este tipo de documento é caracterizado por uma estrutura formal, granular e sem mistura de conteúdo. Como exemplo de documento XML centrado em dados, pode-se fazer referência aos documentos utilizados para armazenar as informações de cada atendente do sistema de atendimento ao vestibulando:

```
<Atendente Numero="12">
    <Nome>João Paulo Silveira</Nome>
    <Login>joao.paulo</Login>
    <Senha>jsilveira</Senha>
    <Status>Online</Status>
    <Disponivel>true</Disponivel>
</Atendente>
```

Figura 1 Exemplo de documentos centrado em dados

Já os documentos centrados em documentos são caracterizados por uma estrutura irregular ou menos regular que a dos documentos centrados em dados. São

geralmente modelados para que pessoas – e não máquinas – os leiam, como livros, e-mail, etc. Um exemplo desse tipo de documento utilizado no sistema de atendimento ao vestibulando é ilustrado a seguir e tem a função de mostrar ao atendente um resumo das informações mais importantes de um vestibulando ao ser iniciado um atendimento.

```
<Resumo>
O <vestibulando>Felipe de Souza Sobrinho
</vestibulando> tem <idade>17 anos</idade>,
se formou no <escola>Curso e Colégio
Energia</escola> e prestará vestibular para
<curso>Ciências da Computação</curso>, cujo
índice candidato-vaga é de <icv>12.6</icv>.
Seu local de prova é na sala
<sala>201</sala> do <local>Institulo
Estadual de Educação</local>, que fica em
<endProva>Av. Mauro Ramos, 123</endProva>
</Resumo>
```

Figura 2 Exemplo de documento centrado em documento

Na prática, a diferença entre os dois tipos de documentos não é sempre clara. Por exemplo, um típico documento centrado em dados como uma fatura pode conter dados de estrutura irregular como é o caso da descrição da fatura. No caso contrário, um livro pode ser citado, que é um documento centrado em documento e pode conter dados bem estruturados e regulares, como o nome do autor e data de revisão.

Além dos tipos de documentos, também existem diferentes técnicas para armazená-los e recuperá-los, são elas: Abordagem Middleware[3], Banco de Dados Adaptados à XML e os Bancos de Dados Nativos XML. As duas primeiras técnicas se baseiam em bancos de dados relacionais, enquanto a última delas suporta dados XML intrínseca e naturalmente.

2.4.1 Middleware

XML Middlewares são servidores independentes usados para transferir dados entre uma aplicação cliente que utiliza documentos XML e um servidor de banco de dados (geralmente relacional). Utilizam geralmente drivers padrões - JDBC e ODBC

[3] Middleware (Jensen's Technology Glossary): Segunda geração das aplicações de rede que transfere dados de um computador cliente para um computador servidor de Web ou de Banco de Dados.

– para interagir com o banco de dados no servidor e são classificados em duas categorias: orientados a modelos e orientados à modelagem.

Middlewares orientados a modelo processam documentos XML com expressões SQL embutidas. Por exemplo, pode-se recuperar os dados de uma tabela de pedidos e usá-los para construir documentos XML. Já a transferência de dados de documentos XML para o banco de dados é complexa através de middlewares orientados a modelo e por isso a maioria deles não possui tal funcionalidade.

Quanto aos middlewares orientados a modelagem, suas vantagens estão na possibilidade de transferir dados no sentido Documento XML para o Banco de Dados. Essa característica é suportada porque uma modelagem de documentos é feita e então mapeada para o banco de dados.

Vantagens	Desvantagens
Uma vez com o mapeamento configurado, o middleware pode esconder da aplicação os detalhes do banco de dados.	A alteração da modelagem dos documentos XML implica também na alteração da modelagem do banco de dados e também na definição do mapeamento.
Um middleware pode ajudar facilmente a criar redundância dos dados em seu nível a fim de tornar a aplicação mais confiável.	O middleware traz ainda outra camada para a arquitetura da aplicação, o que significa alto custo de manutenção e aumento da complexidade.
A aplicação pode passar diretamente ao middleware objetos em memória correspondentes ao documento XML, sem a necessidade de processar o documento.	Middlewares são mais lentos do que uma conexão direta com o banco de dados.

Tabela 1 Vantagens e desvantagens do Middleware

2.4.2 Bancos de Dados Adaptados/Habilitados à XML

São definidos como bancos de dados relacionais convencionais que se ajustaram para que pudessem armazenar documentos XML centrados em dados, geralmente. Isso se deve ao fato de que as tabelas não são especificamente construídas

como modelos de documentos XML. Porém, muitos desses bancos de dados podem armazenar em uma única coluna um documento centrado em documento como um todo, necessitando assim de processador de texto para as consultas.

O que difere um sistema como esse das outras técnicas de armazenamento é que além de servir como um repositório de documentos XML, também provê funcionalidades de gerenciamento típicas dos bancos relacionais como integridade dos dados, segurança e procedimentos de recuperação de danos.

Vantagens	Desvantagens
Depois de configurado, é o método mais rápido quando muito processamento ocorre no banco de dados.	Requer um tempo considerável de programação antes que possa armazenar, recuperar e consultar documentos XML.
A aplicação pode fazer uso de várias funcionalidades do banco de dados, como integridade e consistência.	Força o uso de linguagens proprietárias para executar operações menos triviais, em vez do uso da XML.
Fácil adicionar funcionalidades da XML às aplicações que já usam os mesmos dados.	Quanto mais documentos armazenados como um todo, mais espaço é consumido.

Tabela 2 Vantagens e desvantagens de um Banco de Dados Adaptado à XML

2.4.3 Bancos de Dados Nativos XML

É sabido que qualquer modelo hierárquico de dados pode ser traduzido em relações [BAL01]. Portanto, em princípio, qualquer documento XML pode ser decomposto e armazenado em banco de dados relacionais. Porém, existem documentos cuja estrutura é tão irregular que impossibilita o mapeamento para o modelo relacional.

Os bancos de dados nativos XML têm como sua unidade fundamental (lógica) de armazenamento um documento XML de qualquer estrutura, assim como o registro de uma tabela é a unidade fundamental dos bancos relacionais.

Todas as funcionalidades de um banco de dados como esse são construídas para lidar especificamente com a XML, incluindo o acesso e as linguagens

de consulta e alteração. Por armazenar documentos XML em sua forma pura e nativa, sua utilização é maior quando a aplicação possui dados semi-estruturados ou documentos centrados em documentos.

Segundo Ronald Bourret [BOU02], pesquisador e escritor especializado em banco de dados e XML, uma regra geral pode ser aplicada entre os tipos de documentos XML e os métodos de armazenamento e recuperação dos mesmos. Essa regra diz que quanto menos rígida é a estrutura de um documento XML, mais difícil é sua modelagem em modelos relacionais. Como mencionado anteriormente, o SAVE faz uso tanto de documentos de estrutura regular, caso das informações de um atendente, quanto de documentos de estrutura menos rígida, por exemplo, o resumo das informações de um vestibulando. Esse é um dos motivos pelo qual escolheu-se um banco de dados nativo XML para implementação do sistema.

Vantagens	Desvantagens
Economiza tempo de programação com uma boa modelagem.	Menos confiável do que os bancos de dados relacionais
Não há perda de performance em conversões de dados para XML e vice-versa.	Ainda em desenvolvimento, e muitas características precisam ser alcançadas.
Mudanças na modelagem de um documento XML não causa alterações no mapeamento dos dados.	Provê baixa integridade dos dados.

Tabela 3 Vantagens e desvantagens de um Banco de Dados Nativo XML

Nesta parte do trabalho, foram abordadas as técnicas existentes para armazenar dados cujo conteúdo esteja em formato XML. Na próxima seção, será feita uma abordagem detalhada sobre o funcionamento, incluindo conceitos e características, de um banco de dados nativo XML, que deste ponto em diante também pode ser referenciado pela sigla NXD (*Native XML Database*) ou NXDs para o seu plural.

2.5 BANCOS DE DADOS NATIVOS XML

2.5.1 Conceito

Não há uma definição formal do que é um NXD, entretanto a organização que desenvolve as especificações de um banco de dados nativo XML - XML:DB Initiative – diz que um NXD:

1) Define um modelo lógico para um documento XML e armazena e recupera documentos de acordo com esse modelo. No mínimo, o modelo deve incluir atributos, elementos e ordem de documentos.

2) Tem um documento XML como unidade fundamental (lógica) de armazenamento, assim como um banco de dados relacional tem uma linha como unidade fundamental (lógica) de armazenamento, por exemplo.

3) Não necessariamente tem um documento XML como modelo fundamental de armazenamento físico. Ele pode, por exemplo, ser construído sobre um banco relacional, hierárquico ou objeto-relacional.

Em resumo, um banco de dados nativo XML é simplesmente um banco de dados desenvolvido para armazenar e acessar XML utilizando a própria XML para isso, o que difere de um banco de dados relacional, onde os dados XML devem ser armazenados em formato tabular e acessados utilizando SQL.

2.5.2 Arquiteturas dos NXDs

As arquiteturas de um banco de dados nativo XML se subdividem em duas categorias: *baseadas em texto* e *baseadas em modelo* [BOU02].

2.5.2.1 Arquiteturas baseadas em texto

Um banco de dados com essa arquitetura armazena XML como texto, que pode ser um arquivo texto, um tipo BLOB (Binary Large Object) em um banco de dados relacional ou um formato proprietário. Essa característica dá a esses sistemas um

ganho de velocidade muito grande quando se quer recuperar documentos inteiros ou fragmentos de documentos, isto porque só é necessária a busca por um índice e, assumindo que o fragmento esteja armazenado continuamente no disco, basta uma leitura. Entretanto, quando se deseja recuperar documentos de diversas partes, são necessárias várias buscas por índices e várias leituras de disco.

2.5.2.2 Arquiteturas baseadas em modelo

Ao invés de armazenarem os documentos XML como texto, os sistemas com arquiteturas baseadas em modelo constroem um modelo interno do documentos e armazena esse modelo. A forma na qual o modelo é armazenado depende do banco de dados. Alguns armazenam o modelo em um banco de dados relacional, outros usam algum formato proprietário otimizado para o modelo.

Os bancos de dados nativos XML construídos *sobre* outros bancos de dados têm performance semelhante à desses na recuperação de documentos, pela razão óbvia de poder delegar tal tarefa aos bancos de dados sobre os quais foram construídos.

Os bancos de dados que usam um formato proprietário de armazenamento têm performance similar aos bancos baseados em texto quando recuperando dados na ordem que foram armazenados. Assim como os bancos baseados em texto, os baseados em modelo também têm problemas quando precisam recuperar dados de forma diferente de como eles foram armazenados.

2.5.3 Características dos NXDs

As principais características dos bancos de dados nativos XML são apresentadas nas seções seguintes.

2.5.3.1 Coleções de documentos

As coleções têm papel similar às tabelas do modelo relacional ou aos diretórios em um sistema de arquivos. Elas podem ser formadas de documentos XML e de sub-coleções. Um exemplo de coleção se dá quando é preciso armazenar as

informações de todos os atendentes de uma central de atendimento. Essa necessidade sugere a criação de uma coleção chamada "*atendentes*" para armazenar os documentos que representam um atendente.

2.5.3.2 Linguagens de consulta XML

Diferentemente da SQL, que é a linguagem de consulta e alteração dos bancos de dados relacionais, a maioria das linguagens de consulta XML é incapaz de realizar qualquer modificação no banco de dados. É o caso da XPath, a linguagem de consulta e navegação mais popular entre os banco de dados nativos XML, juntamente com a XQL.

O cenário de uma consulta pelo nome de um atendente utilizando a XPath seria o seguinte:

Consulta: O nome do atendente cujo login é igual a 'paulo'

/callcenter/pessoas/atendente[login='paulo']/nome

Resultado: formato XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nome>Paulo Coelho</nome>
```

Apesar de apenas um elemento ter sido consultado (nome do atendente), o resultado possui formato XML. Isso implica o uso de processadores de documentos XML, como SAX e DOM, para traduzir os dados em variáveis utilizáveis pela aplicação.

2.5.3.3 Modificações

Apesar de ser uma das características dos NXDs que precisam ser melhoradas, existe uma grande variedade de estratégias para atualizar ou remover um documento XML, desde a simples substituição de um documento por outro com as alterações feitas até o uso de linguagens que especificam como alterar ou remover

fragmentos de um documento. Como regra geral, cada produto capaz de modificar fragmentos XML, mantém sua própria linguagem para isso, embora a XML:DB Initiative tenha o padrão XUpdate, suportado por grande parte dos NXDs existentes.

2.5.3.4 Transações, travamento e concorrência

As transações são, de fato, características suportadas pela maioria dos bancos de dados nativos XML. Já o travamento é feito em nível de documento e não em nível de fragmentos de documento; conseqüentemente, a concorrência entre usuários deve ser relativamente baixa.

Essa característica depende da aplicação em si e de como os documentos e coleções estão dispostos na estrutura de dados. Por exemplo, se os dados de um vestibulando são armazenados em documentos separados, ou seja, um documento XML para cada vestibulando, dificilmente haverá problemas de concorrência, acontecendo o inverso caso as informações de todos os vestibulandos estivessem armazenadas em apenas um documento.

2.5.3.5 Round-Tripping

Uma importante característica dos bancos de dados nativos XML é a capacidade de armazenar um documento XML e poder recuperar exatamente o mesmo documento. Essa característica é vital para aplicações centradas em documento e está presente em todos NXDs.

2.5.3.6 Dados remotos

Alguns NXDs podem incluir dados remotos em documentos armazenados no banco. Geralmente esses dados são recuperados de um banco relacional usando ODBC, OLE DB ou JDBC, por exemplo.

2.5.3.7 Índices

Assim como em qualquer banco de dados, os índices são usados para aumentar a performance em consultas. Entretanto, tal característica não se aplica a todos os bancos de dados nativos XML.

2.5.3.8 Normalização

Normalização se refere à definição de um formato lógico adequado para estruturas de dados, com o objetivo de evitar duplicação de dados e minimizar o espaço utilizado por eles [BOR02].

Uma boa modelagem dos documentos deve garantir que nenhum dado seja repetido, causando inconsistências. Uma vantagem da normalização em bancos de dados nativos XML é que suportam propriedades multivaloradas, enquanto que a maioria dos relacionais não o faz. Isso torna possível normalizar os dados de uma forma mais simples e intuitiva.

Voltando ao exemplo da central de atendimento. Para armazenar as mensagens enviadas por um atendente em um banco relacional, deve-se criar uma outra tabela contendo as mensagens de cada atendente, uma mensagem por atendente, por linha. Em um NXD, esta informação pode ser armazenada em um único documento sem redundância, já que a própria natureza hierárquica de XML garante isso, pois um elemento pai pode ter vários elementos filhos.

2.5.3.9 Integridade referencial

Assim como a normalização, a integridade referencial em NXDs também é semelhante à do modelo relacional. Em resumo, serve para garantir a validade de ponteiros entre dados de diferentes tabelas ou documentos.

Em bancos relacionais, a integridade referencial garante que chaves estrangeiras apontem para chaves primárias válidas. Em bancos de dados nativos XML, a integridade referencial é garantida pelos XLinks (mecanismos de “linkagem”), que apontam para documentos ou fragmentos de documentos válidos.

2.5.3.10 Escalabilidade

Escalabilidade é a capacidade de manter a alta performance mesmo com um aumento significativo do tamanho ou carga do banco. Assim como bancos hierárquicos e relacionais, os NXDs usam índices para pesquisar e encontrar dados. Isto significa que localizar documentos e fragmentos de documentos está relacionado com o tamanho do índice e não com tamanho ou quantidade de documentos e que a performance dos NXDs é a mesma comparada a outros tipos de bancos de dados.

Ao contrário dos bancos relacionais, NXDs "*linkam*" dados relacionados fisicamente. Por esse motivo, eles deveriam ter uma boa escalabilidade com relação à consulta de dados visto que, para esta tarefa, links físicos são mais rápidos que os lógicos. No entanto, esta escalabilidade é limitada porque a ligação física somente se aplica a uma hierarquia particular, assim a recuperação de dados em uma hierarquia diferente não é tão rápida..

Para minimizar esse problema, os NXDs fazem grande uso dos índices, chegando até a indexar todos os elementos e atributos de um documento. Isto pode ser uma solução para bancos que são mais usados para consultas, mas quando muitas atualizações são feitas, essa performance cai drasticamente [BOU02].

Além das diferenças entre bancos de dados relacionais e os NXDs já citadas nesta seção, há ainda algumas características intrínsecas da própria XML que são contrárias ao modelo relacional e resumidas neste quadro:

XML	Modelo Relacional
Dados em estrutura hierárquica	Dados em tabelas
Os nós possuem valores de elemento e/ou atributo	As células têm apenas um valor
Elementos podem ser aninhados	Os valores das células são indivisíveis
Esquemas são opcionais	Esquemas são obrigatórios
Consulta com padrões XML (XQuey, XPath)	Consulta com SQL aperfeiçoado para XML

Tabela 4 Diferenças entre o modelo relacional e a XML (Fonte: [CHA01])

2.6 OS PRODUTOS E A ESCOLHA

Como já dito anteriormente, há vários produtos cuja finalidade é armazenar e recuperar documentos XML. Nesta seção do trabalho, citarei os bancos de dados nativos XML que foram mais encontrados nas referências e os dividirei em gratuitos e comerciais.

2.6.1 NXDs Comerciais

2.6.1.1 Tamino XML Server

O Tamino XML Server é desenvolvido pela empresa Software AG, proprietária do banco de dados relacional Abadas e certamente é o banco de dados nativo XML mais maduro dentre os existentes. É composto por produtos construídos em três camadas: serviços centrais, serviços de habilitação e soluções. O NXD propriamente dito é implementado na camada de serviços centrais e possui mecanismos de mapeamento que descrevem onde os dados de um determinado documento XML são armazenados. Isso permite que um único documento possa ser composto de dados que estejam separados em diferentes fontes de dados.

Oferece interface de programação para várias linguagens, incluindo Java e C++. Possui suporte para a linguagem de consulta XPath, além de sua linguagem proprietária.

O Tamino pode ser utilizado em Windows e na maioria dos sistemas Unix. Seu preço inicial é de U$45.000,00 por CPU.

2.6.1.2 eXtensible Information Server (XIS)

O NXD da empresa eXcelon Corp. armazena e manipula dados XML com ênfase em alta performance e controle de concorrência, o que permite a consulta de documentos que estão sofrendo alterações.

Permite a criação de procedimentos chamados gatilhos, que são executados ao sinal de um evento. É o único NXD a oferecer importação de outras

fontes de dados através de OLE DB e ODBC. É compatível com o Windows, Solaris e HP-UX e pode ser comprado por U$40.000,00.

2.6.2 NXDs gratuitos

2.6.2.1 Apache Xindice

O Xindice é um banco de dados nativo XML de código-aberto totalmente desenvolvido em Java e projetado para armazenar grande quantidade de pequenos documentos XML. Está incluso no pacote de desenvolvimento de Serviços Web da Sun Microsystems e possui implementações da XPath e XUpdate, além de dar suporte a várias linguagens de programação através de um mecanismo de chamada remota de procedimentos.

2.6.2.2 Exist

É um NXD que permite armazenar o conteúdo XML tanto e sua forma nativa quanto em bancos de dados relacionais, como MySQL, PostgreSQL e Oracle. Seu mecanismo de consulta foi projetado para prover rapidez, usando índices em todos os elementos de um documento.

Assim como o Xindice, o Exist é desenvolvido em Java e pode ser executado em qualquer computador que tenha a máquina virtual Java instalada.

2.6.3 A escolha

Os produtos comerciais acima descritos são extremamente superiores aos bancos de dados de código aberto. Além do motivo evidente para isso, existem ainda fatores que justificam tal superioridade, como o tempo de vida de cada um dos produtos. As empresas que comercializam NXDs começaram os trabalhos de desenvolvimento há pelo menos cinco anos, além de possuírem conhecimento de experiências anteriores sobre bancos de dados. Já os NXDs gratuitos foram criados a no máximo dois anos e ainda estão em suas primeiras versões.

Infelizmente, o preço de um produto comercial é muito elevado, impossibilitando sua utilização nesse projeto. Dessa maneira, uma solução de código aberto foi a melhor escolha. Analisando as características de ambos os bancos de dados gratuitos, optou-se pelo Xindice pelos seguintes motivos:

- O Xindice está incluso no pacote de desenvolvimento da Sun Microsystem;
- O software Exist ainda não possui sua primeira versão finalizada;
- Segundo seus próprios desenvolvedores, o Exist é recheado de erros;
- O Xindice, assim como o SAVE, foi projetado para armazenar pequenos documentos, porém em grande quantidade;
- O NXD da Apache possui implementação da linguagem XUpdate, que permite fazer alterações em documentos XML. Tal característica não é suportada pelo Exist.

Sendo o Xindice escolhido para ser usado no SAVE como banco de dados nativo XML, no próximo capítulo deste trabalho será feita uma abordagem de sua arquitetura e funcionamento.

3 XINDICE

3.1 HISTÓRICO

Os primeiros passos no desenvolvimento do Xindice, antes chamado de dbXML, foram dados por Tom Bradford em 1999, quando a linguagem utilizada ainda era o C/C++. No ano de 2000, o código-fonte migrou para Java e foi doado à *The Apache Software Foundation* em Dezembro de 2001.

3.2 O QUE É

O Xindice é uma banco de dados nativo XML gratuito que armazena e indexa documentos XML com o objetivo de disponibilizar o conteúdo para uma aplicação cliente com a vantagem de utilizar pouco processamento no lado servidor [XIN02].

3.3 O MODELO

A idéia principal por trás do Xindice é prover uma forma simples de armazenar e gerenciar um grande número de documentos XML. Essa meta é alcançada criando coleções de documentos, onde cada documento é armazenado em formato comprimido. Tal característica eleva a velocidade ao trabalhar com dados em XML e provê fácil mecanismo para consulta e manipulação dos documentos como um conjunto.

O Xindice é otimizado para armazenar grande número de pequenos documentos XML de até 50Kb. Pode-se também adicionar documentos maiores, porém este não é o melhor cenário. Há, inclusive, uma recomendação da própria Apache para que isso não aconteça. O tamanho máximo de um documento XML que pode estar em uma coleção do Xindice é de 5Mb.

3.4 FUNCIONAMENTO

Quando está sendo executado, o Xindice armazena uma séria de informações em objetos dentro da máquina virtual Java. Dentre esse dados, merecem destaque:

- A representação da hierarquia de coleções ;
- O estado da conexão com o cliente;
- Dados em *cache*.

Além dessas informações, uma instância do Xindice também precisa de acesso aos arquivos que contêm dados XML, que são armazenados em uma hierarquia de diretórios, começando na chamada *raiz do banco de dados*, cuja função e estrutura será detalhada posteriormente.

3.4.1 Modos de Execução

Existem duas maneiras de configurar a execução do Xindice, o modo embutido e o modo servidor.

No primeiro caso, uma aplicação Java completa configura uma instância do Xindice em sua própria máquina virtual e apenas essa aplicação é capaz de acessar e manipular os dados contidos no Xindice.

No modo servidor, o Xindice é executado como uma aplicação padrão Java, que pode também estar dentro de um container de aplicações web, como o Apache Tomcat. Nesse modo, vários clientes podem acessar o Xindice de diferentes máquinas virtuais, possivelmente em outro computador, utilizando XML-RPC, um padrão XML para chamadas remotas de procedimento sobre o protocolo HTTP.

3.4.2 Organização

No Xindice, todos os dados XML armazenados são organizados logicamente em hierarquia de coleções. Uma coleção é exatamente o que seu nome

sugere: contém um determinado número de documentos XML além de suas próprias coleções e portanto, provendo hierarquia.

A coleção raiz do banco de dados (/db) é a única que contém características especiais como a ausência de coleção-pai e de documentos XML e é criada na primeira execução do Xindice. Seu conteúdo inicialmente é composto apenas por outra coleção chamada *system*, que contêm o dicionário de dados de todo o Xindice.

A coleção *system* contém duas coleções filhas: a SysConfig e a SysSymbols. Esta última contém vários documentos, que são tabelas de símbolos utilizadas para armazenamento dos nomes dos elementos e atributos de todo o conteúdo XML do banco de dados. Já a coleção SysConfig possui um documento chamado database.xml, cuja função é de manter as configurações de todas as coleções do banco de dados. A estrutura desse arquivo é ilustrada a seguir:

```
<?xml version="1.0" ?>
- <database name="db">
    <xmlobjects />
  - <collections>
    - <collection compressed="true" name="callcenter">
        <filer class="org.apache.xindice.core.filer.BTreeFiler" gzip="true" />
        <indexes />
        <xmlobjects />
      - <collections>
        - <collection compressed="true" name="mensagem">
            <filer class="org.apache.xindice.core.filer.BTreeFiler" gzip="true" />
            <indexes />
            <xmlobjects />
          - <collections>
            + <collection compressed="true" name="at">
            + <collection compressed="true" name="vt">
            </collections>
          </collection>
        - <collection compressed="true" name="pessoa">
            <filer class="org.apache.xindice.core.filer.BTreeFiler" gzip="true" />
            <indexes />
            <xmlobjects />
          - <collections>
            + <collection compressed="true" name="admin">
            + <collection compressed="true" name="atendente">
            - <collection compressed="true" name="vestibulando">
                <filer class="org.apache.xindice.core.filer.BTreeFiler" gzip="true" />
              + <indexes>
                <xmlobjects />
              </collection>
            </collections>
          </collection>
        </collections>
      </collection>
    </collections>
</database>
```

Figura 4 Conteúdo parcial do arquivo de configuração de coleções do Xindice

3.4.2.1 Diretórios

Fisicamente, cada diretório de uma coleção contém no mínimo um arquivo binário contendo todos dados dos documentos XML armazenados naquela coleção. O nome deste arquivo é composto pelo nome da coleção na qual estão os documentos seguido da extensão "*.tbl". No diretório */db/callcenter*, por exemplo, existe um arquivo chamado *callcenter.tbl*. Esse arquivo é o modelo fundamental de armazenamento físico proprietário do Xindice. Como citado na seção 2.5.1, existem NXDs cujo modelo de armazenamento está baseado em bancos de dados relacionais ou objeto-relacionais.

3.5 CARACTERÍSTICAS

O Xindice, em sua versão atual (1.0), possui a maioria das características de um NXD, porém as transações e o controle de concorrência são as maiores carências desta distribuição.

O gerenciamento do banco de dados Xindice se dá através de uma rica linha de comandos. Com ela, pode-se executar tarefas desde de uma simples inserção de documento em uma coleção até a exportação de uma árvore de diretórios.

3.5.1 Linguagem de consulta

A linguagem de consulta com a qual o Xindice trabalha é a XPath, que endereça partes de um documento XML utilizando uma sintaxe hierárquica que lembra o endereço de uma página na Internet. No Xindice, a XPath pode ser utilizada para consultar dados tanto em programas escritos conforme a especificação quanto na linha de comando oferecida pelo banco de dados.

Uma das deficiências da XPath é a falta de propriedades como agrupamento e ordenação, tão comuns à linguagem de consulta de banco de dados relacionais, a SQL.

3.5.2 Modificações

Na documentação do banco de dados Xindice, destaca-se o suporte à linguagem XUpdate, padrão criado pela XML:DB Initiative. Entretanto, a implementação desta linguagem na versão atual está incompleta e existem vários erros reportados à Apache em relação ao funcionamento de modificações no banco de dados.

3.5.3 Índices

O suporte à criação de índices em campos de documentos XML é uma importante característica do Xindice. Sua utilização se dá através da linha de comando e garante maior velocidade na busca de documentos XML.

A exemplo abaixo ilustra a adição de um índice chamada *idx_inscricao* no elemento *inscrição* em todos os documentos da coleção /db/*callcenter/pessoa/vestibulando*:

xindice ai -c /db/ callcenter/pessoa/vestibulando -n idx_inscrição –p inscricao

3.5.4 Controle de usuários

Atualmente, o Xindice não possui qualquer sistema de autenticação perante o acesso ao banco de dados. A previsão de implementação desta característica está prevista para a versão 1.1. do Xindice.

4 IMPLEMENTAÇÃO

Este capítulo apresenta a modelagem criada para o desenvolvimento da plataforma de atendimento ao vestibulando, partindo da especificação dos requisitos e indo até a estrutura de dados.

4.1 ESPECIFICAÇÃO DOS REQUISITOS

Para a completa implantação do SAVE na sede de uma instituição de ensino, requisitos são necessários e expostos a seguir.

Primeiramente, por ser uma aplicação que estará disponível na Internet, a arquitetura cliente/servidor será necessária para sua implementação. O cenário básico necessário para esse tipo de aplicação é composto por um navegador (vestibulando), a aplicação do atendente, um servidor WEB, o software servidor e a rede de comunicação. O uso de objetos distribuídos também ajudará na construção deste sistema, já que seu objetivo é abstrair os detalhes da comunicação distribuída e torná-la responsabilidade da própria infra-estrutura e não do desenvolvedor [CON99].

Uma outra grande necessidade de um sistema distribuído, como o que está sendo proposto neste trabalho, é que ele seja seguro. Isso inclui o fato de somente atendentes cadastrados podem utilizar o sistema e também a exigência de que as informações que trafegam na grande rede mundial sejam visualizadas somente pelo atendente e pelo vestibulando, entre outras características.

4.2 AMBIENTE DE DESENVOLVIMENTO

O processo de implementação do sistema de atendimento ao vestibulando foi inteiramente executado no Sistema Operacional Conectiva Linux 8.0, utilizando várias outras ferramentas e tecnologias citadas abaixo:

- **Máquina Virtual Java™: a** versão 1.4.1 da linguagem Java™ foi utilizada para desenvolver o SAVE.

- **Apache Tomcat:** é um *container*[4] de aplicativos escritos em Java™ e JSP que serve como referência de implementação dessas duas linguagens. Sua versão 4.1.18 foi utilizada para publicar as páginas nas quais o vestibulando navega para obter atendimento.

- **VIM:** Editor de texto padrão de qualquer ambiente UNIX. Nele foi escrita a maioria das classes que compõem o SAVE.

- **JBuilder:** Poderoso ambiente de desenvolvimento para a linguagem Java™ que permite a construção mais rápida de interfaces gráficas. Utilizei sua sexta versão para esboçar as telas da aplicação do atendente.

- **Apache Xindice:** Banco de dados nativo XML escrito em Java™ e ainda em desenvolvimento. Fez-se uso de sua versão 1.0 nesse projeto.

4.3 ARQUITETURA DO SISTEMA

Para que a escalabilidade e reutilização sejam características do SAVE, ele foi desenvolvido separando *apresentação*, *regras de negócios* e *acesso aos dados* em camadas distintas, cujas funções são abordadas a seguir. Nas próximas seções, o funcionamento e construção das aplicações de cada camada serão detalhados.

4.3.1 Camada de Apresentação

A camada de apresentação do SAVE é representada pela interface na qual ocorre a interação entre o usuário e o sistema. Por existirem dois tipos de usuários, os vestibulandos e os atendentes, há também duas aplicações diferentes na camada de apresentação. Ambas as aplicações são chamadas "*thin client"*, ou seja, aplicações cujas funcionalidades estão restritas à manipulação da interface com o usuário, ao envio de requisições à camada de negócios e ao recebimento das suas respostas.

[4] Entidade que provê algum tipo de serviço e possui procedimentos de gere nciamento e distribuição do mesmo.

4.3.2 Camada de Negócios

É nessa camada que está a inteligência de todo o sistema, e conseqüentemente é a que exige mais esforço de programação. Essa camada interage tanto com a camada de apresentação quanto com a camada de dados.

A função básica e mais executada pela camada de negócios é a de receber uma requisição da camada de apresentação, acionar a camada de dados e receber informações da mesma e enviá-las como resposta à camada de apresentação.

Na arquitetuta do SAVE, essa camada é representada por um processo executado no Servidor de Aplicações e Dados, que é o computador que hospeda, além do SAVE, o servidor de páginas, o Xindice.

4.3.3 Camada de Dados

A camada de dados é a que contém todas as informações necessárias ao sistema, ou seja, dados dos vestibulandos, dos atendentes, mensagens trocadas entre eles, etc. Relaciona-se apenas com a camada de negócios, com a tarefa de disponibilizar os dados nela contidos.

O banco de dados nativos XML, Xindice, representa esta camada na arquitetura geral do SAVE, ilustrada na figura seguinte.

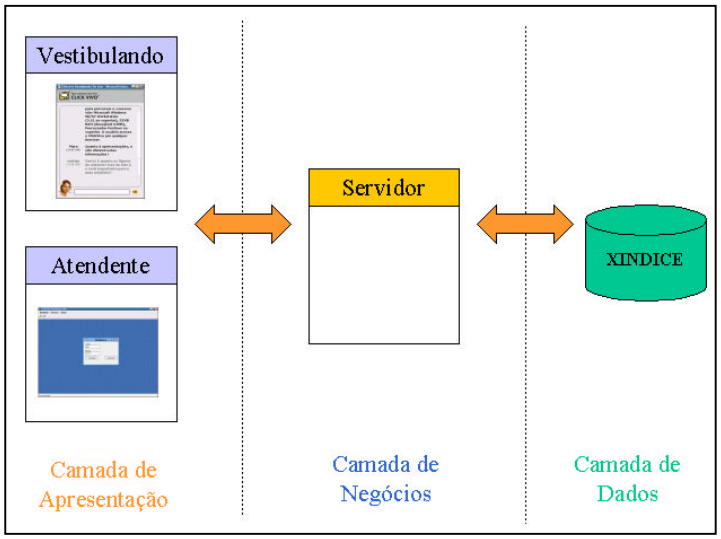

Figura 5 Arquitetura SAVE

4.4 IMPLEMENTAÇÃO DA CAMADA DE DADOS

O SAVE possui a camada de dados baseada no Xindice. Portanto, para que esta camada estivesse apta para interagir com as outras, foi preciso:

1. Instalar e configurar o Xindice;
2. Modelar a estrutura de dados;
3. Criar no Xindice a estrutura modelada;
4. Gerar dados;
5. Disponibilizar classe de acesso ao Xindice.

4.4.1 Instalação do Xindice

O Xindice vem acompanhado com um bom material de apoio à sua instalação e por isso esta fase foi executada com muita rapidez e não exigiu grandes esforços. Todo o procedimento tomado para ter o Xindice apto ao uso se resumiu a descompactar os arquivos da distribuição e em seguida executar o *script*[5] de inicialização do banco de dados.

4.4.2 Modelagem dos dados

Como já frisado anteriormente (seção 2.5.2.3), a performance de um banco de dados nativo XML, assim como o nível de concorrência entre usuários depende de uma boa modelagem dos dados. Por isso, a estrutura de dados foi desenvolvida pensando nas seguintes premissas:

1. **Documentos XML devem ser pequenos:** o Xindice não foi projetado para armazenar arquivos de grande porte (5MB). Tal característica juntamente com o fator concorrência motivaram a decisão de separar cada *"registro"* de uma entidade em apenas um documento XML. Exemplificando, a coleção destinada aos vestibulandos contém vários documentos XML, cada um contendo as informações de

[5] Arquivo texto que contém comandos e instruções a serem executados em algum ambiente Unix

apenas um vestibulando. Essa abordagem é muito mais eficiente do que ter um só documento XML para armazenar os dados de todos os vestibulandos.

2. **Consulta aos dados deve ser rápida:** como o Xindice utiliza a linguagem de consulta XPath, é de bom senso criar uma estrutura que ofereça endereços absolutos para consultar o banco de dados. Por exemplo, ao consultar o histórico de mensagens de um atendente, o ideal seria ter na estrutura de dados uma coleção nomeada de acordo com um identificador exclusivo do atendente (login) e dentro dela documentos XML com as mensagens enviadas e recebidas anteriormente. O endereço absoluto então poderia ser "*//callcenter/mensagens/atendente/<login>/historico*". Essa estratégia evita uma pesquisa de mensagens cujo remetente ou destinatário fosse "Fulano", por exemplo.

Seguindo essas duas premissas, a modelagem de dados final desenvolvida para o SAVE é apresentada abaixo.

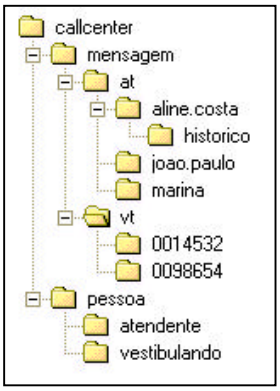

Figura 6 Estrutura de dados no Xindice

A figura ilustra apenas as coleções criadas e alguns exemplos do endereço absoluto já discutido na segunda premissa. Dentro da coleção *mensagem* existem duas outras que correspondem às mensagens enviadas para vestibulandos e para atendentes. Percebe-se também o compromisso com a primeira premissa ao criar coleções distintas para as informações de atendente e vestibulandos. O conteúdo dessas coleções é formado por diversos documentos XML representando cada um deles.

4.4.3 Criação da modelagem no banco de dados

Para criar a estrutura de dados descrita na seção anterior, fiz uso dos comandos de administração do Xindice. O *script* para criação do banco de dados é mostrado na figura abaixo:

```
#!/bin/bash
# Script para criação do banco de dados SAVE

# Criando a coleção 'callcenter'
xindice add_collection -c /db -n callcenter

# Criando a coleção 'mensagem' dentro de 'callcenter'
xindice add_collection -c /db/callcenter -n mensagem

# Criando a coleção 'pessoa' dentro de 'callcenter'
xindice add_collection -c /db/callcenter -n pessoa

# Criando a coleção 'at' dentro de 'mensagem'
xindice add_collection -c /db/callcenter/mensagem -n at

# Criando a coleção 'vt' dentro de 'mensagem'
xindice add_collection -c /db/callcenter/mensagem -n vt

# Criando a coleção 'atendente' dentro de 'pessoa'
xindice add_collection -c /db/callcenter/pessoa -n atendente

# Criando a coleção 'vestibulando' dentro de 'pessoa'
xindice add_collection -c /db/callcenter/pessoa -n vestibulando
```

Figura 7 Script de criação da estrutura de dados

4.4.4 Geração de Dados

Depois de criada a estrutura dos dados no Xindice, foi preciso povoar as coleções com documentos XML reais. Nesse sentido, um programa à parte foi desenvolvido para adicionar dez mil documentos XML na coleção de vestibulandos. O restante das coleções poderia ser preenchido com documentos XML criados pela própria aplicação SAVE.

Essa fase também teve caráter de teste de rapidez e eficácia do Xindice. Observou-se que o tempo decorrido para a inserção os dez mil documentos XML, cada qual representando um vestibulando, foi de dois minutos e dezesseis segundos (136 segundos) não incluindo o tempo de conexão com o banco de dados, que está em torno de um segundo.

Para efeito de comparação, foi feito o mesmo teste para o banco de dados relacional líder no mercado, o Oracle 9i. Uma tabela para os vestibulandos foi criada

com os mesmos campos existentes no documento XML. A inserção de dez mil registros neste banco de dados através de um programa Java durou 310 segundos, ou seja, mais que o dobro do tempo decorrido no teste feito com o Xindice.

Tanto no programa de inserção de documentos XML no Xindice quanto no programa que insere registros no Oracle, um laço com dez mil iterações foi realizado. Utilizando as funcionalidades do Oracle, criou-se um procedimento armazenado que fizesse as 10000 inserções na tabela de vestibulandos. Com isso, a aplicação Java pôde executar apenas este procedimento, o que levou 3,5 segundos.

Os testes foram realizados em um computador com a seguinte configuração: Processador Pentium III, 733MHz e Memória de 128 Mb. O quadro abaixo resume o resultado obtido com os testes realizados:

Inserção de 10.000 vestibulandos através de uma aplicação Java		
Xindice	**Oracle + SQL**	**Oracle + Procedimento**
136 segundos	*310 segundos*	*3,5 segundos*

Tabela 5 Resumo dos testes realizados na geração dos dados

4.4.5 Acesso ao banco de dados

Como em toda aplicação composta de três camadas, a função da camada de dados é permitir que a camada de negócios possa acessar, fazer consultas e alterações no dados, independentemente de qual seja o sistema que os gerencie.

Neste projeto, uma classe Java foi criada especificamente para intercambiar mensagens entre o Xindice e a camada de negócios. Dentre os métodos mais importantes dessa classe estão: *Conectar, Pesquisar, Inserir Documento, Criar Coleção, Atualizar e Remover Documento*.

Com exceção do método *Atualizar,* todos os outros foram implementados de acordo com o que recomendam os desenvolvedores do Xindice. Apesar de sua documentação divulgar o suporte à linguagem XUpdate, a versão 1.0 do Xindice

apresentou problemas ao executar tarefas de atualização no banco de dados. Essa dificuldade será erradicada em breve na distribuição da versão 1.2 do NXD da Apache.

Para que os dados pudessem ser alterados pela aplicação, fez-se uso de um método pouco convencional, que é o de remover o documento e substituí-lo por outro já com os dados alterados.

Merece destaque também o método implementado para a pesquisa. Uma seqüência de testes foi realizada a fim de avaliar a performance do Xindice com os dez mil vestibulandos registrados. Os testes consistiram na comparação da pesquisa por campos indexados e não indexados. O quadro abaixo resume o resultado obtido:

	Campo não indexado	**Campo indexado**
1. Consultar o nome de todos os vestibulandos cadastrados.	*121 segundos*	*82 segundos*
2. Consultar o nome de do vestibulando cuja inscrição é igual a 0295344.	*71 segundos*	*0,141 segundos*

Tabela 6 Resultado dos testes de consulta no Xindice

A tabela acima mostra que a simples adição de um índice pode fazer com que a performance alcance melhorias notáveis. No primeiro caso, a adição de um índice representou uma diminuição de 47% no tempo de resposta, enquanto que no segundo teste, o índice criado fez com que o resultado da consulta fosse obtido 99,8% mais rapidamente.

Com o exposto, todas as responsabilidades da camada de dados estavam implementadas para o devido uso da camada de negócios.

4.5 IMPLEMENTAÇÃO DA CAMADA DE NEGÓCIOS

A camada de negócios é o cérebro do sistema de atendimento idealizado neste trabalho e é responsável por disponibilizar serviços para utilização da camada de apresentação e ainda interagir com a camada de dados.

4.5.1 Interação com a camada de apresentação

A comunicação entre esta camada e a camada de apresentação se dá através do protocolo padrão para objetos distribuídos da linguagem Java™ chamado RMI (Remote Method Invocation) ou Chamada Remota de Métodos. Com este protocolo as aplicações do Atendente e do Vestibulando presentes na camada de apresentação podem solicitar serviços remotos registrados pela camada de negócios. A figura a seguir exemplifica como é feita uma chamada de método remoto de uma aplicação cliente para a aplicação servidor do SAVE.

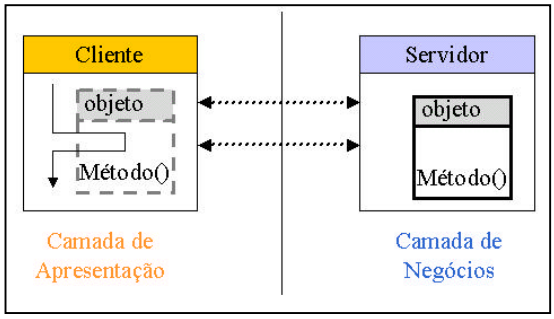

Figura 8 Chamada remota de método entre as camadas de apresentação e negócios

Alguns dos métodos registrados pelo servidor mais utilizados pelas aplicações da camada de apresentação são:

Conectar: esse método é responsável por checar as informações de um atendente ou vestibulando e permitir o acesso ao sistema. Retorna o valor *verdadeiro* se as informações estão corretas, caso contrário sua resposta é *falso*.

Enviar Mensagem: procedimento disponível também para vestibulandos e atendentes, esse método permite a troca de mensagens entre eles.

Receber Mensagem: verifica se há mensagem para o requisitante. Se sim, retorna um objeto Mensagem com todos os seus atributos. Caso contrário, o valor obtido como resposta é nulo.

Encerrar Atendimento: tanto os vestibulandos quanto os atendentes podem encerrar um atendimento.

4.5.2 Interação com a camada de dados

Camada de negócios e camada de dados se comunicam através da classe criada especificamente para o banco de dados descrita na seção 4.4.5.

4.6 IMPLEMENTAÇÃO DA CAMADA DE APRESENTAÇÃO

Apesar de existirem duas aplicações na camada de aplicação, esta é a camada que exige menos esforço de programação. O motivo para isso é que, como já mencionado anteriormente, ambas as aplicações são classificadas como *thin client*, ou seja, aplicações pequenas cuja lógica de programação se restringe a exibir informações provenientes da camada de negócios.

Na aplicação do atendente, cada atendente do sistema de atendimento SAVE poderá tirar dúvidas de quatro vestibulandos simultaneamente. A figura a seguir ilustra como funciona o atendimento a um vestibulando:

Figura 9 Atendimento feito a três vestibulandos simultaneamente

A interface com a qual o vestibulando interage é acessada através da Internet, na página da instituição de ensino, por exemplo. A seguir, é apresentada a figura que representa a aplicação do vestibulando:

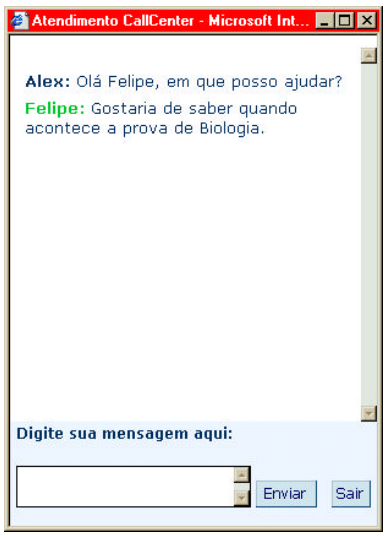

Figura 10 Janela na qual o vestibulando tira suas dúvidas

5 CONCLUSÃO

Com o crescimento espantoso no uso da Internet nos últimos anos, surgiu a necessidade de interoperabilidade entre as plataformas de hardware e software existentes na rede. Dessa necessidade, nasceu a XML e um aumento expressivo no volume de informações sendo transportado e armazenado por sistemas de bancos de dados. Conseqüentemente, surgiram dificuldades na manipulação e integração destes dados com bancos de dados já existentes.

Ao avaliar as pesquisas na área de Banco de Dados Nativos XML realizadas neste trabalho, percebeu-se que esse tipo de sistema ainda é um pouco imaturo se comparados à robustez e à confiabilidade de um banco de dados relacional. Essa imaturidade é compreensível, visto que tal tecnologia nasceu há pouco mais de quatro anos. Além dessa barreira inicial, NXDs enfrentam a falta de padronização de consultas, armazenamento e manipulação de documentos XML.

Entretanto, é grande o entusiasmo da comunidade desenvolvedora em aumentar a qualidade dos NXDs de torná-los a opção mais utilizada para armazenamento e recuperação de documentos XML. Para tanto, todos se espelham no trabalho já realizado no modelo relacional.

Vê-se também um crescente interesse das empresas fabricantes de bancos de dados relacionais nas qualidades da XML, visto que a maioria de seus produtos já se ajustou a fim de prover suporte ao gerenciamento de dados nesse formato. Por isso, vejo a XML hoje como centro das atenções e acredito numa possível convergência dos NXDs e bancos relacionais em um novo e único produto padronizado para o armazenamento e recuperação de documentos XML.

5.1 TRABALHOS FUTUROS

Na continuidade deste trabalho, há ainda uma série de propostas que podem ser pesquisadas e implementadas. Entre elas, destacam-se:

- **Utilização da XUpdate:** Na atual implementação do banco de dados nativo XML da Apache, não há possibilidade de atualizar os dados de um documento XML através da se sua linguagem XUpdate. Esse problema será corrigido na próxima versão e de fato trará benefícios para aplicação apresentada neste trabalho.

- **Comparações práticas:** Apesar de algumas comparações terem sido feitas entre os NXDs e bancos relacionais como o Oracle, seria bastante válida a experiência de executar uma bateria de testes com as ações mais comuns que um banco dados provê: consulta, inserção, alteração e remoção de dados.

- **Raciocínio Baseado em Casos:** Atualmente, a Inteligência Artificial (IA) gera grande interesse na pesquisa e desenvolvimento de sistemas que possam gerar conhecimento a partir da análise do conhecimento já obtido. Em sistemas de atendimento como o SAVE, essa ferramenta da IA ganha importância e pode agilizar o atendimento da seguinte forma: questões freqüentemente feitas por vestibulandos podem ser armazenadas no banco de dados como uma lista juntamente com suas respostas. À medida que outros vestibulandos fizessem perguntas iguais ou parecidas com as que estão na lista, a implementação do RBC poderia recuperar as respostas automaticamente.

- **Cifragem das Mensagens:** Qualquer sistema de informação, seja ele utilizado na Internet ou não, deve prezar pela segurança de seus dados. Em sua atual versão, o SAVE é desprovido de qualquer mecanismo de cifragem das mensagens que trafegam na grande rede mundial. Pretende-se, portanto, criar procedimentos de cifragem das mensagens antes que elas sejam expostas. Para isso pode-se utilizar os pacotes da própria linguagem Java específicos para este tipo de tarefa ou ainda fazer uso dos padrões do W3C, que recentemente anunciou sua recomendação para Cifragem e Decifragem de documentos XML [XEN02].

- **Análise Estatística:** Uma ferramenta espetacular para um sistema de atendimento é a geração de relatórios estatísticos. Com ela, poderíamos analisar a eficácia do atendimento dado ao vestibulando e responder perguntas como: "Quantos vestibulando são atendidos por dia?", "Qual é o perfil do vestibulando que procura

respostas para suas dúvidas na Internet?", "A quantidade de atendentes é proporcional ao número de vestibulandos que procuram atendimento?".

Pela forma como a arquitetura do SAVE foi concebida, sempre se pode agregar novos módulos ou substituí-los por mais avançados, ajudando à sua própria evolução e atendendo às exigências de usuários e do mercado também.

6 REFERÊNCIAS BIBLIOGRÁFICAS

[BAL01] BANSAL, Vinay; ALAM, Asna. *Study and Comparison of Techniques to Efficiently Store and Retrieve XML Data*. Disponível em: http://www.cs.duke.edu/~vkb/advdb/xmldatabase/documents/FinalReport.doc. Acesso em: 10 out. 2002.

[BAT02] BATES, James. *Xindice Internals*. Disponível em: http://xml.apache.org/xindice/dev/guide-internals.html. Acesso em: 25 nov. 2002.

[BOO00] BOOCH, G. *UML, Guia do Usuário*; TRAD de Fábio Freitas da Silva. Rio de Janeiro: Campus, 2000.

[BOR02] BORELLI, Graciele. *O que é normalização de dados*. Disponível em: http://www.unirondon.br/grad/tpd/port_bd/3ano_02/tpd/graciele/curriculum.htm. Acesso em: 18 jan. 2002.

[BOU02] BOURRET, Ronald. *XML and Databases*. Disponível em: http://www.rpbourret.com/xml/XMLAndDatabases.htm . Acesso em: 10 out. 2002.

[CHA01] CHAMPION, M. *Storing XML in Databases*; Eai Jounal. 2001.

[CON99] CONNALEN, J. *Building Web Applications with UML*. Toronto: Addison-Wesley, 1999.

[COX02] COX, John. *Working out the bugs in XML Databases*. Disponível em: http://www.nwfusion.com/news/2002/0107specialfocus.html . Acesso em: 12 out. 2002.

[CYC00] CYCLADES BRASIL *Guia Internet e Conectividade*. São Paulo: SENAC, 2000.

[DEI01] DEITEL, H.M; DEITEL P.J. *Java™ Como Programar*; TRAD de Edson Furnankiewicz. 3. ed. Porto Alegre: Bookman, 2001.

[DYC02] DYCK, T. *Going Native: XML Databases* – PC Magazine. New York City: jun. 2002.

[ECK00] ECKERSDORFF, R. *Chat – A Atual fronteira do atendimento online*. Disponível em: http://www.callcenter.inf.br/materias-conteudo.asp?coddocumento=21 . Acesso em: 16 out. 2002

[HEI01] HEITLINGER, Paulo. *O Guia Prático da XML*. Lisboa: Centro Atlântico, 2001.

[LON00] LONEY, K. *Oracle 8i: O Manual do DBA*; TRAD de Kátia Roque. Rio de Janeiro: Campus, 2000.

[JEN97] JENSEN, Bob. *Technology Glossary*. Disponível em: http://www.trinity.edu/~rjensen/245glosf.htm#M-Terms. Acesso em: 07 jan. 2003.

[LIO02] LIOTTA, Matt. *Apache's Xindice Organizes XML Data Without Schema*. Disponível em: http://www.devx.com/xml/Article/9796/1954. Acesso em: 18 dez. 2002.

[STA01] STAKEN, Kimbro. *Introduction to XML Databases*. Disponível em: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html. Acesso em: 19 out. 2002.

[SUN00] SUN, M. *Java™ Remote Method Invocation (RMI)*. Disponível em: http://java.sun.com/j2se/1.4/docs/guide/rmi/ . Acesso em 17 ago. 2002.

[TAM02] TAMINO XML SERVER. *Tamino XML Server Home Page*. Disponível em: http://www.softwareag.com/tamino/ . Acesso em 18 ago. 2003.

[XDB02] XML:DB INITIATIVE FOR XML DATABASES. *XML:DB Frequently Asked Questions*. Disponível em: http://www.xmldb.org/faqs.html . Acesso em: 27 set. 2002.

[XEN02] WORLD WIDE WEB CONSORTIUM. *XML Encryption Syntax and Processing*. Disponível em: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/ . Acesso em: 10 dez. 2002.

[XIN02] THE APACHE SOFTWARE FOUNDATION. *Xindice Frequently Asked Questions*. Disponível em: http://xml.apache.org/xindice/faq.html. Acesso em: 26 set. 2002.

[W3C96] WORLD WIDE WEB CONSORTIUM. *Extensible Markup Language (XML)*. Disponível em: http://www.w3.org/TR/WD-xml-961114.html#sec1. Acesso em: 06 jan. 2003.

ANEXO A – CÓDIGO FONTE

Gerente.java

```
package com.save.client.gui;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.beans.*;
import java.rmi.*;
import javax.swing.Timer;


import com.save.object.*;
import com.save.client.util.*;
import com.save.client.gui.basic.*;
import com.save.server.rmi.*;


public class Gerente extends BasicMDIFrame {

  /**     Título da Aplicação */
  protected static final String TITULO_APP = "Central de Atendimento CAOL";

  /** Ações */
  protected Action acaoConectar;
  protected Action acaoDesconectar;
  protected Action acaoIniciarAtendimento;
  protected Action acaoSair;
  protected Action acaoAtdNovo;
  protected Action acaoAtdEditar;
  protected Action acaoPreConfig;
  private          Action acaoMostrarPagina;

  /** Componentes Visuais */
  private JMenu menuFile;
  private JMenu menuGerencia;
  private JMenu menuGerAtendente;
        private JMenu menuGerPreferencias;
        private JLabel statusbar;
        private FrmSenha frmSenha;
        private FrmAtendente frmAtendente;
        private FrmCadAtendente frmCadAtendente;

        /** Instancia de constante */
```

```
        public Constantes constantes;

        /** Instancia de RMI */
        protected CallcenterOnline servidor;
protected Atendente atendente;

        /** Timer de verificação de mensagens */
        protected Timer timer;

        /** Conjunto de telas */

        public Gerente() {
        constantes = new Constantes();
  configAcoes();
  configMenu();
  configFrame();
        configServidor();
}

public void setVisible(boolean visible) {
  super.setVisible(visible);
  if (visible) {
        desktopManager.showAll();
  }
}

        public void verifiqueMensagens() {
                timer = new Timer(10000,tarefaGetMensagem);
                timer.start();
        }

        ActionListener tarefaGetMensagem =  new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                        try {
                                String mensagem  = servidor.receiveQuestion(atendente);
                                if (mensagem!=null)
                                        crieFrameInterno(frmAtendente,"FrmAtendente");
                        }
                        catch (Exception e){}
                }
        };


public void habilitaBtnConexao() {
                acaoConectar.setEnabled(acaoDesconectar.isEnabled());
                acaoDesconectar.setEnabled(!acaoDesconectar.isEnabled());
        }

        public boolean conecte(String login, String senha){
```

```
                atendente = new Atendente(login,senha);
                try {
                        if (servidor.conecteAtendente(atendente)){
                                habilitaBtnConexao();
                                verifiqueMensagens();
                                statusbar.setText(atendente.getLogin() + " pronto para
atendimento.");
                                return true;
                        }
                        else {
                                return false;
                        }
                }
                catch (Exception e){
    JOptionPane.showMessageDialog(this,constantes.msgServidorInativo,
                                                "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
                        return false;
                }
        }


  private void configServidor() {

   try {
     servidor = (CallcenterOnline)Naming.lookup(constantes.enderecoServidor);
   }
   catch(Exception ce) {
     JOptionPane.showMessageDialog(this,constantes.msgServidorInativo,
                                                        "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
   }
  }


  private void configAcoes() {
    acaoConectar = new DefaultAction("Conectar...", loadIcon("conectar.gif"));
    acaoDesconectar = new DefaultAction("Desconectar...",
loadIcon("desconectar.gif"));
    acaoIniciarAtendimento = new DefaultAction("Iniciar Atendimento");
    acaoMostrarPagina = new DefaultAction("Na Internet...", loadIcon("web.gif"));
    acaoAtdNovo = new DefaultAction("Novo...");
    acaoAtdEditar = new DefaultAction("Editar dados...");
    acaoPreConfig = new DefaultAction("Configurações...");
                acaoSair = new DefaultAction("Sair");
                acaoDesconectar.setEnabled(false);
  }
```

```
private void configMenu() {
  JMenuItem menuItem;
  JMenuBar menuBar = new JMenuBar();

  menuFile = new JMenu("Atendente");
    menuFile.add(acaoConectar).setIcon(loadIcon("open_small.gif"));
    menuFile.addSeparator();
    menuFile.add(acaoDesconectar).setIcon(loadIcon("open_small.gif"));
    menuFile.addSeparator();
    menuFile.add(acaoMostrarPagina);
    menuFile.addSeparator();
    menuFile.add(acaoSair).setIcon(loadIcon("exit.gif"));

  menuGerencia                          = new JMenu("Gerencia");
                menuGerAtendente        = new JMenu("Atendente");
                menuGerPreferencias = new JMenu("Preferências");

                menuGerencia.add(menuGerAtendente);
                menuGerencia.add(menuGerPreferencias);

                menuGerAtendente.add(acaoAtdNovo);
                menuGerAtendente.add(acaoAtdEditar);
  menuGerPreferencias.add(acaoPreConfig);

  menuBar.add(menuFile);
  menuBar.add(menuGerencia);
  menuBar.add(menuWindow);
  setJMenuBar(menuBar);

}

private void configFrame() {
                Container contentPane = getContentPane();
                setTitle(TITULO_APP);
                contentPane.add(configToolbar(),BorderLayout.NORTH);
                setIconImage(loadIcon("logo.gif").getImage());
                statusbar = new JLabel();
                contentPane.add(statusbar, BorderLayout.SOUTH);
                statusbar.setText("Desconectado");
}

private JToolBar configToolbar() {
  JToolBar toolBar = new JToolBar();
  toolBar.add(acaoConectar);
  toolBar.add(acaoDesconectar);
  toolBar.addSeparator();
  toolBar.setFloatable(false);
  return toolBar;
}
```

```
  private void facaMostrarPagina(String urlSpec) {
    String commandLine = null;
    if (System.getProperty("os.name").toLowerCase().startsWith("win")) {
          commandLine = "rundll32.exe url.dll,FileProtocolHandler " + urlSpec;
    }
    else {
          //commandLine = "netscape " + urlSpec;
          commandLine = "kfmclient exec " + urlSpec;
    }
    try {
          Runtime.getRuntime().exec(commandLine);
    }
    catch (IOException ex) {
    }
  }


  protected void crieFrameInterno(BasicInternalFrame frame, String tipo) {
          if (tipo.equals("FrmSenha")){
                frame = new FrmSenha(desktopManager,"Identifique-se");
          }
          else if (tipo.equals("FrmAtendente")){
                frame = new FrmAtendente(desktopManager,"Atendimento - " +
atendente.getNome());
          }
          else if (tipo.equals("FrmCadAtendente")){
                frame = new FrmCadAtendente(desktopManager);
          }
    try {
          frame.setMaximum(false);
    }
    catch (PropertyVetoException ex) {
    }
  }


  private class DefaultAction extends AbstractAction {
    public DefaultAction(String name) {
                        super(name);
    }

    public DefaultAction(String name, Icon icon) {
                        super(name, icon);
    }

    public void actionPerformed(ActionEvent ev) {
      if (this == acaoConectar) {
```

```
        crieFrameInterno(frmSenha,"FrmSenha");
    } else if (this == acaoDesconectar) {
                                        try {
                                                servidor.desconecteAtendente(atendente);
                                                habilitaBtnConexao();
         timer.stop();
                                                statusbar.setText("Desconectado");
                                        }
                                        catch (Exception e) {}
    } else if (this == acaoSair) {
        facaSair();
    } else if (this == acaoAtdNovo) {
        crieFrameInterno(frmSenha,"FrmCadAtendente");
    } else if (this == acaoIniciarAtendimento) {
        crieFrameInterno(frmAtendente,"FrmAtendente");
    } else if (this == acaoMostrarPagina) {
        facaMostrarPagina("http://www.inf.ufsc.br/~mariano");
    }
  }
}



 public static void main(String[] args) {
                try {

        com.incors.plaf.alloy.AlloyLookAndFeel.setProperty("alloy.licenseCode",
"2003/02/08#xyzabcv@mail.com#rd57jm#1986ew");
                        com.incors.plaf.alloy.AlloyTheme theme = new
com.incors.plaf.alloy.themes.glass.GlassTheme();
                        javax.swing.LookAndFeel alloyLnF = new
com.incors.plaf.alloy.AlloyLookAndFeel(theme);
                        javax.swing.UIManager.setLookAndFeel(alloyLnF);

                }
                catch (javax.swing.UnsupportedLookAndFeelException ex) {
                        ex.printStackTrace();
                }

   Gerente frame = new Gerente();
   frame.setVisible(true);

   // JInternalFrame can only be selected in visible JDesktopPane
   JInternalFrame[] frames = frame.desktopPane.getAllFrames();
   if (frames.length > 0) {
     try {
         frames[0].setSelected(true);
     }
     catch (PropertyVetoException ex) {
```

```
    }
   }
  }

}
```

FrmAtendente.java

```
package com.save.client.gui;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;
import java.rmi.*;
import java.awt.Toolkit;

import com.save.object.*;
import com.save.client.util.*;
import com.save.client.gui.basic.*;
import com.save.server.rmi.*;

public class FrmAtendente  extends BasicInternalFrame {

        /** Ações */
  protected Action acaoEnviarMensagem;
  protected Action acaoEncerrar;
  protected Action acaoSupervisor;


        /** Componentes Visuais */
  private JPanel painel                                          = new JPanel();
  private JTextArea txtConversa   = new JTextArea();
  private JTextField txtResposta  = new JTextField();
  private JButton btnEnviar             = new JButton();
  private JButton btnFuncoes            = new JButton();
  private JButton btnSair                       = new JButton();
  private JCheckBox cbPendente        = new JCheckBox();
  private JLabel lbResposta             = new JLabel();

        /** Instâncias dos meus objetos */
        private Mensagem msgEnviada;
        private String msgRecebida;
        private String login;
        private Timer timer;
```

```
        /**
          Variável que armazena a mensagem que será
          apresentada na tela em caso de erro ou aviso
        */
        String msgAviso;

public FrmAtendente(BasicDesktopManager desktopManager, String titulo) {
  super(desktopManager,titulo);
  configComponentes();
  configAcoes();
  configFrame();
  configObjetos();

  VerifiqueMensagens(10);

}


private void configAcoes() {
  acaoEncerrar = new Acao("Encerrar", loadIcon("exit.gif"));
  acaoSupervisor = new Acao("Supervisor",loadIcon("supervisor.gif"));
          acaoEnviarMensagem = new Acao("Enviar", loadIcon("enviar.gif"));

        btnSair.setAction(acaoEncerrar);
        //btnFuncoes.setAction(acaoSupervisor);
        btnEnviar.setAction(acaoEnviarMensagem);
}

private void configObjetos() {
                msgEnviada    = new Mensagem("","","");
                //msgRecebida           = new Mensagem("","","");
                msgRecebida  = "";
                login = getGerente().atendente.getLogin();
        }

private void configComponentes() {

                /** Componentes de texto */
          txtConversa.setBorder(BorderFactory.createLoweredBevelBorder());
          txtConversa.setLineWrap(true);
          txtConversa.setBounds(new Rectangle(25, 78, 354, 174));
          txtResposta.setBounds(new Rectangle(25, 297, 357, 22));

          /** Botões*/
          btnEnviar.setBounds(new Rectangle(25, 323, 100, 20));
          btnSair.setBounds(new Rectangle(130, 323, 120, 20));
          btnFuncoes.setBounds(new Rectangle(255, 323, 120, 20));
```

```
        /** CheckBox */
        cbPendente.setText("Marcar como pendente");
        cbPendente.setBounds(new Rectangle(25, 59, 160, 19));

        /** Labels */
        lbResposta.setText("Resposta a enviar:");
        lbResposta.setBounds(new Rectangle(26, 281, 150, 15));
}



protected void configFrame() {
  Container contentPane = getContentPane();
  contentPane.setLayout(null);
       contentPane.setBounds(0, 0, 250, 180);

              contentPane.add(btnEnviar);
        contentPane.add(btnSair);
        //contentPane.add(btnFuncoes);
        contentPane.add(txtResposta);
        contentPane.add(txtConversa);
        contentPane.add(lbResposta);
        contentPane.add(cbPendente);

  super.setupInternalFrame(new Rectangle(10, 10, 600, 400));

       }


public void VerifiqueMensagens(int pSegundos) {
  timer = new Timer(pSegundos*1000,tarefaGetMensagem);
              timer.start();
}


ActionListener tarefaGetMensagem =       new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
       try {
       msgRecebida = getGerente().servidor.receiveQuestion(getGerente().atendente);
                          if (msgRecebida!=null && !msgRecebida.equals("")) {
       txtConversa.append(msgRecebida + '\n');
                                   msgRecebida="";
                          }
    }
    catch(java.rmi.ConnectException ce) {
        msgAviso = getGerente().constantes.msgServidorInativo;
      System.err.println(msgAviso);
    }
    catch(Exception e) {
```

```
      System.err.println("Exceção no método tarefaGetMensagem: " + e.getMessage());
    }
  }
 };


 public void activate() {
  desktopManager.getDesktopPane().setSelectedFrame(this);
 }


 private Gerente getGerente() {
        return (Gerente)desktopManager.getParentFrame();
 }


        public void enviaResposta() {
                String strMensagem = txtResposta.getText();
  if (!strMensagem.trim().equals("")) {
           try {
             msgEnviada.setMensagem(strMensagem);
             msgEnviada.setRemetente(login);
             msgEnviada.setReceptor("14532");
             getGerente().servidor.sendAnswer(msgEnviada);
             txtConversa.append(msgEnviada.toString() + '\n');
             txtResposta.setText("");
           }
           catch(java.rmi.ConnectException ce) {
                                msgAviso = "O servidor de mensagens está inativo!";
            JOptionPane.showMessageDialog(this,msgAviso,"Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
           }
           catch(Exception e) {
                                msgAviso = "Ocorreu um erro ao enviar sua mensagem!";
            JOptionPane.showMessageDialog(this,msgAviso,"Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
           }
  }
  else {
                                msgAviso = "Digite a mensagem antes de enviá-la!";
                                JOptionPane.showMessageDialog(this,msgAviso,"Leia a
mensagem",JOptionPane.INFORMATION_MESSAGE);
  }
 }


 /** Classe Acao */
 private class Acao extends AbstractAction {
```

```
   public Acao(String name) {
                        super(name);
   }

   public Acao(String name, Icon icon) {
                        super(name, icon);
   }

   public void actionPerformed(ActionEvent ev) {
     if (this == acaoEnviarMensagem){
        enviaResposta();
     } else if (this == acaoEncerrar) {
                                          timer.stop();
        facaSair();;
     }
   }
  }



}
```

FrmCadAtendente.java

```
package com.save.client.gui;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

import com.save.object.*;
import com.save.client.gui.basic.*;

public class FrmCadAtendente extends BasicInternalFrame {

        /** Ações */
  protected Action acaoCadastrar;
  protected Action acaoCancelar;


        /** Componentes Visuais */
        JLabel lbLogin          = new JLabel();
        JLabel lbSenha          = new JLabel();
        JLabel lbNome           = new JLabel();
        JTextField txtNome = new JTextField();
```

```
        JTextField txtLogin = new JTextField();
        JTextField txtSenha = new JPasswordField();
        JButton btnCadastrar= new JButton();
        JButton btnCancelar = new JButton();


public FrmCadAtendente(BasicDesktopManager desktopManager) {
  super(desktopManager,"Cadastro de Atendente");
  configComponentes();
  configAcoes();
  configFrame();
}


public void activate() {
  desktopManager.getDesktopPane().setSelectedFrame(this);
}


private void configAcoes() {
  acaoCadastrar = new Acao("Cadastrar");
          acaoCancelar = new Acao("Cancelar");
        btnCadastrar.setAction(acaoCadastrar);
        btnCancelar.setAction(acaoCancelar);
}

private void configComponentes() {
        /* Labels */
        lbNome.setBounds(new Rectangle(10, 10, 100, 15));
          lbLogin.setBounds(new Rectangle(10, 55, 100, 15));
          lbSenha.setBounds(new Rectangle(10, 100, 100, 15));
          lbNome.setText("Nome:");
          lbLogin.setText("Login:");
          lbSenha.setText("Senha:");

          /* Componentes de texto */
          txtNome.setBounds(new Rectangle(10, 25, 200, 22));
          txtLogin.setBounds(new Rectangle(10, 70, 100, 22));
          txtSenha.setBounds(new Rectangle(10, 115, 100, 22));

          /* Botões */
          btnCadastrar.setBounds(new Rectangle(10, 145, 90, 20));
          btnCancelar.setBounds(new Rectangle(135, 145, 90, 20));

}


protected void configFrame() {
  Container contentPane = getContentPane();
```

```
   contentPane.setLayout(null);
          contentPane.setBounds(0, 0, 300, 220);

          contentPane.add(lbNome);
          contentPane.add(lbLogin);
          contentPane.add(lbSenha);
          contentPane.add(txtNome);
          contentPane.add(txtLogin);
          contentPane.add(txtSenha);
                contentPane.add(btnCadastrar);
          contentPane.add(btnCancelar);

   super.setupInternalFrame(new Rectangle(5,5,300,220));

  }

  public void cadastre() {
         if (txtNome.getText().equals("")) {
     JOptionPane.showMessageDialog(this,"Preencha o nome antes de continuar",
                                                      "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
         }
         else if (txtLogin.getText().equals("")) {
     JOptionPane.showMessageDialog(this,"Preencha o login antes de continuar",
                                                      "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
         }
         else if (txtSenha.getText().equals("")) {
     JOptionPane.showMessageDialog(this,"Preencha a senha antes de continuar",
                                                      "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
         }
           else {
                        try {
                              Atendente atd = new
Atendente(txtNome.getText(),txtLogin.getText(),txtSenha.getText());
              if (getParentFrame().servidor.cadastreAtendente(atd)){
                                    txtNome.setText("");
                                    txtLogin.setText("");
                                    txtSenha.setText("");
                  JOptionPane.showMessageDialog(this,"Atendente cadastrado com
sucesso!",
                                                      "Leia a
mensagem",JOptionPane.INFORMATION_MESSAGE);
              }
                              else {
                  JOptionPane.showMessageDialog(this,"Atendente não cadastrado!",
                                                      "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
```

```
                                    }
                        }
                    catch(Exception e) {
                                    System.out.println(e.getMessage());
                JOptionPane.showMessageDialog(this,"A conexão com o servidor não foi
estabelecida!",
                                                                "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
                    }
                        }
  }



  private Gerente getParentFrame() {
        return (Gerente)desktopManager.getParentFrame();
  }


  /* Classe Acao */
  private class Acao extends AbstractAction {

    public Acao(String name) {
                        super(name);
    }

    public Acao(String name, Icon icon) {
                        super(name, icon);
    }

    public void actionPerformed(ActionEvent ev) {
      if (this == acaoCadastrar){
         cadastre();
      } else if (this == acaoCancelar) {
          facaSair();
      }
    }
  }

}
```

FrmSenha.java

```
package com.save.client.gui;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

import com.save.object.*;
import com.save.client.gui.basic.*;

public class FrmSenha extends BasicInternalFrame {

        /** Ações */
  protected Action acaoConectar;
  protected Action acaoCancelar;


        /** Componentes Visuais */
        JLabel lbLogin                  = new JLabel();
        JLabel lbSenha                  = new JLabel();
        JTextField txtLogin = new JTextField();
        JTextField txtSenha = new JPasswordField();
        JButton btnConectar  = new JButton();
        JButton btnCancelar = new JButton();
        FrmAtendente frmAtendente;

  public FrmSenha(BasicDesktopManager desktopManager, String titulo) {
   super(desktopManager,titulo);
   configComponentes();
   configAcoes();
   configFrame();
  }


  public void activate() {
   desktopManager.getDesktopPane().setSelectedFrame(this);
  }


  private void configAcoes() {
   acaoConectar = new Acao("Conectar");
           acaoCancelar = new Acao("Cancelar");
         btnConectar.setAction(acaoConectar);
         btnCancelar.setAction(acaoCancelar);
  }
```

```
 private void configComponentes() {
        /* Labels */
          lbLogin.setBounds(new Rectangle(10, 10, 100, 15));
          lbSenha.setBounds(new Rectangle(10, 55, 100, 15));
          lbLogin.setText("Login:");
          lbSenha.setText("Senha:");

          /* Componentes de texto */
          txtLogin.setBounds(new Rectangle(10, 25, 100, 22));
          txtSenha.setBounds(new Rectangle(10, 70, 100, 22));

          /* Botões */
          btnConectar.setBounds(new Rectangle(10, 100, 90, 20));
          btnCancelar.setBounds(new Rectangle(135, 100, 90, 20));

 }


 protected void configFrame() {
  Container contentPane = getContentPane();
  contentPane.setLayout(null);
          contentPane.setBounds(0, 0, 250, 180);

                  contentPane.add(btnConectar);
          contentPane.add(btnCancelar);
          contentPane.add(txtLogin);
          contentPane.add(txtSenha);
          contentPane.add(lbLogin);
          contentPane.add(lbSenha);

  super.setupInternalFrame(new Rectangle(5,5,240,180));

 }

 public void conecte() {
        if (txtLogin.getText().equals("")) {
    JOptionPane.showMessageDialog(this,"Preencha o login antes de continuar",
                                                    "Leia a
mensagem",JOptionPane.INFORMATION_MESSAGE);
        }
        else if (txtSenha.getText().equals("")) {
    JOptionPane.showMessageDialog(this,"Preencha a senha antes de continuar",
                                                    "Leia a
mensagem",JOptionPane.INFORMATION_MESSAGE);
        }
          else {
                                  if
(getGerente().conecte(txtLogin.getText(),txtSenha.getText())){
                  facaSair();
```

```
            }
                                else {
                    JOptionPane.showMessageDialog(this,"Login Inválido",
                                                        "Leia a
mensagem",JOptionPane.ERROR_MESSAGE);
                                }
                    }
  }



  private Gerente getGerente() {
          return (Gerente)desktopManager.getParentFrame();
  }


  /* Classe Acao */
  private class Acao extends AbstractAction {

    public Acao(String name) {
                            super(name);
    }

    public Acao(String name, Icon icon) {
                            super(name, icon);
    }

    public void actionPerformed(ActionEvent ev) {
      if (this == acaoConectar){
          conecte();
      } else if (this == acaoCancelar) {
           facaSair();
      }
    }
  }



}
```

BasicDesktopManger.java

```
package com.save.client.gui.basic;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Rectangle;
import java.beans.*;
import java.util.*;

public class BasicDesktopManager extends DefaultDesktopManager
                        implements VetoableChangeListener,
                                InternalFrameListener {

   /* Padrões de deslocamento do eixo X e Y do Frame Interno */
   protected static int NEW_INTERNAL_X_OFFSET = 22;
   protected static int NEW_INTERNAL_Y_OFFSET = 22;

   /* Padrões de largura e altura */
   protected static int NEW_INTERNAL_WIDTH = 600;
   protected static int NEW_INTERNAL_HEIGHT = 400;


   protected BasicMDIFrame parentFrame;
   protected JDesktopPane desktopPane;

   protected int newInternalX = 0;
   protected int newInternalY = 0;
   /* Rollover counter for y offsets */
   protected int rollover = 0;


   /*
   - Variável que sabe se um frame está sendo maximizado ou não
   - Mapa que vincula os frames aberto com os item do Menu Janela
   - Lista dos frames internos abertos
   - Menu index fo separator in the window menu
   - Índice do frame que será mostrado quando ShowAll() for executado  */
   protected boolean maximizationInProgress;
   protected HashMap frameToMenuItem = new HashMap();
   protected LinkedList openFrames = new LinkedList();
   protected int separatorMenuIndex = -1;
   protected int activeFrameIndex = -1;


   /* Construtor */
   public BasicDesktopManager(BasicMDIFrame parentFrame) {
```

```
  this.parentFrame = parentFrame;
  desktopPane = parentFrame.desktopPane;
}

/* Retorna o Frame-pai */
public BasicMDIFrame getParentFrame() {
    return parentFrame;
}

/* Retorna o DesktopPane */
public JDesktopPane getDesktopPane() {
    return desktopPane;
}

/* Retorna uma lista de frames abertos */
public java.util.List getOpenFrames() {
    return openFrames;
}

/* Retorna um retângulo que serve como os limites de um frame interno */
public Rectangle getNextInternalFrameBounds() {
  if (newInternalY + NEW_INTERNAL_HEIGHT > desktopPane.getHeight()) {
    rollover++;
    newInternalY = 0;
    newInternalX = rollover * NEW_INTERNAL_X_OFFSET;
  }

  Rectangle nextBounds = new Rectangle(newInternalX,newInternalY,
                          NEW_INTERNAL_WIDTH,NEW_INTERNAL_HEIGHT);

  newInternalX += NEW_INTERNAL_X_OFFSET;
  newInternalY += NEW_INTERNAL_Y_OFFSET;

  return nextBounds;
}

/* Seta o índice do frame ativo */
public void setActiveFrameIndex(int activeFrameIndex) {
  this.activeFrameIndex = activeFrameIndex;
}

/* Mostra todos os frames */
public void showAll() {
  Iterator it = openFrames.iterator();
  while (it.hasNext()) {
                                ((BasicInternalFrame)it.next()).setVisible(true);
  }
  if (activeFrameIndex > -1) {
```

```
        JInternalFrame activeFrame =
(JInternalFrame)openFrames.get(activeFrameIndex);
        try {
            activeFrame.setSelected(true);
        } catch (PropertyVetoException ex) {
        }
      }
    }

    /* Adiciona um frame interno no Desktop */
    public void addInternalFrame(JInternalFrame frame) {
      if (frameToMenuItem.size() == 0) {
        separatorMenuIndex = parentFrame.menuWindow.getMenuComponentCount();
        parentFrame.menuWindow.addSeparator();
      }
      Action action = new WindowActivateAction(frame);
      JMenuItem menuItem = parentFrame.menuWindow.add(action);

      desktopPane.add(frame);
      frameToMenuItem.put(frame, menuItem);
      openFrames.add(frame);
      setWindowActionsEnabled(true);
    }


    /* Adiciona um frame interno no Desktop */
    public void removeInternalFrame(JInternalFrame frame) {
      JMenuItem menuItem = (JMenuItem)frameToMenuItem.remove(frame);
      if (menuItem != null) {
        parentFrame.menuWindow.remove(menuItem);
        openFrames.remove(frame);
        if (frameToMenuItem.size() == 0 && separatorMenuIndex > -1) {
          parentFrame.menuWindow.remove(separatorMenuIndex);
          separatorMenuIndex = -1;
          setWindowActionsEnabled(false);
        }
      }
    }


    /* Mostra o próximo frame aberto */
    public void cycleToNextWindow() {
        cycleWindows(true);
    }


                /* Mostra o frame aberto anterior */
    public void cycleToPreviousWindow() {
        cycleWindows(false);
```

```
}

/* Arranja os frames lado a lado */
public void tileWindows() {

  int framesCount = openFrames.size();
  if (framesCount == 0) {
    return;
  }

  int sqrt = (int)Math.sqrt(framesCount);
  int rows = sqrt;
  int cols = sqrt;
  if (rows * cols < framesCount) {
    cols++;
    if (rows * cols < framesCount) {
                                          rows++;
    }
  }

  Dimension size = desktopPane.getSize();

  int width = size.width/cols;
  int height = size.height/rows;
  int offsetX = 0;
  int offsetY = 0;

  JInternalFrame currentFrame;
  Iterator it = openFrames.iterator();
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols && (i * cols + j < framesCount); j++) {
      currentFrame = (JInternalFrame)it.next();
      normalizeFrame(currentFrame);
      resizeFrame(currentFrame, offsetX, offsetY, width, height);
      offsetX += width;
    }
    offsetX = 0;
    offsetY += height;
  }
}

/* Arranja os frames em cascata */
public void stackWindows() {
  newInternalX = newInternalY = rollover = 0;
  Rectangle currentBounds;
  JInternalFrame currentFrame;
  Iterator it = openFrames.iterator();
  while (it.hasNext()) {
    currentFrame = (JInternalFrame)it.next();
```

```
        normalizeFrame(currentFrame);
        currentBounds = getNextInternalFrameBounds();
        resizeFrame(currentFrame, currentBounds.x,currentBounds.y,
                currentBounds.width, currentBounds.height);
        try {
           currentFrame.setSelected(true);
        }
        catch (PropertyVetoException ex) { }
      }
    }

    private void normalizeFrame(JInternalFrame frame) {
      try {
        if (frame.isIcon()) {
          frame.setIcon(false);
        }
        if (frame.isMaximum()) {
           frame.setMaximum(false);
        }
      } catch (PropertyVetoException ex) {}
    }

    private void cycleWindows(boolean forward) {
      JInternalFrame currentFrame = desktopPane.getSelectedFrame();
      JInternalFrame nextFrame = null;

      ListIterator it = openFrames.listIterator();
      while (it.hasNext() && it.next() != currentFrame) {
      }
      if (forward) {
        if (it.hasNext())
          nextFrame = (JInternalFrame)it.next();
        else
                                              nextFrame =
(JInternalFrame)openFrames.getFirst();
      }
      else {
        if (it.hasPrevious() && it.previous() != null && it.hasPrevious())
          nextFrame = (JInternalFrame)it.previous();
        else
           nextFrame = (JInternalFrame)openFrames.getLast();
      }

      try {
        if (nextFrame.isIcon())
           nextFrame.setIcon(false);
        nextFrame.setSelected(true);
      }
      catch (PropertyVetoException ex) {}
```

```
}

private void setWindowActionsEnabled(boolean enabled) {
  parentFrame.actionNextWindow.setEnabled(enabled);
  parentFrame.actionPreviousWindow.setEnabled(enabled);
  parentFrame.actionTileWindows.setEnabled(enabled);
  parentFrame.actionStackWindows.setEnabled(enabled);
}

private void maximizeAllFrames(JInternalFrame source, boolean isMaximum) {
  synchronized (this) {
    if (maximizationInProgress)
        return;
    maximizationInProgress = true;
  }

  try {
    JInternalFrame[] frames = desktopPane.getAllFrames();
    for (int i = 0; i < frames.length; i++) {
      if (frames[i] == source)
        continue;
      try {
          frames[i].setMaximum(isMaximum);
      }
      catch (PropertyVetoException ex) {}
    }
  }
  finally {
    maximizationInProgress = false;
  }
}

public void vetoableChange(PropertyChangeEvent changeEvent)
  throws PropertyVetoException {

  String eventName = changeEvent.getPropertyName();

  if (JInternalFrame.IS_MAXIMUM_PROPERTY.equals(eventName)) {
    if (maximizationInProgress) {
      return;
    }

    boolean isMaximum = ((Boolean)changeEvent.getNewValue()).booleanValue();
    JInternalFrame source = (JInternalFrame)changeEvent.getSource();
    maximizeAllFrames(source, isMaximum);
  }
}

public void internalFrameDeiconified(InternalFrameEvent event) {
```

```
    }

    public void internalFrameOpened(InternalFrameEvent event) {
    }

    public void internalFrameIconified(InternalFrameEvent event) {
    }

    public void internalFrameClosing(InternalFrameEvent event) {
      JInternalFrame frame = event.getInternalFrame();
      removeInternalFrame(frame);
    }

    public void internalFrameActivated(InternalFrameEvent event) {
    }

    public void internalFrameDeactivated(InternalFrameEvent event) {
    }

    public void internalFrameClosed(InternalFrameEvent event) {
      parentFrame.desktopPane.remove(event.getInternalFrame());
    }


    /* Classe WindowActivateAction */
    private class WindowActivateAction extends AbstractAction {

      private JInternalFrame frame;

      public WindowActivateAction(JInternalFrame frame) {
         super(frame.getTitle());
         this.frame = frame;
      }

      public void actionPerformed(ActionEvent ev) {
         try {
           if (frame.isIcon())
               frame.setIcon(false);
           frame.setSelected(true);
         }
         catch (PropertyVetoException ex) {}
      }

    }

}
```

BasicMDIFrame.java

```
package com.save.client.gui.basic;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import java.io.*;
import java.net.URL;
import java.util.*;
import java.beans.*;
import java.lang.reflect.*;


public class BasicMDIFrame extends JFrame {

   public static String DIR_IMAGEM;
   protected static int LARGURA_PADRAO;
   protected static int ALTURA_PADRAO;


   /* Optional path where images are found */
   protected String imagePath;

   /* Ações */
   protected Action actionNextWindow;
   protected Action actionPreviousWindow;
   protected Action actionTileWindows;
   protected Action actionStackWindows;

   /* Componentes visuais */
   protected JDesktopPane desktopPane;
   protected BasicDesktopManager desktopManager;
   protected JMenu menuWindow;


   public BasicMDIFrame() {
         loadSettings();
     setupImagePath();
     setupActions();
     setupMenu();
     setupFrame();
   }

   /* Cria um DesktopManager */
   protected BasicDesktopManager createDesktopManager() {
                        return new BasicDesktopManager(this);
   }
```

```
    /* Sair da aplicação */
    protected void facaSair() {
        dispose();
        System.exit(0);
    }

    /* Carrega um arquivo de imagem do diretório DIR_IMAGEM */
    protected ImageIcon loadIcon(String fileName) {
                            URL imageURL = getClass().getResource("/" + DIR_IMAGEM
+ "/" + fileName);
                            return new ImageIcon(imageURL);
    }


    /* Seta as configurações de acordo com o arquivo config.xml */
    protected void loadSettings(/*String filename*/) {
                    DIR_IMAGEM = "images";
                    LARGURA_PADRAO = 800;
                    ALTURA_PADRAO = 600;


      setSize(LARGURA_PADRAO, ALTURA_PADRAO);

      /*Properties props = new Properties();
      InputStream in = null;
      try {
        in = new BufferedInputStream(new FileInputStream(new File(filename)));
        props.load(in);
      }
      catch (IOException ex) {
                                    props = null;
      }
      finally {
        if (in != null) {
          try {
              in.close();
          }
          catch (IOException ex2) {}
        }
      }
      return props;*/
    }


    private void setupImagePath() {
          imagePath = DIR_IMAGEM + File.separator;
    }
```

```
private void setupActions() {
  actionNextWindow = new WindowAction("Próxima Janela");
  actionNextWindow.setEnabled(false);

  actionPreviousWindow = new WindowAction("Janela Anterior");
  actionPreviousWindow.setEnabled(false);

  actionTileWindows = new WindowAction("Lado a Lado");
  actionTileWindows.setEnabled(false);

  actionStackWindows = new WindowAction("Em cascata");
       actionStackWindows.setEnabled(false);
}

private void setupMenu() {
  menuWindow = new JMenu("Janela");
  menuWindow.add(actionPreviousWindow).setAccelerator(
     KeyStroke.getKeyStroke(KeyEvent.VK_F2, Event.CTRL_MASK));
  menuWindow.add(actionNextWindow).setAccelerator(
     KeyStroke.getKeyStroke(KeyEvent.VK_F3, Event.CTRL_MASK));
  menuWindow.add(actionTileWindows);
  menuWindow.add(actionStackWindows);
}

private void setupFrame() {
  setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

  Container contentPane = getContentPane();
  contentPane.setLayout(new BorderLayout(5,5));
  contentPane.add(buildDesktopPane(), BorderLayout.CENTER);

  addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent event) {
       facaSair();
     }
  });
}

private JDesktopPane buildDesktopPane() {
  desktopPane = new JDesktopPane();
  desktopManager = createDesktopManager();
  desktopPane.setDesktopManager(desktopManager);
  desktopPane.setBackground(new Color(56, 112, 168));
  return desktopPane;
}

private class WindowAction extends AbstractAction {

  public WindowAction(String name) {
```

```
        super(name);
    }

    public WindowAction(String name, Icon icon) {
        super(name, icon);
    }

    public void actionPerformed(ActionEvent ev) {
      if (this == actionPreviousWindow) {
        desktopManager.cycleToPreviousWindow();
      } else if (this == actionNextWindow) {
        desktopManager.cycleToNextWindow();
      } else if (this == actionTileWindows) {
        desktopManager.tileWindows();
      } else if (this == actionStackWindows) {
        desktopManager.stackWindows();
      }
    }
  }

}
```

BasicInternalFrame.java

```
package com.save.client.gui.basic;

import com.save.client.gui.event.*;
import java.awt.Rectangle;
import javax.swing.*;
import java.beans.*;
import java.util.*;
import java.net.URL;


public class BasicInternalFrame extends JInternalFrame {


   protected final BasicDesktopManager desktopManager;
   protected LinkedList listeners = new LinkedList();

   public BasicInternalFrame(BasicDesktopManager desktopManager, String title) {
      super(title, true, true, true, true);
      this.desktopManager = desktopManager;
  }


   public void addMaximizedListener(MaximizedListener listener) {
      listeners.add(listener);
```

```
    }

    public void removeMaximizedListener(MaximizedListener listener) {
        listeners.remove(listener);
    }

    public void setMaximum(boolean maximum) throws PropertyVetoException {
      super.setMaximum(maximum);

      if (maximum) {
        Iterator it = listeners.iterator();
        MaximizedEvent event = new MaximizedEvent(this);
        while (it.hasNext()) {
            MaximizedListener listener = (MaximizedListener)it.next();
            listener.frameMaximized(event);
        }
      }
    }

    /* Carrega um arquivo de imagem do diretório DIR_IMAGEM */
    protected ImageIcon loadIcon(String fileName) {
                            URL imageURL = getClass().getResource("/images/" +
fileName);
                            return new ImageIcon(imageURL);
    }

    /* Sair da aplicação */
    protected void facaSair() {
          dispose();
                  desktopManager.removeInternalFrame(this);
    }

    protected void setupInternalFrame(Rectangle dimensoes) {

      //setBounds(desktopManager.getNextInternalFrameBounds());
      setBounds(dimensoes);

      addVetoableChangeListener(desktopManager);
      addInternalFrameListener(desktopManager);
      desktopManager.addInternalFrame(this);

      if (desktopManager.getParentFrame().isVisible()) {
        setVisible(true);
      }
    }

}
```

Constantes.java

```
package com.save.client.util;

import java.io.IOException;
import nu.xom.Builder;
import nu.xom.Comment;
import nu.xom.Document;
import nu.xom.Element;
import nu.xom.Node;
import nu.xom.ParseException;

public class Constantes {

        public static String msgLoginInvalido;
        public static String enderecoServidor;
        public static String msgServidorInativo;


 public Constantes() {
  try {
    Builder parser = new Builder();
    String xml = System.getProperty("user.dir") +
                                              System.getProperty("file.separator") +
"config.xml";
    Document doc = parser.build(xml);
    list(doc);
  }
  catch (ParseException ex) {
    System.out.println("Arquivo de configuração  mal formado.");
    System.exit(1);
  }
  catch (IOException ex) {
    System.out.println("O Parser não conseguiu ler o arquivo de configuração.");
    System.exit(1);
  }
 }


 public static void list(Node node) {
  for (int i = 0; i < node.getChildCount(); i++) {
    Node child = node.getChild(i);

    if (child instanceof Element) {
        String valor = child.toString();
        if (valor.indexOf("msgLoginInvalido")>0){
                msgLoginInvalido = child.getValue();
                //System.out.println("Login: " + child.getValue());
```

```
            }
            else if (valor.indexOf("enderecoServidor")>0) {
                    enderecoServidor = child.getValue();
                    //System.out.println("Endereco: " + child.getValue());
            }
            else if (valor.indexOf("msgServidorInativo")>0){
                    msgServidorInativo = child.getValue();
            }
        }
        list(child);
      }
    }


}
```

Pessoa.java

```
package com.save.object;

import java.io.*;
import java.util.*;


public class Pessoa implements Serializable {

        private String nome;
        private String sexo;


        /** Construtor sem parâmetros  */
        public Pessoa(){
                this("","");
        }

        /** Construtor com os parâmetros Nome e Sexo */
        public Pessoa(String pNome, String pSexo){
                //setNome(pNome);
                setSexo(pSexo);
        }

        /** Método que seta o nome da Pessoas */
        public void setNome(String pNome) {
                nome = pNome;
        }

        /** Método que seta o sexo da Pessoas */
        public void setSexo(String pSexo)    {
```

```
            sexo = pSexo;
        }

        /** Função que fornece o nome da Pessoas */
        public String getNome(){
            return nome;
        }

        /** Função que fornece o sexo da Pessoas */
        public String getSexo(){
            return sexo;
        }


}
```

Vestibulando.java

```
package com.save.object;

import java.io.*;

public class Vestibulando extends Pessoa implements Serializable{

        private String rg;
        private String inscricao;

        /** Construtor da classe Vestibulando */
        public Vestibulando(){
        }

        /** Construtor da classe Vestibulando */
        public Vestibulando(String pRg, String pInscricao){
            setRg(pRg);
            setInscricao(pInscricao);
        }


        /** Método que seta o RG do vestibulando */
        public void setRg(String pRg)          {
            rg = pRg;
        }

        /** Método que seta a inscrição do vestibulando */
        public void setInscricao(String pInscricao)   {
            inscricao = pInscricao;
        }
```

```
	/** Função que fornece o Rg do vestibulando */
	public String getRg(){
		return rg;
	}

	/** Função que fornece a inscrição do vestibulando */
	public String getInscricao(){
		return inscricao;
	}

 /**
	Método emXML retorna as informações
	do Vestibulando em formato XML
	**/
	public String emXML(){
		StringBuffer sb = new StringBuffer();
		sb.append( "<vestibulando>\n" );
		sb.append( "\t<nome>"+getNome()+"</nome>\n" );
		sb.append( "\t<rg>"+rg+"</rg>\n" );
		sb.append( "\t<inscricao>"+inscricao+"</inscricao>\n" );
		sb.append( "</vestibulando>\n" );
		return sb.toString();
	}

}
```

Atendente.java

```
package com.save.object;

import java.io.*;
import java.util.*;


public class Atendente extends Pessoa implements Serializable{

	private String login;
	private String senha;
	private int chamadas;                                   /* qtde de chamadas
atendidas no momento*/
	private int limiteChamadas = 4; /* qtde máxima de chamadas que pode atender
*/
	private boolean logado;                                 /* True=Logado;
	False=Inativo */
	private boolean disponivel;             /* True=Disponível; False=Ocupado */

	/** Construtor da classe Atendente  */
	public Atendente (){
```

```
        inicie();
}

/** Construtor da classe Atendente com os parâmetros Login e Senha */
public Atendente (String pLogin, String pSenha){
        setLogin(pLogin);
        setSenha(pSenha);
        inicie();
}

/** Construtor da classe Atendente com os parâmetros Nome, Login e Senha */
public Atendente (String pNome, String pLogin, String pSenha){
        setNome(pNome);
        setLogin(pLogin);
        setSenha(pSenha);
        inicie();
}

/** Método que faz parte dos construtores */
public void inicie(){
        logado                  = false;
        disponivel      = false;
        chamadas                = 0;
}


/** Método que seta o login do Atendente */
public void setLogin(String pLogin) {
        login = pLogin;
}

/** Método que seta o senha do Atendente */
public void setSenha(String pSenha) {
        senha = pSenha;
}

/** Função que fornece o login do Atendente */
public String getLogin(){
        return login;
}

/** Função que fornece a senha do Atendente */
public String getSenha(){
        return senha;
}

/**
Função que fornece o número de Vestibulandos
que o Atendente está ajudando
```

```
*/
public int getChamadas(){
	return chamadas;
}

/** Método */
public void facaLogin()	{
	logado = true;
	altereDisponibilidade(true);
}

/** Método */
public void facaLogout()	{
	logado = false;
	altereDisponibilidade(false);
}

/**
	Método que torna o Atendente disponível
	para atender ou não
*/
public void altereDisponibilidade(boolean pEstado){
	disponivel = pEstado;
}

/**
	Método que aumneta o número de chamadas
	quando um atendimento é iniciado
*/
public void incrementaChamadas()	{
	chamadas+=1;
	if (chamadas == limiteChamadas){
		altereDisponibilidade(false);
	}
}

/**
	Método que diminui o número de chamadas
	quando um atendimento é encerrado
*/
public void decrementaChamadas()	{
	if (chamadas > 0)
		chamadas-=1;
	altereDisponibilidade(true);
}

/** Função que determina se o Atendente está conectado */
public boolean estaLogado(){
	if (logado){
```

```
                return true;
            }
            return false;
        }

        /**
            Função que determina se o Atendente pode
            ajudar o vestibulando ou não
        */
        public boolean podeAtender()        {
            if (disponivel){
                return true;
            }
            return false;
        }

    /**
    Método emXML retorna as informações
    do Atendente em formato XML
    **/
    public String emXML(){
        StringBuffer sb = new StringBuffer();
            sb.append( "<atendente>\n" );
            sb.append( "\t<nome>"+getNome()+"</nome>\n" );
            sb.append( "\t<login>"+login+"</login>\n" );
            sb.append( "\t<senha>"+senha+"</senha>\n" );
            sb.append( "\t<logado>"+logado+"</logado>\n" );
            sb.append( "\t<disponivel>"+disponivel+"</disponivel>\n" );
            sb.append( "</atendente>\n" );
            return sb.toString();
        }

}
```

Mensagem.java

```
package com.save.object;

import java.io.*;

public class Mensagem implements Serializable {

        private String remetente;
        private String receptor;
        private String corpo;
        private String tipo;

        /** construtor da classe sem atributos */
```

```
public Mensagem() {
}

/** construtor da classe com todos seus atributos */
public Mensagem( String pRemetente, String pCorpo, String pReceptor ) {
  setRemetente(pRemetente);
  setReceptor(pReceptor);
  setCorpo(pCorpo);
}

/** Procedimento que seta o Remetente da Mensagem */
public void setRemetente( String pNome ) {
  remetente = pNome;
}

/** Função que retorna o Remetente da Mensagem */
public String getRemetente() {
  return remetente;
}

/** Procedimento que seta o conteudo da Mensagem */
public void setCorpo( String pCorpo ) {
  corpo = pCorpo;
}

/** Procedimento que seta o tipo da Mensagem */
public void setTipo( String pTipo ) {
  tipo = pTipo;
}

/** Função que retorna o conteúdo da Mensagem */
public String getCorpo() {
  return corpo;
}

/** Procedimento que seta o Receptor da Mensagem */
public void setReceptor( String pNome ) {
  receptor = pNome;
}

/** Função que retorna o Receptor da Mensagem */
public String getReceptor() {
  return receptor;
}

/** Função que retorna o Tipo da Mensagem */
public String getTipo() {
  return tipo;
}
```

```
	/** Função que retorna a Mensagem representada como String */
	public String toString() {
		if (corpo!=null)
		return remetente + ": " + corpo;
		else
			return null;
	}

	/** Função que retorna a Mensagem representada como String */
	public String toHtml() {
		if (corpo!=null)
		return "<b>" + remetente + ": </b>" + corpo;
		else
			return null;
	}

 /**
	Método emXML retorna as informações
	do Vestibulando em formato XML
	**/
	public String emXML(){
		StringBuffer sb = new StringBuffer();
		sb.append( "<mensagem>\n" );
		sb.append( "\t<remte>"+getRemetente()+"</remte>\n" );
		sb.append( "\t<corpo>"+getCorpo()+"</corpo>\n" );
		sb.append( "\t<dst>"+getReceptor()+"</dst>\n" );
		sb.append( "\t<data>Arrumar</data>\n" );
		sb.append( "\t<hora>Arrumar</hora>\n" );
		sb.append( "</mensagem>\n" );
		return sb.toString();
	}
}
```

CallcenterOnline.java

```
package com.save.server.rmi;

import java.rmi.*;
import com.save.server.xindice.*;
import com.save.object.*;

public interface CallcenterOnline extends Remote {

  public boolean	conecteVestibulando( Vestibulando pVest) throws
RemoteException;
```

```
  public void              desconecteVestibulando(Vestibulando pVest) throws
RemoteException;

  public boolean           conecteAtendente( Atendente pAtendente) throws
RemoteException;

  public void              desconecteAtendente(Atendente pAtendente) throws
RemoteException;

  public void              sendQuestion(Mensagem pMensagem) throws RemoteException;

  public Mensagem          receiveAnswer(Vestibulando pVestibulando) throws
RemoteException;

  public void sendAnswer(Mensagem pMensagem) throws RemoteException;

        public Mensagem       receiveQuestion(Atendente pAtendente) throws
RemoteException;

  public boolean           cadastreAtendente( Atendente pAtendente) throws
RemoteException;
}
```

CallcenterOnlineImpl.java

```
package com.save.server.rmi;

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import com.save.object.*;
import com.save.server.sax.*;
import com.save.server.xindice.*;

public class CallcenterOnlineImpl extends UnicastRemoteObject implements
CallcenterOnline {

        private Xindice xindice;
        private Set atendentes = new HashSet();
        private Set vestibulandos = new HashSet();

  public CallcenterOnlineImpl() throws RemoteException {
                super();
    xindice = new Xindice();
  }

        public boolean conecteVestibulando( Vestibulando pVest){
```

```
                try {
                        if (xindice.conecteVestibulando(pVest)) {
                synchronized (vestibulandos) {
                vestibulandos.add(pVest);
                }
                System.out.println("Vestibulandos Conectados: " + vestibulandos.size());
                                return true;
                        }
                        else
                                return false;
                }
                catch (Exception e) {
                        System.out.println(e.getMessage());
                        return false;
                }
        }


  public void desconecteVestibulando(Vestibulando pVest) {
   synchronized ( vestibulandos ) {
     vestibulandos.remove( pVest );
   }
   System.out.println( "Vestibulandos Conectados: " + vestibulandos.size());
  }

  public boolean conecteAtendente(Atendente pAtendente) {
                try {
                        if (xindice.conecteAtendente(pAtendente)) {
                    synchronized ( atendentes ) {
                                atendentes.add( pAtendente );
                                }
                System.out.println("Atendente "+
pAtendente.getNome()+"("+atendentes.size()+")"+" conectado.");
                return true;
                        }
                        else {
                                pAtendente = null;
                                return false;
                        }
                }
                catch (Exception e) {
                        System.out.println(e.getMessage());
                        return false;
                }
  }

  public void desconecteAtendente(Atendente pAtendente) {
                synchronized ( atendentes ) {
                        atendentes.remove( pAtendente );
```

```
                }
                System.out.println( "Atendentes ativos: " + atendentes.size());
 }

 public void sendQuestion(Mensagem pMensagem) {
                try {
                        pMensagem.setTipo("at");
                xindice.addMensagem(pMensagem);
                }
                catch (Exception e){
                        System.out.println("Exceção em
CallcenterOnlineImpl(sendQuestion)" + e.getMessage());
                }
 }

 public Mensagem receiveAnswer(Vestibulando pVestibulando) {
                try {
                return xindice.getMsgVestibulando(pVestibulando);
                }
                catch (Exception e){
                        System.out.println("Exceção em
CallcenterOnlineImpl(receiveAnswer)" + e.getMessage());
                        return null;
                }
        }

 public void sendAnswer(Mensagem pMensagem) {
                try {
                        pMensagem.setTipo("vt");
                xindice.addMensagem(pMensagem);
                }
                catch (Exception e){
                        System.out.println("Exceção em
CallcenterOnlineImpl(sendAnswer)");
                }
        }


        public Mensagem receiveQuestion(Atendente pAtendente){
                try {
                return xindice.getMsgAtendente(pAtendente);
                }
                catch (Exception e){
                        return null;
                }
 }

 public boolean cadastreAtendente(Atendente pAtendente) {
                try {
```

```
                        System.out.println("Atendente em XML no
CallcenterOnlineImpl: \n" + pAtendente.emXML());
                        return xindice.addAtendente(pAtendente);
                }
                catch (Exception e) {
                        System.out.println("Exceção em
CallcenterOnlineImpl(cadastreAtendente)");
                        return false;
                }
        }

  public static void main(String args[]){
        String nomeServidor = "rmi://192.168.0.1/CallcenterOnline";
    try {
        CallcenterOnlineImpl servidor = new CallcenterOnlineImpl();
      Naming.rebind( nomeServidor, servidor );
    }
    catch( Exception e ) {
      System.out.println(e.getMessage());
                        System.exit(1);
    }
    System.out.println("****************************************");
    System.out.println("*      Servidor com.save rodando.      *");
    System.out.println("*                by                    *");
    System.out.println("*           Alex Mariano               *");
    System.out.println("****************************************");
        }
}
```

SaxAnalyser.java

```
package com.save.server.sax;

import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.sun.xml.parser.*;
import com.save.object.*;

public class SaxAnalyser {


        private String nomeClasse     = "com.sun.xml.parser.Parser";
        private org.xml.sax.Parser    parser;
        private SaxHandler                          handler;


        /* Construtor */
```

```
        public SaxAnalyser(String strXML){
    try{

                        /* Transforma a String XML em um InputStream */
                        ByteArrayInputStream conteudo = new
ByteArrayInputStream(strXML.getBytes());

                        /* Seta a fonte XML como a String passada por parâmetro */
                        InputSource is =
Resolver.createInputSource("text/xml;charset=ISO-8859-1",conteudo,true,"US-
ASCII");

                        /* Cria um handler SAX*/
      handler = new SaxHandler();

                        /* Cria um parser da SUN */
                        parser = ParserFactory.makeParser(nomeClasse);

                    /* Analisa um documento através do parser setadoL */
      parser.setDocumentHandler( handler );
      parser.setErrorHandler( handler );
      parser.parse(is);

    }
    catch(Exception e){
                        e.printStackTrace();
    }
        }


        public Mensagem getMensagem(){
    return handler.getMensagem();
        }

        public String getNomeAtendente(){
    return handler.getAtendente().getNome();
        }

        public String getNomeVestibulando(){
    return handler.getVestibulando().getNome();
        }

}//fim da classe SaxAnalyser
```

SaxHandler.java

```
package com.save.server.sax;

import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.ParserFactory;
import com.sun.xml.parser.Resolver;
import com.save.object.*;

public class SaxHandler extends HandlerBase{

        private Mensagem                msg;
        private Atendente       atd;
        private Vestibulando  vest;
        private String                  tagAtual;


        public Mensagem getMensagem(){
   return msg;
        }

        public Atendente getAtendente(){
   return atd;
        }

        public Vestibulando getVestibulando(){
   return vest;
        }

        public void startDocument() throws SAXException{
        }

        public void endDocument() throws SAXException{
        }

 /*
        Este método é chamado quando uma tag de elemento é aberta.
        Deve saber qual tag foi aberta para que o método character()
        possa fazer algo com os dados lidos pelo parser
        */
        public void startElement( String name , AttributeList atts ) {
                if( name.equalsIgnoreCase("mensagem") ) {
    msg = new Mensagem();
   }
                else if(name.equalsIgnoreCase("atendente")){
    atd = new Atendente();
                }
                else if(name.equalsIgnoreCase("vestibulando")){
```

```
    vest = new Vestibulando();
                }
                tagAtual =  name;
        }


/*
        Este método é chamado quando o parser encontra os dados entre as tags.
        This method is called when the SAX parser encounters #PCDATA or CDATA.
        É importante saber qual é a tag que acabou de ser aberta. Assim pode-se
        setar os dados de maneira correta.
        */
        public void characters( char ch[], int start, int length ){
   String value = new String( ch , start , length );
   if(!value.trim().equals("") && tagAtual!=null && value!=null) {
        if( tagAtual.equalsIgnoreCase("remte") ) {
                                System.out.println("Vai setar REMETENTE : " + value);
        msg.setRemetente(value);
    }
    else if( tagAtual.equalsIgnoreCase("dst") ) {
                                System.out.println("Vai setar DST : " + value);
        msg.setReceptor(value);
    }
    else if( tagAtual.equalsIgnoreCase("corpo") ) {
                                System.out.println("Vai setar CORPO : " + value);
        msg.setCorpo(value);
    }
    else if( tagAtual.equalsIgnoreCase("login") ) {
                                System.out.println("Vai setar LOGIN : " + value);
        atd.setLogin(value);
    }
    else if( tagAtual.equalsIgnoreCase("senha") ) {
                                System.out.println("Vai setar SENHA : " + value);
        atd.setSenha(value);
    }
    else if( tagAtual.equalsIgnoreCase("inscricao") ) {
                                System.out.println("Vai setar INSCRICAO : " + value);
        vest.setInscricao(value);
    }
    else if( tagAtual.equalsIgnoreCase("rg") ) {
                                System.out.println("Vai setar RG : " + value);
        vest.setRg(value);
    }
    else if( tagAtual.equalsIgnoreCase("nome") ) {
                                System.out.println("Vai setar NOME : " + value);
        if (atd!=null)
                                        atd.setNome(value);
        if (vest!=null)
                vest.setNome(value);
```

```
                                System.out.println("Setando nome no
SAXHANDLER!!!");
     }
   }
        }


        /*
        Este método é chamado quando uma tag de elemento é fechado.
        Quando o elemento mensagem é fechado, o corpo desta mensagem
        deve ser enviado ao destinatário
        */
        public void endElement( String name ){
        }


}//fim da classe SaxHander
```

Xindice.java

```
package com.save.server.xindice;

import java.io.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;
import org.xmldb.api.modules.XUpdateQueryService;
import org.w3c.dom.*;
import org.apache.xpath.*;
import org.apache.xpath.objects.*;
import org.apache.xindice.client.xmldb.services.*;
import org.apache.xindice.xml.dom.*;

import com.save.object.*;
import com.save.server.sax.*;

public class Xindice implements Serializable {

        private String strConexao;
        private String strColecao;
        private String strDriver;
        private String cabecalho;
        private String registros;
        private String strIdentificador;
        private ResourceIterator riConsulta;
        private Collection colecao;
```

```
	/** Construtor da Classe Xindice */
	public Xindice(){
		registros	= "";
		strConexao	= "xmldb:xindice:///db";
	  cabecalho	= "<?xml version='1.0' encoding='ISO-8859-1'?>\n";
	strDriver		= "org.apache.xindice.client.xmldb.DatabaseImpl";
		conecte();
	}


	/**
		Função: conecte()
		Descrição: faz a conexão com o banco de dados Xindice
		Parametros: nenhum
		Retorno: Retorna uma coleção do banco de dados
	*/
	public void conecte(){
		try {
			Class c = Class.forName(strDriver);
	Database banco = (Database) c.newInstance();
	DatabaseManager.registerDatabase(banco);
		}
		catch (Exception e) {
			System.out.println("Excecao na conexao com o Xindice");
		}
	}

  public void setColecao(String caminho){
		try {
			strColecao = caminho;
			colecao =  DatabaseManager.getCollection(strConexao +
caminho);
		}
		catch (Exception e) {
			//System.out.println("Exceção em Xindice(setColecao).\n
Mensagem: " + e.getMessage());
		}
	}

	/**
		Função: facaPesquisa()
		Descrição: executa uma consulta ao banco de dados
		Parametros: strPesquisa
		Retorno: Retorna um recordset, ou seja, o conjunto de registros
encontrados
	*/
	public boolean facaPesquisa(String strPesquisa) throws Exception{
		XMLResource xmlResource;
```

```
                try {
        synchronized ( colecao ) {
                        /** Faz a pesquisa e retorna um recordset */
        XPathQueryService servico = (XPathQueryService)
colecao.getService("XPathQueryService", "1.0");
        ResourceSet resultset = servico.query(strPesquisa);
                        riConsulta = resultset.getIterator();

                        /** O resultado da pesquisa é atualizado para a variável 'registros'
*/
                        if (riConsulta!=null) {
                                Resource res = riConsulta.nextResource();
                                xmlResource = (XMLResource) res;
                                strIdentificador = xmlResource.getDocumentId();
                                registros = (String) res.getContent();
                        }

                        }

                        return true;
                }

                catch (Exception e) {
                        //System.out.println("Exceção em Xindice(facaPesquisa).
Mensagem: " + e.getMessage());
                        return false;
                }
                finally {
                        fecheColecao();
                }


        } /** Fim da Função Pesquisa*/



  /** Função que retorna os registros consultados */
        public String getResultado(){
                return registros;
        }


        /**
                Função: trataResultado()
                Descrição: faz a conexão com o banco de dados Xindice
                Parametros:
                Retorno: Retorna uma coleção do banco de dados
        */
        public String trateResultado() throws Exception{
```

```
   StringBuffer retorno = new StringBuffer();
                try {
                        if (riConsulta!=null) {
                                Resource res = riConsulta.nextResource();
                                retorno.append((String) res.getContent());
                        }
                }
                catch (Exception e) {
                        return null;
                }
                return retorno.toString();
        }


        /**
                Função: atualize()
                Descrição: usa XUpdate para atualizar um campo no Xindice
                Parametros: xpath, campo, valor
        */
        public void atualize(String xpath, String campo, String valor) throws Exception{
        String xupdate =        cabecalho +

        "<xupdate:modifications version=\"1.0\" " +
                                                                                "
xmlns:xupdate=\"http://www.xmldb.org/xupdate\">" +

        "<xupdate:update select=\""+ xpath +"\">" + valor + "</xupdate:update>" +

        "</xupdate:modifications>";
                System.out.println("");
                System.out.println(xupdate);
                System.out.println("");

                if (colecao == null) {
                        System.out.println("COLEÇÃO EH NULA. CONECTANDO AO
BANCO DE DADOS");
                        setColecao("/db/callcenter");
                }

                XUpdateQueryService servico = null;
                try {
                        servico = (XUpdateQueryService)
colecao.getService("XUpdateQueryService", "1.0");
                        servico.update(xupdate);
                }
                catch (XMLDBException e) {
                        System.err.println("Servico XUpdate nao disponivel!");
                }
                finally {
```

```
                    fecheColecao();
                }

        }

        /**
                Função: addAtendente()
                Descrição: Insere um Atendente ao Banco de Dados
                Parametros: pAtendente
        */
        public boolean addAtendente(Atendente pAtendente) throws Exception{
                try {
                        setColecao("/callcenter/pessoa/atendente");
                        boolean existe =
facaPesquisa("/atendente[login="'+pAtendente.getLogin()+"']");
                        setColecao("/callcenter/pessoa/atendente");
                        if (!existe) {

                                /** Insere documento XML com as informações do
Atendemte */
                                XMLResource documento = (XMLResource)
colecao.createResource(pAtendente.getLogin(),"XMLResource");
                                String conteudo = cabecalho + pAtendente.emXML();
                documento.setContent(conteudo);
                                colecao.storeResource(documento);

                                /** Cria uma nova colecao cujo nome é o login do
atendente */

        crieColecao("/callcenter/mensagem/at",pAtendente.getLogin());

                                return true;
                        }
                        else
                                return false;
                }
                catch (Exception e) {
                        //System.out.println("Exceção em Xindice
(addAtendente).\nMsg:" + e.getMessage());
                        return false;
                }
                finally {
                        fecheColecao();
                }
        }


        /**
                Função: conecteAtendente()
```

```
		Descrição: Retorna o nome do Atendente quando o Login e a Senha
estiverem OK
		Parametros: pAtendente
	*/
	public boolean conecteAtendente(Atendente pAtendente) throws Exception{
		try {
			setColecao("/callcenter/pessoa/atendente");
			String strQuery = "/atendente[login='"+pAtendente.getLogin()+"'
and senha='"+pAtendente.getSenha()+"']";
			if (facaPesquisa(strQuery)){
				String nome = new
SaxAnalyser(getResultado()).getNomeAtendente();
				pAtendente.setNome(nome);
				return true;
			}
			else {
				return false;
			}
		}
		catch (Exception e) {
			//System.out.println("Exceção Xindice(conecteAtendente). \n
Mensagem: " + e.getMessage());
			return false;
		}
		finally {
			fecheColecao();
		}

	}


	/**
		Função: conecteVestibulando()
		Descrição:
		Parametros: pVest
	*/
	public boolean conecteVestibulando(Vestibulando pVest) throws Exception{
		try {
			setColecao("/callcenter/pessoa/vestibulando");
			String strQuery =
"/vestibulando[inscricao='"+pVest.getInscricao()+"' and rg='"+pVest.getRg()+"']";
			if (facaPesquisa(strQuery)){
				String nome = new
SaxAnalyser(getResultado()).getNomeVestibulando();
				pVest.setNome(nome);
				System.out.println("Vestibulando. \n " +
pVest.emXML());
				return true;
			}
```

```
                else {
                        return false;
                }
        }
        catch (Exception e) {
                System.out.println("Exceção Xindice(conecteVestibulando). \n
Mensagem: " + e.getMessage());
                return false;
        }
        finally {
                fecheColecao();
        }
}



/**
        Função: getMsgVestibulando()
        Descrição: Fornece mensagem do vestibulando
        Parametros: pAtendente
*/
public Mensagem getMsgVestibulando(Vestibulando pVestibulando) throws
Exception{
        try {
setColecao("/callcenter/mensagem/vt/"+pVestibulando.getInscricao());
                if (facaPesquisa("/mensagem")) {
                        Mensagem msg = new
SaxAnalyser(getResultado()).getMensagem();
                        removaDocumento();
                        return msg;
                }
                else {
                        return null;
                }
        }
        catch (Exception e) {
                return null;
        }
        finally {
                fecheColecao();
        }
}



/**
        Função: getMsgAtendente()
        Descrição: Fornece mensagem do vestibulando
        Parametros: pAtendente
```

```
	*/
	public Mensagem getMsgAtendente(Atendente pAtendente) throws Exception{
		try {
	setColecao("/callcenter/mensagem/at/"+pAtendente.getLogin());
			if (facaPesquisa("/mensagem")) {
				Mensagem msg = new
SaxAnalyser(getResultado()).getMensagem();
				removaDocumento();
				return msg;
			}
			else {
				return null;
			}
		}
		catch (Exception e) {
			//System.out.println("Exceção em Xindice
(getMsgAtendente).\nMsg:" + e.getMessage());
			return null;
		}
		finally {
			fecheColecao();
		}
	}



	/**
		Função: addMensagem()
		Descrição: Atribui uma mensagem a um Vestibulando ou Atendente
		Parametros: pMensagem
	*/
	public void addMensagem(Mensagem pMensagem) throws Exception{
		try {

	setColecao("/callcenter/mensagem/"+pMensagem.getTipo()+"/"+pMensagem.ge
tReceptor());

			/** Insere documento XML com o conteúdo da Mensagem */
			XMLResource documento = (XMLResource)
colecao.createResource(null,"XMLResource");
			String conteudo = cabecalho + pMensagem.emXML();
	documento.setContent(conteudo);
			colecao.storeResource(documento);
		}
		catch (Exception e) {
			//System.out.println("Exceção em Xindice
(getMsgAtendente).\nMsg:" + e.getMessage());
		}
		finally {
```

```
                fecheColecao();
            }
        }



        /**
            Função: crieColecao()
            Descrição: Cria uma coleção dentro de outra
            Parametros:  colecaoAtual, colecaoCriada
        */
        public void crieColecao(String colecaoAtual, String colecaoCriada) throws
Exception{
            setColecao(colecaoAtual);
            try {
                CollectionManager service = (CollectionManager)
colecao.getService("CollectionManager", "1.0");
                String configuracao =
                    "<collection compressed=\"true\" name=\"" +
colecaoCriada + "\">" +
                    "   <filer class=\"org.apache.xindice.core.filer.BTreeFiler\"
gzip=\"true\"/>" +
                    "</collection>";

        service.createCollection(colecaoCriada,DOMParser.toDocument(configuracao));
            }
            catch (Exception e){
            }
            finally {
                fecheColecao();
            }
        }


        /**
            Função: removaDocumento()
            Descrição: Remove um documento XML do Banco de Dados
            Parametros:  identificador
        */
        public void removaDocumento() throws Exception {
            try {
                if (strIdentificador!=null) {
                    setColecao(strColecao);
                    Resource documento =
colecao.getResource(strIdentificador);
            colecao.removeResource(documento);
                }
            }
            catch (Exception e){
```

```
                System.out.println("Exceção em Xindice(removaDocumento)");
            }
            finally {
                fecheColecao();
            }
        }


        /**
            Função: fecheColecao()
            Descrição: Fecha uma colecao
            Parametros:
        */
        public void fecheColecao() throws Exception {
            if (colecao != null) {
                colecao.close();
            }
        }


} /** Fim da Classe BancoDeDados */
```

ANEXO B – ARTIGO

Sistema de Atendimento ao Vestibulando Utilizando Banco de Dados Nativo XML

Alex Mariano Costa de Oliveira

Bacharelado em Ciências da Computação, 2003
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
mariano@inf.ufsc.br

Resumo

Este artigo descreve a utilização de um banco de dados nativo XML na implementação de um sistema de atendimento aos vestibulandos de uma Universidade. Trata-se de um novo conceito em armazenamento e recuperação de documentos XML que está em pleno desenvolvimento. Serão apresentadas as formas de armazenamento mais comuns e em seguida a ênfase serão os bancos de dados nativos XML.

Palavras-chave: Banco de Dados Nativo XML, Xindice.

Abstract

This paper describes the use of native XML databases in an online attendance system. It is is a new XML Documents storage approach, which is in growing development. The most common storage techniques will be presented e then a native XML database.

Key-words: Native XML Databases, Xindice.

Introdução

XML está rapidamente se tornando o formato de dados escolhido por uma grande parte das soluções existentes na área de tecnologia da informação, emergindo como um padrão para troca de dados na grande rede mundial. De acordo com o IDC , estima-se que os gastos em Bancos de Dados Nativos XML ultrapassem 280 milhões de dólares em 2006 representando uma taxa de crescimento anual superior a quarenta por cento.

Fazem parte das aplicações que utilizam XML o envio de documentos em sistemas B2B, a troca de mensagens para integração de sistemas legados, armazenamento de dados e várias outras atividades de manipulação de dados. Os benefícios geralmente associados a essa meta-linguagem são a independência de plataforma, capacidade de extensão e o baixo custo de implantação. Neste trabalho, a aplicação da XML está no armazenamento, manipulação e apresentação dos dados.

Para manter os dados de uma aplicação em um sistema de banco de dados, eles devem ser armazenados e recuperados de forma consistente, confiável e eficiente. Dentre os modelos de banco de dados existentes, o modelo relacional é o dominante e também o mais estudado. Por isso, utilizar essa estrutura parece ser uma boa maneira de organizar e gerenciar dados em formato XML. Porém, o mapeamento

de documentos XML em uma estrutura rígida como a dos bancos de dados relacionais não é elementar e pode ainda resultar em esquemas pouco elegantes [LIO02], desperdiçando espaço em disco e trazendo ineficiência nas consultas. Essa dificuldade ocorre porque o modelo relacional criado por E.F Codd é consideravelmente diferente do modelo de dados XML. Os bancos de dados relacionais organizam os dados em estrutura tabular e utilizam ligações relacionais para representar a hierarquia dos dados [XIN02], enquanto a XML é intrinsecamente uma estrutura de árvore hierárquica.

História e Conceito da XML

Há alguns anos, a publicação de dados eletrônicos estava limitada a poucas áreas científicas e técnicas, mas atualmente, trata-se de uma atividade universal. O uso da HTML na Internet possibilitou que os dados fossem apresentados em uma estrutura simples e de fácil leitura. Entretanto, a HTML apresenta limitações fundamentadas em sua própria concepção, baseada em marcações fixas. A emergência da XML como um padrão para a representação de dados na Internet pode facilitar a publicação em meios eletrônicos, por prover uma sintaxe simples, legível para computadores e seres humanos.

A XML (Linguagem de Marcação Estendida) é um subconjunto de uma linguagem de marcação criada em 1986 pelo W3C chamada SGML, que permitia que uma marcação específica fosse criada para especificar idéias e compartilhá-las. Apesar de ser uma linguagem de marcação poderosa, a SGML era demasiadamente complexa para ser utilizada. Em 1996, o engenheiro da Sun Microsystems, Jon Bosak, convenceu o W3C a formar uma equipe com o objetivo de utilizar a SGML na internet. Em Novembro daquele ano, foi criado a XML, inicialmente como uma versão simplificada da SGML, e em fevereiro de 1998, a XML tornou-se uma especificação formal, reconhecida pelo W3C.

A XML é uma linguagem de marcação compacta, enxuta e flexível que provê meios para descrever, armazenar, intercambiar e manipular dados estruturados. Faz parte também de uma iniciativa para estabelecer um padrão mundial para troca de dados.

Técnicas de Armazenamento de Documentos XML

Existem diferentes técnicas para armazenar e recuperar documentos XML. São elas: Abordagem Middleware , Banco de Dados Adaptados à XML e os Bancos de Dados Nativos XML. As duas primeiras técnicas se baseiam em bancos de dados relacionais, enquanto a última delas suporta dados XML intrínseca e naturalmente.

XML Middlewares são servidores independentes usados para transferir dados entre uma aplicação cliente que utiliza documentos XML e um servidor de banco de dados (geralmente relacional). Utilizam geralmente drivers padrões - JDBC e ODBC - para interagir com o banco de dados no servidor e são classificados em duas categorias: orientados a modelos e orientados à modelagem.

Já os bancos de dados adaptados à XML são definidos como bancos de dados relacionais convencionais que se ajustaram para que pudessem armazenar documentos XML centrados em dados, geralmente. Isso se deve ao fato de que as tabelas não são especificamente construídas como modelos de documentos XML. Porém, muitos desses bancos de dados podem armazenar em uma única coluna um documento centrado em documento como um todo, necessitando assim de processador de texto para as consultas.

Por fim, os bancos de dados nativos XML ou NXDs têm como sua unidade fundamental (lógica) de armazenamento um documento XML de qualquer estrutura, assim como o registro de uma tabela é a unidade fundamental dos bancos relacionais.

Xindice

O Xindice é uma banco de dados nativo XML gratuito que armazena e indexa documentos XML com o objetivo de disponibilizar o conteúdo para uma aplicação cliente com a vantagem de utilizar pouco processamento no lado servidor.

Os primeiros passos no desenvolvimento do Xindice, antes chamado de dbXML, foram dados por Tom Bradford em 1999, quando a linguagem utilizada ainda era o C/C++. No ano de 2000, o código-fonte migrou para Java e foi doado à The Apache Software Foundation em Dezembro de 2001.

A idéia principal por trás do Xindice é prover uma forma simples de armazenar e gerenciar um grande número de documentos XML. Essa meta é alcançada criando coleções de documentos, onde cada documento é armazenado em formato comprimido. Tal característica eleva a velocidade ao trabalhar com dados em XML e provê fácil mecanismo para consulta e manipulação dos documentos como um conjunto.

O Xindice é otimizado para armazenar grande número de pequenos documentos XML de até 50Kb. Pode-se também adicionar documentos maiores, porém este não é o melhor cenário. Há, inclusive, uma recomendação da própria Apache para que isso não aconteça. O tamanho máximo de um documento XML que pode estar em uma coleção do Xindice é de 5Mb.

Quando está sendo executado, o Xindice armazena uma séria de informações em objetos dentro da máquina virtual Java. Dentre esse dados, merecem destaque:

- A representação da hierarquia de coleções ;
- O estado da conexão com o cliente;
- Dados em cache.

Além dessas informações, uma instância do Xindice também precisa de acesso aos arquivos que contêm dados XML, que são armazenados em uma hierarquia de diretórios, começando na chamada raiz do banco de dados, cuja função e estrutura será detalhada posteriormente.

Existem duas maneiras de configurar a execução do Xindice, o modo embutido e o modo servidor.

No primeiro caso, uma aplicação Java completa configura uma instância do Xindice em sua própria máquina virtual e apenas essa aplicação é capaz de acessar e manipular os dados contidos no Xindice.

No modo servidor, o Xindice é executado como uma aplicação padrão Java, que pode também estar dentro de um container de aplicações web, como o Apache Tomcat. Nesse modo, vários clientes podem acessar o Xindice de diferentes máquinas virtuais, possivelmente em outro computador, utilizando XML-RPC, um padrão XML para chamadas remotas de procedimento sobre o protocolo HTTP.

No Xindice, todos os dados XML armazenados são organizados logicamente em hierarquia de coleções. Uma coleção é exatamente o que seu nome sugere: contém um determinado número de documentos XML além de suas próprias coleções e portanto, provendo hierarquia.

A coleção raiz do banco de dados (/db) é a única que contém características especiais como a ausência de coleção-pai e de documentos XML e é criada na primeira execução do Xindice. Seu conteúdo inicialmente é composto apenas por outra coleção chamada system, que contêm o dicionário de dados de todo o Xindice.

A coleção system contém duas coleções filhas: a SysConfig e a SysSymbols. Esta última contém vários documentos, que são tabelas de símbolos utilizadas para armazenamento dos nomes dos elementos e atributos de todo o conteúdo XML do banco de dados. Já a coleção SysConfig possui um documento chamado database.xml, cuja função é de manter as configurações de todas as coleções do banco de dados.

O Xindice, em sua versão atual (1.0), possui a maioria das características de um NXD, porém as transações e o controle de

concorrência são as maiores carências desta distribuição.

O gerenciamento do banco de dados Xindice se dá através de uma rica linha de comandos. Com ela, pode-se executar tarefas desde de uma simples inserção de documento em uma coleção até a exportação de uma árvore de diretórios.

A linguagem de consulta com a qual o Xindice trabalha é a XPath, que endereça partes de um documento XML utilizando uma sintaxe hierárquica que lembra o endereço de uma página na Internet. No Xindice, a XPath pode ser utilizada para consultar dados tanto em programas escritos conforme a especificação quanto na linha de comando oferecida pelo banco de dados.

Uma das deficiências da XPath é a falta de propriedades como agrupamento e ordenação, tão comuns à linguagem de consulta de banco de dados relacionais, a SQL.

Sistema de Atendimento

Para que a escalabilidade e reutilização sejam características do SAVE, ele foi desenvolvido separando apresentação, regras de negócios e acesso aos dados em camadas distintas, cujas funções são abordadas a seguir.

A camada de apresentação do SAVE é representada pela interface na qual ocorre a interação entre o usuário e o sistema. Por existirem dois tipos de usuários, os vestibulandos e os atendentes, há também duas aplicações diferentes na camada de apresentação. Ambas as aplicações são chamadas "thin client", ou seja, aplicações cujas funcionalidades estão restritas à manipulação da interface com o usuário, ao envio de requisições à camada de negócios e ao recebimento das suas respostas.

É nessa camada que está a inteligência de todo o sistema, e conseqüentemente é a que exige mais esforço de programação. Essa camada interage tanto com a camada de apresentação quanto com a camada de dados.

A função básica e mais executada pela camada de negócios é a de receber uma requisição da camada de apresentação, acionar a camada de dados e receber informações da mesma e enviá-las como resposta à camada de apresentação.

Na arquitetuta do SAVE, essa camada é representada por um processo executado no Servidor de Aplicações e Dados, que é o computador que hospeda, além do SAVE, o servidor de páginas, o Xindice.

A camada de dados é a que contém todas as informações necessárias ao sistema, ou seja, dados dos vestibulandos, dos atendentes, mensagens trocadas entre eles, etc. Relaciona-se apenas com a camada de negócios, com a tarefa de disponibilizar os dados nela contidos.

O banco de dados nativos XML, Xindice, representa esta camada na arquitetura geral do sistema.

Conclusão

Com o crescimento espantoso no uso da Internet nos últimos anos, surgiu a necessidade de interoperabilidade entre as plataformas de hardware e software existentes na rede. Dessa necessidade, nasceu a XML e um aumento expressivo no volume de informações sendo transportado e armazenado por sistemas de bancos de dados. Conseqüentemente, surgiram dificuldades na manipulação e integração destes dados com bancos de dados já existentes.

Ao avaliar as pesquisas na área de Banco de Dados Nativos XML realizadas neste trabalho, percebeu-se que esse tipo de sistema ainda é um pouco imaturo se comparados à robustez e à confiabilidade de um banco de dados relacional. Essa imaturidade é compreensível, visto que tal tecnologia nasceu há pouco mais de quatro anos. Além dessa barreira inicial, NXDs enfrentam a falta de padronização de consultas, armazenamento e manipulação de documentos XML.

Entretanto, é grande o entusiasmo da comunidade desenvolvedora em

aumentar a qualidade dos NXDs de torná-los a opção mais utilizada para armazenamento e recuperação de documentos XML. Para tanto, todos se espelham no trabalho já realizado no modelo relacional.

Vê-se também um crescente interesse das empresas fabricantes de bancos de dados relacionais nas qualidades da XML, visto que a maioria de seus produtos já se ajustou a fim de prover suporte ao gerenciamento de dados nesse formato. Por isso, vejo a XML hoje como centro das atenções e acredito numa possível convergência dos NXDs e bancos relacionais em um novo e único produto padronizado para o armazenamento e recuperação de documentos XML.

Referências

[1] BANSAL, Vinay; ALAM, Asna. Study and Comparison of Techniques to Efficiently Store and Retrieve XML Data. Disponível em: http://www.cs.duke.edu/~vkb/advdb/xmldatabase/documents/FinalReport.doc . Acesso em: 10 out. 2002.

[2] BOURRET, Ronald. XML and Databases. Disponível em: http://www.rpbourret.com/xml/XMLAndDatabases.htm . Acesso em: 10 out. 2002.

[3] BATES, James. Xindice Internals. Disponível em: http://xml.apache.org/xindice/dev/guide-internals.html. Acesso em: 25 nov. 2002.

[4] CHAMPION, M. Storing XML in Databases; Eai Jounal. 2001.

[5] DEITEL, H.M; DEITEL P.J. Java™ Como Programar; TRAD de Edson Furnankiewicz. 3. ed. Porto Alegre: Bookman, 2001.

[6] DYCK, T. Going Native: XML Databases - PC Magazine. New York City: jun. 2002.

[7] HEITLINGER, Paulo. O Guia Prático da XML. Lisboa: Centro Atlântico, 2001.

[8] LIOTTA, Matt. Apache's Xindice Organizes XML Data Without Schema. Disponível em: http://www.devx.com/xml/Article/9796/1954. Acesso em: 18 dez. 2002.

[9] STAKEN, Kimbro. Introduction to XML Databases. Disponível em: http://www.xml.com/pub/a/2001/10/31/nativexmldb.html. Acesso em: 19 out. 2002.

[10] SUN, M. Java™ Remote Method Invocation (RMI). Disponível em: http://java.sun.com/j2se/1.4/docs/guide/rmi/ . Acesso em 17 ago. 2002.

[11] TAMINO XML SERVER. Tamino XML Server Home Page. Disponível em: http://www.softwareag.com/tamino/ . Acesso em 18 ago. 2003.

[12] XML:DB INITIATIVE FOR XML DATABASES. XML:DB Frequently Asked Questions. Disponível em: http://www.xmldb.org/faqs.html . Acesso em: 27 set. 2002.