Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

NOVAS ABORDAGENS PARA COMPRESSÃO DE DOCUMENTOS XML

Autor: Márlon Amaro Coelho Teixeira

Orientador: Prof. Dr. Leonardo de Souza Mendes

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Telecomunicações e Telemática**.**

Banca Examinadora

Prof. Dr. Leonardo de Souza Mendes (presidente) — DECOM/FEEC/UNICAMP
Prof. Dr. Alexandre Carlos Brandão Ramos — UNIFEI
Prof. Dr. Gean Davis Breda — DECOM/FEEC/UNICAMP

Campinas – SP
26/01/2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

T235n

Teixeira, Márlon Amaro Coelho
Novas abordagens para compressão de documentos XML / Márlon Amaro Coelho Teixeira. – Campinas, SP: [s.n.], 2011.

Orientador: Leonardo de Souza Mendes.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Compressão de dados (Computação). 2. Compressão de dados (Telecomunicações). 3. XML (Linguagem de marcação de documento). I. Mendes, Leonardo de Souza. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês:	New approaches for compression of XML documents
Palavras-chave em Inglês:	Data compression (Computer science), Data compression (Telecommunication), XML (document markup Language)
Área de concentração:	Telecomunicações e Telemática
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Alexandre Carlos Brandão Ramos, Gean Davis Breda, Leonardo de Souza Mendes
Data da defesa:	26/01/2011
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Márlon Amaro Coelho Teixeira

Data da Defesa: 26 de janeiro de 2011

Título da Tese: "Novas abordagens para compressão de documentos XML"

Prof. Dr. Leonardo de Souza Mendes (Presidente): _____

Prof. Dr. Alexandre Carlos Brandao Ramos: _____

Prof. Dr. Gean Davis Breda: _____

Resumo

Atualmente, alguns dos fatores que determinam o sucesso ou fracasso das corporações estão ligados a velocidade e a eficiência da tomada de suas decisões. Para que estes quesitos sejam alcançados, a integração dos sistemas computacionais legados aos novos sistemas computacionais é de fundamental importância, criando assim a necessidade de que velhas e novas tecnologias interoperem. Como solução a este problema surge a linguagem XML, uma linguagem auto-descritiva, independente de tecnologia e plataforma, que vem se tornando um padrão de comunicação entre sistemas heterogêneos. Por ser auto-descritiva, a XML se torna redundante, o que gera mais informações a ser transferida e armazenada, exigindo mais recursos dos sistemas computacionais. Este trabalho consiste em apresentar novas abordagens de compressão específicas para a linguagem XML, com o objetivo de reduzir o tamanho de seus documentos, diminuindo os impactos sobre os recursos de rede, armazenamento e processamento.

São apresentadas 2 novas abordagens, assim como os casos de testes que as avaliam, considerando os quesitos: taxa de compressão, tempo de compressão e tolerância dos métodos a baixas disponibilidades de memória. Os resultados obtidos são comparados aos métodos de compressão de XML que se destacam na literatura. Os resultados demonstram que a utilização de compressores de documentos XML pode reduzir consideravelmente os impactos de desempenho criados pela linguagem.

Palavras-chave: Compressão de Dados, XML.

Abstract

Actually, some of the factors that determine success or failure of a corporation are on the speed and efficiency of making their decisions. For these requirements are achieved, the integration of legacy computational systems to new computational systems is of fundamental importance, thus creating the need for old and new technologies interoperate. As a solution to this problem comes to XML, a language self-descriptive and platform-independent technology, and it is becoming a standard for communication between heterogeneous systems. Being self-descriptive, the XML becomes redundant, which generates more information to be transferred and stored, requiring more resources of computational systems. This work presents new approaches to specific compression for XML, in order to reduce the size of your documents, reducing the impacts on the reducing the impact on network, storage and processing resources. Are presented two new approaches as well as test cases that evaluate, considering the questions: compression ratio, compression time and tolerance of the methods to low memory availability.The results are compared to the XML compression methods that stand out in the literature. The results demonstrate that the use of compressors XML documents can significantly reduce the performance impacts created by the language.

Keywords: Data Compression, XML.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AAL1 - *Asynchronous Transfer Mode Adaptation Layer Type 1*
ATM - *Asynchronous Transfer Mode*
B-ISDN - *Broadband Integrated Service Digital Network*
CCITT - *Consultative Committee for International Telegraphy and Telephony*
DTD - *Document Type Descriptor*
EXI - *Efficient eXtensible Markup Language Interchange*
HTML - *HyperText Markup Language*
HTTP - *Hypertext Transfer Protocol*
IBM - *International Business Machinesr*
ISO - *International Organization for Standardization*
ITU-T - *International Telecommunication Union-Telecommunication Standardization Sector*
J2EE - *Java 2 Enterprise Edition*
J2ME - *Java Plataform Micro Edition*
J2SE - *Java 2 Standard Edition*
JPEG - *Joint Photographic Experts Group*
Lorel - *Lore's Query Language*
LZ77 - *Ziv and Lempel 1977*
MP3 - *Moving Picture Experts Group-1/2 Audio Layer 3*
ONE - *Sun Open Net Enviroment*
PDAs - *Personal Digital Assistants*
PHP - *Hypertext Preprocessor*
RAM - *Random Access Memory*
RMI - *Remote Method Invocation*
SAX - *Simple Application Programming Interface for eXtensible Markup Language*
SGML - *Standard General Markup Language*
SOAP - *Simple Object Access Protocol*
SQL - *Structured Query Language*
UDDI - *Universal Description, Discovery and Integration*
URI - *Uniform Resource Identifier*
W3C - *World Web Consortium*
WBXML - *Wireless Application Protocol Binary eXtensible Markup Language*

WSDL - *Web Service Definition Language*
WWW - *World Wide Web* XBC - *eXtensible Markup Language Binary Characterization*
XML - *eXtensible Markup Language*
XML-QL - *Query Language for eXtensible Markup Language*
XML-RPC - *eXtensible Markup Language - Remote Procedure Call protocol*
XPath - *eXtensible Markup Language Path Language*
XQL - *eXtensible Markup Language Query Language*
XS - *eXtensible Markup Language Schema*
XSD - *eXtensible Markup Language Schema Definition*

Capítulo 1

Introdução

A empresa IBM (*International Business Machines*) em meados da década de 70 possuía grandes volumes de dados armazenados e precisava de uma forma eficiente de organizá-los. Para atender a esta necessidade, a empresa criou a linguagem SGML (*Standard General Markup Language*) capaz de descrever diversos tipos de dados e facilitar o compartilhamento de informações na Internet (Travis and Ozkan, 2002).

Com a popularização da Internet, criou-se a necessidade de distribuir de maneira rápida e fácil as informações. Foi então criada uma linguagem derivada da SGML chamada HTML (*HyperText Markup Language*) (Lambert et al., 2005),(w3schools, 1999),(Bray et al., 2008),(Quin, 2010),(Harold, 1999).

A linguagem HTML propiciou a qualquer usuário da Internet uma forma simples de publicar informações para outros usuários em qualquer parte do mundo, mas se mostrou limitada para as novas necessidades que foram surgindo, pois possuía um conjunto fixo e pré-definido de etiquetas impossibilitando a criação de extensões da linguagem. Devido a esta limitação foi criada a linguagem XML (*eXtensible Markup Language*) (Lambert et al., 2005),(Travis and Ozkan, 2002), (Quin, 2010),(Bray et al., 2008).

A linguagem XML combina a flexibilidade da linguagem SGML com a simplicidade da HTML, oferecendo um método estruturado, auto-descritivo e capaz de transferir informação independentemente de qualquer plataforma (w3schools, 1999),(Harold, 1999).

Pelo fato da linguagem XML ser totalmente independente de tecnologia ou plataforma, ela vem se tornando um padrão de comunicação entre sistemas computacionais em ambientes heterogêneos, onde existe a necessidade que várias tecnologias coexistam e principalmente interoperem (w3schools, 1999), (Bray et al., 2008), (Quin, 2010).

A construção de ambientes heterogêneos vem da crescente necessidade de acesso às informações de formas rápida e eficaz. Em se tratando de corporações públicas, este cenário vem ganhando des-

taque graças as exigências das comunidades, empresas e cidadãos por uma maior responsabilidade e eficiência na gestão de recursos e serviços disponibilizados pelos governantes (Nações Unidas, 2008).

Para cumprir essas exigências os governos estão tornando mais transparentes e proativas suas administrações, buscando formas mais eficientes de gerenciar seus recursos financeiros, humanos e tecnológicos. Para atingir estes objetivos, um importante conceito ganha destaque, denominado Governo Eletrônico (do inglês *Electronic Government* - e-Gov) (Nações Unidas, 2008).

Realizando uma conceituação mais ampla de Governo Eletrônico, pode-se definir como a utilização de tecnologias de comunicação e de informação com o objetivo de facilitar as práticas diárias da administração pública e aproximar as instituições governamentais dos outros atores que compõem a sociedade (cidadãos, corporações públicas e privadas) (Sprecher, 2000).

Algumas características na construção de sistemas de Governo Eletrônico tornam o emprego de novas tecnologias uma tarefa mais árdua que em outros ambientes. Nas instituições públicas há uma maior resistência a mudanças e um maior isolamento entre os departamentos, fazendo com que cada departamento adote seus equipamentos e tecnologias específicas, causando dificuldades na integração para o compartilhamento das informações (Chen, 2002). Por exemplo, pode-se encontrar em algumas instituições (públicas ou privadas) grandes quantidades de informação armazenadas em banco de dados Adabas (AG, 2010) que foi desenvolvido na década de 70. O banco Adabas foi um dos primeiros sistemas gerenciador de banco de dados não-relacional produzido comercialmente. A linguagem de acesso as informações contidas no Adabas são acessadas pela linguagem Natural (AG, 2010), que é específica para este banco de dados. Possuem também sistemas desenvolvidos em linguagens de programação antigas, como o Pascal, que foi projetada sobre o paradigma estrutural e criada na década de 70.

Em contrapartida, as instituições possuem também sistemas desenvolvidos em tecnologias de última geração. Por exemplo, sistemas desenvolvidos em linguagens de programação como o Java (Oracle, 2010), criada na década de 90 sobre o paradigma de orientação a objetos acessando um banco de dados Oracle (Oracle, 2010), que é um banco relacional e utiliza a linguagem de consulta SQL.

Os cenários de isolamento tecnológico, resistência a mudanças e de desenvolvimento de sistemas ao longo do tempo descritos acima, demonstram como os ambientes heterogêneos podem apresentar tecnologias de diferentes épocas, paradigmas, plataformas e arquiteturas tendo a necessidade de coexistirem e interoperarem, criando sérias dificuldades para os desenvolvedores e projetistas de sistemas de computação. São nestes cenários que a linguagem XML tem sido empregada para prover a integração de diferentes tecnologias.

Entretanto, a linguagem XML é auto-descritiva o que a torna redundante, acarretando aos dados acréscimos de informações, aumentando o tráfego sobre a rede e exigindo mais recursos de armazenamento e processamento (Liefke and Suciu, 2000),(Geer, 2005).

Para diminuir este impacto negativo sobre os recursos computacionais, a solução mais explorada é a compressão de dados. A compressão de dados retira informações inúteis ou repetidas, fazendo com que o tamanho da informação a ser transferida ou armazenada diminua (Geer, 2005), (Davis and Burnett, 2005).

Com o objetivo de melhorar o desempenho da linguagem XML foram criados os compressores de XML. Alguns exemplos dessas ferramentas encontradas na literatura são: XMill (Liefke and Suciu, 2000), WBXML (Martin and Jano, 1999), EXI (Schneider and Kamiya, 2008) e MPEG-7/BIM (Niedermeier et al., 2002). Estes compressores além de retirarem os dados redundantes, exploram características específicas da linguagem para atingirem altas taxas de compressão (Augeri et al., 2007).

Neste trabalho são propostas duas abordagens específicas para a linguagem XML que denominamos de Proposta *Schema* e Híbrida. Estas abordagens exploram uma característica da linguagem XML que os métodos encontrados na literatura não exploram: o fato da linguagem XML ser hierárquica e estruturada como uma árvore, criando redundância de informação em seus níveis de mesma hierarquia.

Além de propor novas abordagens de compressão, este trabalho também tem o objetivo de estudar o comportamento dos novos métodos com relação aos quesitos de taxas de compressão, tempo de compressão e tolerância a baixas disponibilidades de memória, comparando os resultados obtidos com os métodos existentes na literatura e no mercado.

O primeiro capítulo é o introdutório, situando o leitor do ambiente em que o trabalho está inserido. O segundo capítulo mostra de forma mais detalhada o que é a linguagem XML e seus elementos, criando o embasamento teórico necessário para o melhor entendimento do trabalho. Também são descritos algumas tecnologias e ambientes onde a linguagem é empregada. O terceiro capítulo define de forma mais ampla o que é compressão de dados, apresentando algumas técnicas mais presentes na literatura juntamente com os métodos de compressão específicos da linguagem XML. No quarto capítulo são apresentadas as propostas deste trabalho: a Proposta *Schema* e a Híbrida. São descritas em detalhes as técnicas que cada uma utiliza para comprimir documentos XML. No quinto capítulo são realizados os testes dos métodos discutidos no trabalho, onde os resultados dos quesitos de taxas de compressão, tempo de compressão e tolerância a baixas disponibilidades de memória foram avaliados e comparados. No sexto capítulo são apresentadas as considerações finais.

Capítulo 2

Linguagem XML

Este capítulo tem por objetivo apresentar as tecnologias e os conceitos envolvidos neste trabalho, criando um embasamento teórico para que as informações contidas no texto sejam compreendidas. É apresentado como ocorreu o surgimento da linguagem XML, os objetivos de sua criação, vantagens, desvantagens e sua composição. Serão descritas algumas tecnologias que utilizam a linguagem XML, como exemplo, os *Web Services*, que estão sendo amplamente utilizados.

2.1 XML

Os desenvolvedores da IBM mensuraram a grande quantidade de documentação digital que a empresa possuia e tinham em mente que poderia aumentar ainda mais devido as diversas áreas de atuação. Eles desenvolveram então a linguagem GML, que tinha por objetivo classificar e descrever qualquer documento e principalmente processá-lo quando necessário. A linguagem obteve resultados significativos, tendo como consequência a sua padronização pela ISO (*International Organization for Standardization*) surgindo a SGML (*Standard General Markup Language*) (Lambert et al., 2005), (Travis and Ozkan, 2002), (w3schools, 1999), (Bray et al., 2008), (Quin, 2010).

A linguagem SGML se mostrou poderosa, já que conseguia atender uma variada gama de novos requisitos, como por exemplo, as linguagens de *markup* e *hiperlinks* que nasceram com o surgimento da Internet. A partir da SGML, foi criado a linguagem HTML (*HyperText Markup Language*) (Lambert et al., 2005), (Travis and Ozkan, 2002), (w3schools, 1999), (Bray et al., 2008), (Quin, 2010).

A linguagem HTML é uma linguagem de marcação (utiliza *tags*) capaz de construir páginas Web, que são interpretadas pelos navegadores (*browsers*). Com a popularização da Internet, criou-se a possibilidade de distribuir informação de maneira fácil e com baixo custo a qualquer usuário que tenha acesso a Internet em qualquer parte do mundo. Simples usuários passaram a ter o poder de divulgar o que desejassem. Porém, à medida que o número de utilizadores cresceu, as limitações

das tecnologias se tornaram mais evidentes. Estas limitações surgiram principalmente pelo fato de que o HTML possui apenas um conjunto fixo e pré-definido de etiquetas impossibilitando a criação de extensões da linguagem. Desta forma, a criação de uma nova linguagem tornou-se necessária (Lambert et al., 2005), (w3schools, 1999), (Bray et al., 2008), (Quin, 2010), (Harold, 1999),(W3C, 2009).

Muitas soluções foram propostas, mas a falta da criação de um padrão culminou no fracasso da maioria delas, pois a interoperabilidade das diferentes aplicações foi prejudicada. Surgiu então a linguagem XML (*eXtensible Markup Language*) que ofereceu um método estruturado, auto-descritivo, capaz de transferir informação e independente de qualquer plataforma. Com isso, as aplicações heterogêneas poderiam trocar informações sem qualquer problema de padronização de comunicação e diferenças de plataformas (Quin, 2010), (Bray et al., 2008), (w3schools, 1999), (Harold, 1999).

As principais vantagens do XML são (w3schools, 1999), (Bray et al., 2008):

- O formato XML é legível tanto para os seres humanos quanto para as aplicações, graças a sua característica auto-descritiva;
- Permite a realização de validação do documento a partir de um *Schema*, auxiliando na complexidade da aplicação, já que o programador não precisa ter a preocupação de realizar a implementação de rotinas de validação;
- O formato hierárquico permite que os dados sejam categorizados, assim atendendo um leque diversificado de problemas, o que torna o XML uma linguagem poderosa;
- É independente de tecnologia, o que é crucial na comunicação de sistemas heterogêneos. A linguagem XML, por exemplo, viabiliza a comunicação entre banco de dados incompatíveis.
- A linguagem XML é muito difundida na comunidade científica e entre as corporações, o que a torna uma tecnologia madura e consistente;
- O processamento dos dados fica a cargo do cliente, diminuindo a carga de processamento no servidor.
- A linguagem XML é escalável, ou seja, ela vai agregando funcionalidade e complexidade com o decorrer do desenvolvimento da aplicação.

As principais desvantagens do XML são (w3schools, 1999), (Bray et al., 2008), (Geer, 2005):

- Se comparado com outras formas de armazenamento e de transferência de dados, a sintaxe do XML é muito redundante, e com isso, os seus arquivos se tornam muito maiores que os das outras tecnologias que possuem a mesma finalidade;

- Dependendo da aplicação, a criação de interpretadores pode-se tornar complexa;
- Quanto maior a quantidade de dados, maior será o número de *tags* para representar os dados, com isso, levando ao aumento do tempo de processamento;
- O modelo hierárquico limita a utilização em outros paradigmas, como por exemplo, o relacional.

2.1.1 XML *Schema*

Para a construção de um arquivo XML, podem-se definir documentos contendo metalinguagens que determinam a sua validade, estes documentos são chamados de validadores de XML. Os objetivos dos validadores é de definir um conjunto de regras para comprovar a integridade dos dados que compõem o documento, tirando da aplicação a responsabilidade de realizar a validação dos dados (Fallside and Walmsley, 2004).

Um dos validadores de XML mais utilizados foi o DTD *Document Type Descriptors*, que também era o validador da linguagem SGML. Com o decorrer do tempo, as limitações do DTD foram ficando mais evidentes, como por exemplo: era uma linguagem específica, todos os tipos são interpretados como texto, o não suporte ao caracter espaço e a obrigatoriedade de uma ordem pré-determinada dos elementos (Fallside and Walmsley, 2004).

Para suprir estas limitações do DTD foi desenvolvido o XSD (*Xml Schema Definition*) ou XSXML (*Schema*), que se tornou a recomendação oficial do W3C desde 2001 para validação da linguagem XML. Os principais propósitos do XML *Schema* são (Fallside and Walmsley, 2004), (Shanmugasundaram et al., 1999):

- Definir elementos e atributos que podem aparecer em um documento;
- Definir hierarquia dos elementos;
- Definir a ordem e quantidade dos elementos filhos;
- Definir se um elemento é vazio ou pode incluir texto;
- Definir tipos de dados para elementos e atributos;
- Definir valores padrão e fixos para elementos e atributos.

Uma grande vantagem do XML *Schema* é de ser escrito em XML. Para criá-lo não é necessário que o desenvolvedor aprenda uma nova linguagem ou novas ferramentas, não comprometendo assim o tempo de desenvolvimento. Outras vantagens podem ser apontadas (Fallside and Walmsley, 2004):

- São extensíveis, com isto o desenvolvedor possui a liberdade de reutilizar *Schemas* já criados, criar tipos derivados e referenciar outros *Schemas* no mesmo documento;
- Suportam tipos de dados, sendo assim, realizar a validação dos dados, trabalhar com dados de um banco de dados, realizar a conversão de dados e definição de padrões e formatos se tornam mais fáceis;
- Suportam namespaces;
- É recomendação oficial da W3C.

A figura 2.1 apresenta um trecho de um documento XSD. Ele define as regras dos dados que são aceitos ou não no documento XML. Caso essas regras não sejam respeitadas uma exceção é gerada, informado que os dados não estão coerentes com as regras descritas no XSD.

```
<xsd:element name="clientes">
  <xsd:complexType>
      <xsd:sequence>
          <xsd:element name="cliente" type="Cliente"
              minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="Cliente">
  <xsd:sequence>
    <xsd:element name="empresa" type="xsd:string" minOccurs="1"
      maxOccurs="1" nillable="false"/>
    <xsd:element name="tipo" type="xsd:string" minOccurs="1"
      maxOccurs="1" nillable="false"/>
    <xsd:element name="nome" type="xsd:string" minOccurs="1"
      maxOccurs="1"/>
    <xsd:element name="sexo" type="xsd:string" minOccurs="1"
      maxOccurs="1" nillable="false"/>
  </xsd:sequence>
</xsd:complexType>
```

Fig. 2.1: Trecho de um documento XSD.

2.1.2 *Namespace*

Na linguagem XML, o usuário tem a total liberdade de criar suas próprias *tags*, o que permite uma grande flexibilidade na criação dos documentos XML. Com esta liberdade surge um problema: como garantir que as *tags* no documento XML possuam um nome único? A resposta para esta pergunta

é o *namespace*. Uma das principais motivações de utilização do *namespaces* é justamente evitar os conflitos quando se utiliza e re-utiliza vários dicionários (Bray et al., 1999).

A definição de *namespace* é uma coleção de nomes identificados por uma URI (*Uniform Resource Identifier*), que são utilizados nos documentos XML como tipos de documentos e nomes de atributos. Este conceito possui vários pontos de similaridade com o conceito de pacotes em Java. Assim como em um pacote no Java é permitido que várias classes e interfaces sejam reutilizadas, no *namespace* é possível ter vários elementos e atributos que podem ser re-utilizados. Para se utilizar uma classe ou interface presente em um pacote é necessário qualificar a classe ou interface com o pacote a que ela pertence, isso também ocorre no *namespace*, os elementos e atributos devem ser qualificados com o *namespace* ao qual eles pertencem. No Java uma classe pode pertencer indiretamente a um pacote. Assim ocorre também com o *namespace*, atributos e elementos podem herdar os *namespaces* de seus pais, pertecendo assim indiretamente a eles (Srivastava,Rahul, 1999).

Na figura 2.2 o documento XML possui *tags* com o mesmo nome. As *tags* <cli:nome> e <cli:telefone> estão qualificadas ao *namespace* cli. Já as *tags* filhas de <prod:produto> (<nome> e <valor>) não estão qualificadas a nenhum *namespace*, mas como os seus pais estão, elas herdam o *namespace* prod, por isso, mesmo possuindo *tags* de mesmo nome em um mesmo documento XML elas podem coexistir.

```
<?commentstylexml commentstyleversion="1.0" enconding="UTF-8"?>
<clientes exemplos xmlns:cli="http://cliente.com.br"
xmlns:proc="http://produto.com.br"
<cli:clientes>
    <cli:nome>Cliente1</cli:nome>
    <cli:telefone>42562100</cli:telefone>
    <prod:produto>
        <nome>Produto1</valor>
        <valor>42562100</valor>
    </prod:produto>
</cli:clientes>
```

Fig. 2.2: Exemplo de um documento XML que utiliza namespace

2.1.3 *Elementos*

Os elementos são divididos em 2 grupos: elementos simples e complexos. Os elementos simples não possuem atributos ou elementos. Podemos criar nossos próprios tipos simples e também utilizar os tipos pré-existentes que são: *string*, *decimal*, *integer*, *boolean*, *date* e *time*.

Para definir o tipo de um elemento ou atributo, o atributo *type* deve ser definido utilizando um dos tipos simples apresentados acima. A definição do atributo *type* é muito útil, pois pode-se criar

validações no próprio *Schema* tirando da aplicação a responsabilidade de realizá-las. Por exemplo, levando em consideração o elemento <xs:element name="dataNascimento"type="xs:date"/>, caso o valor "Endereço 1"seja atribuído ao elemento dataNascimento, uma exceção é gerada, não permitindo que um dado que não seja do tipo *date* possa ser atribuído. Os elementos podem possuir alguns atributos para melhorar a especificação pelo programador, por exemplo:

- *name*: determina o nome do elemento;
- *type*: especifica o tipo ao qual o elemento pertence (*date*, *string*);
- *minOccurs*: especifica o mínimo de vezes que o elemento deve aparecer;
- *maxOccurs*: especifica o mínimo de vezes que o elemento deve aparecer. Para determinar um número ilimitado usa-se a palavra *unbounded*.

Os elementos complexos são elementos XML que contém pelo menos um elemento e/ou atributo. Na figura 2.3 é apresentado um exemplo de criação de um elemento complexo em um documento XSD.

```
<xs:element name="clientes">
  <xs:complexType>
      <xs:sequence>
          <xs:element name="firstname" type="xs:string"/>
          <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fig. 2.3: Criação de elementos complexos em um documento XSD

2.1.4 *Atributos XSD*

Os atributos são utilizados para realizar uma melhor descrição dos elementos. A principal diferença surge no momento da criação do atributo, em vez de usar a palavra reservada xs:element usa-se xs:attribute. Exemplo: <xs:attribute name="lang"type="xs:string"/>

Os atributos podem ser usados para armazenar dados, realizando assim a função dos elementos. Não há uma regra que define quando se deve usar elementos ou atributos, isso fica a critério de cada desenvolvedor. Na figura 2.4 que representa um trecho de um documento XML, a informação sexo é um atributo da *tag* <pessoa>, já na figura 2.5, a informação sexo é um elemento da *tag* <pessoa> e as duas formas podem ser utilizadas.

```
<pessoa sexo="feminino">
    <primeironome>Pessoa 1</primeironome>
    <ultimonome>último nome</ultimonome>
</pessoa>
```

Fig. 2.4: Trecho de um documento XML utilizando atributo.

```
<pessoa>
    <sexo>feminino</sexo>
    <primeironome>Pessoa 1</primeironome>
    <ultimonome>último nome</ultimonome>
</pessoa>
```

Fig. 2.5: Trecho de um documento XML utilizando elemento.

2.2 Linguagem de Consultas para XML

Com o passar dos anos houve um aumento na quantidade de informação que são armazenadas, transferidas e apresentadas utilizando a linguagem XML. A capacidade das fontes de dados de prover aos usuários formas inteligentes de consultas se torna cada dia mais importante. Umas das principais vantagens da linguagem XML é a flexibilidade na representação dos dados e as linguagens de consulta fornecem recursos para recuperar e interpretá-los (Scott et al., 2007).

As linguagens de consultas para XML são criadas com o objetivo de extrair as informações dos documentos XML em forma de consulta. Elas são capazes de obter partes dos dados contidos no documento, evitando que toda a informação seja processada. Exercem a mesma função da linguagem SQL que é utilizada para consultar informações em banco de dados. Foram criadas muitas linguagens de consultas, por exemplo: Lorel (Abiteboul, 2004), (Goldman et al., 1999), XML-QL (Deutsch et al., 1998), XQL (Robie et al., 1998), XPATH (Clark and DeRose, 1999) e XQUERY (Boag et al., 2007).

2.3 Web Services

O termo *Web Services* é muito utilizado nos dias de hoje de formas diferentes. *Web Services* são quaisquer serviços disponíveis na Internet que utilizam a linguagem XML como padrão de comunicação, independente de sistema operacional ou linguagem de programação (Alonso et al., 2004), (Cerami, 2002).

As tecnologias e os conceitos variam de acordo com a interpretação de cada autor ou entidade. A UDDI (*Universal Description, Discovery and Integration*) define *Web Services* como aplicações autosuficientes, compostas de negócios modulares utilizando interfaces baseadas em padrões orientados a Internet. Esta definição propõe que o serviço seja aberto, ou seja, que de alguma forma seja publicado e acessado por meio da Internet (Alonso et al., 2004), (Cerami, 2002).

A W3C define *Web Services* como um *software* identificado por uma URI, na qual suas interfaces e ligações são capazes de serem definidas, descritas e descobertas por artefatos XML. Um *Web Service* suporta interações de outros *softwares* diretamente utilizando mensagens XML e protocolos na Internet. Esta definição da W3C abrange os conceitos de que um *Web Service* deve possuir as propriedades de ser definido, descrito e descoberto. Indiretamente esta definição reforça a idéia de acessibilidade, tornando mais concreta a noção de serviço orientado a Internet e baseado em padrões de interface (Alonso et al., 2004), (Cerami, 2002).

Levando em consideração textos mais técnicos, a definição de *Web Services* pela Webopedia (Webopedia, 2010) é de que *Web Services* é uma forma padronizada de interação de aplicações baseadas na Web que utilizam XML, SOAP (Consortium, 2004), WSDL (Christensen et al., 2001) e UDDI sobre os protocolos da Internet. O papel da linguagem XML é rotular os dados, o protocolo SOAP é usado para transferir os dados, o WSDL (*Web Service Definition Language*) tem a função de descrever os serviços que serão disponibilizados e o UDDI é utilizado para listar os serviços disponíveis. Esta definição traz os padrões que são utilizados para realizar a interação com *Web Services*, mas estes padrões não servem para definir *Web Services*, o conceito de *Web Services* deve ser desvinculado dos padrões que ele utiliza (Webopedia, 2010).

Existem muitas implementações de *Web Services*, sendo as principais: Microsoft's .NET, IBM *Web Services* e *Sun Open Net Enviroment* (ONE). Cada *framework* possui características diferentes, mas compartilham o mesmo conjunto de tecnologias, principalmente o SOAP, WSDL e UDDI (Alonso et al., 2004), (Cerami, 2002).

2.3.1 *SOAP*

O protocolo SOAP *Simple Object Access Protocol* é utilizado para troca de informações que estão descentralizadas e distribuídas. As mensagens são escritas em XML e trocadas sobre o protocolo HTTP. Ele é uma maneira padronizada de aplicações se conectarem e invocarem métodos remotos (Cerami, 2002).

Existem *frameworks* que também disponibilizam funcionalidades similares ao SOAP, como são os casos do CORBA, DCOM e do Java RMI, mas estes *frameworks* trabalham com mensagens em uma linguagem própria, não havendo uma padronização das mensagens de requisição e resposta. Com isso, o SOAP possui uma grande vantagem sobre essas tecnologias, já que suas mensagens são

escritas no padrão XML, tornando interoperável aplicações escritas em diversas linguagens e rodando sobre plataformas diferentes (Cerami, 2002). Nas figuras 2.6 e 2.7 são apresentados exemplos de mensagens XML de requisição e de resposta do protocolo SOAP.

```
POST /InStock HTTP/1.1
Host www.example.org
Content-Type: application/soap+xml;charset=utf-8
Content-Length: nnn
<?commentstylexml commentstyleversion="1.0"?>
    <soap:Envelope
        xmlns:soap="http://www.w3c.org/2001/12/soap-envelope"
        soap:encodingStyle = "http://www.w3c.org/2001/12/
            soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
            <m:GetStockPrice>
                <m:StockName>IBM</m:StockName>
            </m:GetStockPrice>
        </soap:Body>
    </soap:Envelope>
```

Fig. 2.6: Mensagem de requisição de um serviço do protocolo SOAP.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml;charset=utf-8
Content-Length: nnn
<?commentstylexml commentstyleversion="1.0"?>
    <soap:Envelope
        xmlns:soap="http://www.w3c.org/2001/12/soap-envelope"
        soap:encodingStyle = "http://www.w3c.org/2001/12/
            soap-encoding">
        <soap:Body xmlns:m="http://www.example.org/stock">
            <m:GetStockPriceResponse>
                <m:Price>34,5</m:Price>
            </m:GetStockPriceResponse>
        </soap:Body>
    </soap:Envelope>
```

Fig. 2.7: Mensagem de resposta de um serviço do protocolo SOAP.

2.3.2 XML-RPC

O protocolo XML-RPC (*Remote Procedure Calling*) é uma especificação e um conjunto de implementações que permite que uma aplicação em execução seja acessada pela rede por outras ferramentas

rodando em ambientes diferentes. A especificação XML-RPC utiliza o protocolo HTTP para transportar e a linguagem XML para descrever as mensagens que podem possuir estruturas complexas (Software, 2003).

Esta tecnologia pode ser empregada para solucionar problemas na integração e na construção de sistemas computacionais distribuídos. Com a utilização do XML-RPC pode-se disponibilizar interfaces de serviços para qualquer aplicação que esteja rodando em qualquer parte de mundo, basta que a aplicação cliente converse e entenda a interface XML-RPC (Alonso et al., 2004). Nas figuras 2.8 e 2.9 são apresentados exemplos de mensagens XML de requisição e de resposta do XML-RPC.

```
POST /RPC2 HTTP/1.0
User-Agent:Frontier/5.1.2(WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length : 181
<?commentstylexml commentstyleversion="1.0"?>
<methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
        <param>
            <value><i4>41</i4></value>
        </param>
    </params>
</methodCall>
```

Fig. 2.8: Mensagem de requisição de um serviço XML-RPC.

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml
Content-length : 158
Date: Fri 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT
<?commentstylexml commentstyleversion="1.0"?>
<methodResponse>
    <methodName>examples.getStateName</methodName>
    <params>
        <param>
            <value><String>41</String></value>
        </param>
    </params>
</methodResponse>
```

Fig. 2.9: Mensagem de resposta de um serviço XML-RPC.

2.3.3 Dispositivos computacionais móveis

Os dispositivos computacionais móveis permitem aos usuários se comunicar utilizando tecnologias de redes sem fio. Estes dispositivos disponibilizam aos usuários portabilidade, mobilidade e conectividade, e graças a estas características um novo paradigma foi criado. "Computação móvel pode ser representada como um novo paradigma computacional que permite que usuários desse ambiente tenham acesso a serviços independentemente de sua localização, podendo inclusive, estar em movimento"(Figueiredo and Nakamura, 2003).

Com a popularização dos dispositivos computacionais móveis, eles se apresentam de formas mais variadas, tais como: laptops, PDAs(*Personal Digital Assistants*), smartphones e telefones celulares, entre outros. Com isso, os serviços agregados a estes dispositivos vão crescendo, bem como o poder de processamento e comunicação. Tudo isso faz com que as tecnologias de redes sem fio tenham que evoluir rapidamente (Figueiredo and Nakamura, 2003).

Como a maioria dos dispositivos móveis não possui ainda um poder de processamento e comunicação das estações de trabalho, isto vem motivando a comunidade científica a buscar soluções para que o desempenho e a qualidade dos serviços oferecidos aos usuários nestes ambientes sejam satisfatórios (França et al., 2001).

A portabilidade, mobilidade e conectividade introduzem restrições aos sistemas e aplicações. Apesar da evolução natural da tecnologia, acredita-se que as limitações dos dispositivos portáteis, como bateria, tamanho do display, capacidade de processamento, capacidade de armazenamento e largura de banda da rede móvel, permanecerão limitados, principalmente se comparados ao ambiente de rede fixa (Satyanarayanan, 2001).

Considerando as limitações dos dispositivos móveis, a compressão de dados se tornou uma solução de vital importância para o sucesso das aplicações desenvolvidas para este tipo de ambiente. Um dos benefícios da compressão de dados, por exemplo, é o aumento da autonomia de energia dos dispositivos, pois a transmissão de dados é o processo de maior consumo de energia (França et al., 2001).

Capítulo 3

Compressão de Dados

Neste capítulo serão definidos os conceitos de compressão de dados e os princípios utilizados por estes métodos para diminuírem o tamanho da informação a ser transferida ou armazenada. Serão definidos também os conceitos de compressão com perda e sem perda de dados e suas características.

Como exemplo, serão apresentados os métodos de compressão de Huffman (Vitter, 1987), LZ77 (Ziv and Lempel, 1977), JPEG (Pennebaker and Mitchell, 1992) e MP3 (Brandenburg, 1999). Depois de mostrar os métodos clássicos serão apresentados métodos específicos de compressão de arquivos XML e suas estratégias para conseguir tornar o tamanho dos documentos XML menores, mostrando também suas vantagens e desvantagens.

3.1 Definição

A compressão de dados começou a chamar atenção da comunidade científica nos últimos 20 anos. Isto pode ser comprovado verificando a quantidade e a qualidade dos textos publicados neste período (Salomon, 2004).

Compressão de dados é o processo de converter um fluxo de entrada de dados (dados originais) em outro fluxo de saída (dados comprimidos), que possui um menor tamanho comparado ao fluxo original. Fluxo de dados pode ser, por exemplo, um arquivo gravado em disco ou dados armazenados na memória (Salomon, 2004).

O fluxo de dados original pode conter diversos formatos de dados: binário, texto, pixel ou outra forma de representação de dados. Este fluxo é emitido por uma fonte de informação que deve ser codificada (representada em outro formato de menor tamanho) antes de ser enviada para as aplicações destinatárias (Salomon, 2004).

Foram projetados muitos métodos de compressão que utilizam estratégias diferentes para se realizar a compressão dos dados, mas todos utilizam um princípio em comum: a retirada dos dados

redundantes. Qualquer coleção de dados desde que não seja uma coleção gerada aleatoriamente, possui um padrão que pode ser explorado para representá-lo em uma forma mais compacta (Salomon, 2004).

Uma das razões para se comprimir dados é que os usuários costumam guardar informações e não gostam de elimina-las. Não importa o quão grande seja o espaço de armazenamento, em algum momento ele será preenchido. E além do mais, quando o usuário está navegando na internet, seja à espera de um download ou apenas querendo visualizar uma página que contenha alguma informação de seu interesse, os usuários não se sentem confortáveis em esperar um longo tempo (Salomon, 2004).

Nos últimos anos houve um considerável aumento no poder de processamento e de armazenamento das máquinas computacionais. Com a disseminação da WWW (*World Wide Web*) inúmeras redes foram interligadas criando um ambiente heterogêneo em disponibilidade de recurso de rede (largura de banda). Podemos encontrar usuários utilizando tanto conexões de banda larga, que atingem *Mbytes* de velocidade, quanto usuários utilizando modens de 56Kbps conectados pela linha telefônica. Não há garantia de qualidade de serviço, por este motivo, o recurso de rede se tornou um dos gargalos na comunicação de dados na WWW e muito esforço vem sendo desprendido com o objetivo de aproveitá-lo ao máximo, excluíndo as informações inúteis ou redundantes (Muller, 1998).

A solução mais explorada para diminuir o impacto de transfêrencia de dados na rede é a compressão de dados. A compressão de dados acarreta um aumento no processamento da informação, pois é aplicado um método (algoritmo) sobre os dados para retirar as informações inúteis ou repetitivas, assim transferindo ou armazenando somente as informações essenciais. Em contrapartida, uma quantidade menor de dados é transmitida ou armazenada, fazendo com que menos recursos sejam utilizados. Existem duas formas de compressão de dados: compressão sem perda e com perda de dados (Muller, 1998), (Salomon, 2004), (Sayood, 2000).

3.2 Compressão sem perda de dados

Na compressão sem perda os dados são comprimidos e ao descomprimir, todas as informações são recuperadas de forma idêntica a original. Este método é utilizado quando a informação original não pode sofrer nenhum tipo de perda no processo de compressão (Salomon, 2004). Os métodos de compressão Huffman e o LZ77 são alguns exemplos de algoritmos com estas caracteristicas que se destacam na literatura.

3.2.1 LZ77

O algoritmo LZ77 (Ziv and Lempel, 1977) foi projetado por Abraham Lempel e Jacob Ziv no ano de 1977. O método LZ77 constrói um dicionário com base na leitura dos dados de entrada, atribuindo um código a cada sequência. Conforme os dados são lidos, ele substitui as sequências que aparecem repetidamente pela posição em que a sequência apareceu anteriormente (Ziv and Lempel, 1977).

O algoritmo define uma estrutura dividida em duas partes: o *search buffer* e o *look-ahead buffer*. O *search buffer* é o dicionário atual, nele estão os símbolos que recentemente serviram de entrada e serão codificados. No *look-ahead buffer* estão os dados que ainda não foram codificados. Na implementação do algoritmo, o *search buffer* possui centenas de *bytes* de tamanhos, enquanto que o *look-ahead buffer* possui dezenas (Ziv and Lempel, 1977).

Considerando a estrutura citada acima é realizada uma busca no *search buffer* a procura da sequência de caracteres presentes no *look-ahead* e quando encontrada é emitida uma saída (posição, tamanho, próximo caractere). A posição representa a posição do começo da palavra dentro da janela onde a sequência começa, o tamanho representa o tamanho da seqüência encontrada e o próximo caractere representa onde vai ser iniciada a busca da próxima sequência dentro da janela. Neste exemplo utilizamos o *search buffer* de 8 *bits* e o *look-ahead* de 4 *bits* (Ziv and Lempel, 1977). Na tabela 3.1 é apresentado um exemplo do algoritmo LZ77.

search buffer	***Look-ahead***	**Entrada**	**Resultado**
	DADA	ACCAACC	(0,0,D)
D	ADAA	CCAACC	(0,0,A)
DA	DAAC	CAACC	(1,2,A)
DADAA	CCAA	CC	(0,1,C)
DADAAC	CAAC	C	(0,1,A)
DADAACCA	ACC		(3,3,EOF)

Tab. 3.1: Exemplo do algoritmo LZ77

3.2.2 Huffman

A codificação de Huffman define códigos para cada símbolo presente no arquivo a ser comprimido. Para os símbolos que possuem maior probabilidade de aparecer são destinados os menores códigos e para os símbolos com menor probabilidade são destinados códigos maiores. A codificação de Huffman é muito eficaz quando há uma diferença muito grande das probabilidades de aparição dos símbolos (Salomon, 2004), (Vitter, 1987).

Exemplo:

Em um texto, as letras A, B, C, D e E aparecem com as freqüências 16, 32, 32, 8 e 8 vezes respectivamente. Para gerar os códigos de Huffman de cada letra seque-se o algoritmo abaixo:

1º passo: As letras são ordenadas com base na suas freqüências.

8(D), 8(E), 16(A), 32(B), e 32(C).

2º passo: As menores freqüências formam o 1º nível da árvore, como mostrado na figura 3.1.

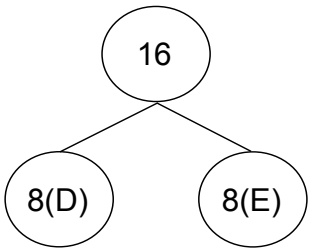

Fig. 3.1: As letras de menor frequência definem os primeiros níveis da árvore de Huffman.

3º passo: A freqüência resultante da soma das duas menores freqüências é adicionada a lista de frequências e a lista é ordenada novamente.

16(D, E), 16(A), 32(B) e 32(C)

4º passo: Repete-se o 2º passo, e a árvore resultante é apresentada na figura 3.2.

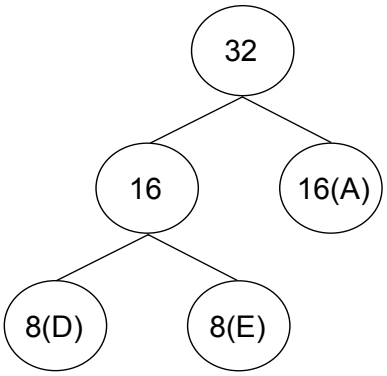

Fig. 3.2: O nó 16(A) que possuí menor frequência é adicionado a árvore.

5º passo: Repete-se o 3º passo.

32(D, E, A), 32(B) e 32(C)

6º passo: Repete-se o 2º passo, e a árvore resultante é apresentada na figura 3.3.

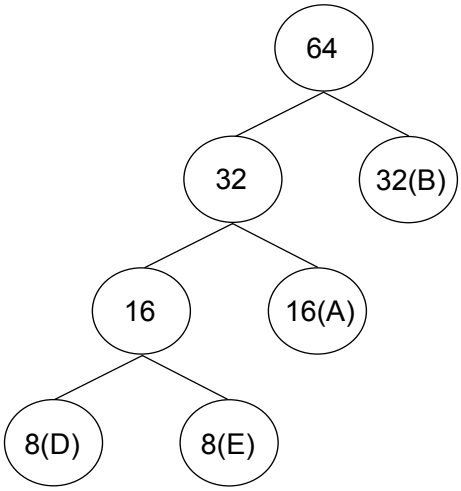

Fig. 3.3: O nó 32(B)que possuí menor frequência é adicionado a árvore.

7º passo: Como só apenas restou um nó, ele é incorporado a árvore, como apresentado na figura 3.4.

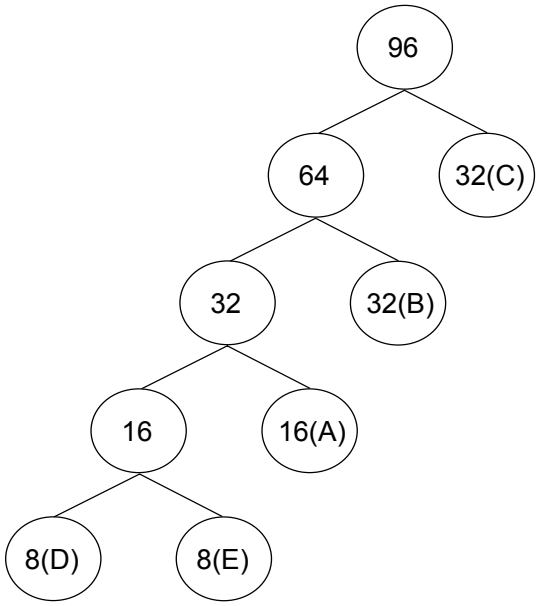

Fig. 3.4: Árvore resultante depois da inserção de todos os nós.

8º passo: Com a árvore construída para cada aresta é atribuído os valores 0 e 1, no exemplo, as arestas da esquerda são atribuídas os valores 0 e as arestas da direta os valores 1, como apresentado na figura 3.5.

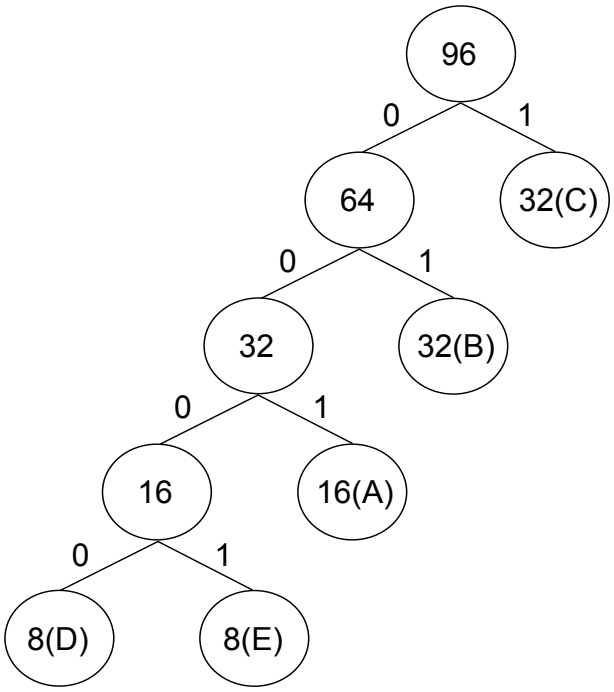

Fig. 3.5: Árvore resultante da codificação de Huffman.

No final dos passos temos que os códigos de Huffman associados a cada letra são:

A: 001 B: 01 C: 1 D: 0000 E: 0001

3.3 Compressão com Perda

Um método de compressão de dados é dito com perda (em inglês *lossy data compression*) quando a informação resultante da descompressão é diferente da informação original, mas não comprometendo o entendimento da informação. Este tipo de compressão é frequentemente utilizado para compactar áudio, imagem e vídeo para a Internet (Salomon, 2008), (Salomon, 2004), (Salomon et al., 2009).

Dependendo da natureza da aplicação os métodos de compressão com perda não são uma solução viável. Existem aplicações onde a perda de informação prejudica seu funcionamento, em contrapartida, existem aplicações onde a perda de informações é tolerada, não influenciando a qualidade do resultado final quando a informação for descomprimida (Salomon, 2008), (Salomon, 2004), (Salomon et al., 2009). O formatos JPEG e o MP3 são alguns dos métodos de compressão com perda disponíveis.

3.3.1 JPEG

Com o passar dos anos, os dispositivos de captura, armazenamento, impressão e de visualização de imagens evoluíram e muitas aplicações de manipulação de imagens digitais foram desenvolvidas. O principal problema encontrado por estas aplicações de manipulação de imagens digitais é a grande quantidade de dados requeridos para representar uma imagem. Em muitos casos a utilização de imagens digitais não se tornou viável devido ao alto custo de transmissão e armazenamento. Para solucionar este problema foram criadas técnicas de compressão de imagens (Pennebaker and Mitchell, 1992), (Wallace, 1991).

A compressão de imagens conseguiu diminuir a grande quantidade de informação necessária para representar uma imagem digital, viabilizando a tecnologia. O problema é que cada corporação criou o seu próprio formato de compressão e com o desenvolvimento de aplicações de transmissão e armazenamento de imagens, as ferramentas desenvolvidas não eram compatíveis com todos os formatos criados. Surgiu então a necessidade de criação de um padrão de compressão de imagens digitais para que a interoperabilidade entre equipamentos e aplicações de diferentes fabricantes fosse possível (Pennebaker and Mitchell, 1992), (Wallace, 1991).

Neste cenário foi desenvolvido o JPEG (*Joint Photographic Experts Group*). O formato JPEG é um método de compressão de imagens desenvolvido em conjunto pela ISO (*International Organization for Standardization*) e pela CCITT (*Consultative Committee for International Telegraphy and Telephony*) com o objetivo de estabelecer o primeiro padrão internacional de compressão de imagens digitais (Pennebaker and Mitchell, 1992), (Wallace, 1991).

Por ser um padrão de compressão de imagem, o JPEG facilitou, entre outras coisas, o compartilhamento de imagens entre diferentes aplicações. Essa característica tornou-se muito importante, pois aplicações que manipulam imagens se tornaram cada vez mais interoperáveis e interconectadas (Pennebaker and Mitchell, 1992), (Wallace, 1991).

3.3.2 MP3

Com a popularização da Internet nos anos 90, todo o tipo de informação começou a ser disponibilizada na rede. A troca de arquivos de áudio, principalmente músicas na Internet, era um dos maiores desejos dos usuários. Uma música de 5 minutos com som *stereo* possuía em média 50 *MBytes* de tamanho, e considerando que os modens na época atingiam velocidades de 56000 *bits* por segundo, transmitir uma música de 5 minutos levaria aproximadamente 2 horas e 20 minutos. Em meio a este ambiente surgiu o MP3, com o objetivo de reduzir o tamanho e viabilizar a troca de arquivos de áudio na Internet (Ruckert, 2005),

O formato MP3 (MPEG-1/2 *Audio Layer* 3) é um padrão aberto de compressão de áudio com

perda de dados quase imperceptível ao ouvido humano. Uma das estratégias adotada para atingir uma alta taxa de compressão é a retirada de frequências de som que são imperceptíveis ao ouvido humano. O ouvido humano é capaz de ouvir sons nas frequências entre 20 Hz e 20 KHz, então frequências fora desta faixa podem ser excluídas que não acarretará perda de qualidade perceptível aos ouvidos humanos (Ruckert, 2005), (Brandenburg, 1999).

No formato MP3 o usuário pode definir vários níveis de taxa de compressão a ser atingida, quanto maior a taxa de compressão, maior é a perda de informação, consequentemente menor é a qualidade do arquivo de áudio resultante (Brandenburg, 1999).

3.4 Compressão de Documentos XML

As coorporações nos dias de hoje utilizam formatos simples de dados, frequentemente projetados para racionalizar o uso eficiente de recursos computacionais pelas aplicações. Estes formatos são atendem as necessidades específicas de cada aplicação, com isso a interoperabilidade entre diferentes aplicações é comprometida. Atualmente, com a necessidade de interligar diferentes aplicações, a linguagem XML tem se tornando cada vez mais utilizada, por ter sido adotada como padrão de comunicação entre sistemas heterogêneos.(Quin, 2010)

Quando os formatos específicos de cada aplicação são transferidos para o formato XML, há um aumento significativo no tamanho do arquivo XML resultante, principalmente pelo fato de que as *tags* do XML são repetitivas (Liefke and Suciu, 2000).

Devido ao fato da linguagem XML ser muito redundante, surgem sérias preocupações sobre a utilização da linguagem, criando impacto sobre processamento de consultas e na transferência dos dados. Existem muitos formatos de dados tanto para transferência quanto para armazenamento que são muito mais econômicos, podendo ser citado o formato binário como exemplo. O formato autodescritivo traz flexibilidade, mas compromete a eficiência (Liefke and Suciu, 2000).

Por exemplo, ambientes móveis em computação sofrem de sérias restrições de recurso de processamento, recurso de memória e autonomia de energia se comparado aos dispositivos fixos. Utilizar um formato que acarrete um menor uso dos recursos de rede, menor dispêndio de energia, processamento e de memória se torna crucial para o sucesso das aplicações. Considerando ambientes móveis como sendo aqueles onde existam dispositivos móveis que matem comunicação com uma central ou entre outros dispositivos por uma rede sem fio (França et al., 2001).

Ainda se tratando de ambientes móveis, a transferência de arquivos assume um papel muito importante para o desenvolvimento de aplicações, pois o dispêndio de energia é um fator muito importante na autonomia dos dispositivos e é sabido que o processo de maior consumo de energia é a transmissão de dados (França et al., 2001).

Existem várias implementações de compressores específicos para XML na literatura. Eles se baseiam em características específicas da linguagem XML para atingir melhores taxas de compressão. Os métodos de compressão de XML estão divididos em 3 categorias: métodos baseados em redundância, métodos baseados no *schema* e métodos híbridos.

Métodos de compressão baseados em redundância de dados são utilizados a algum tempo. Este tipo de compressão não possui nenhum tipo de conhecimento prévio sobre a estrutura e nem do conteúdo do documento XML. Eles se baseiam apenas na busca por redundância de grupos de dados e realizam a troca destes grupos de dados por códigos de menor tamanho. O compressor WBXML (*WAP Binary Extensible Markup Language*) utiliza esta técnica para realizar a compressão. Em geral, a taxa de compressão é mais satisfatória em arquivos grandes, tornando menos efetiva quando o tamanho dos arquivos vai se tornando menor (Cokus and Winkowski, 2002).

Os métodos baseados em *Schema* são cada vez mais utilizados e sua premissa está no conhecimento sobre o conteúdo e sobre a estrutura do documento. As *tags* não são comprimidas e o documento é reconstruído usando conhecimento da sua estrutura. Este método permite que as *tags* e o conteúdo sejam recuperados sem que o arquivo seja descomprimido, ou seja, os dados podem ser recuperados sem que o arquivo comprimido seja transformado no arquivo original novamente (Cokus and Winkowski, 2002), (Imamura and Maruyama, 2001).

Os híbridos aplicam a combinação dos métodos baseado em redundância e no *schema*, o método XMill é um exemplo. A compressão baseada no *schema* é aplicado primeiro, depois é aplicada a compressão baseado em redundância (Liefke and Suciu, 2000), (Cokus and Winkowski, 2002).

Nas próximas seções serão abordadas em maiores detalhes as técnicas utilizadas pelos compressores WBXML, XMill e EXI , destacando suas características principais.

3.4.1 WBXML

O formato WBXML (*WAP Binary Extensible Markup Language*) é resultado de um trabalho contínuo da *Open Mobile Alliance*, grupo que desenvolve padrões para a indústria de telefonia móvel, para desenvolver um padrão que permite que documentos XML sejam transmitidos de maneira compacta sobre ambientes móveis. O WBXML é uma representação binária compacta da linguagem XML que visa reduzir o tamanho de transmissão de documentos sem perda de dados (Martin and Jano, 1999).

Os argumentos para se transformar documentos XML em um formato binário são fundamentados na necessidade de se utilizar uma menor largura de banda e espaço de armazenamento e os interpretadores de documentos em formato binário são mais eficientes que os interpretadores em formato texto, já que as máquinas lidam melhor com números que texto (B'far, 2004).

O WBXML representa *tags* com um número de índice de uma tabela para evitar a transmissão da

mesma tag centenas de vezes. A tabela que mapeia as *tags* em índices é fornecida separadamente do próprio documento comprimido, mas ela não precisa ser enviada através da rede para que os arquivos sejam descomprimidos, já que ela é conhecida previamente tanto por parte do servidor quanto por parte do cliente.

O maior benefício do WBXML é que ele utiliza o SAX (*Simple API for XML*) para interpretar o documento binário com muita rapidez, dispensando a transformação do arquivo binário no arquivo XML para que os dados sejam extraídos. Para descomprimir o arquivo WBXML são realizadas buscas na tabela de mapeamento (Zhang, 2009).

Uma das preocupações sobre as limitações do WBXML é o espaço de códigos. O número de entradas na tabela de mapeamento para *tags* e atributos é muito limitada, comprometendo sua utilização para documentos muito complexos (Zhang, 2009).

Os dispositivos que utilizam o WBXML não transformam o arquivo WBXML em XML para extrair ou processar as informações. As vantagens são que o formato WBXML é menor e utilizando o SAX para interpretar diretamente o arquivo WBXML, ele apresenta melhor desempenho. Mas por outro lado, ele perde em interoperabilidade e legibilidade, pois apenas dispositivos que processem o formato WBXML podem se comunicar e o arquivo resultante da compressão não é legível aos desenvolvedores (Martin and Jano, 1999). Com isso, o formato XML que foi concebido para prover interoperabilidade entre sistemas heterogêneos é substituído por outro formato que não possui as mesmas características.

3.4.2 XMill

O compressor XMill foi a primeira proposta concisa para compressão de arquivos XML na literatura que realiza a separação da informação estrutural dos dados durante o processo de compressão. Experimentos feitos utilizando o XMill mostram que ele oferece taxas de compressão satisfatórias.

O compressor GZip atinge boas taxas de compressão quando comprime documentos XML, mas quando aplicado no documento específico da aplicação consegue atingir taxas ainda melhores. O compressor XMill no entanto, consegue reduzir o documento XML pela metade em comparação ao documento específico da aplicação comprimido pelo GZip (Liefke and Suciu, 2000).

Para reduzir o tamanho dos arquivos XML, o XMill separa as informações estruturais (*tags*) dos dados (valores das informações) que o documento possui. No final deste processo de separação da informação estrutural dos dados, o compressor utiliza a biblioteca ZLIB (Zlib, 2010), que disponibiliza a funcionalidade GZip, para melhorar a taxa de compressão (Ng et al., 2006), (Liefke and Suciu, 2000).

Para realizar a interpretação do documento XML é utilizando o SAX (*Simple API for XML*), que envia os tokens para o processador de caminho do XMill. Todo token seja ele tag, atributo ou valor,

que formam o documento XML é atribuído a um container. Os valores contidos no documento XML são separados em diversos contentores de acordo com as expressões dos containers e cada container é comprimido independentemente. Os três princípios básicos de funcionamento do XMill são:

- Separar a estrutura dos dados, separando as marcas e os atributos que constituem a estrutura dos valores dos atributos e texto dos elementos;
- Agrupar os dados com significado semelhante num mesmo container, sendo cada container codificado separadamente para uma codificação mais eficaz;
- Aplicar codificadores distintos para os vários containers, uma vez que diferentes containers podem ter diferentes tipos de dados, desta forma é possível escolher o compressor mais eficaz para cada tipo;

O núcleo do XMill é o processador de caminho. É ele quem determina como mapear os valores para os containers, para cada valor encontrado, o processador de caminho compara seu caminho com cada caminho presente no container de expressões, caso não esteja presente é criado um novo container (Liefke and Suciu, 2000).

Os containers são mantidos em janelas de 8 *MBytes* de tamanho, podendo ser configurado pelo usuário. Quando o container é completamente preenchido, ele é comprimido utilizando o compressor GZip. Os dados comprimidos são gravados em disco e a compressão recomeça. No final do processo, o arquivo de entrada está comprimido em vários containers separados (Liefke and Suciu, 2000).

O compressor XMill apresenta uma série de desvantagens e limitações. Uma das desvantagens é o fato de não ser capaz de trabalhar em conjunto com um processador de consulta, tendo que ser feita a descompressão do documento XML para que as informações sejam acessadas. Outra desvantagem é não atingir uma taxa de compressão satisfatória em arquivos menores que 20 *Kbytes* de tamanho (Liefke and Suciu, 2000). Uma das limitações do XMill é ser incapaz de trabalhar com tipo List, o compressor não compreende que o atributo possui multiplos valores e assume que o conjunto de valores é apenas um. Outra limitação é quando o compressor encontra um caracter "/", ele gera a exceção *tokenization errors*, indicando que ele esperava o fim de um elemento (Snyder, 2010).

3.4.3 EXI

EXI (*Efficient XML Interchange*) é o resultado de um extenso trabalho realizado pelos W3C *Binary Characterization* (XBC) e *Efficient XML Interchange (EXI) Work Groups*. A XBC foi designada a investigar os custos e benefícios de uma forma alternativa de representação da linguagem XML. Baseada nas recomendações da XBC, a EXI *Work Groups* foi designada a medir, avaliar e comparar a

performance de varias tecnologias XML e então formulou uma nova especificação de formato XML para a W3C (Schneider and Kamiya, 2008).

O compressor EXI é um formato genérico de troca de dados que funciona em uma variedade de aplicações. Ele foi projetado para otimizar performance enquanto reduz a utilização de largura de banda, aumenta a vida útil da bateria do dispositivo, utiliza menos processamento e memória. Para realizar a compressão, ele funciona com ou sem o *Schema*. Caso seja utilizado o *Schema*, as taxas alcançadas de compressão são maiores (Schneider and Kamiya, 2008).

O EXI foi projetado para alcançar aplicações de vários segmentos, ser flexível e apresentar boa performance. Isto graças a unificação de conceitos da teoria da linguagem formal e da teoria da informação dentro de um único algoritmo. O algoritmo pesquisa no texto as informações que ocorrem em maior probabilidade e atribui a essas informações códigos de menor tamanho. O algoritmo funciona para qualquer linguagem que possa ser descrita por uma gramática, por exemplo, XML, JAVA, HTTP, mas no entanto, ele foi projetado para atender especificamente a linguagem XML (Schneider and Kamiya, 2008).

Para atingir o grau de compressão desejada, o EXI coleta o fluxo de dados XML e utilizando uma gramática, realiza o mapeamento do fluxo para um fluxo de tokens de menor tamanho e de menor entropia. O codificador recolhe o fluxo de tokens e aplica o algoritmo de Huffman. Caso uma maior compressão seja desejada, é aplicado um compressor mais sofisticado para atingir uma taxa de compressão maior. O EXI tem sido adotado por uma grande variedade de aplicações e plataformas, o mercado de telefones celulares, veículos, aeronaves, satélites, PDAs, aplicações servidoras, *Web Services* (Schneider and Kamiya, 2008).

Uma das variações do compressor EXI é o EXI *Compression*. Sua variação consiste em organizar blocos de dados em grupos de baixa entropia, chamados de canais, que se tornam muito favoráveis a métodos de compressão, alcançando taxas mais altas de compressão. Ele também pode combinar canais de dados para evitar o uso excessivo de recurso de processamento, atingindo um melhor desempenho (Schneider and Kamiya, 2008).

3.5 Impactos da Compressão de Dados

A compressão de dados, especialmente de XML, tem como principal objetivo a redução de utilização de recursos computacionais, sejam eles de rede ou de armazenamento. Por exemplo, para um cliente que está conectado a uma rede de baixa velocidade, comprimir os dados que irão trafegar na rede pode ser uma boa solução para reduzir o tempo de espera por um serviço requisitado (Geer, 2005).

Mas a utilização de outro formato mais compacto para a transferência de dados acarreta impac-

tos nas aplicações cliente/servidor. É necessário habilitar os lados envolvidos na transferência dos dados para que sejam capazes de processar o novo formato. Em aplicações que utilizam XML para transferir os dados, o principal impacto na adoção de outros formatos mais compactos é perda da interoperabilidade (Geer, 2005).

Algumas soluções podem ser adotadas para diminuir o impacto na adoção de formatos mais compactos de dados. Uma das soluções é a utilização de *proxies* reversos. *Proxy* reverso é um *proxy* que concentra o fluxo de dados da rede externa para um ou mais servidores específicos na rede interna. Desde modo, quando um fluxo de dados externo for endereçado a um cliente na rede interna, este fluxo passa pelo *proxy* reverso que realiza a descompressão dos dados e envia para o cliente. Quando o cliente enviar dados, o *proxy* reverso assume a função de um *proxy* convencional, onde os dados passam por ele, comprimindo e enviando ao seu destino (Kew, 2004).

Com a utilização de *proxies*, o cliente não necessita de qualquer tipo de intervenção por parte dos desenvolvedores para tornar possível o emprego de outro formato de dados, o cliente continua a utilizar a XML como linguagem de transferência de dados, o papel do *proxy* é tornar a compressão um processo transparente para os clientes, como mostra a figura 3.6.

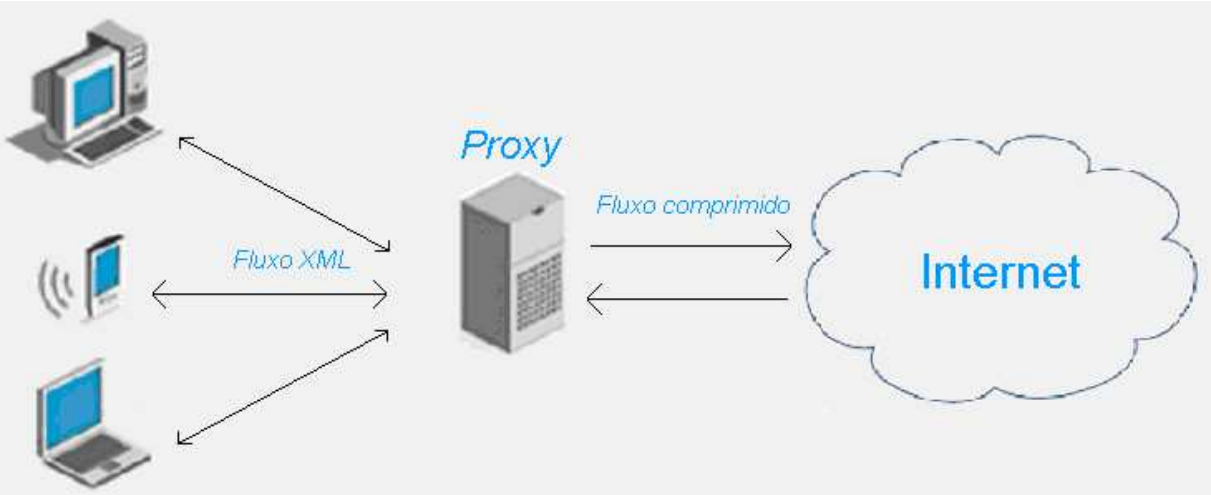

Fig. 3.6: *Proxy* reverso no lado cliente.

Para tornar esse processo transparente também para o lado servidor, pode-se adotar o mesmo mecanismo, eliminando assim a necessidade de qualquer tipo de alteração na aplicação servidora, como mostra a figura 3.7.

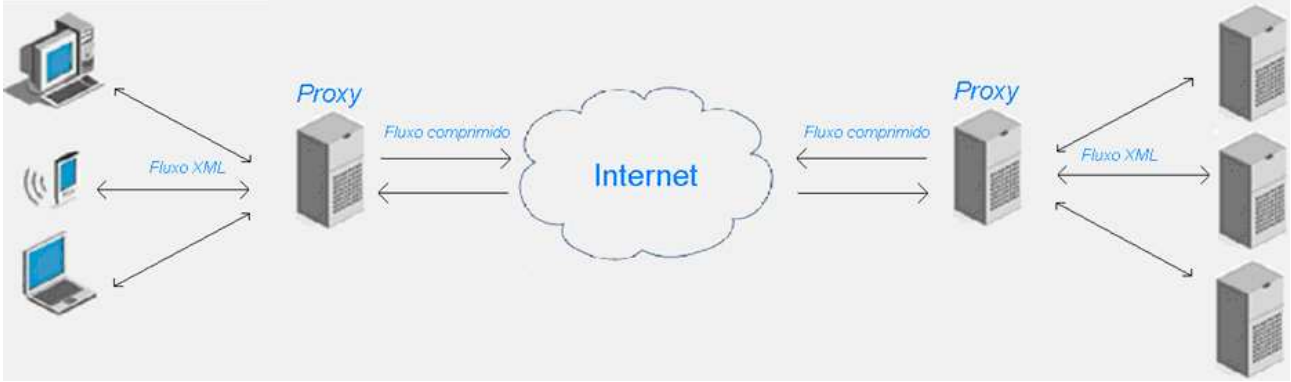

Fig. 3.7: *Proxy* reverso no lado cliente e servidor.

Se as soluções abordadas acima forem aplicadas a interoperabilidade é preservada, já que os *proxies* apenas transformam os dados para que sejam transportados, retirando do conhecimento das aplicações cliente/servidor o novo formato.

Outro impacto criado pela utilização de um formato de dados mais compacto é o custo da compressão e descompressão dos dados. Para que os dados sejam enviados pela rede, deve-se comprimi-los e para obter as informações contidas no fluxo de dados, deve-se descomprimi-los, acarretando assim um aumento na utilização do recurso de processamento, tanto do cliente quanto do servidor, desta maneira, refletindo no tempo de envio dos dados.

Para que o emprego dos formatos de compressão de XML tenha um resultado positivo, o desenvolvedor deve avaliar o ambiente onde sua aplicação está inserida. Tem de ser avaliado se o *Tempo de Compressão* + *Tempo de Envio dos dados comprimidos* + *Tempo de Descompressão* é menor que o *Tempo de Envio dos dados originais*. Esta avaliação pode ser muito complexa de ser realizada, já que os clientes podem possuir diferentes capacidades de processamento e estarem conectados a links de rede com diferentes larguras de banda, e se considerarmos ainda que os *links* onde os clientes estejam conectados podem variar de capacidade de transmissão ao decorrer do tempo, estando menos ou mais congestionados, esta estimativa fica ainda mais complexa.

Outro aspecto que pode ser considerado é a limitação do poder de processamento e memória, como é o caso de alguns dispositivos móveis (celulares, *pda's*, *palmtops*, etc.). Utilizando um formato mais compacto de dados, os clientes teriam de descomprimir os dados para utilizá-los, o que acarretaria uma maior utilização de processamento e de memória, debilitando o desempenho da aplicação e dos dispositivos. Nas soluções abordadas pelas figuras 3.6 e 3.7, este custo de processamento recai sobre os *proxies*. Estas soluções tiram dos clientes e/ou servidores a responsabilidade de comprimir e descomprimir os dados, não os sobrecarregando.

Nas soluções das figuras 3.6 e 3.7, considera-se que os clientes estão localizados próximos uns aos outros, tornando viável a utilização de um *proxy*. Caso os clientes e servidores estiverem localizados

em qualquer parte do mundo, conectados pela Internet, seria necessária a criação de um *proxy* para cada cliente, inviabilizando assim as soluções. Neste caso, os desenvolvedores terão de habilitar as aplicações clientes a manipular o novo formato de dados, compromentendo desta maneira a interoperabilidade. Por ser uma linguagem padrão com o objetivo de tornar interoperável as aplicações de diferentes tecnologias, uma vasta quantidade de linguagens de programação dão suporte ao formato XML, (PHP, C++, C, Perl, JAVA, etc). Para os novos formatos de dados nem toda linguagem oferece suporte, o que limita e traz problemas aos desenvolvedores de aplicações.

Os formatos EXI, WBXML, XMill e o Gzip não possuem suporte de todas as linguagens de programação. O formato EXI possui suporte para as linguagens Java (J2EE, J2SE e J2ME), .NET, C e C++. Possui também um kit para integração ao Axis 1.x, Axis2, .NET Windows Communication Foundation (WCF) e WebLogic versões 9.1 e 9.2 (AgileDelta, 2010). O formato WBXML além de possuir suporte para as linguagens C#, C, .NET, Perl, C++ e Java possui suporte também para o padrão de comunicação SyncML(Synchronization Markup Language) para sincronização de informações. O formato XMill possui apenas suporte para as lingugens C e C++. O formato Gzip, diferentemente dos outros formatos, possui suporte para quase todas as linguagens, por ser mais difundido e estar em um estágio de maturidade maior. O formato GZip também é integrado aos navegadores(*Browsers*) e ao protocolo SOAP (Goodman, 2003).

Capítulo 4

As Novas Abordagens de Compressão

Neste capítulo são propostas duas novas abordagens de compressão de documentos XML. As abordagens foram denominadas de: Proposta *Schema* e Híbrida. Foram realizados testes para avaliar os resultados nos quesitos de taxa de compressão, tempo de compressão e utilização de memória. Os resultados obtidos foram comparados aos métodos de compressão XMill, WBXML, EXI e EXI *Compression*.

A Proposta *Schema* consiste em um método de compressão baseado na estrutura do documento XML para retirar redundâncias das *tags* e não realiza qualquer tipo de compressão dos dados e nem mesmo das *tags*. O resultado da compressão da Proposta *Schema* é um arquivo XML de menor tamanho.

A Proposta Híbrida é uma variante do método *Schema*. Esta variante consiste em aplicar um compressor de dados (GZip) ao resultado da abordagem anterior. Com a aplicação do compressor GZip, além da retirada da redundância das *tags* que é explorada pela Proposta *Schema*, a redundância dos dados presentes no documento também é retirada, com o objetivo de atingir uma maior taxa de compressão.

A seguir descrevemos com mais detalhes as Propostas *Schema* e Híbrida.

4.1 Proposta *Schema*

A Proposta *Schema* consiste na compressão dos arquivos XML baseado na sua estrutura, retirando a redundância das *tags* que estão no mesmo nível da árvore, sem comprimir os dados nem as *tags* presentes no documento. Para demonstrar como essa redundância ocorre, apresentaremos na figura 4.1 um exemplo de um trecho de um documento XML:

```
<clientes>
    <cliente>
        <empresa>Empresa1</empresa>
        <tipo>Tipo1</tipo>
        <nome>Pessoa1</nome>
    </cliente>
    <cliente>
        <empresa>Empresa2</empresa>
        <tipo>Tipo2</tipo>
        <nome>Pessoa2</nome>
    </cliente>
    <cliente>
        <empresa>Empresa3</empresa>
        <tipo>Tipo3</tipo>
        <nome>Pessoa3</nome>
    </cliente>
</clientes>
```

Fig. 4.1: Trecho de um documento XML.

O documento da figura 4.1 possui uma tag raiz <clientes> que possui a tag <cliente>. A tag <cliente> possui as *tags* <empresa>, <tipo> e <nome> e cada uma dessas *tags* possuem seus dados. Para uma melhor visualização da estrutura, o XML acima pode ser representado em forma de árvore, apresentado na figura 4.2.

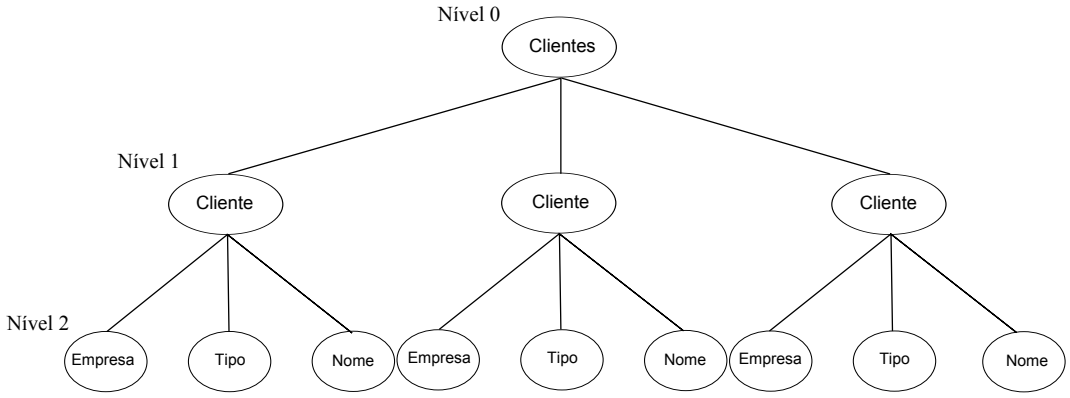

Fig. 4.2: Árvore de representação do documento XML da figura 4.1.

Analisando a estrutura da árvore que representa o documento XML, notamos que os nós de mesmo nível da árvore XML possuem as mesmas *tags*. Com isso, muita informação redundante é enviada pela rede ou é armazenada. É na eliminação da redundância das *tags* de mesmo nível que a proposta se apóia para realizar a redução do tamanho dos documentos XML. Aplicando a proposta para que a redundância das *tags* seja eliminada em cada nível da árvore, os nós de mesmo nível se tornarão

apenas um único nó. Consequentemente, as *tags* repetidas são eliminadas e as informações contidas nos nós serão armazenadas neste único nó resultante. Um detalhe deve ser considerado no processo de armazenamento das informações no nó resultante. As posições dos dados é de extrema relevância, pois é baseada nesta disposição que será feita a descompressão do arquivo. A árvore da figura 4.3 representa o resultado da compressão da estrutura do documento XML da figura 4.1.

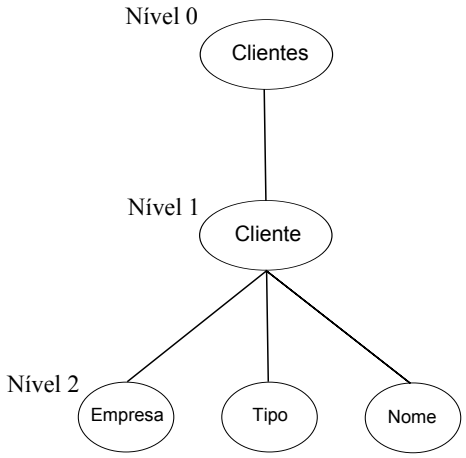

Fig. 4.3: Estrutura do documento XML comprimido.

Esta é a nova representação da árvore XML que nos permite eliminar as *tags* repetidas que aparecem nos níveis da árvore XML. A figura 4.7 mostra como ficam a disposição dos dados contidos nas *tags*.

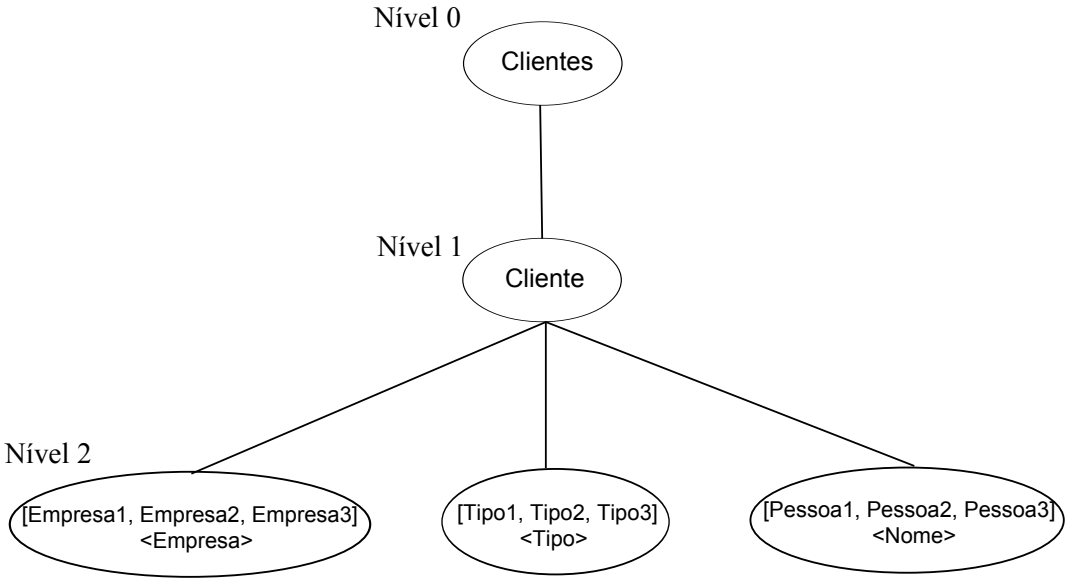

Fig. 4.4: Disposição dos dados na árvore comprimida.

Os nós que contém os dados agora possuem todos os dados contidos nos nós de mesmo nível, e como foi mencionado, a ordem dos dados é mantida para que o processo de descompressão seja realizado com sucesso, recuperando de forma idêntica o XML original. Por exemplo, notamos que no nó resultante que representa a tag <empresa> temos agora os dados [Empresa1, Empresa2, Empresa3]. Se desejarmos recuperar a informação tipo da empresa "Empresa1", devemos realizar a leitura da informação que está presente na primeira posição do nó que possui as informações da tag <tipo>. Assim deve-se proceder com todos os nós resultantes, para que as informações sejam recuperadas. Um exemplo de algoritmo em pseudocódigo para a compressão do documento XML é apresentado nas figuras 4.5 e 4.6.

```
Compressor(raiz)

raiz = LerRaiz();
filhosRaiz = raiz.filhos;
se filhosRaiz.numeroFilhos <> 0 então
    numeroFilhos = filhosRaiz.numeroFilhos;
    para i=0 até i<numeroFilhos faça
        PegarDados(filhosRaiz[i]);
        se filhosRaiz[i].numeroFilhos <> 0 então
            Compressor(filhosRaiz[i]);
        fim
    fim
fim
PegarDados(filhosRaiz);
```

Fig. 4.5: Algoritmo *Compressor* para compressão de documentos XML.

```
elementos = no.elementos;
se elementos.numero <> 0 então
    numeroElementos = elementos.numero;
    para i = 0 até i< numeroElementos faça
        ArmazenaEstrutura(elementos.numero[i]);
    fim
fim
```

Fig. 4.6: Algoritmo do método *PegarDados*.

Depois da aplicação do algoritmo da Proposta *Schema* no documento XML, como resultado obtemos um outro documento XML, que é apresentado na figura 4.7.

```
<clientes>
    <cliente>
        <empresa>[Empresa1, Empresa2, Empresa3] </empresa>
        <tipo>[Tipo1, Tipo2, Tipo3] </tipo>
        <nome>[Pessoa1, Pessoa2, Pessoa3] </nome>
    </cliente>
</clientes>
```

Fig. 4.7: Documento XML resultante da compressão *Schema*.

Nota-se que houve uma redução considerável no tamanho do arquivo XML, principalmente no uso das *tags*. O arquivo compactado reduz apenas o número das *tags* utilizadas, permitindo que os documentos XML sejam transmitidos ou armazenados utilizando menor recurso de rede e armazenamento. Os dados não sofrem nenhum tipo de alteração, ou seja, ficam intactos assim como as *tags*, já que não há nenhum tipo de compressão das informações presentes no documento XML. A compressão se apóia unicamente na eliminação das *tags* que estão presentes no mesmo nível da árvore. Os resultados obtidos em relação a taxa de compressão, tempo de compressão e utilização de memória serão apresentados no capítulo 5.

Esta abordagem, diferentemente dos outros compressores WBXML, XMill e EXI, gera a mesma representação do arquivo XML em outro arquivo XML de menor tamanho. Isto é uma vantagem, pois a característica auto-descritiva da linguagem XML é mantida, e as ferramentas que manipulam XML, como os processadores de XML, podem ainda ser utilizados. Como nenhum formato novo é criado do resultado da compressão, a comunicação entre os sistemas heterogêneos se mantém na linguagem XML, não afetando os requisitos de interoperabilidade que é uma das características da linguagem XML(Geer, 2005).

A posição da informação na nova estrutura de dados indica a que nó da árvore XML original ela pertence. Por exemplo, a figura 4.8 representa um documento XML compactado, a estrutura da árvore é composta de um nível chamado clientes e possui um nó filho cliente, que por sua vez, possui 3 nós filhos denominados: empresa, tipo e nome. O número de filhos do nó cliente (no caso é 3) indica que no documento original o elemento <cliente> possui 3 elementos, sendo eles os elementos <empresa>, <tipo> e <nome>. Analisando mais profundamente os nós filhos do nível cliente, vemos que na estrutura que armazena os dados, existem 3 posições, indicando que o elemento <clientes> possui 3 elementos <cliente>.

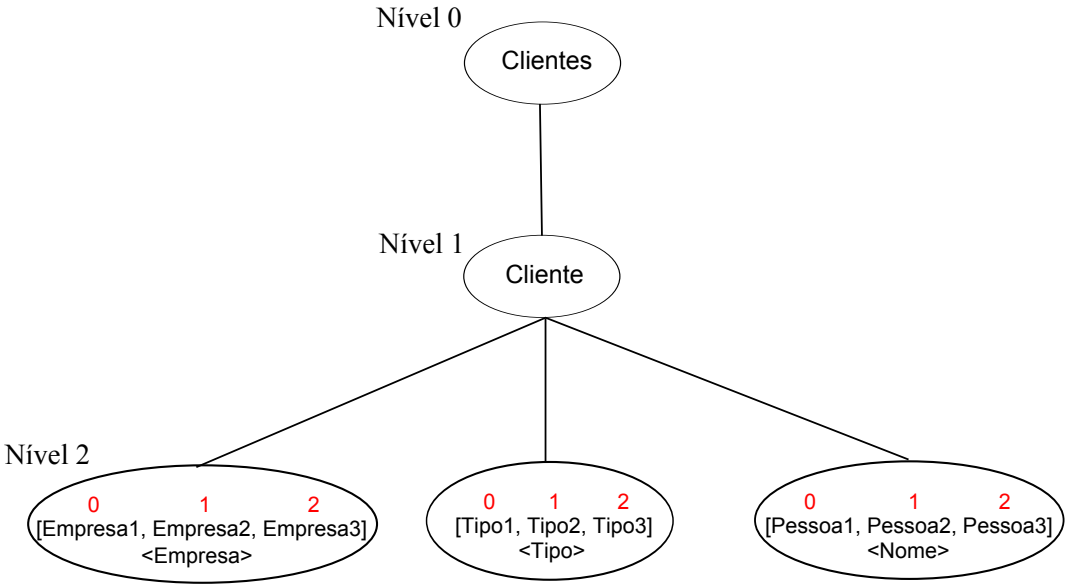

Fig. 4.8: Árvore de um documento XML apresentado as posição dos dados para a descompressão.

Aliado a estas informações, para que a descompressão seja realizada, foi mantida a ordem em que os dados aparecem no documento XML. Esta ordem não é necessariamente uma ordem cronológica de aparecimento dos dados, mas uma ordem para indicar a quem o dado pertence. Por exemplo, voltando na 4.8, a estrutura de dados possui 3 posições, indicando a presença de 3 elementos cliente e cada elemento cliente possui 3 elementos empresa, tipo e nome. O dado Empresa1 está presente no nó empresa e na posição 0 da estrutura de dados, isto indica que todo dado presente na posição 0 das estruturas de dados pertencem a um mesmo cliente, ou seja, na descompressão do documento XML, o documento original possui um cliente com os dados <empresa>Empresa1</empresa>, <tipo>Tipo1</tipo> e <nome>Pessoa1</nome>.

Assim é feito para todas as outras posições do documento compactado resultando no documento XML original. Fica mais claro o processo de descompressão se for analisado do seguinte modo: se a informação Empresa2 for trocada de posição na estrutura de dados com a informação Empresa1, todas as informações das estruturas de dados teriam de ser trocadas, passando as informações da posição 0 para a posição 1 e vice e versa. Este processo garante a obtenção do documento original correntamente, ver figura 4.9.

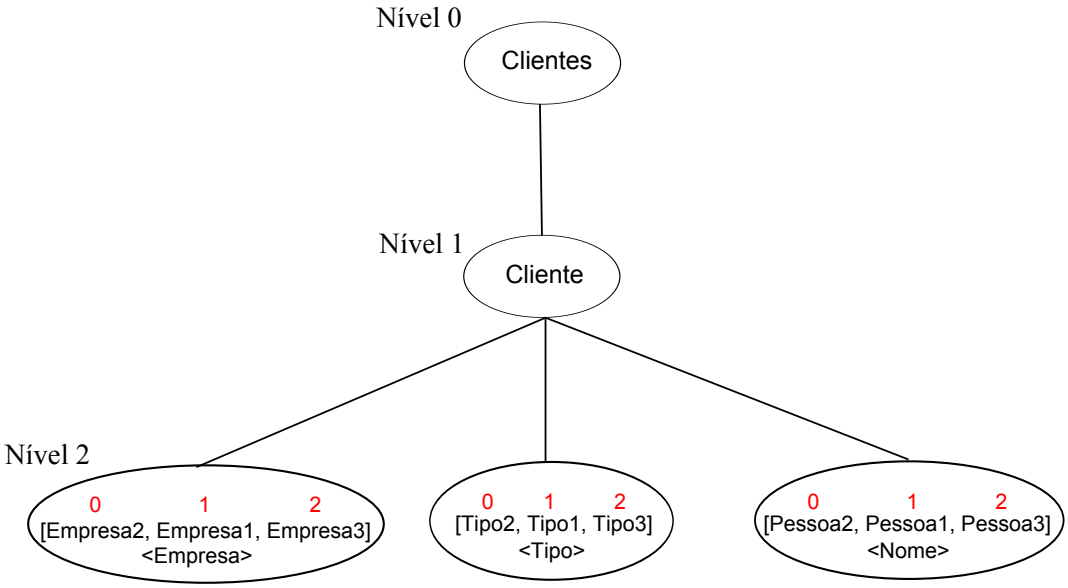

Fig. 4.9: Árvore de um documento XML apresentado a alterção das posição dos dados.

Na figura 4.10 é apresentado um algoritmo em pseudocódigo para realizar a descompressão.

```
raiz = LerRaiz();
filhosRaiz = raiz.filhos;
se filhosRaiz.numeroFilhos <> 0 então
    numeroFilhos = filhosRaiz.numeroFilhos;
    para i=0 até i<numeroFilhos faça
        se filhosRaiz [i].numeroFilhos <> 0 então
            Descompressor(filhosRaiz [i]);
        fim
    fim
fim
InstanciarElementosFolhas(filhosRaiz);
```

Fig. 4.10: Algoritmo *Descompressor* para descompressão de documentos XML.

Esta proposta permite também a criação de uma linguagem de consulta, ou seja, uma linguagem que permita o acesso aos dados do documento compactado sem a necessidade de restabelecer o arquivo original para que as informações sejam acessadas. Criando uma linguagem de consultas para a Proposta *Schema* é possível retirar dados específicos de interesse do desenvolvedor, sem a necessidade da descompressão total do documento. Isto é possível graças ao arquivo resultante da compressão ainda ser um arquivo XML e as informações estarem dispostas em uma ordem definida. O diagrama da figura 4.11 representa uma visão modular da recuperação dos dados.

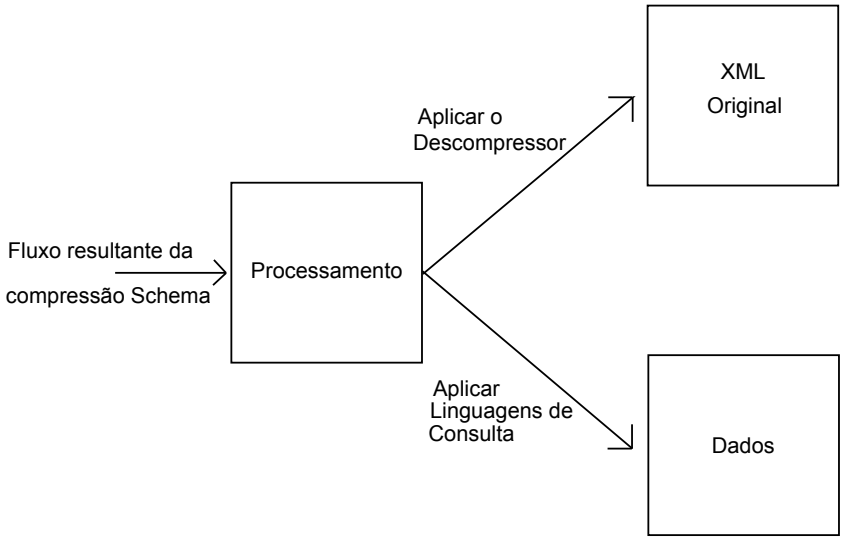

Fig. 4.11: Visão modular de recuperação dos dados da proposta *Schema*.

Para que a compressão seja realizada, o *schema* utilizado pelo documento XML deve ser modificado com o objetivo de comportar a nova estrutura de dados. Cada nó terá o comportamento de uma lista ou vetor, onde cada posição possui as informações que pertenciam ao mesmo nível da árvore do XML original. O novo *schema* tem que estar presente nos sistemas envolvidos na comunicação, para que o processo de compressão e descompressão seja realizado.

O novo *schema* é apenas uma alteração do *schema* original e existem duas formas para que seja feita a adoção do *schema* modificado sem que a interoperabilidade seja prejudicada. A primeira opção é a aplicação de origem enviar o *schema* modificado para o destino da comunicação. Esta solução acarreta um maior uso dos recursos de rede, pois quando ocorrer a primeira comunicação, mais informações terão de ser enviadas. Outra solução são as próprias aplicações realizarem a adaptação do *schema*. Esta solução ocasiona menos tráfego na rede, mas acarreta mais processamento do dispositivo de origem.

4.2 Proposta Híbrida

A Proposta Híbrida é uma variação da proposta *Schema*. Esta variação consiste na aplicação de um compressor de dados no resultado da compressão da proposta *Schema*(ver 4.12). O compressor de dados escolhido foi o GZip, que é amplamente utilizado e difundido na literatura (Snyder, 2010). A figura 4.12 apresenta o diagrama de alteração da proposta *Schema* para a proposta Híbrida.

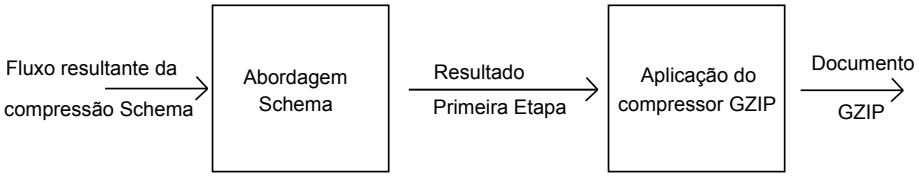

Fig. 4.12: Visão modular da proposta Híbrida

O compressor GZip foi escolhido porque ele é um dos mais populares compressores de dados (loup Gailly and Adler, 2003), (Snyder, 2010). É amplamente utilizado inclusive em compressores especializados em arquivos XML, como por exemplo, o XMill (Davis and Burnett, 2005). A proposta de aplicação do GZip é de realizar a compressão da informação e das *tags*, o que não era feita na primeira abordagem, pois só retirávamos a redundância das *tags* presentes no mesmo nível da árvore XML.

Com esta alteração espera-se uma melhora considerável na taxa de compressão e em contrapartida, uma piora no desempenho do método com relação ao tempo de compressão.

Além do impacto no desempenho desta nova abordagem, outras alterações surgem por consequência desta modificação. O arquivo resultante da compressão não é mais um arquivo XML, agora é um arquivo no formato GZip, ou seja, para que as informações sejam extraídas é necessária a utilização do descompressor GZip.

Aplicando o descompressor GZip teremos restabelecido o XML resultante da Proposta *Schema* e só assim as informações poderão ser extraídas ou processadas. Outra alteração é a perda da legibilidade (o formato GZip é incompreensível para os desenvolvedores) e um maior impacto sobre a interoperabilidade. Outro fato importante é que não é mais possível a utilização das ferramentas disponíveis para a manipulação da linguagem XML.

A perda da interoperabilidade é devido ao fato de que os dispositivos devem possuir meios de descomprimir o formato GZip. Por outro lado, esta perda é menos impactante, pois ao contrário dos outros métodos de compressão que definem um novo formato de documento para a comunicação, a Proposta Híbrida não cria nenhum formato novo, ela utiliza um formato existente com um grau de maturidade alta e amplamente adotado pelas tecnologias (loup Gailly and Adler, 2003),(Snyder, 2010). Na figura 4.13 é apresentado o diagrama que ilustra o processo de recuperação dos dados do documento XML comprimido pela Proposta Híbrida.

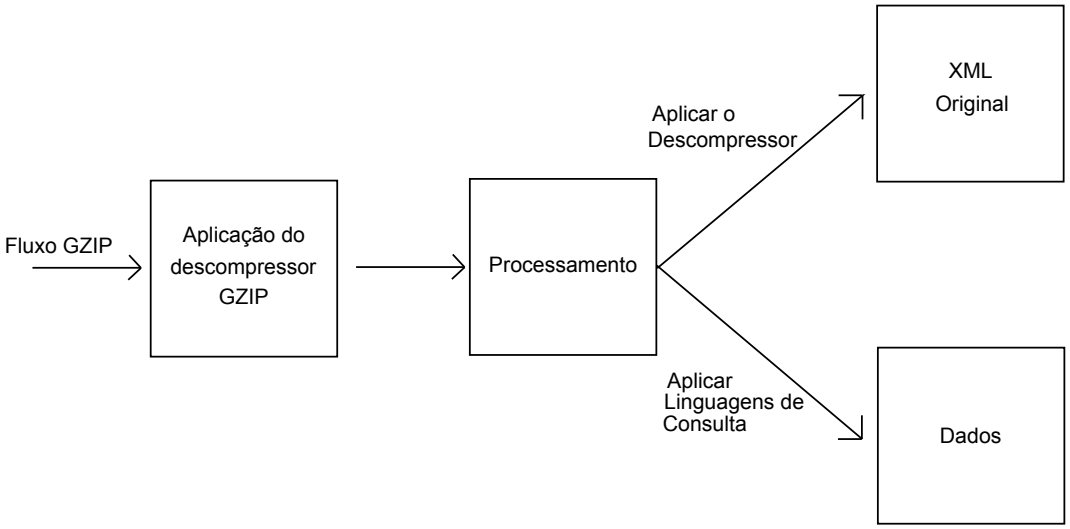

Fig. 4.13: Visão modular de recuperação dos dados da proposta Híbrida.

No próximo capítulo são realizados os testes das abordagens propostas neste trabalho, confrontando os resultados obtidos com os métodos existentes, com o objetivo de identificar suas vantagens e desvantagens.

Capítulo 5

Casos de Testes

Neste capítulo são apresentados os detalhes dos testes a que os métodos foram submetidos. Os resultados obtidos nos quesitos de taxa de compressão, tempo de compressão e tolerância a baixas disponibilidades de memória são avaliados e são feitas análises discutindo as diferenças dos resultados atingidos.

Os testes realizados foram inspirados no artigo "*XML Sizing and Compression Study For Military Wireless Data*", Michael Cokus e Daniel Winkowski. Neste artigo os autores avaliaram os comportamentos dos compressores WinZip, MPEG-7, WBXML, XMill, ASN.1 e algumas de suas variações. Os testes realizados utilizaram tamanho de documentos XML de 6.010 *bytes* (aproximadamente 6 *Kbytes*) a 11.421.822 *bytes* (aproximadamente 11 *Mbytes*).

Seguindo a mesma proposta, neste trabalho foram criados cinco conjuntos de arquivos, cada conjunto possuindo oito arquivos de tamanhos em média: 1 - 268 *Kbytes*, 2 - 538 *Kbytes*, 3 - 1073 *Kbytes*, 4 - 2238 *Kbytes*, 5 - 4482 *Kbytes*, 6 - 8957 *Kbytes*, 7 -17922 *Kbytes* e 8 - 35850 *Kbytes*.

Não foram definidos arquivos menores a 260 *Kbytes*, pois como mencionado anteriormente, alguns compressores não atingem taxas de compressão satisfatórias com arquivos muito pequenos, como é o caso do XMill. Foram escolhidos arquivos de tamanho maiores que 11 *Mbytes* com o objetivo de criar um estudo de comportamento mais abrangente.

Foram definidos 5 conjuntos com 8 arquivos cada com o objetivo de gerar uma média do comportamento dos métodos, para que o estudo não se limitasse a uma simples amostra pontual de dados. Nestes conjuntos de arquivos foram aplicados os métodos EXI, EXI *Compression*, WBXML, XMill e as Propostas *Schema* e Híbrida e foram avaliados nos critérios: taxa de compressão, comportamento dos métodos a pouca disponibilidade de recurso de memória e tempo de compressão. Nas próximas seções serão apresentados os detalhes dos testes realizados e também os resultados obtidos de cada método.

5.1 Taxa de Compressão

Nesta seção vamos avaliar a taxa de compressão atinginda pelos métodos. Os cinco conjuntos de arquivos (ou baterias) foram submetidos aos métodos e de cada arquivo foi analisada a taxa de compressão atingida. Os valores apresentados nos gráficos estão em *Kbytes* e a avaliação da taxa é apresentada no final deste capítulo. Para uma melhor visualização, os resultados obtidos são apresentados nas figuras 5.1, 5.2 e 5.3.

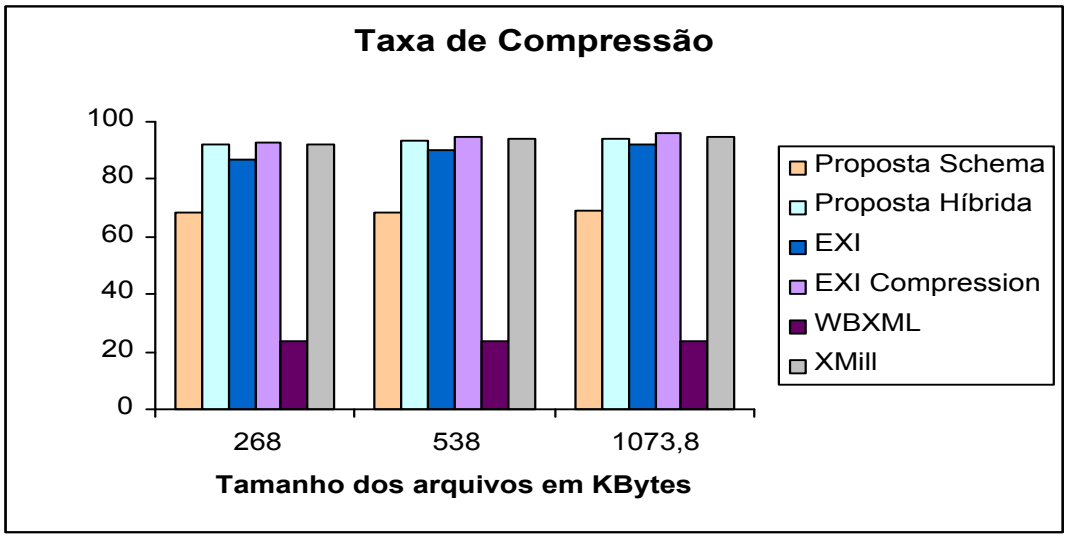

Fig. 5.1: Taxas de compressão obtidas no intervalo de 268 a 1073,8 *Kbytes*.

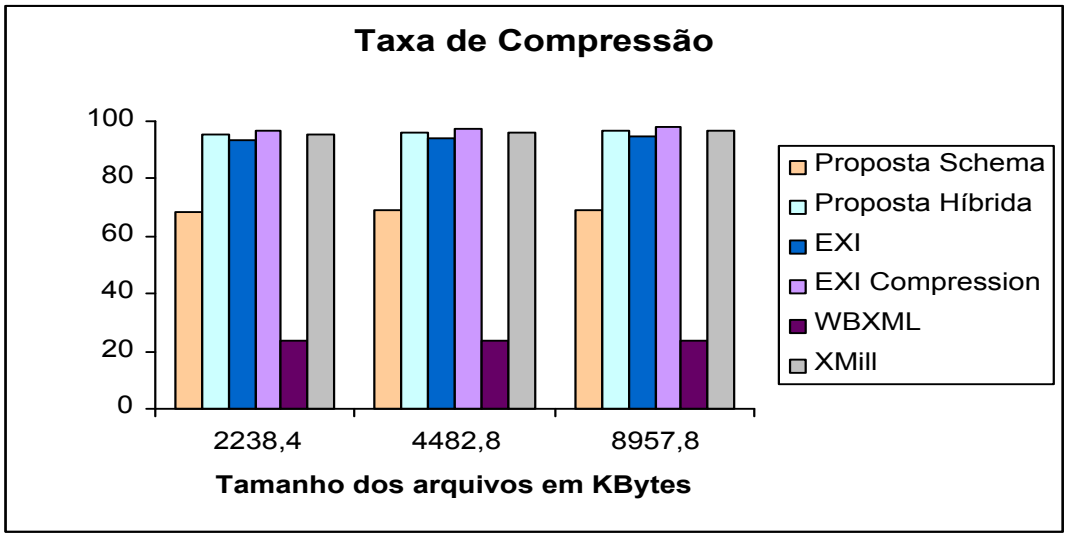

Fig. 5.2: Taxas de compressão obtidas no intervalo de 2238,4 a 8957,8 *Kbytes*.

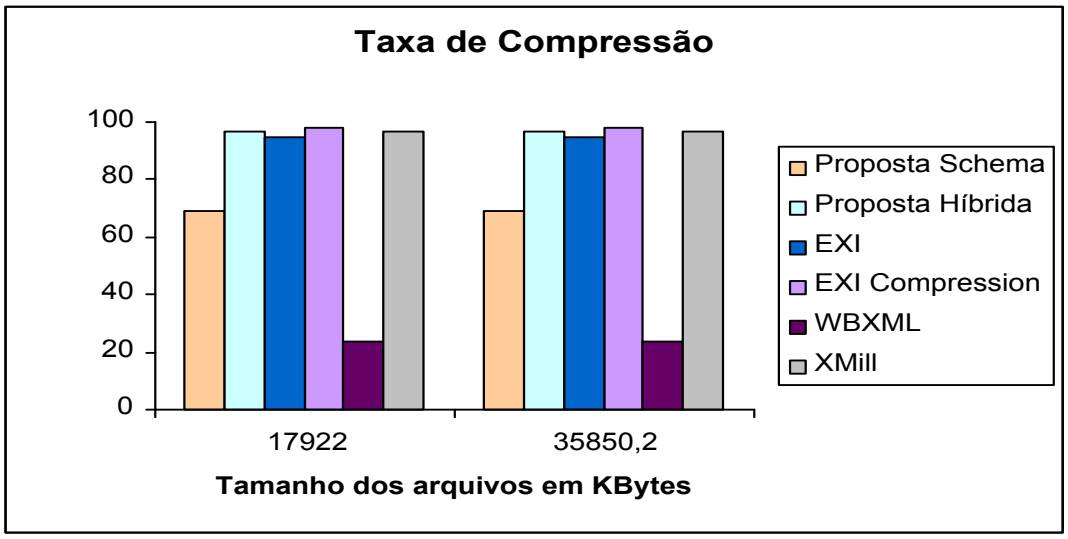

Fig. 5.3: Taxas de compressão obtidas no intervalo de 17922 a 35850,2 *Kbytes*.

Os dados que geraram os gráficos acima são apresentados na tabela 5.1.

Arquivos(*Kbytes*)	**Taxa de compressão em %**					
	EXI	**EXI *Compression***	**Prop. *Schema***	**Prop. Híbrida**	**WBXML**	**XMill**
53,4	86,9	93	68,5	91,9	24	92,1
268,2	90,2	95	68,6	93,2	24	94
536,8	92,4	96,2	68,8	94,4	24	95
1090,8	93,7	97	68,7	95,4	24	95,7
2239,4	94,3	97,4	68,8	95,9	24	96,1
4476,6	94,6	97,8	68,8	96,5	24	96,6
8967,8	94,7	97,9	68,8	96,7	24	96,7
17915,2	94,8	98	68,8	96,8	24	96,6

Tab. 5.1: Média das taxas de compressão atingidas de cada método.

A análise dos gráficos das figuras 5.1, 5.2 e 5.3 e da tabela 5.1 mostra que o método EXI *Compression* atingiu a melhor taxa de compressão para todos os tamanhos de arquivos analisados. Comparando as taxas obtidas pelos métodos XMill e Proposta Híbrida, observa-se valores muito parecidos. O método XMill atingiu taxas levemente melhores para arquivos de até 17922 *Kbytes*, mas considerando arquivos maiores, a Proposta Híbrida obteve melhores resultados. O método WBXML atingiu as piores taxas em todas as amostras realizadas.

5.2 Tempo de Compressão

Outro quesito analisado foi o do tempo de compressão. Para coletar os tempos foram utilizados os mesmos arquivos da análise anterior. Os métodos foram executados em um computador com configuração de processador de 1,33 GHz de freqüência de clock e 1GByte de RAM (*Random Access Memory*). Os gráficos dos resultados obtidos são apresentados nas figuras 5.4, 5.5 e 5.6.

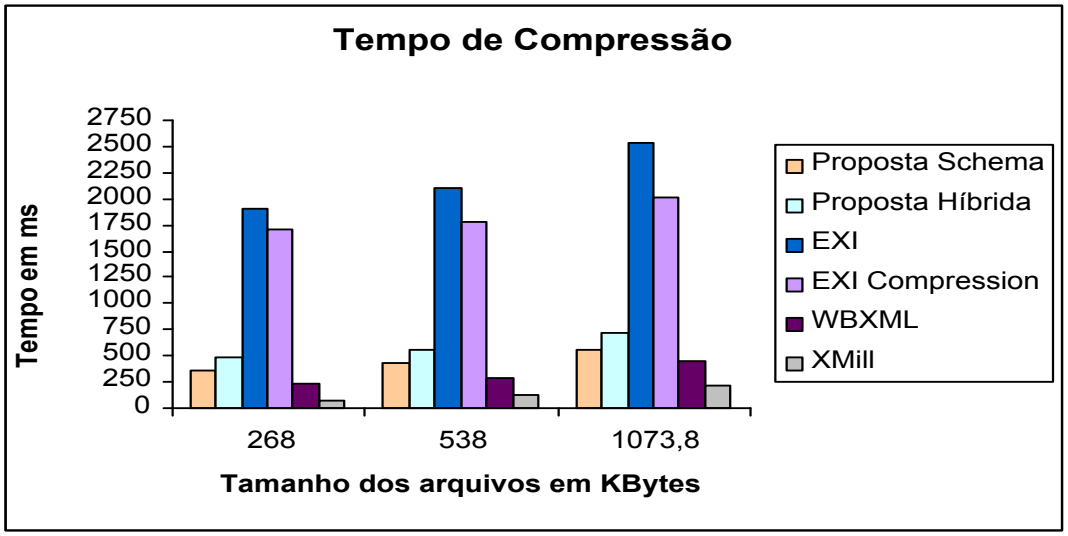

Fig. 5.4: Tempos de compressão obtidos no intervalo de 268 a 1073,8 *Kbytes*.

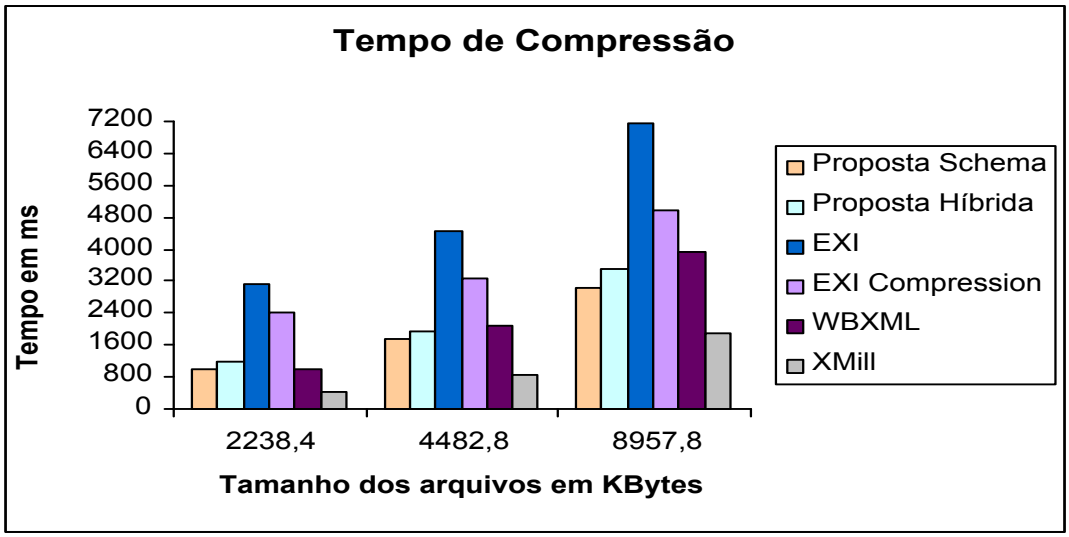

Fig. 5.5: Tempos de compressão obtidos no intervalo de 2238,4 a 8957,8 *Kbytes*.

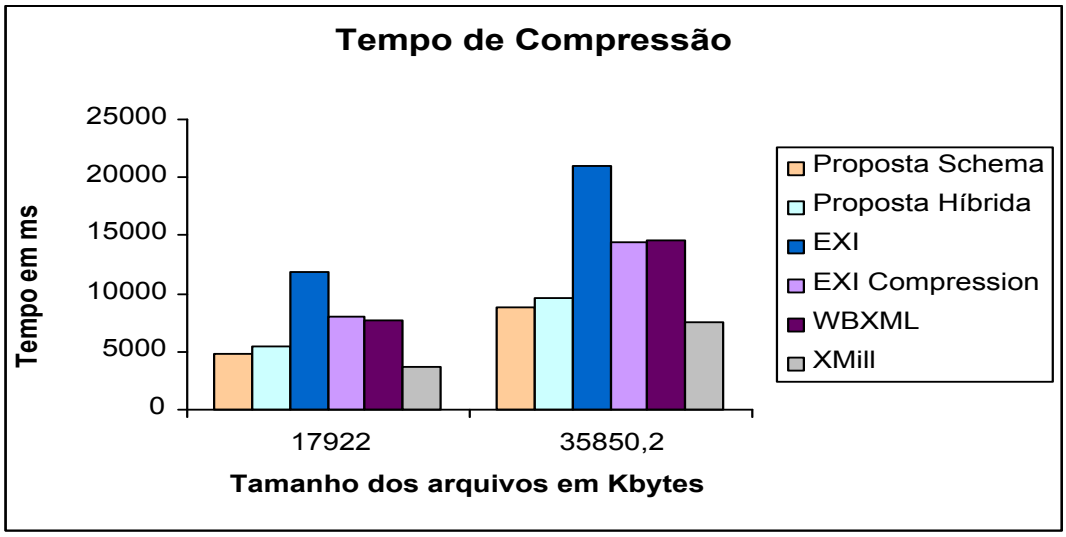

Fig. 5.6: Tempos de compressão obtidos em arquivos de 17922 e 35850,2 *Kbytes*.

Na tabela 5.2 são apresentados os resultados obtidos que gerarm os gráficos apresentados acima.

Arquivos(*Kbytes*)	**Tempo em milisegundos**					
	EXI	**EXI *Compression***	**Prop. *Schema***	**Prop. Híbrida**	**WBXML**	**XMill**
268	1902,8	1699,8	353,4	478,2	228	67,4
538	2109,2	1778,2	431,2	550,2	287,6	128,2
1073,8	2537,6	2009,2	565,6	725,4	446,8	213,8
2238,4	3140,4	2428	997	1187,6	986,8	427,2
4482,8	4459,2	3278	1740,4	1946,8	2068,6	864,8
8957,8	7170	4969,2	3008	3490,8	3934,6	1882,8
17922	11868,8	7959,2	4797	5506,4	7684,4	3616
35850,2	21012,2	14469	8765,6	9687,4	14643,6	7573,6

Tab. 5.2: Média dos tempos de compressão de cada método.

Realizando a análise dos gráficos 5.4, 5.5 e 5.6 e da tabela 5.2, observa-se que o compressor XMill apresentou melhor desempenho para todos os tamanhos de arquivos e em todas as baterias. Um dos motivos para este melhor desempenho é sua implementação feita na linguagem C++, em contrapartida aos outros métodos testados, que são implementados na linguagem Java.

As linguagens C/C++ são linguagens compiladas e possuem melhor desempenho que as linguagens interpretadas, que é o caso da linguagem Java. Outra diferença da linguagem Java que degrada seu desempenho em relação as linguagens C/C++ é o fato de possuir o mecanismo chamado *Garbage Collection* que tem a função de desalocar as memórias que não estão sendo referenciadas, acarretando assim mais custo de processamento.

Visualizando os resultados obtidos nas baterias, nota-se que o WBXML possui melhor desempenho que as Propostas *Schema* e Híbrida para arquivos que variam entre 268 a 2238,4 *Kbytes* e pior desempenho para arquivos variando entre 4482,8 a 8957,8 *Kbytes*. A diferença de desempenho atinge 35% mais rápido em comparação a Proposta *Schema* é 52% mais rápido em comparação a Proposta Híbrida.

Os compressores EXI e EXI *Compression* apresentaram o pior desempenho em todos os tamanhos de arquivos e baterias realizadas. A maior diferença de desempenho relação ao XMill é de aproximadamente 96% mais lentos, e em relação ao WBXML a diferença atinge aproximadamente 88% e 86% mais lentos, respectivamente.

O compressor EXI em relação as Propostas *Schema* e Híbrida é aproximadamente 81% e 74% mais lento, respectivamente. As diferenças de desempenho do EXI e EXI *Compression* em relação aos outros compressores se acentuam em arquivos de menor tamanho, e a medida que o tamanho dos documentos vão crescendo, estas diferenças vão diminuindo.

5.3 Uso de Memória

Outro estudo realizado foi a avaliação do comportamento dos métodos apresentados quando aplicados em um meio onde há escassez de recurso de memória. Um exemplo de ambiente com este perfil são os dispositivos móveis. O navegador Firefox, por exemplo, pode utilizar até 200 *Mbytes* de memória em um PC para sua execução. Pode parecer pouco, mas para um dispositivo que possua apenas 64 *Mbytes* de memória é inviável utilizar uma aplicação que necessite de tanta memória para ser executado.

Este teste simula a quantidade de memória disponível nestes ambientes para avaliar o comportamento dos métodos de compressão. Dentre os dispositivos móveis foi escolhido a configuração de um celular de aproximadamente 30 *Mbytes* de memória RAM.

O sistema operacional *Symbian*, um navegador de internet e a máquina virtual J2ME são as aplicações mais comuns disponíveis em um celular com esta configuração. O *Symbian* utiliza em média 20 *Mbytes* de memória para controlar as funcionalidades básicas do dispositivo. O navegador utiliza em média 15 *Mbytes* para prover o acesso a Internet e a máquina virtual J2ME utiliza em média 5 *Mbytes* para prover recursos as aplicações. A média de utilização dos aplicativos é de aproximadamente 13,33 *Mbytes*, e com a intenção de simular este ambiente, os métodos de compressão foram submetidos a testes com recurso de memória disponível de aproximadamente 10 *Mbytes* (Nokia, 2010).

Expondo os métodos a tal limitação é possível identificar quais métodos melhor otimizam o recurso de memória, podendo ser aplicados neste tipo de ambiente. Os mesmos arquivos das 5 baterias que foram utilizados nos testes anteriores foram submetidos a este novo teste e os resultados são apre-

sentados na tabela 5.3, onde "sim"significa que o compressor efetuou a compressão e "não"significa que a compressão não foi efetuada.

Tamanho em *Kbytes*	EXI	EXI *Compression*	Prop. *Schema*	Prop. Híbrida	WBXML	XMill
268	sim	sim	sim	sim	sim	sim
538	sim	sim	sim	sim	sim	sim
1073,8	sim	sim	sim	sim	não	sim
2238,4	sim	sim	não	não	não	sim
4482,8	sim	sim	não	não	não	sim
8957,8	sim	sim	não	não	não	sim
17922	sim	sim	não	não	não	sim
35850,2	sim	sim	não	não	não	sim

Tab. 5.3: Resultado da compressão com baixa disponibilidade de memória.

Os resultados demonstram que os compressores EXI, EXI *Compression* e o XMill conseguiram comprimir todos os arquivos mesmo com a memória limitada a 10 *Mbytes*. Este comportamento era esperado por parte dos compressores EXI e EXI *Compression*, já que foram projetados para atuar em dispositivos móveis. Já o compressor XMill, que não foi projetado para atuar em um ambiente específico, demonstrou que possui uma boa otimização de memória.

Já por parte das propostas deste trabalho as Propostas *Schema* e Híbrida, demonstraram que não possuem um gerenciamento de recurso de memória tão eficiente quanto as ferramentas EXI, EXI *Compression* e XMill, já que conseguiram comprimir arquivos de até 1073,8 *Kbytes*.

O WBXML foi quem demonstrou pior gerenciamento de recurso de memória, comprimindo arquivos de até 538 *Kbytes*. Mesmo o WBXML sendo projetado para rodar em ambientes onde a memória é escassa e sendo utilizado por aplicações de diversas corporações, como o Windows OS, ele demonstrou que não possui um gerenciamento eficiente de recurso de memória em comparação com os outros métodos.

5.4 Análise Geral

Depois de feita uma análise superficial dos resultados obtidos nos quesitos de taxa de compressão, tempo de compressão e utilização de memória, neste tópico é feita uma análise mais profunda, considerando o cruzamento dos resultados obtidos em cada quesito. Inicialmente são avaliados os métodos XMill, EXI *Compression* e a Proposta Híbrida que foram os métodos que mais se destacaram nos quesitos estudados. Depois é realizada avaliações intermediárias para recolher mais informações relevantes do comportamento de cada método apresentado. Os resultados dos testes realizados são apresentados de forma mais compacta na tabela 5.4.

		268	**538**	**1073,8**	**2238,4**	**4482,8**	**8957,8**	**17922**	**35850,2**
EXI	Taxa	86,9	90,2	92,4	93,7	94,3	94,6	94,7	94,8
	Tempo	1902,8	2109,2	2537,6	3140,4	4459,2	7170	11868,8	21012,2
	Memória	sim	sim	sim	sim	sim	sim	sim	sim
EXI *Compression*	Taxa	93	95	96,2	97	97,4	97,8	97,9	98
	Tempo	1699,8	1778,2	2009,2	2428	3278	4969,2	7959,2	14469
	Memória	sim	sim	sim	sim	sim	sim	sim	sim
Proposta *Schema*	Taxa	68,5	68,6	68,8	68,7	68,8	68,8	68,8	68,8
	Tempo	353,4	431,2	565,6	997	1740,4	3008	4797	8765,6
	Memória	sim	sim	sim	não	não	não	não	não
Proposta Híbrida	Taxa	91,9	93,2	94,4	95,4	95,9	96,5	96,7	96,8
	Tempo	478,2	550,2	725,4	1187,6	1946,8	3490,8	5506,4	9687,4
	Memória	sim	sim	sim	não	não	não	não	não
WBXML	Taxa	24	24	24	24	24	24	24	24
	Tempo	228	287,6	446,8	986,8	2068,6	3934,6	7684,4	14643,6
	Memória	sim	sim	não	não	não	não	não	não
XMill	Taxa	92,1	94	95	95,7	96,1	96,6	96,7	96,6
	Tempo	67,4	128,2	213,8	427,2	864,8	1882,8	3616	7573,6
	Memória	sim	sim	sim	sim	sim	sim	sim	sim

Tab. 5.4: Resultado geral dos critérios avaliados.

Analizando a tabela 5.4 e comparando as taxas de compressão atingidas pela Proposta Híbrida e o compressor XMill, verifica-se valores muito próximos. A média da diferença entre as taxa atingidas tem variação máxima de 12,0%, sendo que para arquivos maiores que 8957,8 *Kbytes*, a Proposta Híbrida apresenta melhor desempenho neste quesito.

Avaliando os métodos XMill, Proposta Híbrida e o EXI *Compression* considerando o quesito tempo de compressão, o que apresentou melhores resultados foi o XMill e a Proposta Híbrida se apresentou 3 vezes e meio mais rápida que o método EXI *Compression*.

Realizando a análise destes 3 métodos considerando o quesito de tolerância a baixas disponibilidades de memória, os métodos EXI *Compression* e o XMill conseguiram comprimir todos os tamanhos de arquivos propostos e a Proposta Híbrida comprimiu arquivos de até 2238,4 *Kbytes*, demonstrando que os métodos EXI *Compression* e o XMill possuem melhor tolerância a ambientes onde o recurso de memória é escasso.

Feitas estas avaliações iniciais, nenhum método se apresentou melhor em todos os quesitos. Ser o melhor método depende dos quesitos avaliados, do tamanho dos arquivos a serem comprimidos e do ambiente onde ele está inserido. Como citado anteriormente, o método que atingiu melhor taxa de compressão é o EXI *Compression*, mas em contrapartida é mais lento que os 2 métodos que mais se aproximam da sua taxa de compressão, o XMill e a Proposta Híbrida. O XMill se apresentou mais rápido que todos os métodos, mas não possui um processador de consultas.

A Proposta Híbrida e o método XMill atingiram taxas de compressão muito parecidas e melhores

desempenhos em comparação com o EXI *Compression*. Além destes fatos, a Proposta Híbrida apresenta uma vantagem de não criar um novo formato de dados, o resultado da compressão é o formato GZip, como citado anteriomente é amplamente estudado e utilizado na literatura.

Outra vantagem é a possibilidade recuperar a informação de forma mais rápida através do processador de consultas. Para descomprimir os dados da Proposta Híbrida é necessário utilizar o descompressor GZip e sobre o dados descomprimidos aplicar o processador de consultas, recuperando as informações necessárias sem transformar o documento XML em sua forma original.

Um fato a ser destacado são os resultados obtidos pelas Propostas *Schema* e Híbrida. Estes resultados podem ser ainda melhores se o algoritmo proposto for desenvolvido utilizando técnicas de otimização de algoritmos e linguagens de melhor desempenho em sua implementação.

Depois destas avaliações dos métodos que obtiveram maior destaque nos quesitos propostos, é feita uma avaliação dos resultados intermediários, para obter mais informações dos comportamentos dos compressores.

Analisando os resultados obtidos pelos métodos WBXML e a Proposta *Schema* no quesito taxa de compressão, observa-se que a Proposta *Schema* atinge aproximadamente taxas até 68,81% e o compressor WBXML atinge aproximadamente taxas de até 24,26%, o que representa uma diferença significativa. Com relação ao quesito tempo de compressão, o compressor WBXML apresenta um melhor desempenho para arquivos de até 2238,4 *Kbytes* e para arquivos maiores que 4482,8 *Kbytes* a Proposta *Schema* é mais rápida. Em relação ao quesito de tolerância a baixa disponibilidade de memória, as Propostas Híbridas e *Schema* se mostraram com um melhor gerenciamento de memória em relação a ferramenta WBXML.

Como esperado, a Proposta Híbrida atingiu taxas de compressão de no mínimo 74,2% e no máximo 89,7% melhor que a Proposta *Schema*. Em relação ao quesito tempo de compressão, como também era esperado, a Proposta *Schema* se apresentou no mínimo 9,5% mais rápida. E analisando em relação à tolerância a baixas disponibilidades de memória as propostas não apresentaram diferenças.

Analisando a diferença numérica da taxa e tempo de compressão das Propostas *Schema* e Híbrida, observa-se que para uma perda de desempenho de no máximo 26%, a Proposta Híbrida apresenta no mínimo uma melhora de 74,2% na taxa de compressão, e ainda não altera os resultados obtidos na utilização de memória. Levando estes fatos em consideração, a Proposta Híbrida se torna uma opção viável do ponto de vista de desempenho, taxa de compressão e utilização de memória. Do ponto de vista de interoperabilidade e legibilidade, como dito anteriormente, a Proposta Híbrida perde estas características em relação a Proposta *Schema*, já que o resultado de sua compressão é um documento no formato GZip.

Capítulo 6

Conclusões

A linguagem XML vem sendo amplamente adotada como um padrão de comunicação entre sistemas computacionais heterogêneos e uns dos fatos que vem chamando a atenção dos pesquisadores e desenvolvedores é a quantidade de informação gerada pela XML. Isto é devido ao fato da linguagem ser redundante e auto-descritiva, consequentemente a representação dos formatos de dados específicos que os sistemas utilizavam transcritos para o formato XML se tornou excessivamente grande, gerando mais dados para serem transmitidos e armazenados. Uma solução para este problema é utilizar compressores específicos da linguagem XML para reduzir o tamanho dos documentos.

Neste trabalho foram apresentados alguns compressores de XML disponíveis na literatura: WBXML, XMill, EXI e sua variação EXI *Compression*. Estes compressores foram escolhidos por estarem em destaque na literatura e por atingirem resultados significativos na compressão de documentos XML. As técnicas de compressão de cada método foram detalhadas com o objetivo de compreender o seu funcionamento, podendo assim, explorar melhor suas vantagens e desvantagens.

Foi proposta uma nova abordagem de compressão utilizando uma técnica ainda não explorada por outros métodos. Esta abordagem retira a redundância das *tags* presentes no mesmo nível da árvore XML para diminuir o tamanho dos documentos e foi denominada de Proposta *Schema*. Também foi proposta uma variação da abordagem *Schema* denominada Proposta Híbrida. Esta última utiliza um compressor de dados para melhorar a taxa compressão dos documentos XML.

Foram elaborados casos de testes e aplicados em todos os métodos. Os quesitos taxa de compressão, tempo de compressão e tolerância a baixas disponibilidades de memória foram avaliados e os resultados comparados. Inicialmente foi feita uma comparação isolada de cada quesito, identificando os métodos que atingiram melhores resultados. Depois desta primeira análise, foi feita uma análise mais refinada dos resultados intermediários para descobrir mais informações sobre o comportamento dos métodos.

Depois da análise dos resultados, concluímos que nenhum compressor foi o melhor em todos os

quesitos. Para avaliar qual o melhor método deve se considerar o ambiente e o tamanho do documento XML comprimido.

Os resultados atingidos pelas propostas foram expressivos, pois alcançaram resultados muitas vezes próximos, e algumas vezes melhores que os compressores já existentes em alguns quesitos analisados. Implementar o algoritmo proposto em uma linguagem como o C ou C++ que apresentam melhor desempenho e o desenvolvedor possui o controle do gerenciamento de memória implicará em uma melhora nos resultados obtidos.

Outra questão abordada pelo trabalho foi o comprometimento da interoperabilidade devido aos métodos de compressão criarem novos formatos. O fato das ferramentas existentes adotarem outros formatos de dados prejudica diretamente a integração de sistemas heterogêneos, pois as aplicações envolvidas devem dar suporte ao novo formato. Existem soluções para amenizar este problema, como foi abordado na seção 3.5, mas não podem ser utilizadas em todos os ambientes.

As propostas deste trabalho comprometem menos a interoperabilidade que os outros métodos existentes viabilizando a compressão de XML nestes ambientes, pois não criam novos formatos de dados. O resultado da compressão da Proposta Schema é um documento XML, mantendo o padrão de comunicação. Na Proposta Híbrida o resultado é o GZip, que é um formato aberto e amplamente utilizado, onde a maioria das tecnologias possuem suporte.

Há muitos fatores a serem explorados ainda na compressão de documentos XML, mas os métodos propostos e as ferramentas existentes já se mostraram de grande importância para poupar recursos de rede e de armazenamento para melhorar o desempenho das aplicações que utilizam esta linguagem para comunicação.

Sugerimos como trabalhos futuros, realizar o refinamento do estudo feito ampliando assim a obtenção de informações sobre os compressores. Uma proposta é aumentar os casos de testes realizados, aumentando o número de amostras de documentos XML para estudar o comportamento de cada compressor. Um estudo que geraria resultados mais precisos é a aplicação dos compressores a documentos que possuam faixas específicas de porcentagens de quantidade de dados e *tags*(metadados). Por exemplo, gerar documentos nas faixas:

- 1% a 20% de dados; 80% a 99% de *tags*.
- 20% a 40% de dados; 60% a 80% de *tags*.
- 40% a 60% de dados; 40% a 60% de *tags*.
- 60% a 80% de dados; 20% a 40% de *tags*.
- 80% a 99% de dados; 1% a 20% de *tags*.

Outro ponto que pode ser desenvolvido para trabalhos futuros é a criação automática do *schema* para a realização da compressão da Proposta *Schema*.

Referências Bibliográficas

Serge Abiteboul. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1:68–88, 2004.

Software AG, 2010. URL `http://www.softwareag.com`.

AgileDelta. Lightning-fast delivery of xml to more devices in more locations, Sep 2010. URL `http://www.agiledelta.com/product_efx.html`.

Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004. ISBN 3-540-44008-9.

Christopher J. Augeri, Dursun A. Bulutoglu, Barry E. Mullins, Rusty O. Baldwin, and Leemon C. Baird, III. An analysis of xml compression efficiency. In *ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science*, page 7, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-751-3. doi: http://doi.acm.org/10.1145/1281700.1281707.

Reza B'far. *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521817331.

Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, and Jonathan Robie Jérôme Siméon. Xquery 1.0: An xml query language. Technical report, Janeiro 2007. URL `http://www.w3.org/TR/xquery/`.

Karlheinz Brandenburg. Mp3 and aac explained. *International Conference on High Quality Audio Coding*, 1999.

Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in xml. Página na internet, World Wide Web Consortium, Janeiro 1999. URL `http://www.w3.org/TR/1999/REC-xml-names-19990114/`.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0. Technical report, Dezembro 2008. URL http://www.w3.org/TR/REC-xml/.

Ethan Cerami. *Web Services Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. ISBN 0596002246.

Xinxiang Chen. The e-gov action plan in beijing. In Roland Traunmüller and Klaus Lenk, editors, *Electronic Government*, volume 2456 of *Lecture Notes in Computer Science*, pages 69–74. 2002.

Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1. Technical report, Março 2001. URL http://www.w3.org/TR/wsdl.

James Clark and Steve DeRose. Xml path language (xpath). Technical report, Novembro 1999. URL XMLPathLanguage(XPath).

Michael Cokus and Daniel Winkowski. Xml sizing and compression study for military wireless data. XML Conference & Exposition, Dezembro 2002.

World Web Consortium. Soap. Technical report, 2004. URL http://www.w3.org/TR/soap/.

Stephen J. Davis and Ian S. Burnett. Efficient delivery within the mpeg-21 framework. In *AXMEDIS '05: Proceedings of the First International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution*, page 205, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2348-X. doi: http://dx.doi.org/10.1109/AXMEDIS.2005.18.

Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for xml. *World Wide Web Consortium*, 1998. URL http://www.w3.org/TR/NOTE-xml-ql/.

David C. Fallside and Priscilla Walmsley. Xml schema part 0: Primer second edition. Technical report, Outubro 2004. URL http://www.w3.org/TR/xmlschema-0/.

Carlos Maurício Seródio Figueiredo and Eduardo Nakamura. Computação móvel: Novas oportunidades e novos desafios. *T&C Amazônia*, 2(2):28, 2003. URL https://portal.fucapi.br/tec/imagens/revistas/ed02_04.pdf.

Lúcio França, Rainer R. P. Couto, and Antonio A. F. Loureiro. Adaptabilidade de compressão em ambientes móveis: Como e quando comprimir. *III Workshop de Comunicação sem Fio e Computação Móvel*, 2001.

David Geer. Will binary xml speed network traffic? *Computer*, 38(4):16–18, 2005. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2005.143.

Roy Goldman, Jason McHugh, and Jennifer Widom. From semistructured data to xml: Migrating the lore data model and query language. *Workshop on The Web and Databases*, pages 25–30, 1999.

Brian D. Goodman. Squeezing soap: Gzip enabling apache axis, Mar 2003. URL `http://www.ibm.com/developerworks/webservices/library/ws-sqzsoap.html`.

Elliotte Rusty Harold. *XML Bible*. Hungry Minds, Incorporated, 1999. ISBN 0764532367.

Takeshi Imamura and Hiroshi Maruyama. Mapping between asn.1 and xml. In *SAINT '01: Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*, page 57, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0942-8.

Nick Kew. Running a reverse proxy in apache. *ApacheWeek*, 2004.

Laura Lambert, Chris Woodford, Hilary W. Poole, and Christos J. P. Moschovitis. *The Internet: a historical encyclopedia*, volume 3. 2005.

Hartmut Liefke and Dan Suciu. Xmill: An efficient compressor for xml data. In *SIGMOD Conference*, pages 153–164, 2000.

Jean loup Gailly and Mark Adler. The gzip home page, Julho 2003. URL `http://www.gzip.org/`.

Bruce Martin and Bashar Jano. Wap binary xml content format. Technical report, Junho 1999. URL `http://www.w3.org/TR/wbxml/`.

Nathan J. Muller. Improving and managing multimedia performance over tcp–ip nets. *Int. J. Netw. Manag.*, 8(6):356–367, 1998. ISSN 1099-1190. doi: http://dx.doi.org/10.1002/(SICI)1099-1190(199811/12)8:6<356::AID-NEM308>3.3.CO;2-9.

Wilfred Ng, Wai Y. Lam, and James Cheng. Comparative analysis of xml compression technologies. *World Wide Web*, 9(1):5–33, March 2006. ISSN 1386-145X. doi: 10.1007/s11280-005-1435-2. URL `http://dx.doi.org/10.1007/s11280-005-1435-2`.

U. Niedermeier, J. Heuer, A. Hutter, and W. Stechele. Mpeg-7 binary format for xml data. In *Proceedings of the Data Compression Conference*, DCC '02, pages 467–, Washington, DC, USA, 2002. IEEE Computer Society. URL `http://portal.acm.org/citation.cfm?id=882455.874980`.

Nokia, 2010. URL http://www.forum.nokia.com/.

Oracle, 2010. URL http://www.oracle.com.

William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Kluwer Academic Publishers, Norwell, MA, USA, 1992. ISBN 0442012721.

Liam Quin. Extensible markup language (xml). Technical report, 2010. URL http://www.w3.org/XML/.

Jonathan Robie, Joe Lapp, and David Schach. Xml query language (xql). In *QL*, 1998.

Martin Ruckert. *Understanding MP3*. SpringerVerlag, 2005. ISBN 3528059052.

David Salomon. *Data Compression: The Complete Reference*. pub-SV, second edition, 2004. ISBN 0-387-40697-2. URL http://www.ecs.csun.edu/~dxs/DC3advertis/Dcomp3Ad.html.

David Salomon. *A Concise Introduction to Data Compression*. Undergraduate topics in computer science. pub-SV, 2008. ISBN 1-84800-071-5.

David Salomon, Giovanni Motta, and David (CON) Bryan. *Handbook of Data Compression*. Springer, 2009.

M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001.

Khalid Sayood. *Introduction to Data Compression 2ed.* 2000.

John Schneider and Takuki Kamiya. Efficient xml interchange (exi) format 1.0. World Wide Web Consortium, Working Draft WD-exi-20080919, September 2008.

Boag Scott, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. Xquery 1.0: An xml query language. Technical report, Janeiro 2007. URL http://www.w3.org/TR/xquery.

Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-615-7.

Sheldon L. Snyder. *Efficient XML Interchange (EXI) Compression and Performance Benefits: Development, Implementation and Evaluation*. PhD thesis, Naval Postgraduate School Monterey, California, 2010.

UserLand Software. Xml-rpc, 2003. URL `http://www.xmlrpc.com/`.

Milford H. Sprecher. Racing to e-government: Using the internet for citizen service delivery. pages 21–22. Government Finance Review, 2000.

Srivastava,Rahul. XML Schema: Understanding Namespaces, Janeiro 1999. URL `http://www.oracle.com/technology/pub/articles/srivastava_namespaces.html`.

Brian E. Travis and Mae Ozkan. *Web Services: Implementation Guide.*, volume Volume 1: Getting Started. 2002.

Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *J. ACM*, 34(4):825–845, 1987. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/31846.42227.

W3C. Some early ideas for html, 2009. URL `http://www.w3.org/MarkUp/historical`.

w3schools. Introduction to xml, 1999. URL `http://www.w3schools.com/xml/`.

Gregory K. Wallace. The jpeg still picture compression standard. *Commun. ACM*, 34(4):30–44, 1991. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/103085.103089.

Webopedia. Web services, 2010. URL `http://www.webopedia.com/TERM/W/Web_services.html`.

M. Zhang. *Artificial Higher Order Neural Networks for Economics and Business*. IGI/ Information Science Reference, 2009. URL `http://eprints.ucl.ac.uk/18094/`.

Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions Information Theory*, 23:337, 1977.

Zlib. A massively spiffy yet delicately unobtrusive compression library, 2010. URL `http://www.zlib.net/`.

Capítulo 7

Apêndices

Abaixo é apresentado o XSD utilizado nos teste e a implementação feita na linguagem java do algoritmo proposto.

```
<?commentstylexml commentstyleversion="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <xsd:element name="clientes">
        <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="cliente" type="Cliente" minOccurs="0"
                  maxOccurs="unbounded"/>
        </xsd:sequence>
            </xsd:complexType>
  </xsd:element>


  <xsd:complexType name="Cliente">
    <xsd:sequence>
      <xsd:element name="empresa" type="xsd:string" minOccurs="1"
        maxOccurs="1" nillable="false"/>
      <xsd:element name="tipo" type="xsd:string" minOccurs="1"
        maxOccurs="1" nillable="false"/>
      <xsd:element name="nome" type="xsd:string" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="sexo" type="xsd:string" minOccurs="1"
        maxOccurs="1" nillable="false"/>
      <xsd:element name="email" type="xsd:string" minOccurs="1"
        maxOccurs="1" nillable="false"/>
      <xsd:element name="telefone" type="xsd:string" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="endereco" type="Endereco" minOccurs="1"
        maxOccurs="unbounded"/>
      <xsd:element name="pais" type="Pais" minOccurs="0"
```

```
        maxOccurs="1"/>
      <xsd:element name="filhos" type="Filhos" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

    <xsd:complexType name="Endereco">
        <xsd:sequence>
            <xsd:element name="rua" type="xsd:string"/>
            <xsd:element name="numero" type="xsd:string"/>
            <xsd:element name="cidade" type="xsd:string"/>
            <xsd:element name="estado" type="xsd:string"/>
            <xsd:element name="cep" type="xsd:string"/>
            <xsd:element name="pais" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Filhos">
        <xsd:sequence>
            <xsd:element name="filho" type="Filho" minOccurs="0"
                    maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Filho">
        <xsd:sequence>
            <xsd:element name="nome" type="xsd:string"/>
            <xsd:element name="sexo" type="xsd:string"/>
            <xsd:element name="email" type="xsd:string"/>
            <xsd:element name="telefone" type="xsd:string"/>
            <xsd:element name="endereco" type="Endereco"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Pais">
        <xsd:sequence>
            <xsd:element name="pai" type="Pai"/>
            <xsd:element name="mae" type="Mae"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Pai">
        <xsd:sequence>
            <xsd:element name="nome" type="xsd:string"/>
            <xsd:element name="profissao" type="xsd:string"/>
            <xsd:element name="email" type="xsd:string"/>
            <xsd:element name="telefone" type="xsd:string"/>
            <xsd:element name="endereco" type="Endereco"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Mae">
        <xsd:sequence>
            <xsd:element name="nome" type="xsd:string"/>
```

```
            <xsd:element name="profissao" type="xsd:string"/>
            <xsd:element name="email" type="xsd:string"/>
            <xsd:element name="telefone" type="xsd:string"/>
            <xsd:element name="endereco" type="Endereco"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Listing 7.1: “Implementação do algoritmo proposto pelo trabalho”

```
package criadorArvoreCompacta;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.xmlbeans.XmlException;
import com.sun.org.apache.xml.internal.serialize.XMLSerializer;
import noNamespace.Cliente;
import noNamespace.ClientesDocument;
import noNamespace.Endereco;
import noNamespace.Filho;
import noNamespace.Filhos;
import noNamespace.Mae;
import noNamespace.Pai;
import noNamespace.Pais;
import noNamespace.ClientesDocument.Clientes;

public class CriadorArvoreCompacta {

    public static void main(String[] args) {
        CriadorArvoreCompacta criadorArvoreCompacta =
            new CriadorArvoreCompacta();

        criadorArvoreCompacta.criadorArvore();

    }

    public void criadorArvore(){

        try {
            ClientesDocument clientesDocument =
                ClientesDocument.Factory.parse(
                        new File("D:\\Xml_original.xml"));

            ClientesDocument clientesDocumentCompactado =
                ClientesDocument.Factory.newInstance();

            Clientes clientes =
                clientesDocument.getClientes();

            Cliente cliente[] =
```

```
    clientes.getClienteArray();

String listaNomeClientes = "",listaEmailClientes = "",listaEmpresaClientes = "",
    listaSexoClientes = "",listaTipoClientes = "",listaTelefoneClientes = "";

String listaRuaEnderecoClientes = "",listaNumeroEnderecoClientes = "",↩
    listaCidadeEnderecoClientes = "",
listaEstadoEnderecoClientes = "",listaCepEnderecoClientes = "",↩
    listaPaisEnderecoClientes = "";

String listaRuaEnderecoPai = "",listaNumeroEnderecoPai = "",listaCidadeEnderecoPai = "↩
    ",
listaEstadoEnderecoPai = "",listaCepEnderecoPai = "",listaPaisEnderecoPai = "";

String listaRuaEnderecoMae = "",listaNumeroEnderecoMae = "",listaCidadeEnderecoMae = "↩
    ",
listaEstadoEnderecoMae = "",listaCepEnderecoMae = "",listaPaisEnderecoMae = "";

String listaEmailPai = "",listaProfissaoPai = "",
listaNomePai = "",listaTelefonePai = "";

String listaEmailMae = "",listaProfissaoMae = "",
listaNomeMae = "",listaTelefoneMae = "";

String listaNomeFilho = "",listaEmailFilho = "",
listaSexoFilho = "",listaTelefoneFilho = "";

String listaRuaEnderecoFilho = "",listaNumeroEnderecoFilho = "",↩
    listaCidadeEnderecoFilho = "",
listaEstadoEnderecoFilho = "",listaCepEnderecoFilho = "",listaPaisEnderecoFilho = "";

for(int contCliente=0;contCliente<cliente.length;contCliente++){

    System.out.println("contCliente: " + contCliente);

    if(contCliente != cliente.length){
        listaNomeClientes +=
            cliente[contCliente].getNome() +";";
        listaEmailClientes +=
            cliente[contCliente].getEmail() +";";
        listaEmpresaClientes +=
            cliente[contCliente].getEmpresa() +";";
        listaSexoClientes +=
            cliente[contCliente].getSexo() +";";
        listaTipoClientes +=
            cliente[contCliente].getTipo() +";";
        listaTelefoneClientes +=
            cliente[contCliente].getTelefone() +";";

        Endereco endereco =
            cliente[contCliente].getEnderecoArray(0);

        listaCepEnderecoClientes += endereco.getCep() + ";";
        listaCidadeEnderecoClientes += endereco.getCidade() + ";";
```

```
listaEstadoEnderecoClientes += endereco.getEstado() + ";";
listaNumeroEnderecoClientes += endereco.getNumero() + ";";
listaPaisEnderecoClientes += endereco.getPais() + ";";
listaRuaEnderecoClientes += endereco.getRua() + ";";

Pais paisCompacto =
    cliente[contCliente].getPais();

listaEmailPai += paisCompacto.getPai().getEmail() + ";";
listaNomePai += paisCompacto.getPai().getNome() + ";";
listaProfissaoPai += paisCompacto.getPai().getProfissao() + ";";
listaTelefonePai += paisCompacto.getPai().getTelefone() + ";";

listaEmailMae += paisCompacto.getMae().getEmail() + ";";
listaNomeMae += paisCompacto.getMae().getNome() + ";";
listaProfissaoMae += paisCompacto.getMae().getProfissao() + ";";
listaTelefoneMae += paisCompacto.getMae().getTelefone() + ";";

Endereco enderecoPai =
    paisCompacto.getPai().getEndereco();

listaCepEnderecoPai += enderecoPai.getCep() + ";";
listaCidadeEnderecoPai += enderecoPai.getCidade() + ";";
listaEstadoEnderecoPai += enderecoPai.getEstado() + ";";
listaNumeroEnderecoPai += enderecoPai.getNumero() + ";";
listaPaisEnderecoPai += enderecoPai.getPais() + ";";
listaRuaEnderecoPai += enderecoPai.getRua() + ";";

Endereco enderecoMae =
    paisCompacto.getMae().getEndereco();

listaCepEnderecoMae += enderecoMae.getCep() + ";";
listaCidadeEnderecoMae += enderecoMae.getCidade() + ";";
listaEstadoEnderecoMae += enderecoMae.getEstado() + ";";
listaNumeroEnderecoMae += enderecoMae.getNumero() + ";";
listaPaisEnderecoMae += enderecoMae.getPais() + ";";
listaRuaEnderecoMae += enderecoMae.getRua() + ";";

Filhos [] filhos =
    cliente[contCliente].getFilhosArray();

for(int quantidadeFilhos=0;quantidadeFilhos<filhos.length;quantidadeFilhos++){
    Filho filho[] =
        filhos[quantidadeFilhos].getFilhoArray();
    for(int j=0;j<filho.length;j++){
        Filho filhoLocal =
            filho[j];

        listaNomeFilho += filhoLocal.getNome() + ";";
        listaTelefoneFilho += filhoLocal.getTelefone() + ";";
        listaSexoFilho += filhoLocal.getSexo() + ";";
        listaEmailFilho += filhoLocal.getEmail() + ";";

        Endereco enderecoFilho =
```

```
                filhoLocal.getEndereco();

            listaRuaEnderecoFilho += enderecoFilho.getRua() + ";";
            listaCidadeEnderecoFilho += enderecoFilho.getCidade() + ";";
            listaCepEnderecoFilho += enderecoFilho.getCep() + ";";
            listaEstadoEnderecoFilho += enderecoFilho.getEstado() + ";";
            listaNumeroEnderecoFilho += enderecoFilho.getNumero() + ";";
            listaPaisEnderecoFilho += enderecoFilho.getPais() + ";";
        }
        listaNomeFilho += ";";
        listaTelefoneFilho += ";";
        listaSexoFilho += ";";
        listaEmailFilho += ";";

        listaRuaEnderecoFilho += ";";
        listaCidadeEnderecoFilho += ";";
        listaCepEnderecoFilho += ";";
        listaEstadoEnderecoFilho += ";";
        listaNumeroEnderecoFilho += ";";
        listaPaisEnderecoFilho += ";";
    }

}
else{
    listaNomeClientes +=
        cliente[contCliente].getNome();
    listaEmailClientes +=
        cliente[contCliente].getEmail();
    listaEmpresaClientes +=
        cliente[contCliente].getEmpresa();
    listaSexoClientes +=
        cliente[contCliente].getSexo();
    listaTipoClientes +=
        cliente[contCliente].getTipo();
    listaTelefoneClientes +=
        cliente[contCliente].getTelefone();

    Endereco endereco =
        cliente[contCliente].getEnderecoArray(0);

    listaCepEnderecoClientes += endereco.getCep();
    listaCidadeEnderecoClientes += endereco.getCidade();
    listaEstadoEnderecoClientes += endereco.getEstado();
    listaNumeroEnderecoClientes += endereco.getNumero();
    listaPaisEnderecoClientes += endereco.getPais();
    listaRuaEnderecoClientes += endereco.getRua();

    Pais paisCompacto =
        cliente[contCliente].getPais();

    listaEmailPai += paisCompacto.getPai().getEmail();
    listaNomePai += paisCompacto.getPai().getNome();
    listaProfissaoPai += paisCompacto.getPai().getProfissao();
    listaTelefonePai += paisCompacto.getPai().getTelefone();
```

```
listaEmailMae += paisCompacto.getMae().getEmail();
listaNomeMae += paisCompacto.getMae().getNome();
listaProfissaoMae += paisCompacto.getMae().getProfissao();
listaTelefoneMae += paisCompacto.getMae().getTelefone();

Endereco enderecoPai =
    paisCompacto.getPai().getEndereco();

listaCepEnderecoPai += enderecoPai.getCep();
listaCidadeEnderecoPai += enderecoPai.getCidade();
listaEstadoEnderecoPai += enderecoPai.getEstado();
listaNumeroEnderecoPai += enderecoPai.getNumero();
listaPaisEnderecoPai += enderecoPai.getPais();
listaRuaEnderecoPai += enderecoPai.getRua();

Endereco enderecoMae =
    paisCompacto.getMae().getEndereco();

listaCepEnderecoMae += enderecoMae.getCep();
listaCidadeEnderecoMae += enderecoMae.getCidade();
listaEstadoEnderecoMae += enderecoMae.getEstado();
listaNumeroEnderecoMae += enderecoMae.getNumero();
listaPaisEnderecoMae += enderecoMae.getPais();
listaRuaEnderecoMae += enderecoMae.getRua();

Filhos [] filhos =
    cliente[contCliente].getFilhosArray();

System.out.println("listaNomeClientes: " + listaNomeClientes);

for(int quantidadeFilhos=0;quantidadeFilhos<filhos.length;quantidadeFilhos++){
    Filho filho[] =
        filhos[quantidadeFilhos].getFilhoArray();
    for(int j=0;j<filho.length;j++){
        Filho filhoLocal =
            filho[j];

        listaNomeFilho += filhoLocal.getNome();
        listaTelefoneFilho += filhoLocal.getTelefone();
        listaSexoFilho += filhoLocal.getSexo();
        listaEmailFilho += filhoLocal.getEmail();

        Endereco enderecoFilho =
            filhoLocal.getEndereco();

        listaRuaEnderecoFilho += enderecoFilho.getRua();
        listaCidadeEnderecoFilho += enderecoFilho.getCidade();
        listaCepEnderecoFilho += enderecoFilho.getCep();
        listaEstadoEnderecoFilho += enderecoFilho.getEstado();
        listaNumeroEnderecoFilho += enderecoFilho.getNumero();
        listaPaisEnderecoFilho += enderecoFilho.getPais();
    }
}
```

```
    }

}

Clientes clientesCompactado =
    clientesDocumentCompactado.addNewClientes();

Cliente clienteCompactado =
    clientesCompactado.addNewCliente();
clienteCompactado.setNome(listaNomeClientes);
clienteCompactado.setEmail(listaEmailClientes);
clienteCompactado.setEmpresa(listaEmpresaClientes);
clienteCompactado.setTelefone(listaTelefoneClientes);
clienteCompactado.setTipo(listaTipoClientes);
clienteCompactado.setSexo(listaSexoClientes);

Endereco enderecoCompactado =
    clienteCompactado.addNewEndereco();

enderecoCompactado.setCep(listaCepEnderecoClientes);
enderecoCompactado.setCidade(listaCidadeEnderecoClientes);
enderecoCompactado.setEstado(listaEstadoEnderecoClientes);
enderecoCompactado.setNumero(listaNumeroEnderecoClientes);
enderecoCompactado.setPais(listaPaisEnderecoClientes);
enderecoCompactado.setRua(listaRuaEnderecoClientes);

Pais paisCompactado =
    clienteCompactado.addNewPais();

Pai paiCompactado =
    paisCompactado.addNewPai();
paiCompactado.setEmail(listaEmailPai);
paiCompactado.setNome(listaNomePai);
paiCompactado.setProfissao(listaProfissaoPai);
paiCompactado.setTelefone(listaTelefonePai);
paiCompactado.addNewEndereco();
paiCompactado.getEndereco().setCep(listaCepEnderecoPai);
paiCompactado.getEndereco().setCidade(listaCidadeEnderecoPai);
paiCompactado.getEndereco().setEstado(listaEstadoEnderecoPai);
paiCompactado.getEndereco().setNumero(listaNumeroEnderecoPai);
paiCompactado.getEndereco().setPais(listaPaisEnderecoPai);
paiCompactado.getEndereco().setRua(listaRuaEnderecoPai);

Mae maeCompactado =
    paisCompactado.addNewMae();
maeCompactado.setEmail(listaEmailMae);
maeCompactado.setNome(listaNomeMae);
maeCompactado.setProfissao(listaProfissaoMae);
maeCompactado.setTelefone(listaTelefoneMae);
maeCompactado.addNewEndereco();
maeCompactado.getEndereco().setCep(listaCepEnderecoMae);
maeCompactado.getEndereco().setCidade(listaCidadeEnderecoMae);
maeCompactado.getEndereco().setEstado(listaEstadoEnderecoMae);
```

```
            maeCompactado.getEndereco().setNumero(listaNumeroEnderecoMae);
            maeCompactado.getEndereco().setPais(listaPaisEnderecoMae);
            maeCompactado.getEndereco().setRua(listaRuaEnderecoMae);

            Filhos filhosCompactado =
                clienteCompactado.addNewFilhos();

            Filho filhoCompactado =
                filhosCompactado.addNewFilho();

            filhoCompactado.setEmail(listaEmailFilho);
            filhoCompactado.setNome(listaNomeFilho);
            filhoCompactado.setSexo(listaSexoFilho);
            filhoCompactado.setTelefone(listaTelefoneFilho);

            Endereco enderecoFilho =
                filhoCompactado.addNewEndereco();

            enderecoFilho.setCep(listaCepEnderecoFilho);
            enderecoFilho.setCidade(listaCidadeEnderecoFilho);
            enderecoFilho.setEstado(listaEstadoEnderecoFilho);
            enderecoFilho.setNumero(listaNumeroEnderecoFilho);
            enderecoFilho.setPais(listaPaisEnderecoFilho);
            enderecoFilho.setRua(listaRuaEnderecoFilho);

            XMLSerializer writer = new XMLSerializer();
            try {
                writer.setOutputByteStream(new FileOutputStream(
                        "D:\\XML_Compactado.xml"));
                //amarrei com um FileInputStream pra sobrescever o arquivo
                writer.serialize(clientesDocumentCompactado.getDomNode());

            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        } catch (XmlException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```