UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO ALVES TEIXEIRA

lattes2latex: Uma Ferramenta para Conversão de Currículos Lattes em Documentos LaTeX

Trabalho de Conclusão apresentado como
requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Prof. Dr. Nicolas Maillard
Orientador

Porto Alegre, julho de 2009

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Teixeira, Marcelo Alves

lattes2latex: Uma Ferramenta para Conversão de Currículos Lattes em Documentos LaTeX / Marcelo Alves Teixeira. – Porto Alegre, 2009.

76 f.: il.

Trabalho de Conclusão (graduação) – Universidade Federal do Rio Grande do Sul. Curso de Bacharelado em Ciência da Computação, Porto Alegre, BR–RS, 2009. Orientador: Nicolas Maillard.

1. Lattes. 2. Latex. 3. Ruby. I. Maillard, Nicolas. II. Título.

“Obstacles are those frightful things you see when you take your eyes off your goal.”
— HENRY FORD

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
MCT	Ministério da Ciência e Tecnologia
C&T	Ciência e Tecnologia
MS-DOS	Microsoft Disk Operating System
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
LMPL	Linguagem de Marcação da Plataforma Lattes
DTD	Data Type Definition
XML	eXtended Markup Language
RTF	Rich Text Format
W3C	World Wide Web Consortium
SGML	Standart Generalized Markup Language
HTML	HyperText Markup Language
XSL	eXtensible Style Language
DTD	Data Type Definition
API	Application Programming Interface
DOM	Document Object Model
SAX	Simple API for XML
XPath	XML Path Language
DOC	Document
MOSP	Microsoft Open Specification Promise
OASIS	Organization for the Advancement of Structured Information Standards
ODF	OpenDocument Format
ISO	Internation Organization for Standardization
OOXML	Office Open XML
ECMA	European Computer Manufacturers Association

PDF	Portable Document Format
CONSCIENTIAS	Comunidade para Ontologias em Ciência, Tecnologia e Informações de Aperfeiçoamento de Nível Superior

LISTA DE FIGURAS

LISTA DE LISTAGENS

RESUMO

O Currículo Lattes é um dos principais instrumentos da Plataforma Lattes utilizado pelo CNPq em suas atividades operacionais e de fomento em relação à pesquisa em Ciência & Tecnologia no Brasil. Com o constante crescimento desta plataforma, têm-se a disposição uma base de dados bastante detalhada e abrangente sobre o histórico de pesquisadores, estudantes e dos projetos de pesquisa desenvolvidos na área. Por ser uma base alimentada pelos próprios pesquisadores, o Currículo Lattes torna-se, também, a principal ferramenta de cadastro destes no que se refere a suas informações acadêmicas e profissionais. Entretanto, o limitado espectro de formatos eletrônicos de saída oferecidos pelo sistema *online* da ferramenta limita o reuso das informações por parte do pesquisador.

Este trabalho objetiva oferecer ao pesquisador uma maneira mais prática de acesso as informações presentes em seu Currículo Lattes, através do padrão LaTeX, um conjunto de macros definidas para o processador de texto TeX, utilizado amplamente na produção de textos matemáticos e científicos.

Para isto, será desenvolvida uma ferramenta que possibilite a conversão das informações do Currículo Lattes, armazenadas em XML, para o formato LaTeX, de acordo com regras de formatação previamente definidas ou especificadas pelo usuário.

Palavras-chave: Lattes, latex, ruby.

lattes2latex: A Tool for Conversion of Lattes Curriculum to LaTeX Documents

ABSTRACT

The Lattes Curriculum is one of the main instruments used by CNPq in its operational activities and on promoting research in Science & Technology in Brazil. With the constant growth of this platform, we have a very detailed and comprehensive data base about the history of researchers, students and research projects developed in the area. As a base fed by the researchers, the Lattes Curriculum it is also the main tool for such registration in relation to their academic and professional information. However, the limited range of output electronic formats supplied by the online tool limits the reuse of information by the researcher.

This study aims to offer researchers a more practical way to access the information in their CV Lattes through the LaTeX standard, a set of macros defined for the TeX word processor, widely used in the production of mathematical and scientific texts.

For this, it will be developed a tool for the conversion of information from the Lattes Curriculum, stored in XML, to the LaTeX format according to predefined rules or format specified by the user.

Keywords: Lattes, latex, ruby.

1 INTRODUÇÃO

Este capítulo contém uma introdução ao trabalho de conclusão de curso. O tema deste trabalho é *lattes2latex - Uma Ferramenta para Conversão de Currículos Lattes em Documentos LaTeX*. As seções a seguir apresentam a contextualização e definição do problema, os objetivos almejados e a organização do trabalho.

1.1 Contextualização: A Plataforma Lattes

Destinada ao fomento da pesquisa científica e tecnológica e à formação de recursos humanos para pesquisas no país, o CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico, é uma agência do Ministério da Ciência e Tecnologia - (MCT, 2008). Sua história está diretamente ligada ao desenvolvimento científico e tecnológico do Brasil contemporâneo (CNPQ, 2008a).

A Plataforma Lattes é a base de dados de currículos e instituições das áreas de Ciência e Tecnologia - C&T - que representa a experiência do CNPq na integração de informações. Sua importância atual se estende, não só as atividades operacionais e de fomento do CNPq, como também à gestão e às ações de outras agências federais e estaduais.

Criado devido à necessidade do CNPq de manter uma base de dados sobre pesquisadores em C&T, o sistema Lattes surgiu em 1993, com a utilização de formulários de papel, um sistema em ambiente MS-DOS[1] e um sistema de currículos específico para credenciamento de orientadores. Deste ano até 1998, a agência acumulou cerca de 35 mil registros curriculares de todo o país, viabilizando a operação de fomento da agência, embora a natureza das informações dificultasse uma plena utilização desta base de dados em outros processos de gestão em C&T.

Percebendo essa dificuldade, em 1999 o CNPq realizou um levantamento para definir um modelo de currículo que atendesse tanto às suas necessidades de operação de fomento como às de planejamento e gestão em C&T. Entre março e abril de 1999 foi avaliado o primeiro protótipo de currículo Lattes, à época denominado CV-Genos.

Em maio de 1999, CNPq e CAPES[2] acordaram a completa compatibilização do novo currículo do CNPq com os dados de pós-graduação catalogados pela CAPES. O encontro entre ambas as agências resultou na modificação do protótipo, que se transformou no

[1] Sistema Operacional comercializado pela Microsoft

[2] Agência de fomento à pesquisa brasileira que atua na expansão e consolidação da pós-graduação *stricto sensu* (mestrado e doutorado) em todos os estados do Brasil

Sistema de Currículos Lattes, lançado em 16 de agosto de 1999, e que, nos dois primeiros anos, teve um aumento no número de currículos de 300%, superando a marca dos 100 mil currículos.

Com esta base crescendo constantemente, em julho de 2000 a Coordenação Geral de Informática do CNPq iniciou um trabalho de intercâmbio com outras instituições ligadas à C&T no país, resultando em ligações dinâmicas com outras bases de dados nos currículos Lattes de pequisadores. Ao mesmo tempo também foi desenvolvida uma ferramenta on-line, sobre uma plataforma *Web*, que permite aos pesquisadores atualizarem seus currículos diretamente na base do CNPq.

A partir desta integração, também no ano de 2000, instituições federais de ensino superior reuniram suas equipes de informática e convidaram as agências federais para construir um modelo único de informação. Foi criada então a Comunidade Virtual LMPL, acrônimo para Linguagem de Marcação da Plataforma Lattes, que definiu o modelo DTD XML do Currículo Lattes. Com este modelo, as universidades brasileiras podem extrair informações do currículo Lattes e gerar informações para o mesmo a partir de seus sistemas corporativos.

Atualmente, a base conta com cerca de 1.100.000 currículos, sendo que 31% destes são de doutores, mestres e estudantes de pós-graduação e 59% de graduados e estudantes de graduação (CNPQ, 2008b).

1.2 Definição do Problema

Com o crescimento cada vez maior da base de dados da Plataforma Lattes, o CNPq e as instituições federais de ensino superior têm a sua disposição informações detalhadas do histórico de pesquisadores e estudantes bem como do andamento das pesquisas na área de C&T em todo o país.

Membros destas instituições de ensino, pesquisadores e estudantes, interessados em contribuir para o constante aprimoramento desta base de dados, fazem do Sistema de Currículos Lattes sua principal ferramenta de cadastro no que se refere a informações acadêmicas, como publicações, participações em eventos ou bancas, e profissionais. No entanto encontram uma deficiência na utilização desta ferramenta: a reutilização das informações existentes. Os recursos disponibilizados pelo Sistema de Currículos Lattes são escassos, permitindo apenas a exportação do currículo no formato RTF ou como uma base de dados XML. Isto dificulta e por vezes impede que pesquisadores e estudantes utilizem-se de suas informações armazenadas.

1.3 Objetivo

Com a possibilidade de acesso a base de dados XML gerada a partir do Sistema de Currículos Lattes e a necessidade de melhor reutilizar as informações existentes neste sistema, o presente trabalho tem como objetivo geral permitir o acesso do pesquisador a seus dados através de uma interface mais amigável: o padrão LaTeX. Para isto destacam-se os seguintes objetivos específicos:

- Identificar e analisar o formato de especificação de currículos da Plataforma Lattes.
- Disponibilizar uma ferramenta para a geração de uma representação de dados intermediária a partir do formato especificado.
- Implementar a tradução dos dados colhidos em código LaTeX através da utilização da ferramenta criada.
- Analisar os resultados obtidos e as alternativas presentes no uso da ferramenta.

1.4 Organização do Texto

O restante deste projeto será organizado da seguinte maneira: o Capítulo 2 apresenta conceitos básicos do formato XML, da especificação de linguagens através do DTD e métodos de análise destes conteúdos. O Capítulo 3 traz definições da linguagem de programação, ferramentas para análise dos dados e conceitos de métodos de programação. O Capítulo 4 é constituído de uma descrição dos possíveis formatos eletrônicos de saída analisados e a motivação na escolha do LaTeX. O Capítulo 5 apresenta a definição do Currículo Lattes além de detalhar a etapa de desenvolvimento dos *parsers* de DTD e XML, apresentando o processo de tradução do Currículo Lattes em base XML para o código LaTeX final. O Capítulo 6 permite analisar o resultado final obtido neste trabalho assim como abordar outras perspectivas de utilização da ferramenta. Por fim, o Capítulo 7 apresenta as conclusões do projeto.

Ao longo do trabalho, conceitos básicos necessários para a compreensão do funcionamento do conversor serão introduzidos e, a medida que isto for sendo feito, a figura 1.1 será atualizada com a inclusão do respectivo conceito e sua importância no processo.

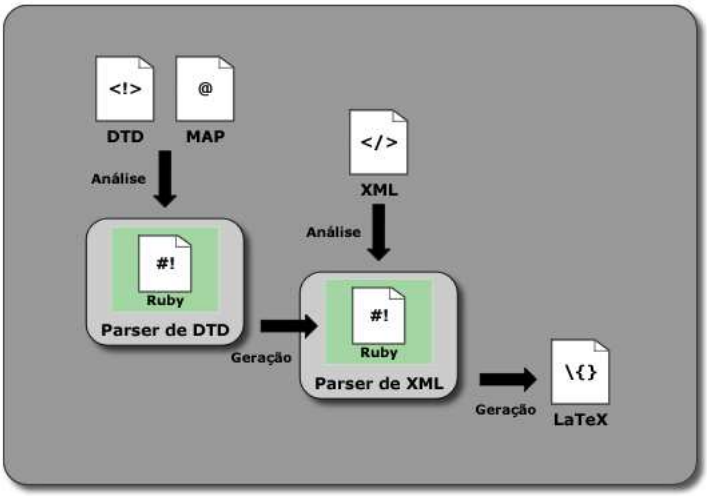


Figura 1.1: Processo de Conversão do Currículo Lattes em Código LaTeX

2 A ENTRADA: XML - EXTENDED MARKUP LANGUAGE

O modo mais comum e intuitivo de armazenamento de informações é através da palavra escrita em papel. Porém, dados mantidos desta forma têm acesso realizado de forma lenta e difícil. Quando surge a oportunidade de manipulação destas informações se faz necessária a intervenção do ser humano, fortemente limitado em sua capacidade de análise e processamento de grandes volumes de dados. Esta limitação acaba gerando o principal gargalo no desempenho de tarefas que envolvam um grande número de informações. No caso específico de informações dinâmicas, de atualização constante, isto torna a correta manutenção dos dados onerosa e inviável em muitos casos.

Um exemplo bastante próximo pode ser encontrado na prática legislativa brasileira: é comum encerrar diplomas legais com a expressão “revogam-se as disposições em contrário”. É praticamente impossível saber quais disposições foram revogadas e, para uma determinada lei, é difícil saber se não foi suplantada por outra posterior. O problema está em consultar e interpretar o grande número de textos legais, uma tarefa que ultrapassa a capacidade de análise do ser humano.

Com o advento e popularização da computação surgiram métodos mais eficientes de armazenamento de informações, permitindo uma consulta ou atualização mais rápida e eficaz. Ao longo dos anos, formatos de armazenamento foram sendo definidos, cada um com um enfoque específico, gerando vantagens em determinados aspectos graças as suas particularidades.

2.1 XML

Estimulado pela insatisfação gerada devido à alta complexidade dos formatos de armazenamento, padronizados ou não, existentes à epoca, em meados da década de 90 o W3C, *World Wide Web Consortium*, começou a trabalhar no desenvolvimento de uma linguagem de marcação[1] que combinasse a flexibilidade da SGML[2] com a simplicidade da HTML[3] - Figura 2.1. A idéia era criar uma linguagem que pudesse ser facilmente lida e interpretada por humanos e computadores, integrando-se com as demais já existentes (W3C, 2008a).

[1] Conjunto de marcações aplicadas a um texto ou a dados com o fim de adicionar informações particulares sobre os mesmos.

[2] Metalinguagem através da qual se pode definir linguagens de marcação para documentos.

[3] Linguagem de marcação utilizada para produzir páginas na *Web*.

O desenvolvimento desta nova linguagem baseou-se nos seguintes princípios:

- Separação entre conteúdo e formatação.
- Simplicidade e legibilidade, tanto para humanos como para computadores.
- Possibilidade de criação de *tags* sem limitação.
- Criação de arquivos para validação de estruturas.
- Integração com bancos de dados distintos.
- Preocupação com a estrutura da informação, e não sua aparência.

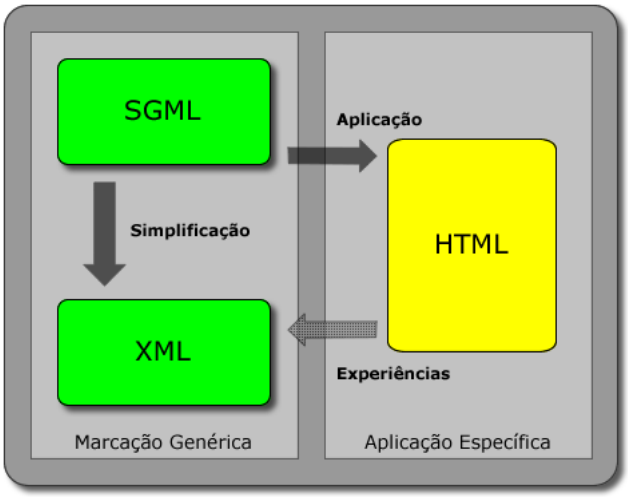


Figura 2.1: Origem da XML

Como resultado deste esforço, em 1998 nascia a versão 1.0 da XML (W3C, 2008a).

XML - *eXtensible Markup Language* - é uma especificação aberta, de propósito geral, definida para permitir a criação de linguagens de marcação personalizadas. É classificada como linguagem extensível pois permite ao usuário criar os elementos de marcação que serão utilizados.

Tem como proposta auxiliar sistemas de informação no compartilhamento de dados através de qualquer meio de transmissão como, por exemplo, a *Internet*. Também pode e costuma ser utilizada no armazenamento e codificação de informações ou, ainda, na serialização de dados[4].

A tecnologia disponibilizada pela XML, combinada com outros padrões, possuí uma característica fundamental: permite armazenar o conteúdo de um documento em separado de sua formatação. Este isolamento facilita o reuso da informação em outras aplicações ou em diferentes ambientes de apresentação (RAY, 2003, p. 15).

Considerada um bom formato para a criação de documentos com dados hierárquicos, a XML provê uma representação dos dados - listagem 2.1 - que se revelou aplicável e fácil

[4]Processo de salvar um objeto em um meio de armazenamento ou transmití-lo por uma conexão de rede.

de ser desenvolvida. Implementações industriais demonstraram a qualidade intrínseca e o potencial do formato estruturado em árvore dos documentos em XML.

```
<?xml version="1.0"?>
<!-- Acervo de Livros -->
<acervo>
  <livro id="1">
    <titulo>Um Livro</titulo>
    <autor>Um Autor</autor>
  </livro>
  <livro id="2">
    <titulo>Outro Livro</titulo>
    <autor>Outro Autor</autor>
  </livro>
  <livro id="3">
    <titulo>Mais um Livro</titulo>
    <autor>Mais um Autor</autor>
  </livro>
</acervo>
```

Listagem 2.1: Documento XML

A representação de um documento XML se dá através de um texto plano com *tags* de marcação e suas respectivas informações. Graficamente podemos considerar este documento como uma árvore rotulada - figura 2.2 - com nós consistindo de:

- Um nome.
- Um conjunto de atributos, cada qual consistindo de um nome e um valor (opcionais).

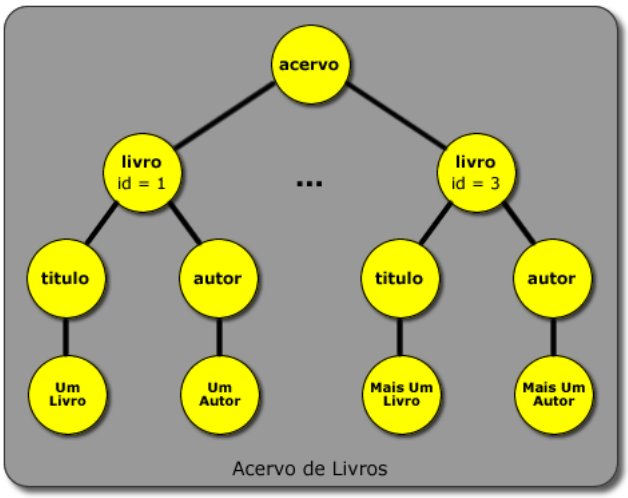


Figura 2.2: Árvore XML

A especificação XML conta ainda, devido ao conceito de isolamento entre informações e a forma de apresentação, com o recurso de folhas de estilos, definidas através da XSL[5] - *eXtensible Style Language* - (W3C, 2008b), para a apresentação dos dados. Este

[5]Linguagem de transformação que permite descrever como arquivos no formato XML serão formatados.

recurso permite a visualização e o processamento da mesma informação através de inúmeras maneiras diferentes, dependendo exclusivamente da folha de estilo aplicada sobre o arquivo XML, como podemos ver na figura 2.3.

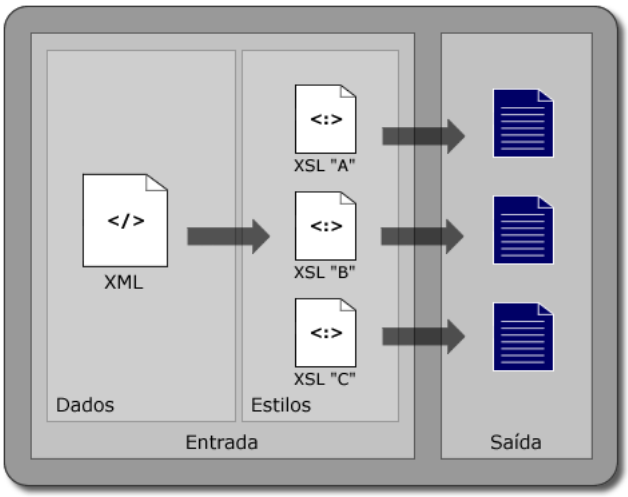


Figura 2.3: Folhas de Estilos em XML: a XSL

Sensível ao contexto, um documento XML é classificado como *well-formed* quando atende a algumas regras básicas que permitem com que os dados sejam lidos e expostos sem nenhuma descrição externa ou conhecimento da semântica das informações (HAROLD, 2004, p. 156). São elas:

- Existe somente uma *tag* raiz.
- Toda *tag* de início deve ser encerrada com uma *tag* de fim correspondente.
- As *tags* de elementos têm que ser apropriadamente aninhadas.

Além de bem formado, um documento XML é definido como *valid* quando atende a regras de estruturação especificadas em um arquivo especial, o DTD.

2.2 DTD

Na especificação XML, apesar de ser definida uma regra de formação dos dados, não existe nenhuma restrição quanto à estrutura da informação. As regras que definem a validade de um documento em relação a sua estrutura são especificadas por arquivos denominados DTD - *Data Type Definition*. O DTD é uma especificação, também originada a partir da SGML, criada como um complemento a utilização da XML (W3C, 2008c).

Uma aplicação, ao processar os dados presentes em um arquivo XML, pode não possuir nativamente um conhecimento das informações. Nesta situação, a validação do documento de entrada costuma ser realizada pela associação deste com uma definição presente em um DTD que especifica a estrutura da informação esperada. Um trecho da aplicação analisa as informações presentes no arquivo XML através de um *parsing* e as compara com as regras de formação contidas na especificação DTD informada - figura 2.4.

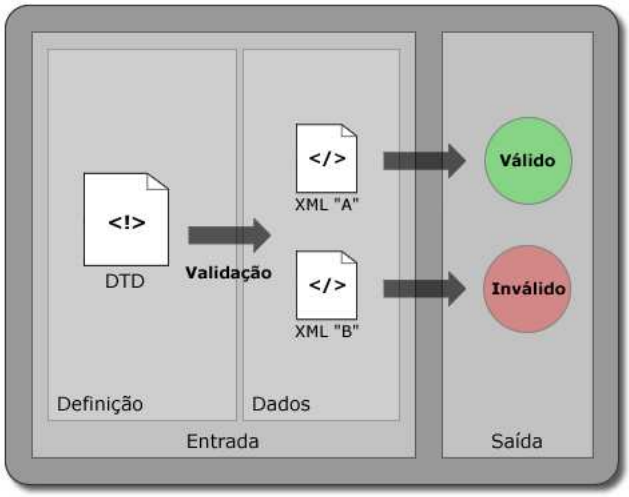


Figura 2.4: Validação de Arquivo XML

Um arquivo XML pode também ser associado diretamente a uma definição DTD permitindo que aplicações realizem a validação do conteúdo XML de maneira transparente ao usuário. Esta associação acontece de duas formas (HAROLD, 2004, p. 189):

Interna - quando as declarações de estrutura em formato DTD estão presentes diretamente no corpo do arquivo XML.

Externa - quando existe uma referência *Document Type* no cabeçalho do arquivo XML, apontando para um segundo arquivo que contenha a definição das informações.

O DTD, que define o conjunto de regras da estrutura do documento XML indicando os elementos que podem ser utilizados e onde podem ser aplicados, é composto pelos seguintes blocos (HUNTER et al., 2007):

```
1 <!ELEMENT elemento categoria>
2 <!ELEMENT elemento (conteudo)>

4 <!ATTLIST elemento nome-atributo tipo-atributo valor-padrao>

6 <!ENTITY entidade "valor">
```

Listagem 2.2: Declarando Blocos no DTD

Elementos são os principais componentes de documentos XML - linhas 1 e 2.

Atributos permitem incluir informações complementares sobre os elementos. Incluídos nas *tags* de abertura, são formados por um par nome/valor - linha 4.

Entidades representam caracteres que possuem um comportamento especial em arquivos XML - linha 6.

Com poucas linhas é possível definir a estrutura de documentos XML complexos - listagem 2.3 - permitindo a validação dos mesmos.

```
1 <!ELEMENT acervo (livro*)>
2 <!ELEMENT livro (titulo, autor)>
3 <!ELEMENT titulo (CDATA)>
4 <!ELEMENT autor (CDATA)>

6 <!ATTLIST livro id ID 0>
```

Listagem 2.3: Arquivo DTD Completo

2.3 *Parsing*

Parsing é o processo de reconhecer uma sequência de entrada (por exemplo, palavras) através de expressões regulares[6] para aferir se sua estrutura gramatical é válida segundo uma determinada gramática formal[7] (PRICE; TOSCANI, 2005). Um *parser* é o componente de um interpretador ou compilador que verifica se uma entrada possui sintaxe correta, gerando uma representação dos dados que simboliza a estrutura sintática das informações presentes no documento - figura 2.5.

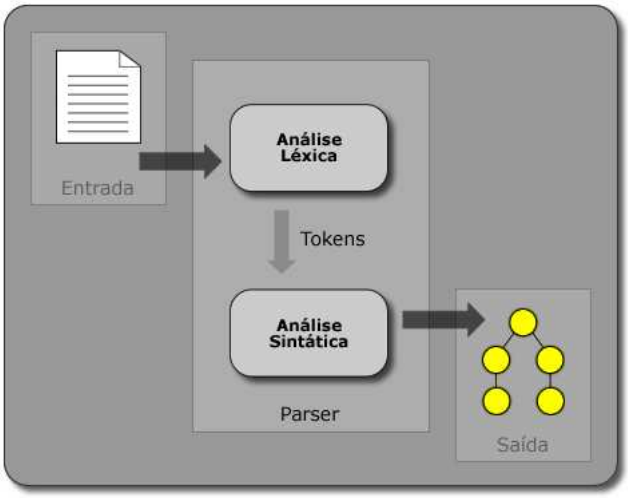


Figura 2.5: Processo de *Parsing*

Este processo de *parsing* é dividido em dois estágios:

Análise Léxica é a responsável pela geração dos *tokens*[8], a partir da sequência de entrada, através da divisão da mesma de acordo com uma determinada regra definida em uma expressão regular.

Análise Sintática é o estágio que verifica se os *tokens* de entrada formam uma expressão válida de acordo com a especificação da gramática formal da linguagem.

[6] Mecanismo que provê uma forma concisa e flexível de identificar sequências de caracteres de interesse sem que haja a necessidade de se listar todos os elementos do conjunto procurado.

[7] Objeto matemático que permite especificar uma linguagem através de um conjunto de regras que descrevem quais sequências formadas a partir de um alfabeto de uma linguagem formal são sintaticamente válidas de acordo com a linguagem.

[8] Segmento de texto ou símbolo que fornece significado ao contexto.

A tarefa do *parsing* é determinar se uma entrada de dados pode ser derivada com as regras de uma gramática formal. Isso pode ser feito de duas maneiras:

Descendente (*top-down*) onde um analisador pode iniciar com o símbolo inicial da gramática e tentar aplicar as regras da mesma até transformá-lo na entrada de dados. Intuitivamente, o analisador inicia dos elementos mais genéricos e os divide em elementos mais especificos, recursivamente, até encontrar a entrada e aceitá-la ou esgotar todas as possibilidades e rejeitá-la.

Ascendente (*bottom-up*) onde um analisador pode iniciar com a entrada de dados e tentar reescrevê-la, através da aplicação das regras da gramática, até formar o símbolo inicial da mesma. Intuitivamente, o analisador localiza os elementos mais básicos e, então, forma elementos mais genéricos, recursivamente, até encontrar o símbolo inicial da gramática e aceitar a entrada ou esgotar todas as possibilidades e rejeitá-la.

Os principais quesitos analisados na avaliação da qualidade de um *parser* são:

Velocidade - ao realizar um *parsing* de uma entrada pequena, com poucas instruções, é provável que a velocidade não seja um diferencial. Entretanto, a medida que esta entrada cresce, passando a apresentar centenas ou até mesmo milhares de *tokens*, a variação da velocidade de análise pode tornar-se significativa.

Complexidade - normalmente relacionada com a velocidade, um *parser* costuma ser rápido em sua análise quando é simples de ser utilizado.

2.3.1 *Parsing* de XML

Todo programa que recebe dados no formato XML deve ter um *parser* para analisar estas informações. O *parser* recebe como entrada o conteúdo do arquivo XML, formatado através de suas *tags*, e gera uma representação estruturada que pode ser percorrida e operada de inúmeras formas.

Internamente, os *parsers* de XML disponíveis atualmente utilizam, em sua grande maioria, pelo menos uma das seguintes APIs[9] - *Application Programming Interface* - para acessar as informações contidas em um documento de entrada (HUNTER et al., 2007) - figura 2.6:

DOM - *Document Object Model* - representa o documento XML através da criação de uma estrutura de árvore interna, permitindo que qualquer elemento seja acessado, alterado e removido e que novos elementos sejam criados percorrendo-se esta estrutura. Sua desvantagem evidente é o alto consumo de memória necessário para manter a árvore da estrutura acessível.

SAX - *Simple API for XML* - é uma API baseada em eventos, ou seja, reporta eventos de *parsing*, como início ou fim de um elemento, diretamente a aplicação chamadora

[9] Conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por programas aplicativos.

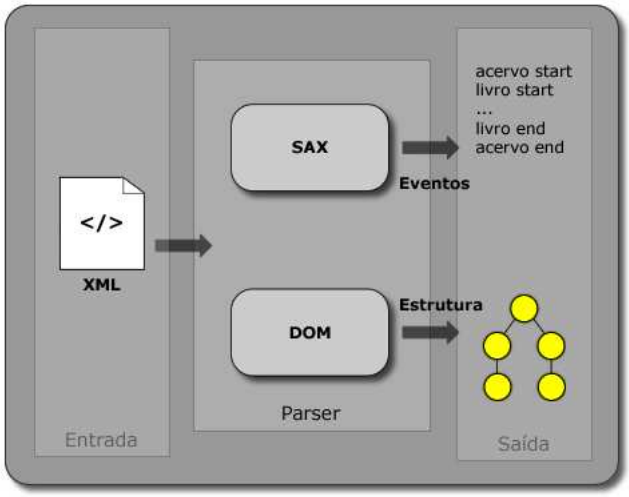


Figura 2.6: Alternativas de *Parsing* de XML

através de *callbacks*[10]. Por não manter a estrutura em memória, é mais eficiente em termos de velocidade do que a alternativa anterior mas não permite edição dos dados.

Além da utilização das APIs referenciadas, os *parsers* de XML podem ser classificados, no que diz respeito a análise sintática de documentos, como *non-validating* ou *validating*. *Parsers non-validating* certificam-se apenas de que o documento XML apresentado é *well-formed*. Os *parsers validating*, por sua vez, realizam uma verificação mais rigorosa, certificando-se também de que o documento é *valid* de acordo com as regras definidas em uma especificação DTD. Estes *parsers* podem inclusive utilizar informações presentes no DTD, como as definições de entidades e de atributos padrão, para complementar os dados recebidos através da XML.

A geração de *parsers* de XML *validating* pode ser feita através da análise do arquivo DTD que especifica a linguagem XML em questão. Esta análise é realizada por outro processo de *parsing*, aplicado sobre a DTD, o *parsing* de DTD.

2.3.2 *Parsing* DTD

Assim como a XML, o DTD também é especificado através de uma linguagem, com uma gramática formal bem definida. Desta forma é possível realizar o processo de *parsing* em arquivos DTD da mesma forma com que são feitos em arquivos XML. Entretanto, ao analisar a estrutura de um DTD, têm-se como retorno uma abstração da definição da linguagem, e não dados como nos arquivos XML.

Através desta abstração obtida é possível especificar-se, de maneira automatizada, um *parser* de XML *validating*, ou seja, que valide as informações de um arquivo XML quanto à estrutura definida no DTD.

As APIs de acesso as informações de *parsers* de XML são também empregáveis em *parsers* de DTD, entretanto a primeira opção - DOM - costuma ser mais utilizada

[10]Código executável passado como argumento a outro trecho de código.

pois permite que se tenha toda a estrutura da definição da linguagem armazenada, o que permite que se gere como saída informações que englobem toda a especificação, e não elementos de maneira pontual.

2.3.3 *Parsers* em Linguagens de Programação

Um sistema que disponibiliza ou consome dados em XML pode ser desenvolvido em qualquer uma das inúmeras linguagens de programação existentes. As linguagens imperativas mais utilizadas atualmente possuem *parsers* XML nativos ou bibliotecas que desempenham esta função de forma plenamente satisfatória. A escolha por uma linguagem se dá através da análise de requisitos do sistema, como desempenho, interatividade, usabilidade, confiabilidade, portabilidade, etc, além da preferência pessoal do programador.

2.3.3.1 Java

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90. Diferentemente das linguagens convencionais é compilada para um código intermediário que é executado por uma máquina virtual (SUN, 2008). Possui uma biblioteca para tratamento de XML chamada Xerces - listagem 2.4 - que suporta integralmente as APIs DOM e SAX (APACHE, 2008).

```
1  import org.w3c.dom.*;
2  import org.apache.xerces.parsers.DOMParser;

4  public class DOM {
5    public static void main(String[] args) {
6      try {
7        DOMParser parser = new DOMParser();
8        parser.parse(args[0]);
9        Document doc = parser.getDocument();

11       NodeList nodes = doc.getElementsByTagName("livro");
12       System.out.println("Existem " + nodes.getLength() +
13                          " livros.");
14     } catch (Exception ex) {
15       System.out.println(ex);
16     }
17   }
18 }
```

Listagem 2.4: *Parser* XML em Java com Xerces

2.3.3.2 C

C é uma linguagem de programação compilada de propósito geral criada em 1972 para desenvolver o sistema operacional UNIX. É uma das linguagens de programação mais usadas, além de ter influenciado na criação de muitas outras linguagens (KERNIGHAN; RITCHIE, 1988). Também possui uma biblioteca de *parser* XML, a eXpat (CLARK, 2008), com suporte somente a API SAX.

2.3.3.3 .NET

.NET é uma iniciativa da Microsoft visando uma plataforma única para desenvolvimento e execução de sistemas e aplicações (MICROSOFT, 2008b). Todo e qualquer código gerado para .NET pode ser executado em qualquer dispositivo que possua um *framework* de tal plataforma. Assemelha-se a idéia do Java, onde o programador deixa de escrever código para um sistema ou dispostivo específico e passa a escrever para uma plataforma.

Uma biblioteca pode ser gerada para .NET a partir de qualquer uma das linguagens pertencentes a lista suportada pelo *framework*. Isto permite que qualquer *parser* XML desenvolvido em uma linguagem integrada a plataforma possa ser utilizado.

2.3.3.4 Ruby

Ruby é uma linguagem de programação interpretada, orientada a objetos, projetada tanto para programação em grande escala quanto para codificação rápida, além de possuir vários repositórios de bibliotecas disponíveis (MATSUMOTO, 2008a). O Ruby traz em sua biblioteca padrão o REXML - listagem 2.5 - uma API para um processador XML relativamente rápido, intuitivo e implementado totalmente em Ruby (MATSUMOTO, 2008b).

```
1 require 'rexml/document'
2 include REXML

4 doc = Document.new(File.new(ARGV[0]))
5 nodes = doc.elements['acervo\livro']
6 puts "Existem #{nodes.size()} livros."
```

Listagem 2.5: *Parser* XML em Ruby com REXML

Optou-se, neste trabalho, por utilizar a linguagem de programação Ruby, discutida no capítulo 3, para:

- Implementar o *parser* de DTD que irá gerar o *parser* de XML a partir da DTD do Currículo Lattes.
- Implementar, através da saída do *parser* de DTD, o *parser* de XML que irá traduzir um arquivo XML do Currículo Lattes em código LaTeX.

Neste capítulo foram introduzidos os conceitos de XML, DTD e *Parsers* - figura 2.7, fundamentais para o entendimento da especificação do Currículo Lattes e do processo que converte uma instância deste currículo no código LaTeX desejado. A seguir teremos um maior detalhamento da linguagem de programação Ruby e das ferramentas que levaram esta a ser escolhida como base para implementação deste trabalho.

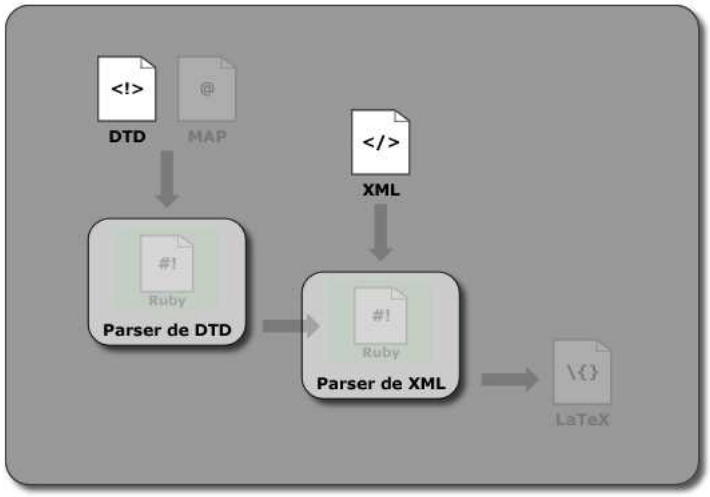


Figura 2.7: Processo de Conversão do Currículo Lattes em Código LaTeX

3 A LINGUAGEM DE PROGRAMAÇÃO

O meio mais eficaz de comunicação entre pessoas é a linguagem. Na programação de computadores, uma linguagem de programação serve como meio de comunicação entre o indivíduo que deseja resolver um determinado problema e o computador escolhido para ajudá-lo na solução. A linguagem de programação deve fazer a ligação entre o pensamento humano (muitas vezes, de natureza não estruturada) e a precisão requerida para o processamento pela máquina.

Uma das principais metas das linguagens de programação é o aumento da produtividade dos programadores, permitindo que estes expressem suas intenções mais facilmente do que quando feito diretamente através de linguagem de máquina (código binário). Para isso, quanto mais alto o nível de abstração de uma linguagem de programação, maior será a facilidade do programado em especificar o que deseja (MITCHELL, 2003, p. 3).

A tradução de uma linguagem de programação em código de máquina se dá através dos tradutores, que assumem dois formatos (SCOTT, 2006, p. 18): compiladores e interpretadores.

Compilação é o processo que realiza a tradução de todo o código do programa para só depois permitir a execução. O código resultante é armazenado de forma que o programa possa ser executado um número indefinido de vezes sem que seja necessária uma nova compilação. Entre as linguagens compiladas mais comuns estão o Pascal e o C.

Interpretação é o formato no qual a tradução do código do programa é feita à medida em que o mesmo é executado, caso de linguagens como Javascript, Python e Ruby. A execução deste tipo de código é geralmente mais lenta do que a de um código compilado, mas este método é também mais flexível, já que pode interagir com o ambiente de maneira mais amigável - figura 3.1.

As linguagens de programação também são classificadas, quanto a conceitos e abstrações utilizados na representação de elementos em um programa em diferentes paradigmas de programação, dentre os quais os seguintes (SCOTT, 2006, p. 18):

Imperativo - é o paradigma que descreve uma sequência de computações através de declarações que alteram o estado do programa. É caracterizado por instruções que informam ao computador como determinado dado deve ser processado. Neste paradigma temos linguagens procedurais, que determinam seu fluxo através de chama-

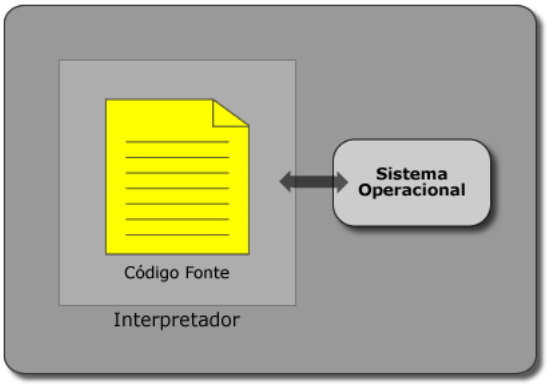


Figura 3.1: Interpretação

das de procedimentos, e as linguagens orientadas à objetos, que utilizam o conceito de objetos, suas propriedades e métodos.

Declarativo é o paradigma que expressa a lógica de uma computação sem especificar seu fluxo. Suas instruções determinam à máquina o que deve ser realizado ao invés de como fazê-lo. A programação funcional, que trata as computações como funções matemáticas, e a programação lógica, que usa a matemática lógica e suas sentenças para representar as computações, são alguns exemplos de linguagens presentes neste paradigma.

3.1 Ruby

Criada em 1993 por Yukihiro Matsumoto, que buscava uma linguagem de *script* mais poderosa que Perl[1] e mais orientada à objetos que Python[2], Ruby é uma linguagem de programação interpretada que combina aspectos de programação funcional e imperativa (FLANAGAN; MATSUMOTO, 2008).

3.1.1 Conceitos Básicos

Ruby é uma linguagem genuinamente orientada à objetos. Tudo que se pode manipular é um objeto, assim como o resultado destas manipulações. Este comportamento, inspirado em SmallTalk[3], é uma das principais diferenças entre Ruby e a maior parte das outras linguagens de *script*. Apesar de definidas como orientadas à objetos, outras linguagens misturam este conceito com tipos primitivos de dados: em Java, por exemplo, é possível obter o valor absoluto de um número através de uma função que recebe como argumento este numero - `valor_absoluto = Math.abs(numero)`. Em Ruby, o próprio número tem capacidade de informar este valor - `valor_absoluto = numero.abs`. O mesmo é aplicado a todos objetos em Ruby: o código C `strlen(nome)`, por exemplo,

[1]Linguagem de programação estável e multiplataforma, usada em aplicações de missão crítica em todos os setores, sendo destacado o seu uso no desenvolvimento de aplicações web de todos os tipos.

[2]Linguagem de programação de alto nível, interpretada, interativa, orientada a objetos, de tipagem dinâmica e forte.

[3]Linguagem de programação frequentemente utilizada como referência em relação a orientação à objeto.

equivale a `nome.length` em Ruby (THOMAS; HUNT, 2000).

```
1  def diga_boanoite(nome)
2    result = "Boa Noite, " + nome
3    return result
4  end

6  puts diga_boanoite("Marcelo")
7  puts diga_boanoite("Nicolas")
```

Listagem 3.1: Programa Ruby Básico

Como a listagem 3.1 mostra, a sintaxe de Ruby é bastante simplificada, o que a torna extremamente legível. Não existem ponto-e-vírgulas ao final dos comandos, já que cada um está presente em uma linha. Métodos são definidos pela palavra-chave `def`, seguida pelo nome do método e seus parâmetros entre parênteses. Ao invés de utilizar chaves para delimitar o corpo do método, basta encerrá-lo com a palavra-chave `end`. Nas linhas 6 e 7 o método `diga_boanoite` definido é chamado e seu resultado passado como parâmetro para o método `puts`, que exibe seu argumento na saída padrão, seguido de uma quebra de linha.

Ruby também utiliza uma convenção para distinguir o uso de variáveis: o primeiro caractere do nome da variável indica como esta será usada. Variáveis locais, parâmetros de métodos e nomes de métodos devem começar por uma letra minúscula ou *underscore*. Variáveis globais são prefixadas com cifrão `$` e variáveis de instância começam com o sinal de arroba `@`. Variáveis de classe começam com dois sinais de arroba `@@` e, finalmente, nomes de classes, nomes de módulos e constantes devem começar com uma letra maiúscula.

A linguagem conta ainda com duas estruturas de coleções indexadas: *arrays* e *hashes*. Ambas armazenam coleções de objetos acessíveis por uma chave. Em *arrays*, a chave é um número inteiro, enquanto *hashes* suportam qualquer objeto como chave. O acesso à elementos de *arrays* é mais eficiente, entretanto *hashes* proveem uma maior flexibilidade.

Todas as estruturas de controle comuns estão presentes na linguagem, como o condicional `if` e repetições `while`. Assim como na definição de métodos, o corpo destes comandos não é delimitado pelo uso de chaves, mas sim pela utilização da palavra-chave `end` - listagem 3.4.

```
1  if a < 10
2    puts "a menor que dez"
3  elseif a < 20
4    puts "a menor que vinte"
5  else
6    puts "a maior ou igual a vinte"
7  end

9  while i < 20
10   i += 1
11   puts "contando..."
12 end
```

Listagem 3.2: Estruturas de Controle em Ruby

Além das estruturas de controle comuns, Ruby introduz atalhos quando o corpo dos comandos é formado por apenas uma expressão:

```
1 if a < 10
2   puts "a menor que 10"
3 end

5 while i < 20
6   i += 1
7 end
```

Listagem 3.3: Estruturas de Controle em Ruby

Podem também serem escritos como:

```
1 puts "a menor que 10" if a < 10

3 i += 1 while i < 20
```

Listagem 3.4: Estruturas de Controle em Ruby

3.1.2 Conceitos Avançados

A maioria dos tipos presentes em Ruby é familiar à todos os programadores. *Strings*, inteiros, *floats*, *arrays*, etc. estão presentes na maioria das linguagens de programação. Entretanto, o suporte a expressões regulares é nativo, normalmente, em linguagens de *script*, como Ruby e Perl (THOMAS; HUNT, 2000). Expressões regulares são uma poderosa ferramenta para manipulação de texto e, portanto, bastante utilizadas em *parsers*.

Uma expressão regular é uma maneira simples de especificar um padrão de caracteres para serem casados em uma *string*. Em Ruby, a criação de uma expressão regular é feita através da descrição de um padrão entre barras (`/padrao/`). E, como era de se esperar, expressões regulares são objetos e podem ser manipuladas como tal.

Com um objeto de expressão regular criado é possível utilizar o operador `=~` para casar uma *string* contra a expressão regular. Se o padrão for encontrado, `=~` retorna a posição de início, caso contrário, retorna `nil` - listagem 3.5.

```
1 if linha =~ /padrao/
2   puts "padrao reconhecido..."
3 end
```

Listagem 3.5: Expressões Regulares em Ruby

Uma das maiores forças de Ruby é o conceito de blocos, que nada mais são do que trechos de código entre chaves ou `do...end` - listagem 3.6, que podem ser utilizados para implementar *callbacks*, iteradores ou passar trechos de código como parâmetros.

```
1 { puts "ola" }

3 do
4   lista.push(objeto)
5   objeto.acao
6 end
```

Listagem 3.6: Definição de Blocos em Ruby

Um método pode invocar o código de um bloco associado através do comando `yield` - listagem 3.7. Este comando é como uma chamada de método que executa o bloco associado.

```
1 def chamada_bloco
2   puts "inicio do metodo"
3   yield
4   puts "fim do metodo"
5 end

7 chamada_bloco { puts "no bloco" }
```

Listagem 3.7: Uso de Blocos em Ruby

produz:

```
1 inicio do metodo
2 no bloco
3 fim do metodo
```

É possível passar parâmetros na chamada de um bloco, que serão associados à variáveis declaradas entre barras verticais (`|`) no código do bloco. Este recurso é utilizado em Ruby para implementar iteradores: métodos que retornam elementos sucessivos de uma coleção, como um *array* - listagem 3.8.

```
1 animais = %w( cachorro cavalo macaco gato ) # criacao do array
2 animais.each { | animal | puts animal } # iteracao do conteudo
```

Listagem 3.8: Iteradores em Ruby

produz:

```
1 cachorro
2 cavalo
3 macaco
4 gato
```

O método `each` da classe `Array` é implementado através do uso de blocos com parâmetros - listagem 3.9.

```
1 # na classe Array
2 def each
3   for each element # simplificacao do processo de varredura
4     yield(element)
5   end
6 end
```

Listagem 3.9: Implementação de Iteradores em Ruby

A maioria das construções de repetição presentes em outras linguagens são simples chamadas de métodos em Ruby, com a execução do bloco associado zero ou mais vezes - listagem 3.10.

```
1 3.times { print "*" }
2 4.upto(6) { | i | print i }
```

Listagem 3.10: *Looping* em Ruby

produz `***456`.

Além destas construções nativas, Ruby apresenta diversos repositórios *online*, a partir dos quais pode-se adicionar bibliotecas com o uso do *gems*, um gerenciador de bibliotecas que permite controlar a instalação e remoção das mesmas. No entanto, a biblioteca padrão presente na linguagem já é bastante abrangente, disponibilizando as principais ferramentas utilizadas, como o REXML.

3.1.3 REXML

REXML é um processador de XML puramente escrito em Ruby que incluí *parsing* de arquivos XML, consultas em XPath[4] - *XML Path Language* - e geração de documentos XML. Foi desenvolvido para suprir a necessidade de uma API para acesso direto a informações estruturadas em XML, sem que haja a necessidade de uma referência constante a documentação para a realização de tarefas corriqueiras.

"Mantenha o caso comum simples, e o incomum, possível" (MATSUMOTO, 2008b). Com este princípio, o REXML não foi desenvolvido a partir da API DOM padrão, e sim de uma versão adaptada a simplicidade presente na linguagem, tornando-o uma API XML orientada à programadores Ruby.

O REXML suporta tanto o método de *parsing* baseado em árvore (DOM-*Like*) como em *stream* (SAX-*Like*).

```
1 require 'rexml/document'
2 include REXML

4 xml = Document.new(File.open("acervo.xml"))
5 puts "Elemento Raiz: #{xml.root.name}"
6 puts "ID dos Livros"
7 xml.elements.each("livro") { | e | puts e.attributes["id"] }
```

Listagem 3.11: REXML API em Ruby

Utilizando o arquivo XML da listagem 2.1 como entrada, o código da listagem 3.11 produz:

```
1 Elemento Raiz: acervo
2 ID dos Livros
3 1
4 2
5 3
```

3.2 Metaprogramação

Metaprogramação é a escrita de programas que geram ou manipulam outros programas ou a sí mesmos. A ferramenta mais comum de metaprogramação é o compilador, que permite escrever programas em uma linguagem de alto nível a fim de gerar programas em linguagem de máquina. Em muitos casos, isto permite ao programador desenvolver muito mais no mesmo período de tempo que ele gastaria para codificar tudo manualmente, além

[4]Linguagem de busca de elementos em um documento XML definida pelo W3C.

de prover ao programa uma maior flexibilidade no tratamento de novas situações (FLANAGAN; MATSUMOTO, 2008, p. 266).

A linguagem na qual o metaprograma é escrita é chamada de metalinguagem. Por sua vez, a linguagem do programa manipulado ou gerado é chamada de linguagem objeto.

A metaprogramação é geralmente implementada através de duas formas. A primeira é a exposição do mecanismo interno de execução do código da linguagem através de uma API. A segunda é a execução dinâmica de expressões de texto que contém comandos de programação - figura 3.2.

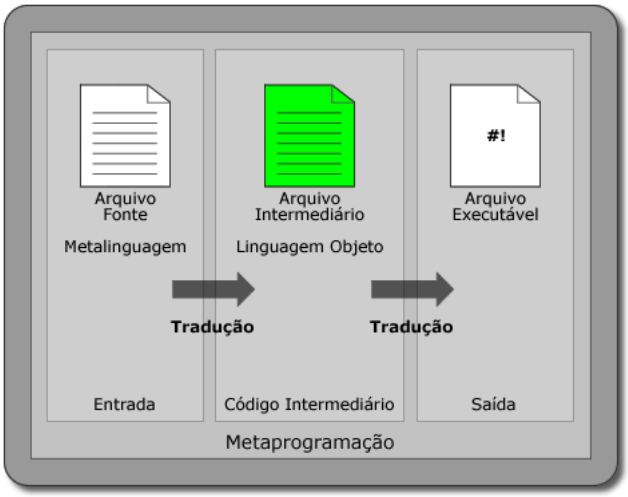


Figura 3.2: Metaprogramação

Um exemplo de metaprogramação em Ruby é o *script* da listagem 3.12, que gera um novo programa - listagem 3.13 - com 7 linhas de código, que imprime os números na faixa de 0 a 4.

```
1 codigo = "#!/usr/bin/ruby\n"
2 codigo << "\n"

4 5.times do | i |
5   codigo << "puts \"#{i}\"\n"
6 end

8 puts codigo
```

Listagem 3.12: Metaprogramação em Ruby

```
1 #!/usr/bin/ruby

3 puts "0"
4 puts "1"
5 puts "2"
6 puts "3"
7 puts "4"
```

Listagem 3.13: Código Gerado Através de Metaprogramação em Ruby

Nem toda metaprogramação envolve programação automática: se os programas são modificados em tempo de execução, então existem técnicas que realizam a metaprogramação sem realmente gerar código fonte.

3.3 Ruby... Por quê?

A definição da linguagem para implementação do trabalho partiu das necessidades impostas pelo mesmo: constante acesso a arquivos de texto plano, necessidade de desenvolvimento de *parsers*, geração de arquivos de código-fonte. Em todos estes quesitos, as linguagens de *script* levam vantagens sobre linguagens compiladas. A análise de informações através do *parsing* de arquivos e do casamento de padrões com a utilização de expressões regulares facilita a tomada de decisão na hora de gerar a saída esperada.

Dentre as linguagens de script disponíves, optou-se por Ruby pois esta oferece o conceito de blocos e iteradores, que facilitam a navegação pela estrutura de arquivos DTD e XML. Além disso, a presença do *parser* REXML auxilia na obtenção dos dados existentes nos arquivos XML e permite abstrair este processo no desenvolvimento do *parser* de XML.

Este capítulo introduziu a principal ferramenta de programação empregada na criação deste conversor, a linguagem Ruby, além da metaprogramação, utilizada na criação do *Parser* de XML a partir do *Parser* de DTD - figura 3.3. No próximo capítulo serão detalhados os formatos eletrônicos estudados como alternativas para a saída do conversor e a motivação na escolha de LaTeX.

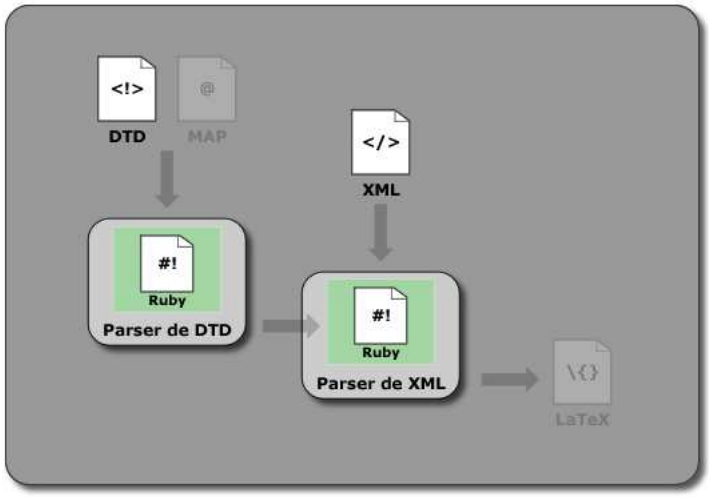


Figura 3.3: Processo de Conversão do Currículo Lattes em Código LaTeX

4 A SAÍDA: FORMATOS ELETRÔNICOS

Durante anos, os documentos públicos foram escritos em tinta sobre papel. O acesso aos mesmos era restrito àqueles que possuíam meios para fazer cópias. Esta situação mudou com a invenção da prensa de tipos móveis, permitindo a produção de cópias de textos a custos muito menores que a contratação de monges copistas[1]. Com a invenção de Gutemberg, todos passaram a ter acesso a cópias legíveis e relativamente baratas de textos e, em especial, de documentos legais.

A chegada dos computadores e da *Internet* anunciou uma nova era na disponibilização de documentos. Até então, o documento e seu suporte, geralmente papel, eram inseparáveis. Havia um original e cópias produzidas a partir dele. Com a chegada dos documentos eletrônicos, o arquivo passou a ser um repositório contendo o documento propriamente dito.

A Internet, por sua vez, tornou possível a distribuição praticamente sem custos dos documentos gerados por órgãos e entidades públicas. Leis, decretos, portarias e diversas outras normas passaram a existir não mais em papel apenas, mas em um mundo virtual onde qualquer um, de qualquer lugar, pode acessá-los.

Este ambiente propiciou a concorrência de diversas empresas como Convergente, Lotus, Microsoft, Word Perfect e Word Star, que trataram de iniciar a produção de ferramentas para editar documentos eletrônicos, competindo pela simpatia das suas clientelas oferecendo recursos cada vez mais avançados.

De forma a acomodar os recursos que cada uma desenvolvia, as empresas criaram não apenas programas mas, principalmente, formatos de arquivos. Formatos fechados que são protegidos por licenças de uso que acabaram com uma das principais características dos documentos como os conhecemos: a liberdade para cópia-los.

Com a criação de formatos fechados para a gravação de documentos eletrônicos, não basta ter autorização ou requisitar a cópia. Quando um documento é armazenado em um formato fechado, é necessário, também, o pagamento de uma licença de uso de um software capaz de abrir o arquivo e exibir o seu conteúdo.

Quando um ente público disponibiliza documentos eletrônicos em formatos fechados, está instituindo a criação de um tributo privado. Como cidadãos, temos que ter acesso a documentos públicos sem nos preocuparmos se temos ou não um programa específico.

[1] Monges que, na Idade Média, se dedicavam a cópia de livros e documentos que nesta época eram escritos à mão.

Temos que ter acesso a documentos públicos de forma irrestrita usando a ferramenta que for mais apropriada para as nossas necessidades (DA. . . , 2008).

Nas próximas seções serão introduzidos formatos eletrônicos que foram estudados como possíveis saídas do conversor *lattes2latex*, assim como a motivação na escolha de LaTeX.

4.1 RTF

Desenvolvido pela Microsoft em 1987, o RTF, acrônimo para *Rich Text Format*, é um padrão de documento eletrônico que visava originalmente a integração multiplataforma de seus conteúdos. Sua especificação provê um formato para texto e gráficos que pode ser utilizado com diferentes dispositivos de saída, ambientes de operação e sistemas operacionais (MICROSOFT, 2008d).

O primeiro editor para RTF, ainda em sua versão 1.0, foi disponibilizado em 1987, como parte do Microsoft Word 3.0[2]. Todos os subsequentes lançamentos do editor Microsoft Word possuem uma ferramenta de leitura e escrita do formato RTF que permitia realizar a tradução entre esta especificação e o formato padrão do editor, o DOC.

Por ser um formato proprietário, apesar de ter sua especificação divulgada e livre para implementação, ainda pertence a Microsoft e encontra-se em sua versão 1.9.1.

- RTF 1.0.0 - 1987.
- RTF 1.3.0 - 1995.
- RTF 1.5.0 - 1997.
- RTF 1.6.0 - 1999.
- RTF 1.7.0 - 2001.
- RTF 1.8.0 - 2004.
- RTF 1.9.1 - 2008.

Internamente um arquivo RTF é formado por texto não formatado, palavras de controle, símbolos de controle e grupos. Para exemplificar a formatação da especificação vamos utilizar o arquivo da figura 4.1, gerado a partir do WordPad versão 5.1.

Apesar da aparente simplicidade do arquivo, constituído apenas de uma lista não ordenada, a representação interna do mesmo - listagem 4.1 - possui certo grau de complexidade.

```
1 {\rtf1\ansi\ansicpg1252\deff0\deflang1046\deflangfe1046
2   {\fonttbl
3     {\f0\fswiss\fcharset0 Arial;}
4     {\f1\fnil\fcharset2 Symbol;}
5   }
6   {\pntext\f1\'B7\tab}Item 1\par
```

[2]Processador de texto produzido pela Microsoft

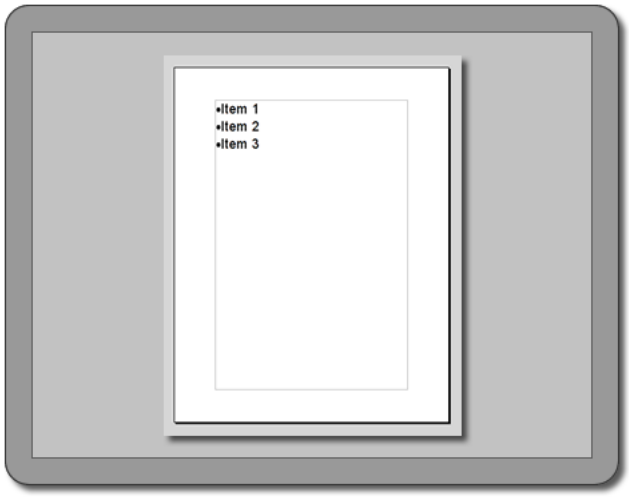


Figura 4.1: Arquivo RTF

```
7    {\pntext\f1\'B7\tab}Item 2\par
8    {\pntext\f1\'B7\tab}Item 3\par
9  }
```

Listagem 4.1: Representação Interna de um Arquivo RTF

Notamos a subdivisão do arquivo em um cabeçalho - linhas 1 a 5, formado neste caso por parâmetros de versão do RTF e de codificação, além da tabela de fontes presentes no arquivo, e corpo - linhas 6 a 8, que contém a especificação da lista não ordenada e seus itens.

4.2 DOC

DOC ou .doc, uma abreviação para *document*, é uma extensão de arquivos utilizada historicamente por diversos processadores de texto. Originalmente, representava documentos criados em texto plano por um grande leque de sistemas operacionais até que, em meados da década de 80, o editor WordPerfect[3] utilizou-a como extensão para seu formato proprietário. Logo a seguir, no início dos anos 90, a Microsoft, ao lançar seu editor de texto, o Microsoft Word, decidiu utilizar a extensão DOC como padrão para seu novo formato de arquivo binário. Foi então que a sigla passou a representar o formato de arquivos do Microsoft Word, afastando-se aos poucos do contexto original (MICROSOFT, 2008a).

O formato DOC possui certa variedade de especificações presentes entre as diversas versões do Microsoft Word. A primeira especificação do formato foi adotada até a versão 97 deste editor. Com o lançamento do Microsoft Word 97 uma segunda especificação foi criada, permanecendo como formato padrão até a chegada do Microsoft Office 2007[4] que trazia um novo formato, o OpenXML.

Apesar de nativos para o Microsoft Word, os arquivos no formato DOC podem ser ge-

[3]Processador de texto comercializado pela Corel como parte do pacote Corel Perfect.
[4]Suíte de aplicativos para escritório produzida pela Microsoft.

rados e lidos por outros editores como OpenOffice.org[5] e AbiWord[6]. Parte desta portabilidade deve-se ao fato de que, em fevereiro de 2008, a Microsoft publicou a especificação do formato DOC como parte da MOSP - *Microsoft Open Specification Promise*[7].

Um arquivo DOC é formado por:

- Fluxo principal.
- Fluxo de informação resumida.
- Fluxo de tabela.
- Fluxo de dados.
- Fluxos de objetos OLE 2.0.

As informações do texto estão contidas no fluxo principal, onde temos um cabeçalho, o corpo do texto e as informações de formatação do mesmo.

4.3 ODF x OpenXML

Com a chegada das novas gerações de suítes de escritório e a crescente diversificação de plataformas e sistemas operacionais existentes, percebeu-se a necessidade do aumento da portabilidade e integração nos diversos formatos de arquivos gerados e manipulados pelos editores de texto.

Com base nesta iniciativa o consórcio OASIS[8], acrônimo para *Organization for the Advancement of Structured Information Standards* (OASIS, 2008a), formado por diversas empresas tecnológicas como IBM, Microsoft, ORACLE, Sun, entre outras, publicou *OpenDocument Format* ou simplesmente ODF, forma abreviada para *OASIS Open Document Format for Office Applications* (OASIS, 2008b). Utilizado para armazenamento e troca de documentos de escritório, como textos, planilhas, bases de dados, gráficos e apresentações, o ODF foi totalmente baseado na linguagem XML, sendo formado exclusivamente por arquivos de texto plano, o que facilita a interpretação e o desenvolvimento de ferramentas para a manipulação destes arquivos. Tais ferramentas podem facilmente ser criadas pois, por ser um formato aberto, o ODF possui sua especificação disponível e sua implementação é livre. Em maio de 2006 a especificação do ODF foi aprovada como norma ISO de documentos eletrônicos (ISO, 2008).

Percebendo a ameaça existente contra sua especificação de documentos binários - o DOC - a Microsoft lançou, em sua suíte de aplicativos Microsoft Office 2007, um novo formato de documento eletrônico, o Office Open XML (MICROSOFT, 2008c). Também referenciado como OOXML, OpenXML ou Open XML, é um formato para armazenamento de documentos, planilhas, gráficos e apresentações muito semelhante ao OpenDocument Format. Um arquivo OpenXML contém, basicamente, arquivos XML formados

[5]Suíte de aplicativos para escritório livres.

[6]Processador de texto de código aberto.

[7]Política adotada pela Microsoft em setembro de 2006 na qual se compromete a não exercer direitos legais sobre algumas patentes ou implementações de suas tecnologias.

[8]Consórcio global que conduz o desenvolvimento, convergência e adoção de padrões para *e-bussiness* e *web services*.

por texto plano comprimidos em um pacote ZIP[9]. Criada originalmente para suceder o antigo formato binário da Microsoft, a especificação tornou-se um padrão *Ecma International*[10] em 2006. Em novembro de 2008, após uma longa batalha corporativa, além da incorporação de mudanças propostas, uma nova versão da especificação original foi publicada como padrão internacional ISO de documentos eletrônicos.

Para exemplificar a semelhança entre os formatos, existente graças à utilizado de XML por ambos, vamos analisar o arquivo da figura 4.1.

Em comparação ao formato RTF, vemos um aumento significativo da quantidade de texto necessária para reproduzir o arquivo, ao mesmo tempo em que notamos uma maior facilidade ao interpretar os dados contidos no mesmo - listagens 4.2 e 4.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document
  ...>
  <w:body>
    <w:p w:rsidR="00FF7463"
      w:rsidRDefault="005A4442" w:rsidP="005A4442">
      <w:pPr>
        <w:pStyle w:val="PargrafodaLista"/>
        <w:numPr>
          <w:ilvl w:val="0"/>
          <w:numId w:val="1"/>
        </w:numPr>
      </w:pPr>
      <w:r><w:t xml:space="preserve">Item 1</w:t></w:r>
    </w:p>
    <w:p w:rsidR="005A4442"
      w:rsidRDefault="005A4442" w:rsidP="005A4442">
      <w:pPr>
        <w:pStyle w:val="PargrafodaLista"/>
        <w:numPr>
          <w:ilvl w:val="0"/>
          <w:numId w:val="1"/>
        </w:numPr>
      </w:pPr>
      <w:r><w:t xml:space="preserve">Item 2</w:t></w:r>
    </w:p>
    <w:p w:rsidR="005A4442"
      w:rsidRDefault="005A4442" w:rsidP="005A4442">
      <w:pPr>
        <w:pStyle w:val="PargrafodaLista"/>
        <w:numPr>
          <w:ilvl w:val="0"/>
          <w:numId w:val="1"/>
        </w:numPr>
      </w:pPr>
      <w:r><w:t xml:space="preserve">Item 3</w:t></w:r>
    </w:p>
  </w:body>
</w:document>
```

Listagem 4.2: Representação Interna de um Arquivo OpenXML

[9] Formato de compactação de arquivos.
[10] Associação dedicada à padronização de sistemas de informação.

```
<?xml version="1.0" encoding="UTF-8">
<office:document-content ... office:version="1.2">
  <office:font-face-decls>...</office:font-face-decls>
  <office:automatic-styles>...</office:automatic-styles>
  <office:body>
    <office:text>
      <text:sequence-decls>
      ...
      </text:sequence-decls>
      <text:list xml:id="list41534633" text:style-name="L1">
        <text:list-item>
          <text:p text:style-name="P1">Item 1</text:p>
        </text:list-item>
        <text:list-item>
          <text:p text:style-name="P1">Item 2</text:p>
        </text:list-item>
        <text:list-item>
          <text:p text:style-name="P1">Item 3</text:p>
        </text:list-item>
      </text:list>
    </office:text>
  </office:body>
</office:document-content>
```

Listagem 4.3: Representação Interna de um Arquivo ODF

4.4 PostScript x PDF

PostScript é uma linguagem de descrição de páginas originalmente criada para impressão, em 1982, por John Warnock e Charles Geschke. A linguagem oferece uma máquina de pilha e comandos específicos para o desenho de letras e figuras, incluindo comandos de traçado e formas de representação de imagens (ADOBE, 2008a).

Possuindo uma grande variedade de operadores gráficos que permitem descrever precisamente a página desejada, PostScript torna-se uma alternativa no desenvolvimento de páginas com formatações específicas. Estes operadores controlam a localização de três tipos de objetos gráficos:

- Texto, em uma larga variedade de fontes, pode ser colocado em uma página em qualquer posição, orientação e escala.
- Figuras geométricas podem ser construídas através da descrição de linhas e curvas de quaisquer tamanhos, orientações e espessuras.
- Imagens digitalizadas são exibidas em qualquer posição, escala e orientação.

Em 1993, também surgida dos laboratórios da Adobe System, foi criada a especificação do formato de arquivos PDF, um acrônimo para *Portable Document Format*. Destinado a representação de documentos independente do aplicativo utilizado, do *hardware* ou do sistema operacional, o PDF tornou-se rapidamente uma alternativa comumente utilizada na disponibilização de documentos através da *Internet* (ADOBE, 2008b).

Um arquivo PDF pode descrever documentos que contenham texto, gráficos e imagens num formato independente de dispositivo e resolução. Sua especificação é aberta e, portanto, qualquer pessoa pode desenvolver aplicativos para a criação e manipulação de arquivos neste padrão. Hoje existem aplicativos gratuitos que trabalham com PDF para Linux, Microsoft Windows e MacOS, alguns deles distribuídos pela própria Adobe Systems. Em 2008 a ISO reconheceu o PDF como padrão internacional de formato de arquivo.

Como ambas as especificações são destinadas a descrição e representação de páginas de documentos prontos para impressão, não possuem estruturas para controle de formatação de texto, ao contrário dos formatos explanados anteriormente. Com base nisso, uma análise do formato de representação interna de listas nestes padrões não teria utilidade prática.

4.5 LaTeX

LaTeX é um conjunto de macros definidas para o processador de texto TeX, utilizado amplamente para a produção de textos matemáticos e científicos, graças a sua alta qualidade tipográfica. Entretanto, devido a sua sintaxe relativamente simples, também pode ser utilizado para a produção de cartas pessoais, artigos, currículos e livros sobre assuntos diversos (LATEX. . . , 2008).

Como um conjunto de macros para o TeX, o sistema LaTeX fornece ao usuário comandos de alto nível sendo, dessa forma, mais fácil a sua utilização por pessoas sem um profundo conhecimento da especificação. Possui abstrações para lidar automaticamente com bibliografias (`\bibliography`), citações (`\cite`), formatos de páginas (`\documentclass`), referências cruzadas (`\ref`) e tudo mais que não seja diretamente relacionado ao conteúdo do documento.

Desenvolvido na década de 80 por Leslie Lamport, encontra-se atualmente na versão denominada LaTeX2e ou LaTeX *2nd edition*.

A definição de LaTeX tem por base distanciar o autor o máximo possível da apresentação visual, pois a constante preocupação com a formatação desvia o pensamento do autor do conteúdo do documento. Ao invés de trabalhar com idéias visuais, o usuário é encorajado ao trabalho com conceitos mais lógicos e, consequentemente, mais independentes de apresentação, como capítulos (`\chapter`), seções (`\section`) e ênfases (`\emph`) sem, contudo, privar o usuário da liberdade de indicar, expressamente, declarações de formatação (LAMPORT, 1986).

Examinando o arquivo LaTeX da listagem 4.4, podemos perceber que a sintaxe deste alia a síntese da representação RTF com a facilidade de interpretação existente em formatos mais extensos, como o OpenXML e ODF. Isto se deve ao grande poder de representação que encontramos em LaTeX.

```
1 \begin{document}
2   \flushleft
3   \begin{itemize}
4     \item Item 1
5     \item Item 2
6     \item Item 3
```

```
7   \end{itemize}
8 \end{document}
```

Listagem 4.4: Representação LaTeX: Simples e Poderosa

4.6 LaTeX... Por quê?

Ao analisar as opções de formatos de documentos eletrônicos disponíveis para representação de um Currículo Lattes, podemos claramente dividir as especificações em três grupos: os formatos utilizados por editores de textos, onde temos RTF, DOC, OpenXML e ODF; os formatos gerados a partir de outros primários, compostos pelo PostScript e o PDF; e o formato utilizado na representação de um processador de texto, neste caso o LaTeX.

De acordo com a finalidade e as necessidades do projeto percebemos facilmente que as especificações pertencentes ao primeiro grupo são voltadas à edição e manipulação de informações, enquanto as especificações presentes no segundo grupo não permitem que se faça tais edições ou manipulações, gerando um documento fiel à descrição inicial. Enquanto isso, o LaTeX alia a fidelidade na representação com a flexibilidade da edição do conteúdo, além de, por possuir uma sintaxe clara e definida, permitir uma rápida e fácil geração de código sem abusar dos recursos do sistema. Estas vantagens permitem que se automatize a geração do currículo mantendo a capacidade de edição do mesmo sem que haja a necessidade da utilização de suítes de escritório ao término do processo.

Ao longo deste capítulo foram detalhados os formatos eletrônicos estudados e as características dos mesmo que levaram a escolha do LaTeX como formato de saída do conversor - figura 4.2. A seguir será exposto o processo de desenvolvimento do conversor *lattes2latex*, realizado através da criação de um *Parser* de DTD e de um *Parser* de XML para analisar o conteúdo do Currículo Lattes e gerar a saída esperada.

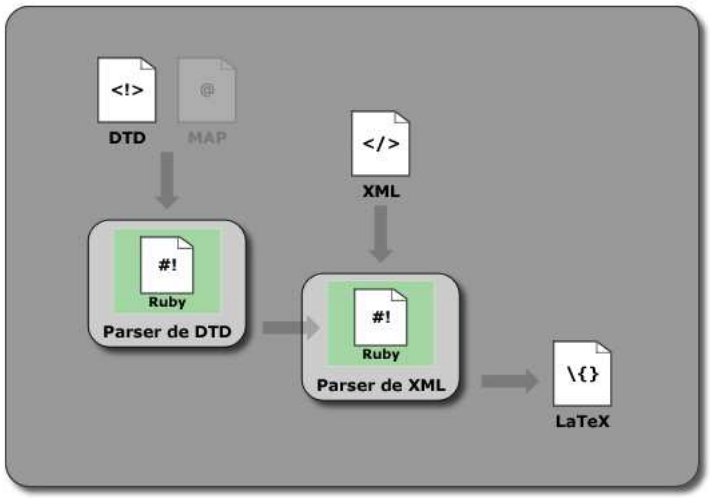


Figura 4.2: Processo de Conversão do Currículo Lattes em Código LaTeX

5 LATTES2LATEX

Lattes2latex é a ferramenta de tradução, desenvolvida ao longo deste Trabalho de Conclusão de Curso, que objetiva permitir a pesquisadores que esses possam reaproveitar as informações presentes em seu currículo na Plataforma Lattes através da conversão do conteúdo deste, disponibilizado em formato XML, em código LaTeX pronto para compilação.

Ao longo deste capítulo será apresentada a definição do Currículo Lattes através de sua especificação DTD e do formalismo criado para determinar a semântica de exibição das informações em código LaTeX. Além disso será detalhado o processo de construção do *parser* de DTD que será responsável pela geração, através de metaprogramação em Ruby, do *parser* de XML específico para a definição do Currículo Lattes.

Na continuação é introduzido um estudo sobre a instância de Currículo Lattes através de seu código XML. A seguir, será analisada a aplicação do *parser* de XML, gerado anteriormente, na tradução do Currículo Lattes de formato XML para código LaTeX.

Devido à extensão de alguns trechos de código, foram necessárias algumas quebras de linha adicionais, indicadas por um caractere de *underscore* (_) ao término da linha. Além disso, trechos de código foram substituidos por ... quando não se faziam fundamentais para a compreensão do assunto abordado.

5.1 A Definição do Currículo Lattes

A estrutura do Currículo Lattes, disponível em formato XML, é definida através de uma especificação DTD. Esta especificação determina quais elementos e atributos podem estar presentes no código XML de um Currículo Lattes, bem como sua estruturação. Através desta definição é possível gerar de maneira automatizada, com o emprego de um *parser* de DTD, sistemas que interajam com instâncias deste Currículo Lattes.

5.1.1 DTD

Com o surgimento e aprimoramento da Plataforma Lattes durante a década de 90, se fez necessária a adoção de um padrão de currículos pelo CNPq em parceria com a Capes. Para isso foi criada a CONSCIENTIAS - Comunidade para Ontologias em Ciência, Tecnologia e Informações de Aperfeiçoamento de Nível Superior - que é uma extensão da Comunidade LMPL estabelecida no ano de 2000 como responsável pela criação e

manutenção das gramáticas que definem a Plataforma Lattes (CONSCIENTIAS, 2008a).

A definição de um padrão de currículos significa a possibilidade de intercâmbio de informações entre diversas bases institucionais e as bases do Sistema de Currículo Lattes.

O padrão XML para o Currículo Lattes foi a primeira unidade de informação definida para a Plataforma Lattes. Esse padrão, mantido pela Comunidade CONSCIENTIAS-LMPL, foi elaborado seguindo as informações e a estrutura delas representadas no Sistema de Currículos Lattes.

Através da definição feita pela Comunidade CONSCIENTIAS-LMPL, para a unidade de informação do *Curriculum Vitae*, o sistema de Currículos Lattes incorporou as funcionalidades de integração de informações em XML, como importação e exportação de dados, disponibilizando-as a partir da versão 1.4 deste sistema. Este padrão XML foi definido através da utilização do DTD.

A especificação completa contida no arquivo DTD do Currículo Lattes - listagem 5.1 - possui mais de duas mil e trezentas linhas de código. Devido a esta extensão vamos detalhá-lo apenas em um primeiro nível (CONSCIENTIAS, 2008b).

```
36 <!ELEMENT CURRICULO-VITAE (DADOS-GERAIS,
37     PRODUCAO-BIBLIOGRAFICA?,
38     PRODUCAO-TECNICA?,
39     OUTRA-PRODUCAO?,
40     DADOS-COMPLEMENTARES?)>
41 <!ATTLIST CURRICULO-VITAE
42         SISTEMA-ORIGEM-XML CDATA #REQUIRED
43         NUMERO-IDENTIFICADOR CDATA  #IMPLIED
44         FORMATO-DATA-ATUALIZACAO NMTOKEN #FIXED "DDMMAAAA"
45         DATA-ATUALIZACAO CDATA  #IMPLIED
46         FORMATO-HORA-ATUALIZACAO NMTOKEN #FIXED "HHMMSS"
47         HORA-ATUALIZACAO CDATA  #IMPLIED
48         xmlns:lattes CDATA  #IMPLIED
49 >
```

Listagem 5.1: Elemento Principal do Currículo Lattes

Nas linhas 36 até 40 está definido o principal elemento de um Currículo Lattes, `<!ELEMENT CURRICULO-VITAE (...)>`, que subdivide o conteúdo em cinco grandes tópicos:

1. Dados Gerais
2. Produção Bibliográfica
3. Produção Técnica
4. Outra Produção
5. Dados Complementares

Entre as linhas 41 e 49 estão definitos os atributos pertencentes ao elemento principal do currículo. Pelo nível genérico no qual se apresentam na estrutura podem ser considerados atributos de controle do currículo.

O elemento DADOS-GERAIS - listagem 5.2 - representa informações sobre o currículo, como resumo, endereço do pesquisador, sua formação, atuações profissionais, áreas de atuação, idiomas e prêmios ou títulos recebidos.

```
50 <!ELEMENT DADOS-GERAIS (RESUMO-CV?,
51     OUTRAS-INFORMACOES-RELEVANTES?,
52     ENDERECO?,
53     FORMACAO-ACADEMICA-TITULACAO?,
54     ATUACOES-PROFISSIONAIS?,
55     AREAS-DE-ATUACAO?,
56     IDIOMAS?,
57     PREMIOS-TITULOS?)>
```

Listagem 5.2: Elemento DADOS-GERAIS do Currículo Lattes

Já o elemento PRODUCAO-BIBLIOGRAFICA - listagem 5.3 - apresenta informações sobre trabalhos em eventos, artigos, livros, jornais e revistas publicados ou com participação do pesquisador.

```
650 <!ELEMENT PRODUCAO-BIBLIOGRAFICA (TRABALHOS-EM-EVENTOS?,
651     ARTIGOS-PUBLICADOS?,
652     LIVROS-E-CAPITULOS?,
653     TEXTOS-EM-JORNAIS-OU-REVISTAS?,
654     DEMAIS-TIPOS-DE-PRODUCAO-BIBLIOGRAFICA?,
655     ARTIGOS-ACEITOS-PARA-PUBLICACAO?)>
```

Listagem 5.3: Elemento PRODUCAO-BIBLIOGRAFICA do Currículo Lattes

O próximo elemento da estrutura do currículo, a PRODUCAO-TECNICA - listagem 5.4 - é responsável pela representação das informações sobre os desenvolvimentos tecnológicos do pesquisador.

```
932 <!ELEMENT PRODUCAO-TECNICA (SOFTWARE*,
933     PRODUTO-TECNOLOGICO*,
934     PROCESSOS-OU-TECNICAS*,
935     TRABALHO-TECNICO*,
936     DEMAIS-TIPOS-DE-PRODUCAO-TECNICA*)>
```

Listagem 5.4: Elemento PRODUCAO-TECNICA do Currículo Lattes

O penúltimo elemento pertencente ao currículo é o OUTRA-PRODUCAO - listagem 5.5 - que apresenta os dados sobre produções que não são abrangidas pelos elementos anteriores.

```
1298 <!ELEMENT OUTRA-PRODUCAO (PRODUCAO-ARTISTICA-CULTURAL*,
1299     ORIENTACOES-CONCLUIDAS*,
1300     DEMAIS-TRABALHOS*)>
```

Listagem 5.5: Elemento OUTRA-PRODUCAO do Currículo Lattes

Por último temos o elemento DADOS-COMPLEMENTARES - listagem 5.6 - com informações adicionais sobre formação, participação em bancas, eventos, orientações, cursos e instituições.

```
1647 <!ELEMENT DADOS-COMPLEMENTARES (FORMACAO-COMPLEMENTAR*,
1648     PARTICIPACAO-EM-BANCA-TRABALHOS-CONCLUSAO?,
1649     PARTICIPACAO-EM-BANCA-JULGADORA?,
```

```
1650      PARTICIPACAO-EM-EVENTOS-CONGRESSOS?,
1651      ORIENTACOES-EM-ANDAMENTO?,
1652      INFORMACOES-ADICIONAIS-INSTITUICOES?,
1653      INFORMACOES-ADICIONAIS-CURSOS?)>
```

Listagem 5.6: Elemento DADOS-COMPLEMENTARES do Currículo Lattes

De acordo com sua definição, o DTD especifica apenas a estrutura válida de uma linguagem XML. Não existe nenhuma informação sobre a semântica de cada elemento. Para gerar o código LaTeX de um determinado Currículo Lattes é preciso especificar a formatação de exibição a nível de elemento (W3C, 2008c).

Para resolver este problema foi definido um formalismo que acrescenta ao *parser* de DTD informações sobre a formatação de cada elemento: o Mapeamento de Elementos.

5.1.2 Mapeamento de Elementos

O arquivo de mapeamento foi definido para permitir que cada elemento presente no DTD possa ter um comportamento singular no momento da execução do *parser* de XML a ser gerado. Para isto, o arquivo de mapeamento associa métodos aos elementos presentes na definição da linguagem, através da utilização de uma variável *hash* - listagem 5.7. Durante o processo de *parsing* do DTD estes métodos são anexados ao corpo das classes definidas no código-fonte do *parser* de XML.

Devido a extensão do arquivo de mapeamento iremos analisá-lo apenas no que diz respeito aos elementos de primeiro nível da definição.

```
1  @map['CURRICULO-VITAE'] = <<EOF
2    def parse
3      puts '\\documentclass[a4paper]{article}'
4      puts '\\usepackage[brazil]{babel}'
5      puts '\\usepackage[utf8]{inputenc}'
6      puts
7      puts '\\begin{document}'
8      @elementos['DADOS-GERAIS'].toTEX
9      puts '\\newpage'
10     @elementos['PRODUCAO-BIBLIOGRAFICA'].toTEX
11     puts '\\newpage'
12     @elementos['DADOS-COMPLEMENTARES'].toTEX
13     puts '\\end{document}'
14   end
15 EOF

17 @map['DADOS-GERAIS'] = <<EOF
18   def toTEX
19     puts '\\section*{Curriculum Vitae}'
20     puts @atributos['NOME-COMPLETO']
21     puts @atributos['CIDADE-NASCIMENTO'] + ', '
22     puts @atributos['PAIS-DE-NASCIMENTO']
23     puts
24     @elementos['RESUMO-CV'].toTEX
25     @elementos['IDIOMAS'].toTEX
26     @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
27   end
28 EOF
```

```
30 @map['PRODUCAO-BIBLIOGRAFICA'] = <<EOF
31   def toTEX
32     puts '\\subsection*{Producao Bibliografica}'
33     @elementos['TRABALHOS-EM-EVENTOS'].toTEX
34     @elementos['ARTIGOS-PUBLICADOS'].toTEX
35     @elementos['LIVROS-E-CAPITULOS'].toTEX
36     @elementos['TEXTOS-EM-JORNAIS-OU-REVISTAS'].toTEX
37     @elementos['ARTIGOS-ACEITOS-PARA-PUBLICACAO'].toTEX
38   end
39 EOF

41 @map['DADOS-COMPLEMENTARES'] = <<EOF
42   def toTEX
43   end
44 EOF

46 ...
```

Listagem 5.7: Mapeamento de Métodos para o Currículo Lattes

O primeiro elemento especificado é o elemento raiz da definição DTD do Currículo Lattes - `CURRICULO-VITAE`. Nele definimos o método `parse` que é o responsável pelo início do processo de *parsing* do arquivo XML. Nesta implementação, o método `parse` cria a estrutura do código LaTeX, com a declaração do documento e a importação de pacotes, e invoca o método `toTEX` das classes `DADOS-GERAIS`, `PRODUCAO-BIBLIOGRAFICA` e `DADOS-COMPLEMENTARES`. O método `toTEX` de cada classe, também definidos no arquivo de mapeamento, é o responsável pela especificação do código LaTeX a ser gerado por cada elemento, utilizando as informações presentes nos atributos do mesmo e invocando os métodos `toTEX` de seus elementos descendentes quando necessário.

Com esta definição de métodos é possível especificar quais atributos de um determinado elemento estarão presentes no código LaTeX gerado pelo *parser* de XML, além da sua formatação, bastando invocar estes métodos durante o processo de *parsing* da XML.

5.2 *Parser* de DTD

Como visto no capítulo 2, é possível criar um *parser* de XML, específico para uma linguagem definida em um DTD, através do processo de *parsing* deste DTD. Neste trabalho foi desenvolvido um *parser* de DTD, em linguagem Ruby, para automatizar a geração do *parser* de XML do Currículo Lattes. O fato deste *parser* de DTD aceitar qualquer definição DTD e associar aos elementos desta os métodos presentes no arquivo de mapeamento permite que se utilize este *parser* de DTD para qualquer especificação DTD, gerando *parsers* de XML para outras especificações e propósitos.

A estrutura do *parser* de DTD é apresentada na figura 5.1, onde:

DTD - classe que armazena toda a informação do DTD. É a responsável por inicializar os processos de *parsing* do DTD e de geração do *parser* de XML.

Element - classe que mantém informações sobre cada elemento especificado no DTD. Armazena o nome do elemento, seus elementos e atributos.

Attribute - classe que mantém informações sobre cada atributo especificado no DTD. Mantém o nome do atributo, seu tipo e seu valor padrão.

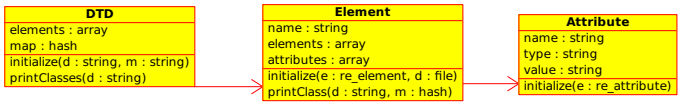


Figura 5.1: Estrutura do *Parser* de DTD

O processo realizado pelo *parser* de DTD é dividido em três fases, detalhadas a seguir:

- Análise do DTD
- Análise do Mapeamento de Elementos
- Geração do *Parser* de XML

5.2.1 Análise do DTD

Ao instanciar um objeto `DTD` é disparado, pelo método construtor deste - `initialize`, o processo de análise do DTD de entrada. Como o DTD é definido através de uma linguagem formal, utilizando texto puro, foi empregado o uso de expressões regulares para o reconhecimento das informações contidas no arquivo.

O estudo da especificação da linguagem do DTD levou à definição de três expressões - listagem 5.8 - que especificam o formato das declarações de elementos e atributos do arquivo, além de uma lista de atributos.

```
1  # [1] = Nome do Elemento [2] = Elementos do Elemento
2  RE_ELEMENT        = _
3    /<!ELEMENT\ (.*)\ (EMPTY|ANY|\(.*\))>/

5  # [1] = Nome do Elemento [2] = Lista de Atributos do Elemento
6  RE_ATTRIBUTE      = _
7    /<!ATTLIST\s*(\S*)\s*([^>]*)>/

9  # [1] = Nome do Atributo [2] = Tipo [3] = Valor
10 RE_ATTRIBUTE_LIST = _
11   /\s*(\S*)\s*(\([^\)]*\)|\S*)\s*(#FIXED\s*\S*|\S*)/
```

Listagem 5.8: Expressões Regulares Utilizadas no *Parser* de DTD

A primeira expressão regular - linha 3 - identifica no corpo do arquivo as declarações do tipo `ELEMENT`. Com sua apliação sobre o DTD é possível obter, para cada resultado retornado, o nome do elemento e seus elementos descendentes. Estas informações são utilizadas na criação de objetos `Element`, posteriormente armazenados em um vetor de elementos - listagem 5.9 - pertencente ao objeto `DTD`.

```
1  @elements = Array.new
2  d.scan(RE_ELEMENT) { | m | @elements.push(Element.new(m, d)) }
```

Listagem 5.9: Busca de Elementos

A segunda expressão regular - linha 7 - é empregada na busca de declarações do tipo `ATTLIST`. A aplicação desta expressão, que retorna o nome do elemento e sua lista de atributos, ocorre a cada objeto `Element` instanciado no processo anterior. Sobre esta lista de atributos retornada é aplicada a terceira expressão regular - linha 11 - que retorna o nome de cada atributo pertencente a lista, seu tipo e valor padrão. Para cada retorno deste é instanciado, e incluído em uma lista de atributos pertencente ao objeto `Element` correspondente, um objeto do tipo `Attribute` - listagem 5.10.

```
1 @attributes = Array.new
2 d.scan(RE_ATTRIBUTE) { | m |
3   m[1].scan(RE_ATTRIBUTE_LIST) { | n |
4     @attributes.push(Attribute.new(n)) unless n[0] == ''
5   } if m[0] == @name
6 }
```

Listagem 5.10: Busca de Atributos

Ao término do processamento das expressões regulares temos a estrutura do arquivo DTD totalmente armazenada no objeto `DTD`. É possível, neste momento, gerar uma descrição de classes que represente o DTD de entrada para ser utilizada no *parser* de XML. Porém, as informações armazenadas não possuem nenhum detalhe específico do comportamento dos elementos. Em uma situação onde todos os elementos existentes são manipulados ou exibidos da mesma forma isto não seria problema. No entando, o enfoque deste *parser* de DTD é permitir que se possa determinar o comportamento de cada elemento de maneira específica.

5.2.2 Análise do Mapeamento de Elementos

A definição da semântica de exibição dos elementos analisados na seção anterior é realizada através da análise do arquivo de mapeamento definido para o DTD. Através da leitura da variável *hash* `@map`, presente no arquivo, é possível determinar quais métodos serão anexados à quais elementos da especificação.

Com os elementos da definição DTD armazenados no objeto `DTD` e os métodos a serem adicionados às classes armazenados na variável `@map` é possível gerar o código-fonte do *parser* de XML que será o responsável por carregar os dados do documento XML e executar sobre estes os métodos definidos através do mapeamento.

5.2.3 Geração do *Parser* de XML

O objeto `DTD` instanciado no processo de análise do DTD possui, em sua definição, um método denominado `printClasses` - listagem 5.11. Através deste método é gerado o código-fonte do *parser* de XML especificado pela estrutura analisada. Este *parser* gerado é composto dos seguintes elementos:

Cabeçalho - sequência de comandos que realiza a importação da biblioteca REXML - linhas 5 e 6.

Classes - definições das classes geradas a partir da análise do DTD, com os respectivos métodos mapeados - linha 10.

Disparador - sequência de comandos que carregam o arquivo XML e disparam o processo de *parsing* do mesmo - linhas 18 a 21.

```
1  def printClasses( d )

3    d = File.new(d, "w")

5    d.puts "require 'rexml/document'"
6    d.puts "include REXML           "
7    d.puts

9    @elements.each do
10     | e | e.printClass(d, @map[e.name])
11     d.puts
12     d.puts
13   end

15   d.puts
16   d.puts

18   d.puts "f       = Document.new(File.new(ARGV[0]))                  "
19   d.puts "oLoader = _
20 #{className(elements[0].name)}.new(f.elements['#{elements[0].name}'])"
21   d.puts "oLoader.parse                                              "

23   d.close

25 end
```

Listagem 5.11: Método de Geração de Classes no Objeto `DTD`

O método `printClass` - listagem 5.12, do objeto `Element`, é o responsável pela geração de cada definição de classe no *parser* de XML. Esta definição é composta pelos seguintes elementos:

Construtor - linha 5 - método responsável pela inicialização do objeto, é definido através da iteração do conteúdo dos vetores de elementos e atributos que gera comandos de carga do arquivo XML, através do uso da biblioteca REXML. De acordo com a cardinalidade de cada elemento presente no vetor `@elements`, define se este vai ser referenciado por uma variável *hash* ou por um *array* de elementos.

Métodos Mapeados - linha 31 - a variável `m` possuí o código-fonte dos métodos definidos no mapeamento que são simplesmente impressos na definição da classe.

```
1  def printClass( d, m )

3    d.puts       "class #{className(@name)}                          "
4    d.puts
5    d.puts       "  def initialize( d )                              "
6    d.puts
7    d.puts       "   return if d == nil                              "
8    d.puts
9    d.puts       "   @atributos = Hash.new                           "
10   @attributes.each do | a |
11     d.puts     "   @atributos['#{a.name}'] = d.attributes['#{a.name}']"
```

```
12    end
13    d.puts
14    if @elements.instance_of?(Hash)
15      d.puts      "   @elementos = Hash.new                                  "
16      @elements.each do | c, v |
17        if v == '' or v == '?'
18          d.puts " _
19  @elementos['#{c}'] = #{className(c)}.new(d.elements['#{c}'])"
20        else
21          d.puts "   @elementos['#{c}'] = Array.new                         "
22          d.puts "   d.elements.each('#{c}') do | i |                       "
23          d.puts "     @elementos['#{c}'].push(#{className(c)}.new(i))      "
24          d.puts "   end                                                    "
25        end
26      end
27    end
28    d.puts
29    d.puts        "  end                                                    "
30    d.puts
31    d.puts        m unless m == nil
32    d.puts
33    d.puts        "end                                                     "

35  end
```

Listagem 5.12: Método de Impressão de Classe no Objeto `Element`

Ao término da execução do método `printClasses` do objeto `DTD` têm-se o *parser* de XML criado em um arquivo indicado como parâmetro no início do processo.

Nesta seção foi detalhado o processo de *parsing* do DTD, através da análise do DTD e do arquivo de mapeamento, - figura 5.2, responsável pela criação do *Parser* de XML que será estudado nas próximas seções.

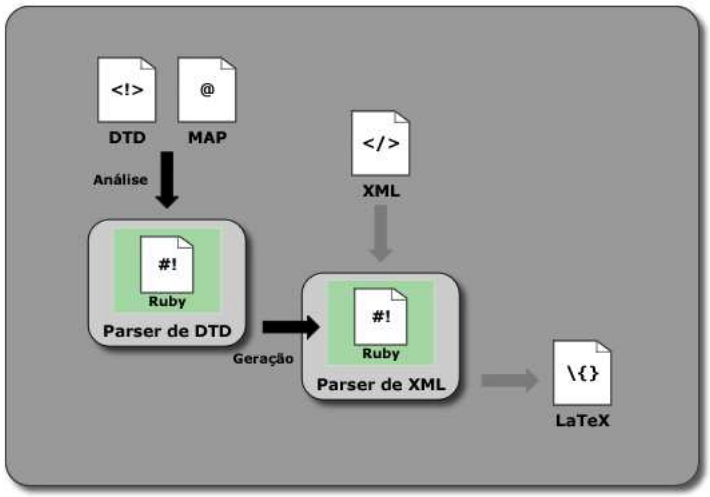


Figura 5.2: Processo de Conversão do Currículo Lattes em Código LaTeX

5.3 A Instância do Currículo Lattes

No escopo deste trabalho, uma instância de Currículo Lattes é formada por uma descrição deste em um arquivo XML. Neste arquivo os dados do pesquisador estão armazenados de maneira estruturada, respeitando a definição da linguagem existente no arquivo DTD que define o formato do Currículo Lattes. Com este arquivo estruturado e com o *parser* de XML gerado através do processo anterior, é possível traduzir o conteúdo do Currículo Lattes do pesquisador em código LaTeX, de acordo com a formatação definida no mapeamento dos elementos.

5.3.1 XML

O detalhamento do arquivo XML do Currículo Lattes a seguir - listagem 5.13 - foi realizado somente para alguns elementos do mesmo já que, devido a extensão e ao minimalismo da especificação, um arquivo completo possui milhares de linhas.

```
1  <CURRICULO-VITAE
2    SISTEMA-ORIGEM-XML="LATTES_OFFLINE"
3    DATA-ATUALIZACAO="24092007"
4    HORA-ATUALIZACAO="183716">
5    <DADOS-GERAIS
6      NOME-COMPLETO="Marcelo Alves Teixeira"
7      NOME-EM-CITACOES-BIBLIOGRAFICAS="TEIXEIRA, M. A."
8      NACIONALIDADE="B"
9      PAIS-DE-NASCIMENTO="Brasil"
10     UF-NASCIMENTO="RS"
11     CIDADE-NASCIMENTO="Osorio"
12     DATA-NASCIMENTO="14051986"
13     SEXO="MASCULINO"
14     NOME-DO-PAI="Rossano Ubirajara Debastiani Teixeira"
15     NOME-DA-MAE="Marcia Terezinha Alves Teixeira"
16     PERMISSAO-DE-DIVULGACAO="NAO">
17     <RESUMO-CV TEXTO-RESUMO-CV-RH="Marcelo Alves Teixeira e
18       estudande do curso de Ciencia de Computacao na UFRGS."/>

20     ...

22     <FORMACAO-ACADEMICA-TITULACAO>
23       <GRADUACAO
24         SEQUENCIA-FORMACAO="1"
25         NIVEL="1"
26         TITULO-DO-TRABALHO-DE-CONCLUSAO-DE-CURSO="lattes2latex
27           - Um Conversor de Curriculos Lattes em Codigo LaTeX."
28         NOME-DO-ORIENTADOR="Nicolas Maillard"
29         CODIGO-INSTITUICAO="000600000001"
30         NOME-INSTITUICAO=
31           "Universidade Federal do Rio Grande do Sul"
32         CODIGO-CURSO="90000000"
33         NOME-CURSO="Ciencia da Computacao"
34         CODIGO-AREA-CURSO="10300007"
35         STATUS-DO-CURSO="CONCLUIDO"
36         ANO-DE-INICIO="2005"
37         ANO-DE-CONCLUSAO="2009"
38         FLAG-BOLSA="NAO"
39         CODIGO-AGENCIA-FINANCIADORA=""
```

```
40          NOME-AGENCIA=""
41          NUMERO-ID-ORIENTADOR=""
42          CODIGO-CURSO-CAPES=""/>
43        </FORMACAO-ACADEMICA-TITULACAO>
44        <AREAS-DE-ATUACAO>
45          <AREA-DE-ATUACAO SEQUENCIA-AREA-DE-ATUACAO="1"
46            NOME-GRANDE-AREA-DO-CONHECIMENTO=
47              "CIENCIAS_EXATAS_E_DA_TERRA"
48            NOME-DA-AREA-DO-CONHECIMENTO="Ciencia da Computacao"
49            NOME-DA-SUB-AREA-DO-CONHECIMENTO=
50              "Metodologia e Tecnicas da Computacao"
51            NOME-DA-ESPECIALIDADE="Engenharia de Software"/>
52        </AREAS-DE-ATUACAO>
53        <IDIOMAS>
54          <IDIOMA IDIOMA="PT" DESCRICAO-DO-IDIOMA="Portugues"
55            PROFICIENCIA-DE-LEITURA="BEM"
56            PROFICIENCIA-DE-FALA="BEM"
57            PROFICIENCIA-DE-ESCRITA="BEM"
58            PROFICIENCIA-DE-COMPREENSAO="BEM"/>
59        </IDIOMAS>
60      </DADOS-GERAIS>

62      ...

64      <DADOS-COMPLEMENTARES>
65        <INFORMACOES-ADICIONAIS-CURSOS>
66          <INFORMACAO-ADICIONAL-CURSO
67            CODIGO-CURSO="90000000"
68            CODIGO-ORGAO=""
69            NOME-ORGAO=""
70            CODIGO-INSTITUICAO="000600000001"
71            NOME-INSTITUICAO=
72              "Universidade Federal do Rio Grande do Sul"
73            NOME-GRANDE-AREA-DO-CONHECIMENTO=
74              "CIENCIAS_EXATAS_E_DA_TERRA"
75            NOME-DA-AREA-DO-CONHECIMENTO="Ciencia da Computacao"
76            NOME-DA-SUB-AREA-DO-CONHECIMENTO=""
77            NOME-DA-ESPECIALIDADE=""
78            NIVEL-CURSO="GRADUACAO"/>
79        </INFORMACOES-ADICIONAIS-CURSOS>
80      </DADOS-COMPLEMENTARES>
81  </CURRICULO-VITAE>
```

Listagem 5.13: Instância do Currículo Lattes - Arquivo XML

Na linha 1 é declarado o elemento raiz do Currículo Lattes - `CURRICULO-VITAE`, do qual todos os demais elementos são descendentes. Este elemento possui alguns atributos que controlam a versão e data de geração do currículo, que não são importantes neste momento. A seguir, na linha 5, temos o elemento que define o primeiro tópico do Currículo Lattes: `DADOS-GERAIS`. Este elemento apresenta informações sobre o currículo como:

Resumo - através do elemento descendente `RESUMO-CV`.

Formação Acadêmica - através da presença do elemento descendente `FORMACAO-ACADEMICA-TITULACAO` e seus atributos. O elemento `FORMACAO-ACADEMICA-TITULACAO` possui zero ou mais elementos de formação, como `GRADUACAO`, `MESTRADO`, `DOUTORADO`, etc.

Áreas de Atuação - são listadas no elemento `AREAS-DE-ATUACAO` e detalhadas em seus descendentes `AREA-DE-ATUACAO`.

Idiomas - presente no elemento `IDIOMAS`, é especificada em cada `IDIOMA` descendente deste.

Analogamente, todos os tópicos detalhados no estudo da definição do Currículo Lattes são especificados, quando presentes no currículo, através da declaração dos elementos definidos no DTD de especificação.

5.4 *Parser* de XML

A criação do *parser* de XML do tradutor *lattes2latex* foi realizada através do emprego do *parser* de DTD descrito anteriormente. Para isto, já que se possuía o arquivo DTD de especificação do Currículo Lattes, foi necessário apenas definir o comportamento, neste caso a geração de código LaTeX, de cada elemento de interesse presente na definição do Currículo Lattes através da elaboração do mapeamento dos elementos.

A estrutura do *parser* de XML gerado, que possuí centenas de classes definidas, é apresentada parcialmente, devido à sua extensão, na figura 5.3, onde:

CurriculoVitae - classe que representa o elemento raiz da definição do DTD. É a responsável por disparar o processo de *parsing* do Currículo Lattes.

Demais Classes - todos os outros elementos especificados no DTD possuem uma classe que os define. O método construtor de cada classe é o responsável por realizar a análise dos dados na XML que pertencem ao elemento. Além disso, cada classe possui seus métodos especificos, definidos no arquivo de mapeamento, que serão disparados no decorrer do processo de *parsing*.

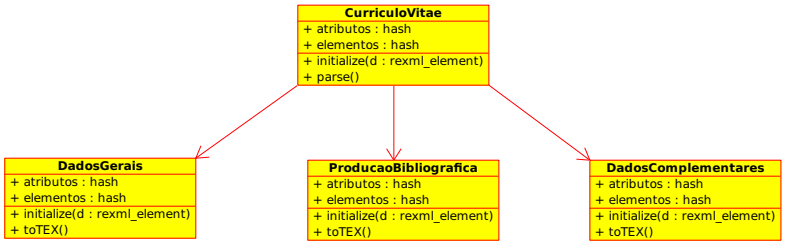


Figura 5.3: Estrutura Parcial do *Parser* de XML para o Currículo Lattes

O processo realizado pelo *parser* de XML do Currículo Lattes é disparado pela sequência de comandos da listagem 5.14 e dividido em duas fases, detalhadas a seguir:

- Análise do XML
- Geração do Código LaTeX

```
1 f       = Document.new(File.new(ARGV[0]))
2 oLoader = CurriculoVitae.new(f.elements['CURRICULO-VITAE'])
3 oLoader.parse
```

Listagem 5.14: Geração dos Objetos

5.4.1 Análise do XML

A carga do arquivo XML - linha 1 - é realizada através da criação de um objeto do tipo `Document`, da biblioteca REXML. Este objeto armazena a estrutura e os dados do arquivo XML de entrada na variável `f`, permitindo o acesso aos mesmos através do uso da linguagem XPath.

A análise das informações acontece na instanciação do objeto `CurriculoVitae` - linha 2, que representa o elemento raiz do Currículo Lattes - `CURRICULO-VITAE`. A criação deste objeto dispara seu método construtor - listagem 5.15, linha 4 - que carrega as informações de atributos do elemento existentes no arquivo XML utilizando o objeto `Document` da biblioteca REXML que contém as informações.

De maneira recursiva, cada elemento descendente do elemento atual é inicializado através da chamada do método construtor de sua classe, como ocorre com o elemento `DADOS-GERAIS` - linha 13. Ao término desta carga têm-se uma variável `oLoader` que representa o nó raiz do Currículo Lattes e apartir da qual pode-se acessar todos os elementos e atributos do currículo, assim como os métodos definidos nas classes que representam os mesmos.

```
1  require 'rexml/document'
2  include REXML
3  class CurriculoVitae
4    def initialize( d )
5      return if d == nil
6      @atributos = Hash.new
7      @atributos['SISTEMA-ORIGEM-XML'] = _
8        d.attributes['SISTEMA-ORIGEM-XML']
9      ...

11     @elementos = Hash.new
12     @elementos['DADOS-GERAIS'] = _
13       DadosGerais.new(d.elements['DADOS-GERAIS'])
14     @elementos['PRODUCAO-BIBLIOGRAFICA'] = _
15       ProducaoBibliografica.new(d.elements['PRODUCAO-BIBLIOGRAFICA'])
16     @elementos['PRODUCAO-TECNICA'] = _
17       ProducaoTecnica.new(d.elements['PRODUCAO-TECNICA'])
18     @elementos['OUTRA-PRODUCAO'] = _
19       OutraProducao.new(d.elements['OUTRA-PRODUCAO'])
20     @elementos['DADOS-COMPLEMENTARES'] = _
21       DadosComplementares.new(d.elements['DADOS-COMPLEMENTARES'])
22   end

24   ...

26 end

28 class DadosGerais
29   def initialize( d )
30     return if d == nil
31     @atributos = Hash.new
32     @atributos['NOME-COMPLETO'] = _
33       d.attributes['NOME-COMPLETO']
34     ...

36     @elementos = Hash.new
```

```
37      @elementos['ENDERECO'] = _
38        Endereco.new(d.elements['ENDERECO'])
39      @elementos['RESUMO-CV'] = _
40        ResumoCV.new(d.elements['RESUMO-CV'])
41      @elementos['IDIOMAS'] = _
42        Idiomas.new(d.elements['IDIOMAS'])
43      @elementos['ATUACOES-PROFISSIONAIS'] = _
44        AtuacoesProfissionais.new(d.elements['ATUACOES-PROFISSIONAIS'])
45      @elementos['AREAS-DE-ATUACAO'] = _
46        AreasAtuacao.new(d.elements['AREAS-DE-ATUACAO'])
47      @elementos['PREMIOS-TITULOS'] = _
48        PremiosTitulos.new(d.elements['PREMIOS-TITULOS'])
49      @elementos['FORMACAO-ACADEMICA-TITULACAO'] = _
50        FormacaoAcademicaTitulacao.new(_
51  d.elements['FORMACAO-ACADEMICA-TITULACAO'])
52      @elementos['OUTRAS-INFORMACOES-RELEVANTES'] = _
53        OutrasInformacoesRelevantes.new(_
54  d.elements['OUTRAS-INFORMACOES-RELEVANTES'])
55    end

57    ...

59  end
```

Listagem 5.15: *Parser* de XML Gerado para o Currículo Lattes

5.4.2 Geração do Código LaTeX

Com o término da carga do arquivo XML têm-se, além da estrutura do Currículo Lattes, as informações específicas da instância carregadas em memória. Neste momento, a geração do código LaTeX - listagem 5.16 - é disparada através do método `parse` definido para o elemento `CURRICULO-VITAE`. Este método é o responsável por gerar o esqueleto do código, criando a definição de classe do documento, importando os pacotes necessários e delimitando o corpo do documento. O preenchimento das informações do Currículo Lattes se dá através de chamadas para os métodos `toTEX` dos objetos responsáveis por estas informações, como é o caso de `DADOS-GERAIS`, `PRODUCAO-BIBLIOGRAFICA` e `DADOS-COMPLEMENTARES`.

```
1   class CurriculoVitae

3     ...

5     def parse
6       puts '\documentclass[a4paper]{article}'
7       puts '\usepackage[brazil]{babel}'
8       puts '\usepackage[utf8]{inputenc}'
9       puts
10      puts '\begin{document}'
11      @elementos['DADOS-GERAIS'].toTEX
12      puts '\newpage'
13      @elementos['PRODUCAO-BIBLIOGRAFICA'].toTEX
14      puts '\newpage'
15      @elementos['DADOS-COMPLEMENTARES'].toTEX
16      puts '\end{document}'
17    end
```

```
19 end

21 class DadosGerais

23   ...

25   def toTEX
26     puts '\\section*{Curriculum Vitae}'
27     puts @atributos['NOME-COMPLETO']
28     puts @atributos['CIDADE-NASCIMENTO'] + ', '
29     puts @atributos['PAIS-DE-NASCIMENTO']
30     puts
31     @elementos['RESUMO-CV'].toTEX
32     @elementos['IDIOMAS'].toTEX
33     @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
34   end

36 end

38 class ProducaoBibliografica

40   ...

42   def toTEX
43     puts '\\subsection*{Producao Bibliografica}'
44     @elementos['TRABALHOS-EM-EVENTOS'].toTEX
45     @elementos['ARTIGOS-PUBLICADOS'].toTEX
46     @elementos['LIVROS-E-CAPITULOS'].toTEX
47     @elementos['TEXTOS-EM-JORNAIS-OU-REVISTAS'].toTEX
48     @elementos['ARTIGOS-ACEITOS-PARA-PUBLICACAO'].toTEX
49   end

51 end

53 class DadosComplementares

55   ...

57   def toTEX
58   end

60 end
```

Listagem 5.16: Métodos de Geração de Código no *Parser* de XML

Cada método `toTEX` destes objetos inicializa uma nova seção no código LaTeX e imprime no mesmo seus atributos requisitados e a formatação especificada pelo programador.

Quando existe a necessidade de impressão de informações pertencentes a atributos de elementos descendentes, ela é realizada através de chamadas para os métodos `toTEX` destes elementos. Isto faz com que cada elemento definido no Currículo Lattes seja o responsável por gerar suas informações, assim como foi para carregá-las do arquivo XML.

Neste momento, com a conclusão da análise do funcionamento do *Parser* de XML, temos o processo de desenvolvimento do conversor *lattes2latex* finalizado - figura 5.4. No próximo capítulo será explanado o processo de validação da ferramenta e demonstrados

alguns resultados.

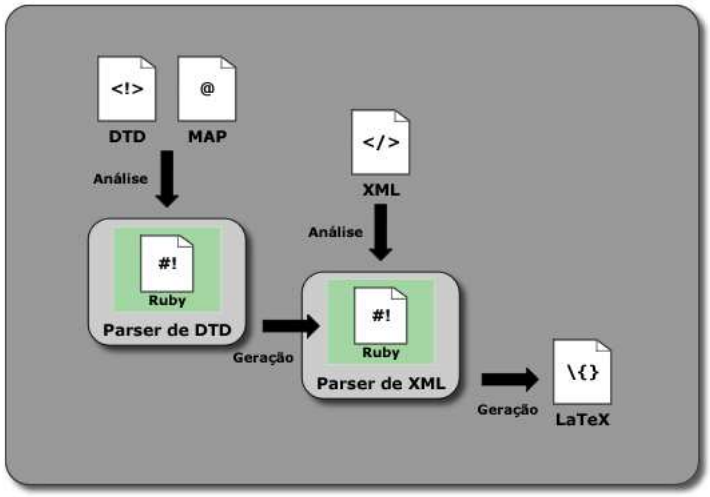


Figura 5.4: Processo de Conversão do Currículo Lattes em Código LaTeX

6 RESULTADOS

A validação do conversor *lattes2latex* foi realizada através da execução de testes com, aproximadamente, dez Currículos Lattes de professores e pós-graduandos do Instituto de Informática da Universidade Federal do Rio Grande do Sul. Neste capítulo, serão apresentados os resultados obtidos através do uso do *lattes2latex*, aplicado a uma instância de Currículo Lattes, bem como outras alternativas de implementação deste conversor e de uso do *Parser* de DTD criado.

6.1 O *lattes2latex* Aplicado à um Currículo Lattes

Conforme analisado nos capítulos anteriores, o conversor *lattes2latex* desenvolvido permite realizar a conversão de um Currículo Lattes armazenado no formato XML em um documento LaTeX formatado de acordo com a especificação desejada. Nesta implementação do conversor foram selecionados como informações presentes no documento final os seguintes elementos de um Currículo Lattes:

- Dados Gerais
  - Nome, Cidade e País de Nascimento
  - Resumo do Curriculum Vitae
  - Idiomas
  - Formação Acadêmica
    - Graduação
    - Mestrado
    - Doutorado
    - Pós-Doutorado
- Produção Bibliográfica
  - Trabalhos em Eventos
  - Artigos Publicados
  - Livros
  - Capítulos
  - Textos em Jornais ou Revistas

  - Artigos Aceitos para Publicação

- Dados Complementares

  - Participação em Bancas
    * Trabalhos de Graduação
    * Trabalhos de Mestrado
    * Trabalhos de Doutorado

Estas informações são obtidas através da análise do arquivo XML realizada pelo *Parser* de XML e, após a criação da estrutura de representação intermediária, geradas como saída em código LaTeX formatado. Nas próximas seções vamos estudar alguns elementos, pertencentes ao Currículo Lattes do Prof. Nicolas Maillard, orientador deste trabalho, desde a sua definição em XML até o resultado final obtido através da compilação do código LaTeX.

6.1.1 Informações Pessoais e Resumo

```
...
<DADOS-GERAIS
  NOME-COMPLETO="Nicolas Maillard"
  NOME-EM-CITACOES-BIBLIOGRAFICAS="MAILLARD, N."
  NACIONALIDADE="E"
  CPF="..."
  PAIS-DE-NASCIMENTO="Franca"
  UF-NASCIMENTO=""
  CIDADE-NASCIMENTO="Nantes"
  DATA-NASCIMENTO="..."
  SEXO="MASCULINO"
  NUMERO-IDENTIDADE="..."
  ORGAO-EMISSOR="..."
  UF-ORGAO-EMISSOR="..."
  DATA-DE-EMISSAO="..."
  NUMERO-DO-PASSAPORTE="..."
  NOME-DO-PAI="Didier Maillard"
  NOME-DA-MAE="Martine Maillard"
  PERMISSAO-DE-DIVULGACAO="NAO">
  <RESUMO-CV
    TEXTO-RESUMO-CV-RH="Possui graduacao em Matematicas Aplicadas
     e Informatica - ENSIMAG, Grenoble, Franca (1996), mestrado
     em Mathematiques Appliques et Paralllisme - Universite de
     Grenoble I (Univ. Joseph Fourier) (1996) e doutorado em
     Sciences et Technologies de l'Information - Univ. Joseph
     Fourier (2001). Atualmente, e professor adjunto da
     Universidade Federal do Rio Grande do Sul, onde leciona
     Sistemas Operacionais I e Compiladores na graduacao, alem de
     duas disciplinas no Programa de Pos-Graduacao em Ciencia da
     Computacao. Tem experiencia na area de Ciencia da Computacao,
     com enfase em Processamento paralelo de alto desempenho,
     atuando principalmente nos seguintes temas: processamento de
     alto desempenho, computacao em Clusters e Grids, MPI,
     escalonamento de processos."/>
  ...
</DADOS-GERAIS>
```

```
...
```

Listagem 6.1: Arquivo XML: Informações Pessoais e Resumos

```
...
@map['DADOS-GERAIS'] = <<EOF
  def toTEX
    puts '\\section*{Dados Gerais}'
    puts @atributos['NOME-COMPLETO']
    puts @atributos['CIDADE-NASCIMENTO']
    puts @atributos['PAIS-DE-NASCIMENTO']
    @elementos['RESUMO-CV'].toTEX
    @elementos['IDIOMAS'].toTEX
    @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
  end
EOF

@map['RESUMO-CV'] = <<EOF
  def toTEX
    puts '\\subsection*{Resumo}'
    puts @atributos['TEXTO-RESUMO-CV-RH']
  end
EOF
...
```

Listagem 6.2: Métodos de Geração do Código LaTeX: Informações Pessoais e Resumos

A listagem 6.1 mostra o trecho de código do arquivo XML do Currículo Lattes que armazena as informações pessoais do pesquisador e o resumo do currículo. Estas informações são populadas, com o uso do REXML, nas estruturas da listagem 6.3, geradas pelo *Parser* de XML a partir do DTD do Currículo Lattes e do arquivo de mapeamento criado - listagem 6.2. Após este processo, as informações são convertidas no código LaTeX da listagem 6.4, que possui a formatação definida através dos métodos mapeados em cada elemento do DTD.

```
class DadosGerais
  attr_reader :atributos, :elementos
  def initialize( d )
    return if d == nil
    @atributos = Hash.new
    @atributos['NOME-COMPLETO'] =
      d.attributes['NOME-COMPLETO']
    @atributos['NOME-EM-CITACOES-BIBLIOGRAFICAS'] =
      d.attributes['NOME-EM-CITACOES-BIBLIOGRAFICAS']
    @atributos['NACIONALIDADE'] =
      d.attributes['NACIONALIDADE']
    @atributos['CPF'] =
      d.attributes['CPF']
    @atributos['NUMERO-DO-PASSAPORTE'] =
      d.attributes['NUMERO-DO-PASSAPORTE']
    @atributos['PAIS-DE-NASCIMENTO'] =
      d.attributes['PAIS-DE-NASCIMENTO']
    @atributos['UF-NASCIMENTO'] =
      d.attributes['UF-NASCIMENTO']
    @atributos['CIDADE-NASCIMENTO'] =
      d.attributes['CIDADE-NASCIMENTO']
    ...
```

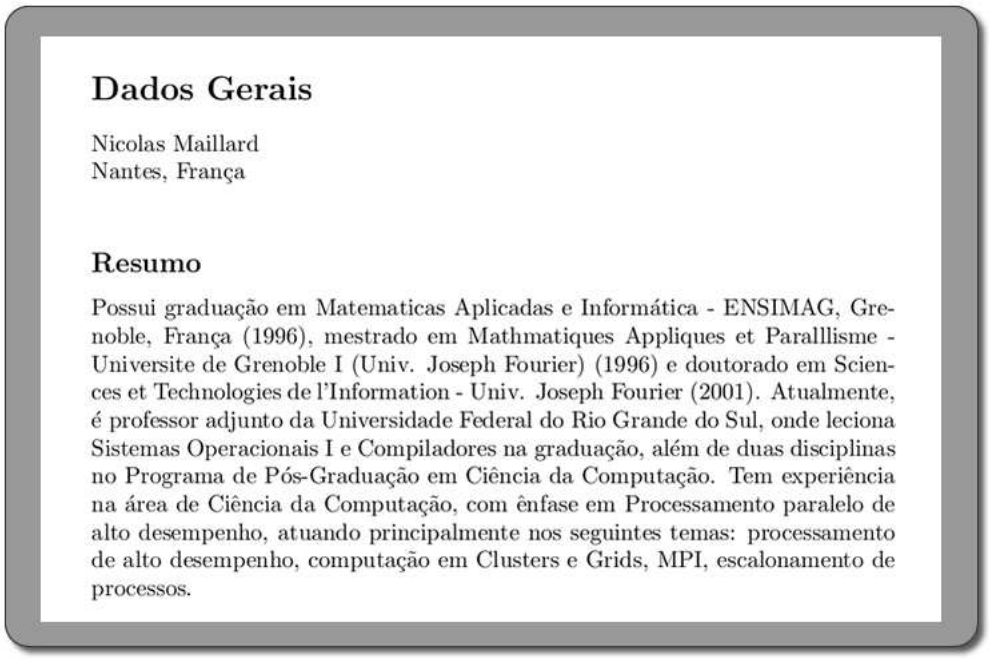
```
24      @elementos = Hash.new
25      @elementos['ENDERECO'] =
26        Endereco.new(d.elements['ENDERECO'])
27      @elementos['RESUMO-CV'] =
28        ResumoCV.new(d.elements['RESUMO-CV'])
29      ...
30    end

32    def toTEX
33      puts '\section*{Dados Gerais}'
34      puts @atributos['NOME-COMPLETO']
35      puts @atributos['CIDADE-NASCIMENTO']
36      puts @atributos['PAIS-DE-NASCIMENTO']
37      @elementos['RESUMO-CV'].toTEX
38      @elementos['IDIOMAS'].toTEX
39      @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
40    end
41  end

43  class ResumoCV
44    attr_reader :atributos, :elementos
45    def initialize( d )
46      return if d == nil
47      @atributos = Hash.new
48      @atributos['TEXTO-RESUMO-CV-RH'] =
49        d.attributes['TEXTO-RESUMO-CV-RH']
50    end

52    def toTEX
53      puts '\subsection*{Resumo}'
54      puts @atributos['TEXTO-RESUMO-CV-RH']
55    end
56  end
```

Listagem 6.3: Estrutura do *Parser* de XML: Informações Pessoais e Resumos

```
1  \section*{Dados Gerais}
2  Nicolas Maillard
3  Nantes, Franca

5  \subsection*{Resumo}
6  Possui graduacao em Matematicas Aplicadas e Informatica -
7  ENSIMAG, Grenoble, Franca (1996), mestrado em Mathematiques
8  Appliques et Paralllisme - Universite de Grenoble I (Univ.
9  Joseph Fourier) (1996) e doutorado em Sciences et Technologies
10 de l'Information - Univ. Joseph Fourier (2001). Atualmente,
11 e professor adjunto da Universidade Federal do Rio Grande do
12 Sul, onde leciona Sistemas Operacionais I e Compiladores na
13 graduacao, alem de duas disciplinas no Programa de Pos-Graduacao
14 em Ciencia da Computacao. Tem experiencia na area de Ciencia
15 da Computacao, com enfase em Processamento paralelo de alto
16 desempenho, atuando principalmente nos seguintes temas:
17 processamento de alto desempenho, computacao em Clusters e Grids,
18 MPI, escalonamento de processos."/>
```

Listagem 6.4: Código LaTeX: Informações Pessoais e Resumos

Este trecho de código LaTeX, após ser compilado, gera a seleção presente na figura 6.1.



Dados Gerais

Nicolas Maillard
Nantes, França

Resumo

Possui graduação em Matematicas Aplicadas e Informática - ENSIMAG, Grenoble, França (1996), mestrado em Mathmatiques Appliques et Paralllisme - Universite de Grenoble I (Univ. Joseph Fourier) (1996) e doutorado em Sciences et Technologies de l'Information - Univ. Joseph Fourier (2001). Atualmente, é professor adjunto da Universidade Federal do Rio Grande do Sul, onde leciona Sistemas Operacionais I e Compiladores na graduação, além de duas disciplinas no Programa de Pós-Graduação em Ciência da Computação. Tem experiência na área de Ciência da Computação, com ênfase em Processamento paralelo de alto desempenho, atuando principalmente nos seguintes temas: processamento de alto desempenho, computação em Clusters e Grids, MPI, escalonamento de processos.

Figura 6.1: Documento Gerado: Informações Pessoais e Resumos

6.1.2 Artigos Publicados

Nesta seção, na listagem 6.5, temos o armazenamento dos artigos publicados pelo pesquisador. Após processadas pelo *lattes2latex* estas informações têm como saída o código LaTeX da listagem 6.6.

```
<ARTIGOS-PUBLICADOS>
  <ARTIGO-PUBLICADO
    SEQUENCIA-PRODUCAO="1">
    <DADOS-BASICOS-DO-ARTIGO
      NATUREZA="COMPLETO"
      TITULO-DO-ARTIGO=
       "Using Computer Algebra to Diagonalize some Kane Matrices"
      ANO-DO-ARTIGO="2000"
      PAIS-DE-PUBLICACAO=""
      IDIOMA="Ingls"
      MEIO-DE-DIVULGACAO="IMPRESSO"
      HOME-PAGE-DO-TRABALHO="[http://www-id.imag.fr/~maillard]"
      FLAG-RELEVANCIA="NAO"
      DOI=""/>
    <DETALHAMENTO-DO-ARTIGO
      TITULO-DO-PERIODICO-OU-REVISTA=
       "Journal of Physics. A, Mathematical and General"
      ISSN="03054470"
      VOLUME="33"
      FASCICULO=""
      SERIE=""
      PAGINA-INICIAL="2857"
```

```
      PAGINA-FINAL="2870"
      LOCAL-DE-PUBLICACAO="London"/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Claude Pierre Jeannerod"
      NOME-PARA-CITACAO="JEANNEROD, Claude Pierre"
      ORDEM-DE-AUTORIA="1"/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Nicolas Maillard"
      NOME-PARA-CITACAO="MAILLARD, N."
      ORDEM-DE-AUTORIA="2"/>
    <PALAVRAS-CHAVE
      PALAVRA-CHAVE-1="perturbation theory"
      PALAVRA-CHAVE-2="Eigenvalues"
      PALAVRA-CHAVE-3="schroedinger equation"
      PALAVRA-CHAVE-4=""
      PALAVRA-CHAVE-5=""
      PALAVRA-CHAVE-6=""/>
    ...
  </ARTIGO-PUBLICADO>
  <ARTIGO-PUBLICADO
    SEQUENCIA-PRODUCAO="27">
    <DADOS-BASICOS-DO-ARTIGO
      NATUREZA="COMPLETO"
      TITULO-DO-ARTIGO=
       "The I-Cluster Cloud: Distributed Management of Idle
        Resources for Intense Computing"
      ANO-DO-ARTIGO="2005"
      PAIS-DE-PUBLICACAO="Holanda"
      IDIOMA="Ingls"
      MEIO-DE-DIVULGACAO="IMPRESSO"
      HOME-PAGE-DO-TRABALHO=""
      FLAG-RELEVANCIA="SIM"
      DOI=""/>
    <DETALHAMENTO-DO-ARTIGO
      TITULO-DO-PERIODICO-OU-REVISTA="Parallel Computing"
      ISSN="01678191"
      VOLUME="31"
      FASCICULO=""
      SERIE=""
      PAGINA-INICIAL="813"
      PAGINA-FINAL="838"
      LOCAL-DE-PUBLICACAO=""/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Bruno Richard"
      NOME-PARA-CITACAO="RICHARD, Bruno"
      ORDEM-DE-AUTORIA="1"/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Csar A F De Rose"
      NOME-PARA-CITACAO="ROSE, Csar A F de"
      ORDEM-DE-AUTORIA="3"/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Reynaldo Novaes"
      NOME-PARA-CITACAO="NOVAES, Reynaldo"
      ORDEM-DE-AUTORIA="4"/>
    <AUTORES
      NOME-COMPLETO-DO-AUTOR="Nicolas Maillard"
      NOME-PARA-CITACAO="MAILLARD, N."
      ORDEM-DE-AUTORIA="2"/>
```

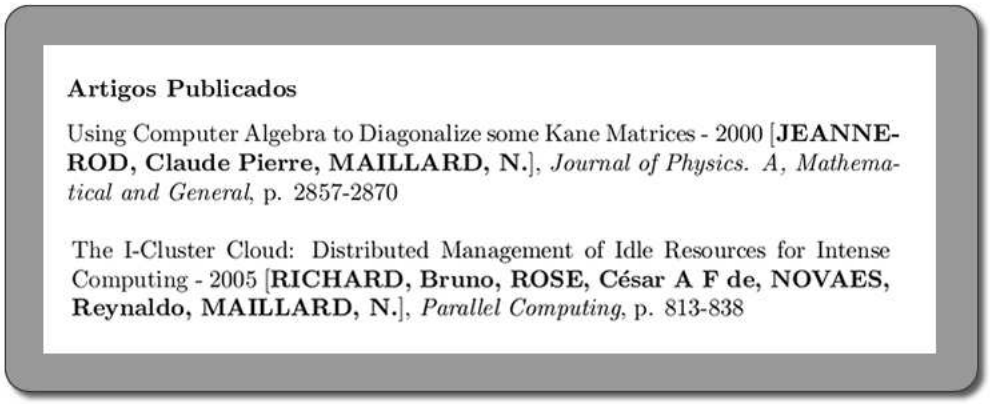
```
81      ...
82    </ARTIGO-PUBLICADO>
83  </ARTIGOS-PUBLICADOS>
```

Listagem 6.5: Arquivo XML: Artigos Publicados

```
2  \subsubsection*{Artigos Publicados}
3  Using Computer Algebra to Diagonalize some Kane Matrices - 2000
4  [\textbf{JEANNEROD, Claude Pierre, MAILLARD, N.}],
5  \textit{Journal of Physics. A, Mathematical and General},
6  p. 2857-2870

8  The I-Cluster Cloud: Distributed Management of Idle Resources for
9  Intense Computing - 2005
10 [\textbf{RICHARD, Bruno, ROSE, Csar A F de, NOVAES, Reynaldo,
11 MAILLARD, N.}],
12 \textit{Parallel Computing}, p. 813-838
```

Listagem 6.6: Código LaTeX: Artigos Publicados

Novamente, a compilação do código LaTeX acima produz o resultado exibido na figura 6.2.



Artigos Publicados

Using Computer Algebra to Diagonalize some Kane Matrices - 2000 [**JEANNEROD, Claude Pierre, MAILLARD, N.**], *Journal of Physics. A, Mathematical and General*, p. 2857-2870

The I-Cluster Cloud: Distributed Management of Idle Resources for Intense Computing - 2005 [**RICHARD, Bruno, ROSE, César A F de, NOVAES, Reynaldo, MAILLARD, N.**], *Parallel Computing*, p. 813-838

Figura 6.2: Documento Gerado: Artigos Publicados

Como visto na figura 6.2 as instruções de formatação presentes no arquivo de mapeamento levaram o *lattes2latex* a realçar, em cada artigo, os títulos dos *proceedings* em ítálico e os autores em negrito. Para fins de legibilidade, apenas algumas publicações constam na figura. O Currículo Lattes completo gerado pode ser encontrado no Anexo A.

6.2 Alternativas na Implementação do Conversor *lattes2latex*

Nesta seção serão estudadas algumas alternativas de uso do conversor *lattes2latex*, como a alteração na formatação do código LaTeX gerado, a possibilidade de validar o conteúdo de um Currículo Lattes, entre outras.

6.2.1 Alterando a Formatação do Documento Gerado

Como detalhado no capítulo 5, toda a semântica que define a formatação de cada elemento do Currículo Lattes foi definida através do arquivo de mapeamento. Portanto, é bastante intuitivo que, ao se desejar uma formatação diferente ou a inclusão/remoção de informações no currículo, basta que se altere os métodos mapeados nos elementos afetados e se recrie o *Parser* de XML a partir da execução do *Parser* de DTD.

```
...
@map['DADOS-GERAIS'] = <<EOF
  def toTEX
    puts '\\section*{Dados Gerais}'
    puts @atributos['NOME-COMPLETO'].upcase
    puts @atributos['CIDADE-NASCIMENTO']
    puts @atributos['PAIS-DE-NASCIMENTO']
    @elementos['RESUMO-CV'].toTEX
    @elementos['IDIOMAS'].toTEX
    @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
  end
EOF
...
```

Listagem 6.7: Métodos de Geração do Código LaTeX: Alternativas de Formatação

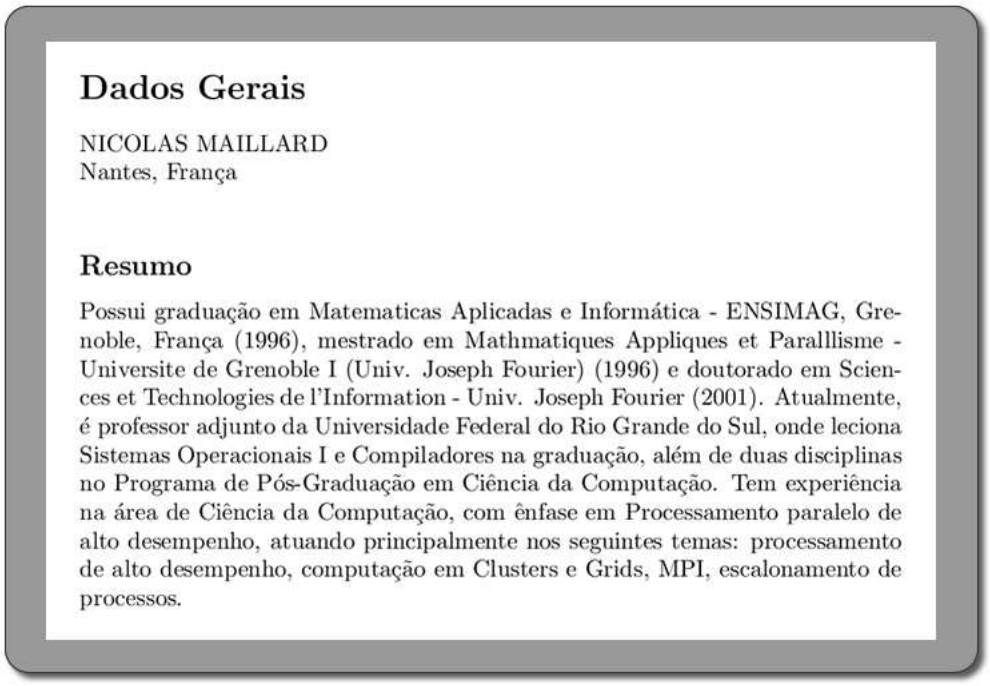
O código da listagem 6.7 demonstra uma pequena alteração na formatação do nome do pesquisador, que passará a ser exibido com todos os caracteres em letras maiúsculas. Para realizar esta mudança bastou alterar o método `toTEX` do elemento `DADOS-GERAIS` para aplicar a função `upcase` ao atributo `NOME-COMPLETO` antes de imprimí-lo no arquivo de saída. Quando executado novamente, o *Parser* de DTD irá gerar a estrutura do *Parser* de XML já com esta modificação, como pode ser visto na listagem 6.8.

```
class DadosGerais
  attr_reader :atributos, :elementos
  def initialize( d )
    return if d == nil
    @atributos = Hash.new
    @atributos['NOME-COMPLETO'] =
      d.attributes['NOME-COMPLETO']
    @atributos['NOME-EM-CITACOES-BIBLIOGRAFICAS'] =
      d.attributes['NOME-EM-CITACOES-BIBLIOGRAFICAS']
    @atributos['NACIONALIDADE'] =
      d.attributes['NACIONALIDADE']
    @atributos['CPF'] =
      d.attributes['CPF']
    @atributos['NUMERO-DO-PASSAPORTE'] =
      d.attributes['NUMERO-DO-PASSAPORTE']
    @atributos['PAIS-DE-NASCIMENTO'] =
      d.attributes['PAIS-DE-NASCIMENTO']
    @atributos['UF-NASCIMENTO'] =
      d.attributes['UF-NASCIMENTO']
    @atributos['CIDADE-NASCIMENTO'] =
      d.attributes['CIDADE-NASCIMENTO']
    ...

    @elementos = Hash.new
    @elementos['ENDERECO'] =
      Endereco.new(d.elements['ENDERECO'])
```

```
27      @elementos['RESUMO-CV'] =
28        ResumoCV.new(d.elements['RESUMO-CV'])
29      ...
30    end

32    def toTEX
33      puts '\section*{Dados Gerais}'
34      puts @atributos['NOME-COMPLETO'].upcase
35      puts @atributos['CIDADE-NASCIMENTO']
36      puts @atributos['PAIS-DE-NASCIMENTO']
37      @elementos['RESUMO-CV'].toTEX
38      @elementos['IDIOMAS'].toTEX
39      @elementos['FORMACAO-ACADEMICA-TITULACAO'].toTEX
40    end
41  end
```

Listagem 6.8: Estrutura do *Parser* de XML: Alternativas de Formatação

Uma nova compilação deste código LaTeX produz o trecho exibido na figura 6.3.



Dados Gerais

NICOLAS MAILLARD
Nantes, França

Resumo

Possui graduação em Matematicas Aplicadas e Informática - ENSIMAG, Grenoble, França (1996), mestrado em Mathmatiques Appliques et Paralllisme - Universite de Grenoble I (Univ. Joseph Fourier) (1996) e doutorado em Sciences et Technologies de l'Information - Univ. Joseph Fourier (2001). Atualmente, é professor adjunto da Universidade Federal do Rio Grande do Sul, onde leciona Sistemas Operacionais I e Compiladores na graduação, além de duas disciplinas no Programa de Pós-Graduação em Ciência da Computação. Tem experiência na área de Ciência da Computação, com ênfase em Processamento paralelo de alto desempenho, atuando principalmente nos seguintes temas: processamento de alto desempenho, computação em Clusters e Grids, MPI, escalonamento de processos.

Figura 6.3: Documento Gerado: Alternativas de Formatação

6.2.2 Validando o Currículo Lattes

O arquivo de mapeamento foi especificado para definir a formatação do Currículo Lattes em formato LaTeX para cada elemento. Entretanto, os métodos presentes neste arquivo não se restringem à geração deste código. Qualquer computação sobre os dados do Currículo Lattes pode ser feita, até mesmo a validação dos mesmos.

Como informado no capítulo 2, o DTD determina os elementos e a estrutura presentes no arquivo XML do Currículo Lattes. Nada é delimitado quanto a validade dos dados informados. Um artigo, por exemplo, pode ter seu ano de publicação definido como

2042. Isto, obviamente, não estaria correto, mas o DTD não tem autonomia para este tipo de checagem. Para isto, os métodos que definem o *parsing* de um Currículo Lattes podem ser criados para verificar esta e muitas outras informações, gerando uma listagem de erros e avisos ao pesquisador.

O método `validate`, na listagem 6.10, por exemplo, verifica para cada elemento `DADOS-BASICOS-DO-ARTIGO` se o ano deste é válido, ou seja, é menor ou igual ao ano atual. Caso seja inválido, é impressa na tela uma mensagem ao pesquisador - figura 6.4, informando o título do artigo e alertando para o ano incorreto.

```
1  ...
2  @map['DADOS-BASICOS-DO-ARTIGO'] = <<EOF
3    def validade
4      if @atributos['ANO-DO-ARTIGO'] > Date.new.year
5        puts 'Ano Invalido! Artigo: '
6        puts @atributos['TITULO-DO-ARTIGO']
7        puts ' (' + @atributos['ANO-DO-ARTIGO'] + ')'
8      end
9    end
10 EOF
11 ...
```

Listagem 6.9: Validando o Currículo Lattes

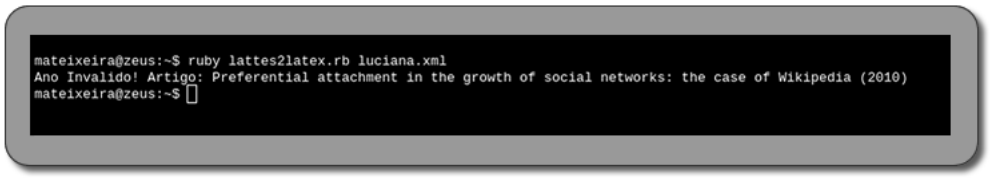


Figura 6.4: Validando o Currículo Lattes: Ano do Artigo Inválido

Qualquer elemento ou atributo do Currículo Lattes pode ser validado desta maneira, o que possibilita também a comparação entre elementos e atributos.

6.3 Alternativas no Uso do *Parser* de DTD

Como descrito na seção 5.2, o *Parser* de DTD criado não se restringe ao uso na definição do Currículo Lattes. Um *Parser* de XML pode ser gerado para qualquer marcação XML definida através de uma especificação DTD. Utilizando como exemplo o DTD descrito na listagem 2.3 associado ao arquivo de mapeamento da listagem 6.10, é possível gerar um *Parser* de XML que, para uma entrada XML de acervo, exiba na tela os títulos dos livros existentes.

```
1  ...
2  @map['acervo'] = <<EOF
3    def parse
4      @elementos['livro'].lista
5    end
6  EOF

8  @map['livro'] = <<EOF
```

```
9   def lista
10    @elementos['titulo'].each { | e | puts e.elementos }
11  end
12 end
13 ...
```

Listagem 6.10: Validando o Currículo Lattes

A aplicação do arquivo XML da listagem 2.1 a este *Parser* de XML gerado produz como saída a tela presente na figura 6.5.

Figura 6.5: Alternativas no Uso do *Parser* de DTD: Acervo de Livros

7 CONCLUSÃO

Este trabalho apresentou o processo de desenvolvimento do *lattes2latex - Uma Ferramenta para Conversão de Currículo Lattes em Documentos LaTeX*. Para isto, foram abordados os principais conceitos que envolveram este desenvolvimento, como XML, DTD, *Parsers*, Ruby e LaTeX.

Com a necessidade do uso de XML como formato de entrada, decidiu-se desenvolver um *parser* de arquivos DTD, denominado *Parser* de DTD, que gerasse um *Parser* de XML específico para a estrutura esperada. Este *Parser* de DTD foi criado utilizando-se a linguagem de programação Ruby e, com o emprego de técnicas de metaprogramação existentes, foi possível alcançar o objetivo proposto. Além disso, durante o desenvolvimento do presente trabalho, foram analisadas alternativas de formatos eletrônicos para a saída do conversor. Devido a questões técnicas, como a fidelidade na representação, a facilidade de geração de código e a possibilidade de alteração do conteúdo gerado, o processador de texto LaTeX acabou sendo escolhido frente a outras alternativas como OpenXML, ODF, PostScript e PDF.

Como a especificação DTD contém somente informações da estruturação dos dados, foi necessária a elaboração de um outro formalismo, denominado arquivo de mapeamento, onde a formatação LaTeX desejada, a nível de elementos, é especificada.

A partir do *Parser* de DTD, do arquivo DTD de especificação do Currículo Lattes e do arquivo de mapeamento criado, foi gerado automaticamente o *Parser* de XML *lattes2latex* que, recebendo um arquivo XML de um Currículo Lattes, converte suas informações, gerando um documento LaTeX pronto para ser compilado. Este documento é estruturado de acordo com as definições existentes no arquivo de mapeamento.

Devido à generalidade do *Parser* de DTD, foi estudada a possibilidade de mudança na formatação do documento LaTeX criado, bastando para isso alterar o arquivo de mapeamento nos pontos onde for necessário. Além disso, os métodos definidos por este arquivo não se restringem à criação de documentos LaTeX, podendo qualquer computação ser realizada, como a geração de outros formatos de arquivo e até mesmo a validação dos dados presentes em um Currículo Lattes.

Assim como é possível alterar o conteúdo do arquivo de mapeamento, também é possível utilizar outra especificação DTD como fonte para o *Parser* de DTD. Isto permite a criação de *Parsers* de XML específicos para qualquer arquivo DTD que defina uma linguagem XML. Com esta liberdade, esta ferramenta torna-se bastante abrangente, podendo ser utilizada em diversas especificações, associadas a inúmeros processos, através

dos arquivos de mapeamento, como a listagem de títulos existentes em um acervo de livros apresentada no capítulo 6.

Como trabalhos futuros, existe a possibilidade da extensão do conversor para que este, através de uma interface gráfica, permita ao usuário selecionar quais mapeamentos de elementos deseja utilizar no momento da conversão de seu currículo. Também é possível abordar a implementação de um arquivo de mapeamento multidimensional que, além de definir um comportamento para cada elemento presente no DTD, também possa gerenciar a existencia e o uso de diversos idiomas pré-configurados para o conversor.

Outra alternativa de trabalho futuro é disponibilizar ao usuário vários arquivos de mapeamento para que sejam selecionados no momento da conversão, através de uma interface gráfica.

A execução do conversor *lattes2latex* pode ser realizada através da utilização de um interpretador Ruby, existente na maioria dos sistemas operacionais disponíveis, ou mediante a utilização de um executável pré-compilado para determinada distribuição de sistema operacional.

REFERÊNCIAS

ADOBE. *Adobe PostScript language specifications*. 2008. Disponível em: <http://partners.adobe.com/public/developer/ps/index_specs.html>. Acessado em: maio/2008.

ADOBE. *PDF Reference and Adobe Extensions to the PDF Specification*. 2008. Disponível em: <http://www.adobe.com/devnet/pdf/pdf_reference.html>. Acessado em: maio/2008.

APACHE. *Xerces Java Parser*. 2008. Disponível em: <http://xerces.apache.org/xerces-j/>. Acessado em: maio/2008.

CLARK, J. *Expat XML Parser*. 2008. Disponível em: <http://sourceforge.net/projects/expat/>. Acessado em: maio/2008.

CNPQ. *Conselho Nacional de Desenvolvimento Científico e Tecnológico*. 2008. Disponível em: http://www.cnpq.br. Acessado em: maio/2008.

CNPQ. *Plataforma Lattes*. 2008. Disponível em: http://lattes.cnpq.br. Acessado em: maio/2008.

CONSCIENTIAS. *Comunidade CONSCIENTIAS-LMPL*. 2008. Disponível em: http://lmpl.cnpq.br/lmpl. Acessado em: maio/2008.

CONSCIENTIAS. *DTD do Currículo Lattes*. 2008. Disponível em: http://lmpl.cnpq.br/lmpl/Gramaticas/Curriculo/DTD/Fontes/LMPLCurriculo.DTD. Acessado em: maio/2008.

DA Importância do ODF. 2008. Disponível em: <http://www.broffice.org/Importancia_ODF>. Acessado em: maio/2008.

FLANAGAN, D.; MATSUMOTO, Y. *The Ruby Programming Language*. illustrated. [S.l.]: O'Reilly, 2008. 429 p.

HAROLD, E. R. *XML 1.1 Bible*. 3, illustrated. ed. [S.l.]: John Wiley and Sons, 2004. 1022 p.

HUNTER, D. et al. *Beginning XML*. 4, illustrated. ed. [S.l.]: John Wiley and Sons, 2007. 1039 p.

ISO. *International Organization for Standardization*. 2008. Disponível em: <http://www.iso.org>. Acessado em: maio/2008.

KERNIGHAN, B. W.; RITCHIE, D. M. *The C Programming Language*. 2, illustrated. ed. [S.l.]: Prentice Hall, 1988. 272 p.

LAMPORT, L. *LaTeX: A document preparation system*. 23, illustrated, reprint. ed. [S.l.]: Addison-Wesley, 1986. 242 p.

LATEX project: LaTeX - A document preparation system. 2008. Disponível em: <http://www.latex-project.org/>. Acessado em: maio/2008.

MATSUMOTO, Y. *Linguagem de Programação Ruby*. 2008. Disponível em: <http://www.ruby-lang.org/pt/>. Acessado em: maio/2008.

MATSUMOTO, Y. *REXML*. 2008. Disponível em: <http://www.germane-software.com/software/rexml/>. Acessado em: maio/2008.

MCT. *Ministério da Ciência e Tecnologia*. 2008. Disponível em: http://www.mct.gov.br. Acessado em: maio/2008.

MICROSOFT. *Microsoft Office File Formats*. 2008. Disponível em: <http://msdn.microsoft.com/en-us/library/cc313118.aspx>. Acessado em: maio/2008.

MICROSOFT. *.NET Framework Developer Center*. 2008. Disponível em: <http://msdn.microsoft.com/pt-br/netframework/default.aspx>. Acessado em: maio/2008.

MICROSOFT. *Office XML File Formats*. 2008. Disponível em: <http://msdn.microsoft.com/en-us/office/bb265236.aspx>. Acessado em: maio/2008.

MICROSOFT. *Word 2007: Rich Text Format (RTF) Specification*. 2008. Disponível em: <http://www.microsoft.com/downloads/details.aspx?FamilyId=DD422B8D-FF06-4207-B476-6B5396A18A2B\&displaylang=en>. Acessado em: maio/2008.

MITCHELL, J. C. *Concepts in Programming Languages*. illustrated. [S.l.]: Cambridge University Press, 2003. 529 p.

OASIS. *OASIS - Advancing open standards for the information society*. 2008. Disponível em: <http://www.oasis-open.org>. Acessado em: maio/2008.

OASIS. *OASIS Open Office Specification*. 2008. Disponível em: <http://docs.oasis-open.org/office/v1.1/OpenDocument-v1.1.html>. Acessado em: maio/2008.

PRICE, A. M. d. A.; TOSCANI, S. S. *Implementação de Linguagens de Programação: Compiladores*. [S.l.]: Sagra Luzzatto, 2005.

RAY, E. T. *Learning XML*. 2, illustrated. ed. [S.l.]: O'Reilly, 2003. 400 p.

SCOTT, M. L. *Programming Language Pragmatics*. 2, illustrated. ed. [S.l.]: Morgan Kaufmann, 2006. 875 p.

SUN. *Developer Resources for Java Technology*. 2008. Disponível em: <http://java.sun.com/>. Acessado em: maio/2008.

THOMAS, D.; HUNT, A. *Programming Ruby: The Pragmatic Programmer's Guide*. illustrated. [S.l.]: Addison-Wesley, 2000. 564 p.

W3C. *Extensible Markup Language*. 2008. Disponível em: <http://www.w3.org/XML/>. Acessado em: maio/2008.

W3C. *The Extensible Stylesheet Language Family*. 2008. Disponível em: <http://www.w3.org/Style/XSL/>. Acessado em: maio/2008.

W3C. *Guide to the W3C XML Specification ("XMLspec") DTD*. 2008. Disponível em: <http://www.w3.org/XML/1998/06/xmlspec-report.htm>. Acessado em: maio/2008.

A ANEXO

Curriculum Vitae

Dados Gerais

Nicolas Maillard
Nantes, França

Resumo

possui graduação em Matématicas Aplicadas e Informática - ENSIMAG, Grenoble, França (1996), mestrado em Mathématiques Appliquées et Parallélisme - Universite de Grenoble I (Univ. Joseph Fourier) (1996) e doutorado em Sciences et Technologies de l'Information - Univ. Joseph Fourier (2001). Atualmente, é professor adjunto da Universidade Federal do Rio Grande do Sul, onde leciona Sistemas Operacionais I e Compiladores na graduação, além de duas disciplinas no Programa de Pós-Graduação em Ciência da Computação. Tem experiência na área de Ciência da Computação, com ênfase em Processamento paralelo de alto desempenho, atuando principalmente nos seguintes temas: processamento de alto desempenho, computação em Clusters e Grids, MPI, escalonamento de processos.

Idiomas

Idioma	Leitura	Fala	Escrita	Compreensão
Francês	Bem	Bem	Bem	Bem
Português	Bem	Bem	Bem	Bem
Inglês	Bem	Bem	Bem	Bem
Alemão	Bem	Razoavelmente	Bem	Bem

Formação Acadêmica

Graduação

1993 - 1996	Matématicas Aplicadas e Informática École Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Mestrado

1995 - 1996	Mathématiques Appliquées Parallélisme Universite de Grenoble I (Scientifique Et Medicale - Joseph Fourier)

Doutorado

1996 - 2001	Sciences Et Technologies de l Information Universite de Grenoble I (Scientifique Et Medicale - Joseph Fourier)

Pós-Doutorado

2002 - 2003	Pontifícia Universidade Católica do Rio Grande do Sul
2004 -	Universidade Federal do Rio Grande do Sul

Produção Bibliográfica

Trabalhos em Eventos

Parallélisation du Calcul ab-initio de l'énergie électronique par la méthode MP2 [**Jean-Louis Roch, Pierre Valiron, MAILLARD, N.**], RenPar'9 - 1997, *RenPar'9 - 9èmes rencontres francophones du parallélisme*, p. 45-49

An Algorithmic Approach for the Symmetric Perturbed Eigenvalue Problem - Application to the solution of the Schroedinger Equation by a kp-perturbation method [**JEANNEROD, Claude Pierre, PFLUEGEL, Eckhard, MAILLARD, N.**], IMACS Conference on Applications of Computer Algebra - 1998

Parallelizing a Dense Matching Region Growing Algorithm for an Image Interpolation Application [**FERNANDES, L. G., DENNEULIN, Y., MAILLARD, N.**], International Conference on Parallel and Distributed Techniques and Applcations - 2001, *Proceedings of the International Conference on Parallel and Distributed Techniques and Applcations*, p. 491-495

The Virtual Cluster: a Dynamic Environment for Exploitation of Idle Network Resources [**ROSE, César A F de, BLANCO, F., SAIKOSKI, K., NOVAES, Reynaldo, RICHARD, O., RICHARD, Bruno, MAILLARD, N.**], SBAC-PAD - 2002, *14th symposium on Computer Architecture and High-Performance Computing (SBAC-PAD 2002)*, p. 141-148

Contrôle amorti des synchronisations pour le test d extasciiacutearrêt des méthodes itératives [**DAOUDI, El Mostafa, MANNEBACK, Pierre, ROCH, Jean Louis, MAILLARD, N.**], RenPar extasciiacute14 - 2002, *14èmes rencontres francophones du parallélisme*, p. 177-182

Tradeoff to minimize extra-computations and stopping criterion tests for parallel iterative schemes [**BEAUMONT, Olivier, DAOUDI, El Mostafa, MANNEBACK, Pierre, ROCH, Jean Louis, MAILLARD, N.**], Parallel Matrix Algorithms and Applications (PMAA'04) - 2004

Automatic Data-Flow Graph Generation of MPI Programs [**SILVA, Rafael Ennes da, PEZZI, Guilherme Peretti, DIVERIO, Tiaraju, MAILLARD, N.**], SBAC-PAD - 2005, *17th Symposium on Computer Architecture and High Performance Computing*, p. 93-100

Scheduling Dynamically Spawned Processes in MPI-2 [**CERA, Márcia Cristina, PEZZI, Guilherme Peretti, PILLA, Maurício, NAVAUX, Philippe, MAILLARD, N.**], 12th Workshop on Job Scheduling Strategies for Parallel Processing - 2006, *Job Scheduling Strategies for Parallel Processing, 12th International Workshop, JSSPP 2006*, p. 33-45

Improving the Dynamic Creation of Processes in MPI-2 [**CERA, Márcia Cristina, PEZZI, Guilherme Peretti, MATTHIAS, Elton Nicoletti, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.**], Euro-PVM/MPI - 2006, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, p. 247-255

Escalonamento Dinâmico de programas MPI-2 utilizando Divisão e Conquista [**PEZZI, Guilherme Peretti, CERA, Márcia Cristina, MATHIAS, Elton Nicoletti, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.**], Workshop em Sistemas Computacionais de Alto Desempenho - 2006, *VII Workshop em Sistemas Computacionais de Alto Desempenho*, p. 71-78

Grid Computing for Mesoscale Climatology: Experimental Comparison of Three Platforms [**VELHO, Haroldo Fraga de Campo, PRETO, A., Stephany, S., RODRIGUES, E., PANETTA, Jairo, Almeida, E., SOUTO, R. P., NAVAUX, Philippe, DIVERIO, Tiaraju, DIAS, Pedro Leite da Silva, MAILLARD, N.**], WCGC'2006 - Workshop on Computational Grids and Clusters - 2006, *vecpar'06: 6th International Conference on High Performance Computing in Computational Sciences.*, p. 1-6

ICE: A Service Oriented Approach to Uniform Access and Management of Cluster Environments [**MARQUEZAN, Clarissa, NAVAUX, Philippe Olivier Alexandre, SCHNORR, Lucas Mello, RIGHI, Rodrigo da Rosa, CARISSIMI, Alexandre da Silva, MAILLARD, N.**], International Workshop on Grid Testbeds (together with CCGRID'2006) - 2006, *Proceedings of CCGRID'06*, p. 1-8

Portal ICE - Uso de Web Services para Atingir Extensibilidade no Gerenciamento de Múltiplos Agregados [**RIGHI, Rodrigo da Rosa, SCHNORR, Lucas Mello, MARQUEZAN, Clarissa, CARISSIMI, Alexandre, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.**], Workshop em Sistemas Computacionais de Alto Desempenho - 2006, *VII Workshop em Sistemas Computacionais de Alto Desempenho*, p. 133-140

Processing Mesoscale Climatology in a Grid Environment [**SOUTO, R. P., ÁVILA, Rafael Bohrer, NAVAUX, Philippe Alexandre Olivier, Py, M., DIVERIO, Tiaraju Asmuz, VELHO, Haroldo Fraga de Campo, Stephany, S., PRETO, A., PANETTA, Jairo, RODRIGUES, E., ALMEIDA, E., MAILLARD, N.**], CCGRID'07 - 2007, *Proceedings of CCGRID'07.*, p. 363-370

Performance Improvement of the Parallel Lattice Boltzmann Method [**SCHEPKE, C., MAILLARD, N.**], SBAC-PAD - 2007, *19th International Symposium on Computer Architecture and High Performance Computing*, p. 71-78

A Component-Oriented Support for Hierarchical MPI Programming on Multi-Cluster Grid Environments [**MATHIAS, Elton Nicoletti, BAUDE, F., Cave, V., MAILLARD, N.**], SBAC-PAD - 2007, *19th International Symposium on Computer Architecture and High Performance Computing*, p. 135-142

Impacto da Migração de Máquinas Virtuais de Xen na Execução de Programas MPI [**VEIGA, M., RIGHI, Rodrigo da Rosa, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.**], Workshop em Sistemas Computacionais de Alto Desempenho - 2007, *VIII Workshop em Sistemas Computacionais de Alto Desempenho*, p. 45-52

On-Line Scheduling of MPI-2 Programs with Hierarchical Work Stealing [**PEZZI, Guilherme Peretti, CERA, Márcia Cristina, MATHIAS, Elton Nicoletti, NAVAUX, Philippe Alexandre Olivier, MAILLARD, N.**], SBAC-PAD - 2007, *19th International Symposium on Computer Architecture and High Performance Computing*, p. 247-254

Deque-Free Work-Optimal Parallel STL Algorithms [**TRAOURE, D., ROCH, Jean Louis, GAUTIER, T., BERNARD, J., MAILLARD, N.**], EUROPAR extasciiacute08 - 2008, *Proceedings of EUROPAR extasciiacute08*, p. 887-897

Controle da Granularidade com Threads em Programas MPI Dinâmicos [**LIMA, J. V. F., MAILLARD, N.**], WSCAD-SSC 2008 - 2008, *IX Simpósio em Sistemas Computacionais - WSCAD-SSC 2008*, p. 125-133

Artigos Publicados

Using Computer Algebra to Diagonalize some Kane Matrices - 2000 [**JEANNEROD, Claude Pierre, MAILLARD, N.**], *Journal of Physics. A, Mathematical and General*, p. 2857-2870

The I-Cluster Cloud: Distributed Management of Idle Resources for Intense Computing - 2005 [**RICHARD, Bruno, ROSE, César A F de, NOVAES, Reynaldo, MAILLARD, N.**], *Parallel Computing*, p. 813-838

Livros Publicados ou Organizados

Anais do 2o Workshop de Processemanto Paralelo e Distribuído - 2004 [**NAVAUX, Philippe Alexandre Olivier, OLIVEIRA, Márcio Ramos de, DIVERIO, Tiarajú Asmuz, MAILLARD, N.**], 190 páginas.

Dados Complementares

Participação em Bancas

Graduação

Hélio Antônio Miranda Silva - *Estudo da Usabilidade da Linguagem Java em Aglomerados de Computadores* - 2003 [PASIN, Marcelo, MÜLLER, F. Martins, MAILLARD, N.]

Caciano dos Santos Machado - *Análise da Biblioteca DECK com as Aplicações do NAS Parallel Benchmark* - 2005 [CARISSIMI, Alexandre da Silva, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.]

Farlon Souto - *Estudo de Sistemas Operacionais Embarcados: uma Abordagem Visando o Chip Dedicado de Kernel* - 2005 [CARISSIMI, Alexandre da Silva, JOÃO NETTO,, MAILLARD, N.]

Mairo Pedrini - *Um Modelo de Arquitetura para Depuração em Ambientes Distribuídos* - 2006 [CARISSIMI, Alexandre da Silva, NAVAUX, Philippe Olivier Alexandre, MAILLARD, N.]

Mestrado

Andre Luis Martinotto - *Resolucao de Sistemas de Equacoes Lineares atraves de Metodos de Decomposicao de Dominio* - 2004 [DIVERIO, Tiaraju Asmuz, GEYER, Cláudio Fernando Resin, NAVAUX, Philippe Oliver Alexandre, MAILLARD, N.]

Rodrigo da Rosa Righi - *Sistema Aldeia: Programação Paralela e Distribuída em Java sobre Infiniband e DECK.* - 2005 [GEYER, Claudio Fernando Resin, AMORIM, Claudio Luis de, MAILLARD, N.]

Márcia Cristina Cera - *Suporte ao Controle e Alocação Dinâmica de Computadores em Java* - 2005 [CHARÃO, Andréa Schwertner, PASIN, Marcelo, MAILLARD, N.]

Leonardo Alves de Paulo e Silva - *Implementação da Biblioteca de Comunicação DECK sobre o padrão de protocolo de comunicação em nível de usuário VIA* - 2005 [DIVERIO, Tiaraju Asmuz, STEIN, Benhur de Oliveira, MAILLARD, N.]

Ricardo de Gasperi Presotto - *Otimizações para a multiplicação Vetor-Descritor através do algoritmo Slice* - 2006 [FERNANDES, Paulo Henrique Lemelle, ROSE, César de, CLAUDIO, Dalcidio Moraes, MAILLARD, N.]

Guilherme Galante - *Métodos Multigrid paralelos em malhas não estruturadas aplicados à simulação de dinâmica de fluídos computacional e transferência de calor* - 2006 [NAVAUX, Philippe Olivier Alexandre, CHARÃO, Andrea Schwertner, MAILLARD, N.]

Everton Hermann - *Dinamismo de Servidores de Dados no Sistema de Arquivos dNFSp* - 2006 [PASIN, Marcelo, Mattoso, M. L., MAILLARD, N.]

Fabio Massaaki Katayama - *O problema da troca de mensagens de diferentes tamanhos em redes multi-aglomerados* - 2006 [Song, S. W., Goldman, A., MAILLARD, N.]

Lucas Janssen Balso - *Predição de Desempenho de Aplicações Paralelas para Máquinas Agregadas Utilizando Modelos Estocásticos* - 2006 [FERNANDES, L.

G., FERNANDES, Paulo Henrique Lemelle, Dotti, F. L., MAILLARD, N.]

Jeysonn Isaac Balbinot - *Projeto de um Serviço Configurável de Detecção de Defeitos* - 2007 [WEBER, T. S., NUNES, R. C., MAILLARD, N.]

Mauro Strelow Storch - *Uma arquitetura para gerência de rede de máquinas virtuais com ênfase na emulação de sistemas distribuídos* - 2008 [FERNANDES, L. G., MAILLARD, N.]

Gustavo Cestari Frainer - *Espaço pervasivo de arquivos: habilitando acesso adaptativo e consciente da aplicação a arquivo em ambiente pervasivo* - 2008 [WEBER, T. S., BARBOSA, J. L. V., MAILLARD, N.]

Doutorado

Carlos Amaral Holbig - *Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas* - 2005 [NAVAUX, Philippe, DIMURO, Graçaliz Pereira, REISER, Renata Hax Sander, MAILLARD, N.]