Monografia - Geração Automática de Metadados para Publicações Eletrônicas

José David Fernández Curado
Orientadora: Renata Wassermann

12 de dezembro de 2010

Sumário

I Parte Objetiva **2**

1 Introdução **2**
1.1 Motivação e Objetivos 2
1.2 Projeto Euclid 2
1.3 Problemas 3
1.4 Estrutura deste Trabalho 3

2 Conceitos e técnicas estudadas **5**
2.1 Metadados 5
2.2 Geração Automática de Metadados 5
2.2.1 Raciocínio baseado em casos 6
2.2.2 Raciocínio baseado em regras 7
2.2.3 Processamento de Linguagem Natural 8
2.2.4 Expressões Regulares e Linguagens formais 10

3 Tecnologias Estudadas **11**
3.1 XML 11
3.2 JavaCC 12

4 O gerador automático **14**
4.1 Parsing 14
4.2 Análise e Busca 15
4.3 Geração do Arquivo 15
4.4 Continuação do Desenvolvimento 15

5 Conclusões **16**
5.1 Resultados 16
5.2 Próximos passos do desenvolvimento 16

6 Bibliografia **17**

II Parte Subjetiva **18**

7 Desafios e Frustrações **18**

8 Disciplinas relevantes **19**

9 Observações **21**

10 Continuação **22**

Parte I
Parte Objetiva

1 Introdução

1.1 Motivação e Objetivos

A maior motivação deste trabalho está na produção de uma ferramenta útil para o CLE [1] de modo que este possa integrar sua revista, a CLE "e-Prints"[2], ao Projeto Euclid [3]. O CLE pretende deixá-la disponível através do site do projeto e, para tanto, precisa se responsabilizar por gerar alguns arquivos que ajudam o site a funcionar.

Um desafio técnico para fazer essa integração é que o Projeto Euclid exige que as revistas enviem, junto de cada edição, um arquivo contendo algumas informações sobre a mesma, em um formato específico, para que seu site possa disponibilizá-las na página da publicação e para acelerar seus mecanismos de busca.

O objetivo deste trabalho é fornecer uma ferramenta que possa fazer a extração automática das informações que são necessárias para construir o arquivo exigido pelo Projeto Euclid, seguindo o padrão definido por ele.

As restrições desse arquivo são, simplesmente, restrições de um DTD[4] para um arquivo XML[5], mas esse arquivo deve refletir os dados referentes à edição enviada, como, por exemplo, o nome dos editores e, para cada artigo da edição, abstract e autores.

1.2 Projeto Euclid

O Projeto Euclid considera como sua missão melhorar a comunicação entre estudiosos das áreas de matemática e estatística. Seu objetivo mais básico era suprir as necessidades de revistas pequenas e de baixo custo. O que o projeto acaba oferecendo é uma plataforma avançada de publicação online com funcionalidades de busca textual, geração de links para referência, que segue os padrões da "Open Archives Initiative" e fornece serviço de armazenamento de informação a longo prazo.

Para poder fornecer um serviço consistente e confiável, o Projeto Euclid exige que, junto com cada edição da revista, seja submetido um arquivo contendo informações da edição. Como vários grupos se utilizam da plataforma oferecida pelo projeto, é necessário que informações como autores, editores e responsáveis sejam enviadas de forma estruturada seguindo um padrão acordado entre a revista e o projeto.

[1] Centro de Lógica e Epistemologia da Unicamp
[2] http://www.cle.unicamp.br/e-prints/articles.html
[3] http://projecteuclid.org/
[4] Document Type Definition
[5] eXtensible Markup Language

O padrão de metadados do Projeto Euclid é simplesmente um arquivo XML seguindo o DTD "euclid_issue.dtd". Esse padrão contém, entre outras coisas, rótulos representando dados de contato, data de publicação, autores, editores e abstract.

1.3 Problemas

Os principais problemas para este projeto foram:

- **Levantar especificações completas da ferramenta a ser criada:** O primeiro problema, e de mais difícil resolução, foi levantar especificações suficientes para a ferramenta. Esta podia ser feitas de várias formas e com diferentes possibilidades de restrições ou features. Definir um escopo factível para a ferramenta demorou mais tempo do que o esperado.
- **Levantar uma estrutura recorrente nos artigos:** Um dos grandes problemas para encontrar as informações nos artigos é que estes estão escritos de forma natural, sem obrigatoriedade do uso da estrutura do LaTeX. Apesar de serem artigos científicos, não existe uma grande preocupação com o código escrito em LaTeX, apenas com a linguagem formal e algumas regras gerais, portanto é necessário um trabalho para entender como os dados aparecem no texto.
- **Entender a estrutura do DTD do Projeto Euclid:** O XML precisa seguir a estrutura do projeto, mas também precisa implementar a semântica desejada, utilizando as tags corretas para os dados levantados.
- **Gerar código XML válido:** O documento XML gerado pela ferramenta deve ser válido de acordo com as normas descritas no arquivo "euclid_issue.dtd". Isso exige algum estudo de validação de XML com relação a um DTD.

1.4 Estrutura deste Trabalho

O trabalho foi dividido em duas partes:

1. Técnicas e conceitos de extração de Metadados
2. Proposição de um algoritmo real de extração

Os capítulos, divididos segundo o objetivo, são:

1. Objetivo e os problemas a serem resolvidos no resto do texto
2. Conceitos básicos envolvidos no problema e técnicas utilizadas para resolução deste
3. Estrutura e as tecnologias usadas na ferramenta

4. O Gerador de metadados
5. Conclusões
6. Referencias bibliográficas

2 Conceitos e técnicas estudadas

Explicaremos agora os conceitos envolvidos neste trabalho (Metadados) e técnicas comuns para extração.

2.1 Metadados

Metadados são informalmente definidos como dados sobre dados e são, também, chamados de dicionário de dados ou metainformção. Na prática, um item de metadado especifica alguma coisa sobre um dado. Podemos também defini-los como uma abstração de dados ou como dados de mais alto nível sobre dados de mais baixo nível. Podem fornecer informação como:

- data de criação
- autor
- editor
- padrões
- identificadores

Por exemplo: arquivos de imagem, em um computador, fornecem informações sobre a dimensão da imagem, resolução e padrão de cores. Outros tipos de documentos podem conter outros tipos de informações.

Metadados facilitam o entendimento e a estruturação de representações dos dados em si. Podemos, então, criar ferramentas para trabalhar com essa representação mais formal do dado, tendo, inclusive, uma semântica para os dados. Como a estrutura é normalmente definida a priori, temos uma semântica associada à estrutura dos dados e isso permite criar sistemas que usem essa semântica para fazer buscas.

Esse não é um conceito novo e já tem sido amplamente usado por bibliotecas há muito tempo na forma de catálogos de livros.

São geralmente representados através de arquivos XML, pois sua estrutura bem definida é perfeita para estruturar os metadados de forma simples e sucinta.

2.2 Geração Automática de Metadados

Muito já se estudou sobre geração automática de metadados e técnicas de extração de metadados de textos. Essa área é importante para a web, pois o uso de metadados facilita a criação de algoritmos de buscas eficientes.

Dado que se dispõe de metadados com um significado bem definido, é possível a criação de algoritmos de buscas semânticas, em oposição a algoritmos de busca textual.

Descreveremos agora algumas das técnicas mais utilizadas para a tarefa.

2.2.1 Raciocínio baseado em casos

Raciocínio baseado em casos [1] é uma técnica de Inteligência Artificial para resolução de problemas. É uma técnica de raciocínio e aprendizado que se baseia em soluções conhecidas para problemas semelhantes para construir a solução de um problema proposto.

É diferente de outras técnicas de Inteligência Artificial, pois não depende de conhecimentos a priori sobre o domínio e nem de regras que os problemas sigam, e, sim, de soluções reais conhecidas. Também é diferente por ser uma técnica de aprendizado continuo, já que, cada vez que um novo problema é resolvido, sua solução é guardada para uso futuro.

Formalmente, esse tipo de sistema tenta imitar o método de resolução de muitos especialistas humanos que utilizam conhecimentos sobre problemas resolvidos para resolver problemas atuais em sua área.

Um exemplo de uso de sistemas baseados em casos é o sistema de perguntas e respostas proposto em [4].

Esses sistemas são divididos em quatro passos:

1. Retrieve: busca casos similares, onde um caso é o problema, sua solução e como a solução foi obtida.
2. Reuse: reusa as informações sobre a solução dos casos recuperados.
3. Revise: revisa a solução proposta.
4. Retain: guarda as partes da experiência que poderão ser úteis no futuro.

As áreas de problemas em Raciocínio baseado em casos, de acordo com [1], são:

1. Representação de Conhecimento: Especificamente a representação do conhecimento sobre os casos e suas resoluções.
2. Métodos de "Retrieve": Em particular métodos de buscas de casos semelhantes.
3. Métodos de "Reuse": Como reusar as soluções conhecidas?
4. Métodos de "Revise": Normalmente é um passo fora do sistema e envolve determinação de erros.
5. Métodos de "Retain": Como extrair novos casos e como indexar essa informação para uso futuro.

No contexto de extração de Metadados, o raciocínio baseado em casos foi utilizado em [5]. O sistema proposto no artigo utiliza regras para agrupar documentos semelhantes e então utiliza "Raciocínio baseado em casos" diretamente para extrair os metadados.

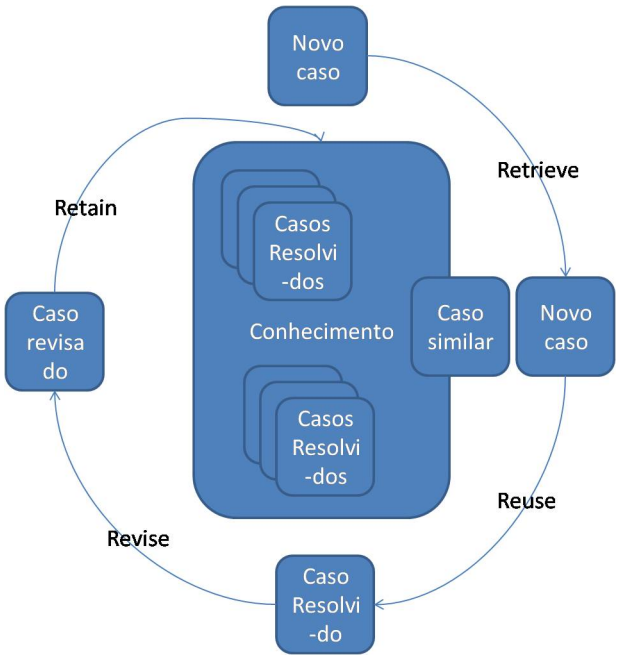

Figura 1: Passos do Raciocinio baseado em casos

No algoritmo proposto a própria representação da solução do problema define marcadores delimitando os metadados, transformando o problema de extração num problema simples de busca de marcas distintas na solução gerada.

Os passos do algoritmo são:

1. Agrupamento de documentos semelhantes (baseado em regras)
2. Análise e marcação do documento (baseado em casos)
3. Extração dos metadados
4. Verificação e correção dos metadados extraídos

Algoritmos baseados em casos têm a grande vantagem de se aperfeiçoar com o tempo, já que aprende com cada novo caso apresentado, mas têm uma grande limitação quanto à diversidade permitida para a estrutura dos casos. Esses algoritmos perdem muita precisão quando são fornecidos casos razoavelmente diferentes entre si.

2.2.2 Raciocínio baseado em regras

Raciocínio baseado em regras é outra técnica de Inteligência Artificial que se utiliza de um conjunto de regras e um mecanismo de inferência para chegar a conclusões sobre um conjunto de premissas.

Normalmente são sistemas que permitem definição de uma base de conhecimento do tipo "If A then B else C", onde A é uma condição e B e C são ações que o sistema pode executar. É uma técnica muito utilizada no desenvolvimento de sistemas especialistas.

No contexto de metadados, o sistema proposto em [3] utiliza regras extraídas de bancos de dados, e de propriedades ortográficas, para definir características de uma palavra. Essa classificação é, então, usada no algoritmo para classificar linhas em separado e em contexto para, por fim, conseguir extrair os metadados que estejam na linha.

1. Extração de características das palavras (baseado em regras)
2. Classificação Independente das linhas
3. Classificação Contextual das linhas
4. Quebra das linhas e extração dos metadados

Os algoritmos são normalmente simples de implementar, pois não é necessário nenhum tipo de treinamento para o algoritmo, mas são naturalmente restritos aos problemas definidos pelas regras.

Como o sistema é definido pelas regras, é necessário de um conjunto completo de regras para cada domínio de problemas. Regras que precisam ser bem especificadas, gerando outro problema que é a especificação das regras.

2.2.3 Processamento de Linguagem Natural

Processamento de Linguagem Natural é uma área de Inteligência Artificial que estuda técnicas de geração e compreensão automática de línguas humanas. Métodos de PLN permitem que humanos interajam mais naturalmente com máquinas, eliminando a necessidade de formas artificiais de interação homem-máquina.

Esse estudo tem um grande caráter interdisciplinar envolvendo as áreas de "Inteligência Artificial", "Teoria da Computação", "Compiladores", "Lingüística Computacional" e outras [2].

Nosso interesse em PLN, para extração automática de metadados, é a criação de um sistema capaz de processar documentos e extrair informações relevantes.

O processamento automático de linguagens, normalmente, segue as seguintes etapas [2]:

1. **Análise Morfológica:** Identificação e classificação de palavras ou expressões em uma sentença.

   No contexto, o analisador deve inferir o tipo da palavra (substantivo, adjetivo, ...) e considerar regras de flexão, como plural e flexão de verbos.

   Esse passo é fundamental para a compreensão da frase, pois, para extrair o significado da frase, é necessário entender todos os vocábulos componentes.

   Um reconhecedor possível para essa fase é o Autômato Finito.

2. **Análise Sintática:** Aplicação de conhecimentos de gramática da lingua.

   O objetivo do analisador sintático é definir as relações entre as palavras. Para tanto ele se utiliza do processamento feito na análise morfológica e procura construir uma árvore de derivação para a sentença seguindo regras de construções definidas pela gramática da língua. Dentre essas regras: concordância e regência nominal e verbal.

   Neste ponto, o maior problema é a desambiguação de frases.

   Para esta fase, técnicas muito utilizadas são:

   - Gramáticas Regulares
   - Gramáticas Livres de Contexto
   - Gramáticas Sensíveis ao Contexto

3. **Análise Semântica:** Definição do significado.

   O problema evidente desta fase é definir o significado das palavras na frase, considerando os possíveis usos de uma mesma palavra. Esta fase define significado considerando componentes das palavras (mercado, hipermercado), ambiguidades de palavras ("tomar um suco", "tomar um banho") e diferenças de sentido ("cachorro", "meu cachorro").

   A semântica também busca uma representação do sentido, que pode ser feito através de decomposição de unidades léxicas, ou através do uso de redes semânticas.

   O resultado é a associação de significado à árvore de derivação da análise sintática.

4. **Pragmática:** Processamento do todo

   Obviamente, o significado completo normalmente está no texto, e não apenas nas partes, portanto a análise final é a análise contextual das frases.

   Nesta etapa, o processo contextualiza as frases para conseguir extrair figuras de linguagem. Ao fim desta etapa espera-se poder perceber situações como sarcasmo e hipérboles.

No contexto de Geração Automática de Metadados, o sistema proposto em [6] utiliza PLN para extrair preferências de usuários através de emails trocados com seus analistas financeiros.

O algoritmo segue os seguintes passos:

1. Identificação de orações
2. Análise sintática
3. Análise morfológica
4. Identificação de conceitos de várias palavras
5. Categorização de conceitos

6. Metadados implícitos

7. Extração de metadados

Notamos que os passos $1 - 4$ são simplesmente passos do processamento de linguagem natural, já os passos $5 - 7$ são os passos reais de extração de metadados.

O sistema é bem completo, mas esbarramos em limitações de computação para PLN, por exemplo, os problemas associados à Análise sintática, como a falta de um método eficiente e forte o suficiente para compreender a gramática de linguagens naturais.

A ausência de um meio de implementar as gramáticas humanas, com todos os seus detalhes, de forma computacionalmente viável, acaba forçando algoritmos incompletos.

Temos, também, a especificidade do sistema, já que um sistema de PLN é específico para uma única língua.

2.2.4 Expressões Regulares e Linguagens formais

A Área de Computação que se refere a Expressões Regulares, Autômatos, e Linguagens formais, de forma mais genérica, é amplamente utilizada no processamento de textos e parsing dentro dos algoritmos de Extração de metadados.

Observamos forte uso de gramáticas de vários tipos da hierarquia de Chomsky em PLN, mas também no parsing de textos em outros sistemas.

Muitos dos algoritmos estudados utilizam parsers e analisadores para linguagens para extrair informações necessárias para executar seus passos.

Duas áreas bem distintas de aplicação desses conhecimentos no projeto foram:

1. parser: criação de uma ferramenta para processar o texto, em seu formato mais desestruturado, gerando uma estrutura lógica que possa ser usada em um processo.

   Utilizamos JavaCC para criar uma gramática para arquivos TEX.

2. extração: extração de trechos de textos, dadas expressões regulares com objetivos mais simples, em seções especificas do texto processado.

3 Tecnologias Estudadas

3.1 XML

XML (eXtensible Markup Language) é um conjunto de regras, derivado do SGML (Standard Generalized Markup Language), para criar formas de representar dados estruturados. Foi definido e é mantido pelo W3C (World Wide Web Consortium). Algumas linguagens definidas com base em XML incluem XHTML e RDF.

Entre as construções permitidas para o XML, como especificado pela W3C, encontram-se, pelo menos:

1. **Tags:** O XML é basicamente definido por suas tags. É onde as informações estão, normalmente, armazenadas.

   Existem três tipos de tags:

   (a) **start-tag**
   São da forma "<title>".

   (b) **end-tag**
   São da forma "</title>".

   (c) **empty-tag**
   São da forma "<title />".

2. **Elemento:** É um componente lógico do documento que consiste, ou de uma empty-tag, ou de uma seqüência start-tag, conteúdo, end-tag, onde conteúdo pode ser uma sequêcia de caracteres ou outros elementos.

   Exemplos:

   - <Greeting>Hello, World!</Greeting>
   - <line-break />

3. **Attribute:** É um par de valores que podem existir numa start-tag ou empty-tag.

   Exemplos:

   - <Greeting lang="en_us">Hello, World!</Greeting>
   -

Um exemplo de um arquivo XML é:

```
1 <?xml-stylesheet type="text/dtd" href="euclid_issue.dtd"?>
2 <euclid_issue>
3   <header>...</header>
4   <issue>
5     <issue_data>...</issue_data>
6     <record type="frontmatter" lang="EN">
7       <title>Matrix - A Computer generated world</title>
```

```
 8        <author>
 9          <name>
10            <given>Anderson</given>
11          </name>
12        </author>
13        <abstract>
14          <p>
15          The Matrix is a Computer generated World to trap humans and
                turn them into...
16          </p>
17        </abstract>
18      </record>
19    </issue>
20 </euclid_issue>
```

É a linguagem mais utilizada para representar metadados.

3.2 JavaCC

Java Compiler Compiler é uma ferramenta para gerar parsers em Java, a partir de uma gramática. Foi usado neste projeto para definir o parser incompleto de TEX.

É semelhante em funcionalidade ao "yacc", pois gera um parser a partir de uma gramática formal. Em particular, o javacc gera parser top-down, limitado a parsers para a classe de gramáticas $LL(k)$ e sem recursão à esquerda.

Um exemplo de código usado para definir a gramática é:

```
1 Document Document() :
2 {
3     Document document = new Document();
4     Element e;
5 }
6 {
7     (
8         "\\begin{document}"
9         (
10            e = Element()
11            {
12                document.getCorpo().add(e);
13            }
14        )*
15        "\\end{document}"
16    )
17    {
18        return document;
19    }
20 }
```

No código acima, definimos um não-terminal "Document" que poderá ser usado para definir novas regras, no estilo "EBNF".

Mais profundamente, o código acima também especifica que, para que um Document seja derivável, precisamos que o input comece com "\begin{document}",

termine com "\end{document}" e contenha zero ou mais elementos deriváveis pelo não-terminal Element. Ainda mais, cada Element encontrado é adicionado ao corpo do Document.

Ao final da execução da regra, um Document é devolvido para a regra que o derivou.

4 O gerador automático

Propomos um algoritmo que recebe os arquivos TEX referentes aos artigos, e gera o arquivo de metadados correspondente. O algoritmo segue três passos bem definidos:

1. Parsing dos arquivos TEX
2. Análise da Estrutura Abstrata do TEX
3. Geração do arquivo de metadados

4.1 Parsing

O primeiro passo do algoritmo é realizar um parsing do arquivo TEX e gerar uma estrutura abstrata em memória que pode ser processada diretamente.

A estrutura foi desenvolvida para conter informações suficientes para que os dados extraídos sejam de boa qualidade e possam refletir, de forma eficaz, o conteúdo de interesse do arquivo TEX .

Para executar essa etapa, desenvolvemos um parser incompleto de TEX, que é capaz de extrair, corretamente, pelo menos a estrutura básica do arquivo. Consideramos que o preâmbulo e o começo do documento contêm a maior parte dos metadados desejados.

O modelo que utilizamos para TEX , implementado no pacote "tex.base" é:

1. **Tex:** É a classe base que representa o documento TEX.

   membros: Um "Preamble" e um "Document".

2. **Preamble:** É o preâmbulo do documento TEX. Genericamente: tudo que está entre "\documentclass... " (incluso) e "\begin{document}".

   Espero que, nesta parte do documento, estejam especificadas definições a serem usadas no documento em si. Entre as definições esperadas, esperamos que esteja definido:

   - title: para a extração do titulo
   - author: para a extração de autores
   - institute (possivelmente): para extrair metadados referentes aos institutos que o autor é afiliado.

   membros: Lista de Definition

3. **Definition:** São comandos TEX da forma "\name{env1}[options]{env2}".

   membros: Lista de Strings (opções) e Lista de Enviroment

4. **Enviroment:** Conceito genérico para um bloco onde podem aparecer os "Element" do TEX

   São da forma $\{Element*\}$.

   membros: lista de "Element"

5. **Element:** Enviroment ou Definition ou Text.

   É uma interface para generalizar chamadas a elementos básicos de um TEX, ou seja, coisas que podem aparecer várias vezes.

6. **Document:** é o documento em si.

   Definimos como documento tudo aquilo que está entre "\begin{document}" e "\end{document}".

   membros: Lista de "Element" e opções.

7. **Text:** texto puro, diretamente processável.

   membros: Lista de Strings

4.2 Análise e Busca

A extração de metadados é feita com base na estrutura do arquivo TEX e com expressões regulares.

Consideramos que o autor irá usar os comandos TEX para especificar título, autor e outras informações.

Alguns metadados não estão presentes nos arquivos em si. Estes serão fornecidos de outra forma, provavelmente como uma configuração do programa.

4.3 Geração do Arquivo

Várias ferramentas foram estudadas para a geração do XML.

A escolha inicial era gerar o XML de uma forma "hard coded", baseado na versão atual do DTD "euclid_issue.dtd" e estender, em uma versão futura da ferramenta, para geração de XML dado um DTD qualquer.

Acreditamos que esta seja a melhor opção, mas já foram estudadas algumas alternativas de gerar o arquivo de forma mais independente.

4.4 Continuação do Desenvolvimento

Seria interessante implementar outros algoritmos de extração de metadados, para complementar os dados extraídos.

- Existem ótimos algoritmos de extração de palavras-chave que podem beneficiar os metadados referentes aos artigos
- Os metadados extraídos podem estar incompletos, existem possibilidades de estender o programa para utilizar mais estratégias ou estratégias melhores

5 Conclusões

5.1 Resultados

Apesar da aparente simplicidade da definição da ferramenta, e da estrutura definida para a linguagem TEX , não foi possível completar o seu desenvolvimento. Problemas com o escopo da ferramenta fizeram com que a fase de programação demorasse muito para começar.

Ao final do desenvolvimento, temos apenas uma ferramenta incompleta. O software ainda não gera os arquivos desejados, apesar de já conter a estrutura básica para tanto. Falta desenvolver:

1. Módulo de geração do arquivo XML
2. Código de busca de informações
3. Melhorar o parser, que ainda tem muitas limitações

5.2 Próximos passos do desenvolvimento

Após o desenvolvimento das características finais da ferramenta, indicadas na seção anterior, ainda há o que melhorar no projeto:

1. **Busca de metadados que não estão presentes nos documentos processados**, mas que existem no arquivo compilado.

   Apesar de podermos encontrar muitos metadados interessantes, de forma fácil, usando a estrutura do TEX, temos o problema de não termos acesso fácil a estrutura do arquivo pdf final.

   Ao compilar o TEX, em um pdf, temos outras características que não aparecem no texto puro, como número de páginas.

2. **Determinação de língua utilizada nos abstract**

   A ferramenta deveria ser capaz de determinar a língua na qual o abstract foi escrito, visto que isso faz parte da especificação do "euclid_issue.dtd". Se não temos informações de idioma, supõe-se o inglês, que pode não ser verdade.

3. **Determinação correta de nomes, separando nome e sobrenome**

   Apesar de podermos simplificar a extração de nomes, temos o problema de extrair a estrutura do nome, isto é, dado que podemos ler "José David Fernández Curado", gostaríamos de determinar que "José David" é o nome e "Fernández Curado" é o sobrenome.

6 Bibliografia

Referências

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, 7(1):39–59, 1994.

[2] Fabio Abreu Dias de Oliveira. Processamento de linguagem natural: princípios básicos e a implementação de um analisador sintático de sentenças da língua portuguesa. Disponível em: http://www.inf.ufrgs.br/gppd/disc/cmp135/trabs/992/Parser/parser.html, ultimo acesso: 12/12/2010.

[3] Hui Han C. Lee Giles, Eren Manavoglu, Hongyuan Zha, Zhenyue Zhang, and Edward A. Fox. Automatic document metadata extraction using support vector machines. In *JCDL '03: Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 37–48, 2003.

[4] Peng Han, Rui-Min Shen, Fan Yang, and Qiang Yang. The application of case based reasoning on q&a system. In Bob McKay and John Slaney, editors, *AI 2002: Advances in Artificial Intelligence*, volume 2557 of *Lecture Notes in Computer Science*, pages 704–713. Springer Berlin / Heidelberg, 2002.

[5] Krisda Khankasikam. A hybrid case-based and rule-based for metadata extraction on heterogeneous thai documents. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, 2010.

[6] Woojin Paik, Sibel Yilmazel, Eric Brown, Maryjane Poulin, Stephane Dubon, and Christophe Amice. Applying natural language processing (nlp) based metadata extraction to automatically acquire user preferences. In *Proceedings of the 1st international conference on Knowledge capture*, K-CAP '01, pages 116–122, New York, NY, USA, 2001. ACM.

Parte II
Parte Subjetiva

7 Desafios e Frustrações

Ao longo deste projeto, me deparei com várias situações de desenvolvimento real de software, que já havia ouvido falar em outras disciplinas, como:

1. Dificuldade de fechar uma especificação para as funcionalidades necessárias para o sistema

   Demorei muito para fechar detalhes suficientes da especificação a fim de me sentir confortável desenvolvendo algumas partes do trabalho.

   Apesar de ser um projeto aparentemente simples, foi necessário definir algumas restrições como:

   - detecção de linguagem do abstract (definimos que será sempre a mesma linguagem)
   - geração do XML flexível ou "hard-coded" (definimos que será hard-coded com a possibilidade de mudar esse aspecto no futuro)
   - questões relacionadas à definição de metadados necessários, mas que, obviamente, não poderiam ser extraídos do textos, como integrantes fixos da revista, por exemplo editores.

2. Dificuldade de aprender as ferramentas disponíveis.

   Apesar de já ter utilizado muitas das ferramentas, problemas com relação a escolha de ferramenta para geração de XML foram frequentes, e problemas com relação a métodos disponíveis de parsing de DTD.

3. Dificuldade de encontrar padrões bem-estabelecidos para determinadas partes do software.

   Apesar de estarmos utilizando elementos bem comuns na computação como TEX e XML, não encontramos ferramentas conhecidas e testadas de parsing de TEX (apenas parsing, sem geração de pdf) e de parsing de DTD, para gerar XML a partir deste DTD.

   Para gerar o XML, escolhemos, mas ainda não implementamos, um método hard-coded que não leva em consideração o DTD.

8 Disciplinas relevantes

Entre os conhecimentos mais usados podemos citar:

1. **Laboratório de Programação:** JavaCC e TEX.

   Da disciplina aprendi a utilizar o JavaCC ao fazer um compilador para uma linguagem definida durante a matéria, e aprendi sobre a ferramenta LATEX.

   Foi a disciplina que me deu base para fazer todo o código do parser, tanto do lado do JavaCC quanto o conhecimento necessário de TEX

2. **Linguagens Formais e Autômatos:** Idéia do algoritmo.

   Todo o algoritmo foi pensado em termos de Gramáticas para TEX e as regras de extração, baseadas em Gramáticas Regulares.

   Inicialmente, e antes mesmo, de estudar outras técnicas de extração de metadados, o algoritmo mais obvio nasceu de conhecimentos de Linguagens Formais e Autômatos.

   Visto que eu sabia que ia processar arquivos TEX e que esses deviam corresponder a alguma gramática, a idéia mais obvia para a resolução deste problema é através de Gramáticas.

   Apesar de ter encontrado pessoas que afirmam que TEX não define realmente uma Gramática simples, pois a linguagem permite extensões e criação de novos comandos e estruturas, o desenvolvimento de um parser simples para um subconjunto muito usado dos comandos de TEX é relativamente fácil, apesar de um pouco trabalhoso.

3. **Inteligência Artificial, Sistemas Baseados em Conhecimentos e Redes Neurais (IPN0007):** Bases para o entendimento de muitos dos Algoritmos conhecidos de extração de Metadados.

   Durante o levantamento de informações sobre outras técnicas de extração de Metadados foram muito úteis os conhecimentos da área de I.A. aprendidos nestas 3 disciplinas, pois ajudaram a compreender mais tranquilamente os algoritmos e as classificações dadas a eles.

   Em particular, é um uso direto deste conhecimento aprender sobre os diversos algoritmos fortemente baseados em I.A. e Sistemas Baseados em Conhecimentos.

4. **Conceitos Fundamentais de Linguagem de Programação:** Parsers e JavaCC

   Apesar de aprender a utilizar a ferramenta JavaCC em Laboratório de Programação II, só foi em Conceitos Fundamentais que muitas das questões envolvendo o Parser gerado pelo JavaCC foram resolvidas.

   Questões como limitações do parser e características da linguagem gerada só foram respondidas com conhecimentos de Conceitos e Autômatos.

Vale lembrar, também, que boa parte do entendimento das gramáticas geradas com o JavaCC, e o entendimento das limitações de Processamento de Linguagem Natural, vem das disciplinas de Conceitos e Autômatos.

Questões de eficiência, relacionadas a gramáticas mais poderosas da hierarquia de Chomsky, só são mencionadas nestas matérias, mas pouco estudadas, mas são bem importantes para aprender bem o conceito necessário para estudar essa área.

9 Observações

A aplicação de conceitos estudados foi bem direta, já que eu trabalhei bastante na construção de um parser, o que é uma aplicação direta dos conhecimentos de Linguagens Formais e Autômatos.

Outras aplicações, no entanto foram ainda mais diretas, como conhecimentos de JavaCC adquiridos em Laboratório de Programação e a experiência prática em desenvolver um parser.

Conhecimento prático de desenvolvimento de parsers ajudou muito na execução do trabalho e na idealização do algoritmo. Certamente o projeto poderia ter sido muito mais complicado se eu não tivesse esses conhecimentos a priori.

Notei também a interação forte entre Linguagens Formais e Inteligência Artificial, que nunca tinha sido tão evidente para mim. Muitos algoritmos estudados são de I.A. e usam, de forma completamente transparente, gramáticas e parsers como ferramentas, mas esse fato nunca apareceu de forma tão obvia em aula.

Essa parte de I.A. é uma parte que eu, infelizmente, não estudei em aula, mas acabei estudando durante este projeto.

10 Continuação

Conhecimentos relevantes para continuar a desenvolver-se nessa área são:

1. **Inteligência Artificial:** especificamente a área de processamento de linguagem natural e raciocinadores

   Obviamente I.A. está presente em muitos algoritmos desta área, então são fundamentais sólidos conhecimentos de I.A. nas áreas de extração de informação e raciocinadores.

   PLN é uma área que tem grande potencial para produzir grandes idéias para a área de extração de Metadados, e técnicas baseadas em programação lógica são muito utilizadas nos algoritmos conhecidos.

2. **Sistemas baseados em Conhecimento:** representação de conhecimento

   Alguns algoritmos desta área levam em conta a capacidade de um programa guardar conhecimento e indexar esse conhecimento. Uma das grandes dificuldades de "Raciocínio baseado em casos" é exatamente essa.

   Ser capaz de armazenar conhecimento e acessá-lo torna-se crucial na execução desses algoritmos, e precisa ser muito bem estudado.

3. **Redes Neurais:** Representação de conhecimento e Algoritmos de Aprendizado

   A área de Redes Neurais pode ser muito útil ao entendimento de muitos dos algoritmos propostos, pois fornece uma boa base de conhecimento sobre algoritmos de aprendizado de máquina.

   Muitos dos modelos de R.N.A.s fornecem muitas experiências úteis para este domínio de conhecimento. Em particular conhecimento sobre memória e modelos para a memória.

   R.N.A.s colocam o estudante em contato com outros exemplos de memória como "memória reflexiva" e enriquecem o conhecimento do aluno sobre essa área.

4. **Linguagens Formais e Autômatos e Conceitos Fundamentais de Linguagem de Programação:** Gramáticas

   É absolutamente necessário que alguém que pretenda estudar esses algoritmos tenha um sólido conhecimento nessas áreas.

   Mais especificamente em Linguagens Formais, que formam a base dos algoritmos de Análises Sintáticas e Léxicas e dos parsers necessários em qualquer processamento de texto.