xRot: Um Roteiro para Autoria de Aplicações que Manipulam Documentos XML na Web

Daniel Facciolo Pires

Orientadora: ***Profa. Dra. Maria da Graça Campos Pimentel***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Área: Ciências de Computação e Matemática Computacional.

USP – São Carlos
Maio de 2000

* Trabalho realizado com o apoio FAPESP.

Ao meu pai

que me incentivou

a fazer este trabalho

AGRADECIMENTOS

À minha mãe Sandra, por tudo.

À Profª Graça Pimentel, pela orientação neste trabalho e por viabilizar minha viagem para o Georgia Institute of Tecnology, Atlanta, EUA.

Aos meus irmãos Alex e Milene, pelo companheirismo.

À Melissa, pela compreensão e carinho durante todo o tempo.

Ao Dani, Guto, Carbol, Emílio, Fefe e Carioca, amigos de república.

Aos amigos do Intermídia, pelo convívio diário.

Aos amigos da pós, em especial ao grupo Niburu, pelos momentos de alegria.

Aos funcionários do ICMC, em especial à Beth, Laura, Marília, Adriana e Sandra, pelos favores prestados.

À Profª Renata Pontim, pela atenção prestada na ausência da Profª Graça.

À Fapesp, pelo apoio financeiro.

E a todos que, de alguma forma, tornaram este trabalho possível.

ÍNDICE

ÍNDICE DE FIGURAS

RESUMO

Orientar o desenvolvimento de aplicações para a Web é um desafio para pesquisadores da área de Hipermídia. O trabalho apresentado nesta dissertação tem como objetivo apoiar a construção de aplicações que se preocupam com o intercâmbio de informações através da utilização da especificação XML (Ex*tensible MarkUp Language*). Apresenta-se *xRot*, um roteiro para apoiar as etapas de definição, geração e apresentação de documentos estruturados manipulados por aplicações para a Internet. O roteiro inclui um algoritmo para geração de documentos XML em um ambiente apoiado por servidores de banco de dados e *World Wide Web*. Também é apresentada a *ArqGDE*, uma arquitetura que suporta as aplicações desenvolvidas com o *xRot*. Como estudos de caso da utilização do roteiro, foram desenvolvidas duas aplicações: AulaML e C2000ML.

ABSTRACT

Many efforts in the area of hypermedia are geared towards supporting the development of Web-based applications. This thesis focuses in supporting the development of applications that need to provide for the interchange of documents with the use of XML — *Extensible MarkUp Language*. It presents *xRot*, a set of directions towards guiding the phases of definition, generation and presentation of structured documents manipulated by Internet-based applications. *xRot* includes an algorithm for the generation of XML documents in an environment supported by database and Web servers. *ArgGDE*, an architecture that support applications developed with *xRot*, is also presented. AulaML and C2000ML are applications built as a cases study of the use of *xRot*.

1. INTRODUÇÃO

1.1. Considerações Iniciais

A falta de padrões para a definição de hiperdocumentos de sistemas hipermídia, ao final da década de 1980, era um problema que impossibilitava o intercâmbio desses documentos entre diferentes sistemas. Nesse contexto, o Dexter Reference Model foi proposto como parte de um esforço para criar padrões com o objetivo de permitir a reutilização do conteúdo e da estrutura de hiperdocumentos por sistemas distintos (Halasz, 1994).

O padrão SGML (*Standard Generalised MarkUp Language*) (ISO, 1986), que define a meta-linguagem SGML, foi proposto para solucionar problema equivalente imposto a instituições, pesquisadores e empresas que necessitavam manipular e realizar o intercâmbio de documentos eletrônicos. SGML foi desenvolvido de modo a permitir que os documentos pudessem ser escritos com sua própria gramática e formato, legíveis por seres humanos e manipulados de modo eficiente por aplicações computacionais.

Quando de sua criação, a Web demandava uma linguagem simples e pequena para especificar a estrutura e a apresentação de documentos estáticos, igualmente simples (HTML, 1992). Dessa forma, muito dos recursos associados à utilização de SGML não eram necessários. Definida como uma aplicação de SGML, HTML (*Hypertext MarkUp Language*) é uma linguagem que fez sucesso por sua simplicidade. Entretanto, uma única linguagem não consegue suportar de modo satisfatório as inúmeras aplicações existentes hoje na Web. Assim, ênfase tem sido dada à definição de padrões, recomendações e formatos para a definição, manipulação e intercâmbio de hiperdocumentos estruturados suportados por sistemas hipermídia.

Nesse contexto, o World Wide Web Consortium, *W3C*[1], vem investindo inúmeros esforços na definição de padrões e recomendações com o objetivo de garantir que a Web cresça de modo aberto e padronizado. Um exemplo é a definição de padrões que regulam a formalização da estrutura de documentos, como a XML (*Extensible Markup Language*) (Connolly et al., 1997) e do seu formato de apresentação, como a XSL (*Extensible Style Language*) (XSL, 2000). Especificamente, a XML permite que projetistas de aplicações definam uma estrutura lógica para os documentos manipulados: o objetivo é prover interoperabilidade entre aplicações e, assim, XML pode ser vista como a proposta do W3C de modelo de representação de dados para aplicações da Web (Connolly, 1998).

A necessidade de se criar metodologias que viessem apoiar o desenvolvimento de aplicações hipermídia fez surgir, por exemplo, HDM (*Hypermidia Design Model*), (Garzotto et al., 1993), RMM (*Relationship Management Methodology*), (Isakowitz et al., 1995) e OOHDM (*Object Oriented Hypermedia Design Model*), (Schwabe et al., 1996). Essas metodologias normalmente não suportam diretamente o intercâmbio de documentos ou utilizam padrões e formatos na definição e manipulação dos seus documentos. As metodologias suportam a criação dessas aplicações de modo mais genérico como, por exemplo, apoiando as fases de modelagem de navegação e de interface.

[1] http://www.w3.org

1.2. Motivação

Este trabalho motiva-se em apoiar o desenvolvimento de aplicações para a Web com o objetivo de promover o intercâmbio das informações — provendo a interoperabilidade entre as aplicações — através da utilização do padrão[2] XML. São propostos o roteiro *xRot* e sua arquitetura subjacente, *ArqGDE - Arquitetura para Geração de Documentos Estruturados*. O roteiro apoia as etapas de definição, geração e apresentação de documentos estruturados manipulados por aplicações para a Internet. A *ArqGDE* suporta as aplicações desenvolvidas com o *xRot*.

1.3. Objetivos

Especificamente, os objetivos do trabalho apresentado são:

- A criação de um roteiro para apoiar o desenvolvimento de aplicações para a Web que promovem o intercâmbio de informações, em particular nas etapas de definição, geração e apresentação de documentos estruturados suportados pelas aplicações.
- Explorar, neste contexto, a utilização do padrão XML para prover a interoperabilidade entre as aplicações.
- Realizar estudos de caso do roteiro proposto através da construção de duas aplicações para a Internet: AulaML e C2000ML.

1.4. Estrutura

O Capítulo 2 discute, inicialmente, padrões de hiperdocumentos estruturados na WWW — SGML, HTML e XML — que têm como objetivo aumentar a independência dos sistemas hipertexto ao permitir a formalização dos documentos por eles manipulados. A seguir, esse capítulo discute a utilização dos recursos Java, XSL, SAX (*Simple API for XML*) e DOM (*Document Object Model*) no desenvolvimento de aplicações XML.

O Capítulo 3 discute uma arquitetura típica de aplicações existentes hoje na Web e apresenta a *ArqGDE*, que suporta aplicações geradas com o roteiro *xRot*. Na seqüência, esse capítulo detalha o roteiro *xRot* e discute alguns trabalhos relacionados.

Os Capítulos 4 e 5 apresentam as aplicações AulaML e C2000ML, respectivamente, desenvolvidas com base no roteiro *xRot*. O Capítulo 6 conclui o trabalho discutindo as contribuições e trabalhos futuros.

[2] As especificações produzidas pelo W3C são denominadas recomendações e não padrões. Entretanto, essas especificações se colocam como padrão "de fato" devido a sua forte adoção pela comunidade de usuários da WWW. Assim, as especificações do W3C são referenciadas neste texto como recomendações ou padrões, sem distinção.

2. DOCUMENTOS ESTRUTURADOS

2.1. Considerações Iniciais

Ao final da década de 1980, a falta de padrões para a definição de hiperdocumentos de sistemas hipermídia impossibilitava o intercâmbio desses documentos entre diferentes sistemas. O Dexter Reference Model foi proposto ao final dos anos 80 como parte de um esforço para permitir a reutilização do conteúdo e da estrutura de hiperdocumentos por sistemas distintos (Halasz, 1994). Ainda neste contexto, o padrão SGML (*Standard Generalised MarkUp Language*) (ISO, 1986) é uma meta-linguagem definida com o objetivo de permitir que os documentos pudessem ser escritos com sua própria gramática e formato. O padrão SGML deu origem a outros padrões que surgiriam com a demanda da Web por novos recursos.

Este capítulo segue com a Seção 2.2 com uma introdução ao padrão SGML. Em seguida, na Seção 2.3 são apresentados o padrão HTML, sua importância e quais suas limitações. O capítulo continua com a Seção 2.4 sobre XML (*Extensible MarkUp Language*), que foi utilizado neste trabalho para promover o intercâmbio de informações entre as aplicações que utilizam o *xRot*. Em seguida, na Seção 2.5, as especificações XSL (*Extensible Style Language*) e CSS (*Cascade Style Sheet*) para transformação e apresentação de documentos estruturados na Web, respectivamente, são introduzidas. Finalmente, na Seção 2.6, é realizada uma discussão sobre manipulação e apresentação de documentos XML utilizando os recursos Java, XSL, DOM (*Document Object Model*) e SAX (*Simple API for XML*). Os dois primeiros recursos, incluindo CSS, foram utilizados no trabalho para manipular documentos estruturados e associar informações de apresentação a documentos XML.

2.2. SGML - Standard Generalised MarkUp Language

A idéia de que documentos estruturados armazenados eletronicamente pudessem ser trocados e manipulados de uma maneira eficiente a partir de um padrão deu origem a vários esforços na década de 1960. Em um destes esforços, o GCA (*Graphic Communications Association*) criou o *GenCode*, um gerador de formatos, com o objetivo de desenvolver padrões genéricos para seus clientes que usavam dados com formatos diferentes para que pudessem trocar informações. Em uma outra tentativa, a IBM desenvolveu o GML (*Generalised MarkUp Language*) para resolver problemas de publicações internas. O GML tinha uma sintaxe simples incluindo as *tags* de marcação <> e </> idênticas e conhecidas hoje (sinais "<" e ">" que indicam a presença de um elemento em um documento, por exemplo, `<elemento>`). Este padrão era de grande utilidade para as pessoas que liam ou escreviam um documento, mas pouco útil para as aplicações de computador (Connolly et al., 1997). Na década de 1980, representantes da GenCode e do GML se juntaram para formar, junto ao ANSI (*American National Standards Institute*), um comitê para discutir linguagens computacionais para a produção de texto com o objetivo de padronizar as maneiras de especificar, definir e usar as linguagens de marcação em documentos no ambiente computacional — surgiu assim, o padrão SGML (*Standard Generalised MarkUp Language*) (ISO, 1986).

O padrão SGML define a meta-linguagem SGML, desenvolvida com o objetivo de permitir que os documentos pudessem ser escritos com sua própria gramática e formato, legíveis por humanos e manipulados de modo eficiente por aplicações. Os documentos SGML possuem uma rigorosa descrição de sua estrutura que podem ser analisadas por computadores e entendidas pelo homem (Herwijnen, 1994).

A representação da estrutura de um documento no padrão SGML tem como vantagens:

- **Extensibilidade**: SGML permite que os usuários especifiquem quais elementos e atributos farão parte de seus documentos.
- **Estruturação**: A estrutura de um documento SGML suporta a especificação de estruturas grandes e complexas, necessárias para representar hierarquias orientadas a objeto e esquemas de banco de dados.
- **Formalização**: As especificações são formalizadas em um documento denominado DTD (*Document Type Definition*) de acordo com as regras da meta-linguagem SGML. Aplicações genéricas podem verificar tanto a validade das especificações como dos documentos a eles correspondentes.

O padrão HyTime (*Hypermedia/Time-Based Structuring Language*) (ISO, 1992) é uma meta-linguagem, definida como uma aplicação SGML, criado para suportar hiperdocumentos mais complexos que não podem ser representados pela linguagem SGML. A complexidade dos hiperdocumentos está relacionada ao seu conteúdo e estrutura. O conteúdo relaciona-se com o escalonamento e sincronização de objetos multimídia como áudio e vídeo. A estrutura diz respeito à representação de ligações hipertexto como, por exemplo, trilhas, anotações, elos multidestino e *guided-tours* (Pimentel et al., 1997).

Definida como uma aplicação de SGML, HTML é uma linguagem que faz muito sucesso por sua simplicidade e pela independência de plataforma. Apesar de seus poucos recursos de ligações de hipertexto, se comparados a HyTime, esta linguagem é ainda o principal formato para hiperdocumentos na Web. A seguir, o padrão HTML será abordado.

2.3. HTML - Hypertext MarkUp Language

Em 1990, a Web necessitava de uma linguagem simples, pequena para especificar a estrutura e a apresentação de documentos simples. Assim, os recursos e complexidade associados à utilização de SGML e HyTime não eram necessários.

A partir da versão 2.0 em 1995, HTML tem sido definida como uma aplicação informal de SGML. Informal porque, apesar de existir uma definição formal da linguagem na forma de um DTD SGML, os *browsers* (ferramentas para navegação na Web) não exigem que os documentos sigam a sintaxe definida pelo DTD. A versão mais recente de HTML é também definida por um DTD SGML -- o que significa, ainda, que a sintaxe é fixada por um DTD.

A falta de extensibilidade da linguagem fez com que autores construíssem páginas na Internet sem se preocupar em definir uma estrutura semântica para o conteúdo da mesma. Esta limitação trouxe agravantes como, por exemplo, a não possibilidade de especificar e criar novas *tags* e atributos e, com isso, explicitar o significado da informação, e dificuldade para manipular os documentos. Outras características limitantes de HTML são (Bray et al., 1997a):

- **HTML tem pouca estrutura semântica**: A linguagem HTML representa a informação em termos de seu *layout* e não através do seu significado. Não existe uma maneira de especificar o significado de um item de uma página em particular em um arquivo HTML.
- **HTML fornece uma única visão da informação**: HTML não oferece recursos de, dinamicamente, fazer com que o conteúdo de uma página na Web seja apresentado de formas diferentes. Uma maneira de escrever arquivos HTML que mostram a mesma informação em

diferentes visões é através de DHTML (*Dynamic Hypertext MarkUp Language*) (DHTML, 1998).

Logo, ficou evidente que uma única linguagem não seria suficiente para a especificação e o suporte à variedade de aplicações da Web. Assim, tornou-se necessária a especificação de uma meta-linguagem que provesse tal suporte, o que demandou a criação da XML.

2.4. XML - Extensible MarkUp Language

XML é uma meta-linguagem definida como um subconjunto da meta-linguagem SGML. Ela tem por objetivo fornecer benefícios não existentes em HTML e ser mais fácil de utilizar do que SGML.

HTML define um conjunto de tipos de elementos (por exemplo, H1, IMG e TABLE) que se preocupa com a aparência da informação, tendo como foco as fontes, cores e outros formatos de apresentação. Por outro lado, XML, sendo uma meta-linguagem, não possui tipos de elementos pré-definidos. Com XML, projetistas podem criar seus próprios elementos de acordo com sua aplicação, de modo a refletir a importância do conteúdo e da estrutura da informação.

XML foi definida para que qualquer grupo de trabalho pudesse criar sua linguagem de marcação própria, de modo a satisfazer as necessidades específicas de suas aplicações de modo rápido, eficiente e lógico (Connolly et al., 1997). Premissas que delinearam o projeto de XML incluem (Bray et al., 1997a):

- XML deveria ser usada diretamente sobre a Internet suportando uma grande variedade de aplicações e compatível com SGML
- O projeto de documentos XML deveria ser fácil e rápido de preparar, formal, conciso e com características opcionais mínimas possíveis, sendo claramente legível para o usuário.

Uma das maiores vantagens da XML é que ela permite que o autor crie a linguagem de seus documentos, definindo os elementos, os atributos, e a sua estrutura que farão parte de seus documentos. Estas definições são expressas como declarações em um documento DTD realizadas de acordo com a meta-linguagem XML. Desta forma, o projetista pode definir no DTD quais os elementos e atributos devem aparecer e quais são opcionais na sua linguagem.

2.4.1. Elementos de um Documento DTD

No DTD são declarados a seqüência e o aninhamento que os elementos de um documento devem obedecer, os valores e os tipos de atributos, as entidades que devem ou podem aparecer e serem referenciadas entre outros recursos. Para que um *parser* XML verifique se um documento está correto ou não (*parser* de validação) ele processa inicialmente o seu DTD correspondente, para verificar a estrutura do documento (Walh, 1997) (Bray et al., 1997b).

Um documento em conformidade com um DTD XML possui, não obrigatoriamente:

DTD: A declaração do DTD associado ao documento XML pode ser através de uma referência no documento XML a um documento DTD contendo a especificação (Figura 1-a) ou embutida no próprio documento XML (Figura 1-b).

```
<!DOCTYPE aulaml SYSTEM "aulaml.dtd">
<aluno><nome> Maria </nome>
<nota> 10.0 </nota>
</aluno>
```

```
<!ELEMENT aluno (nome, nota)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT nota (#PCDATA)>
<aluno><nome> Maria </nome>
<nota> 10.0 </nota>
</aluno>
```

Figura 1 - (a)Referência ao documento DTD **(b)Documento DTD embutido no próprio documento**

Elemento: Elementos correspondem aos componentes estruturais da linguagem. Sua ocorrência é delimitada por colchetes angulares "<" e ">", e a definição de seu conteúdo é feita de acordo com o DTD. Se o elemento *element* não é vazio, ele começa com um *<elemen>* e termina com *</elemen>*. Por exemplo, a declaração:

```
<!ELEMENT Aluno (CDATA) >
```

em um DTD permitiria a ocorrência, em um documento, de:

```
<Aluno> Marcos de Almeida </Aluno>.
```

onde CDATA significa que o elemento *ALUNO* pode conter quaisquer tipos alfanuméricos.

Referência a uma entidade: Uma entidade é composta basicamente de elementos e atributos. Para que uma entidade possa referenciar os elementos e atributos de uma outra entidade, ela faz referência ao seu nome. Uma referência a uma entidade é feita começando com um caracter *ampersand* (&) e terminando com um ponto-e-vírgula (;), como em

```
<Estuda> &Disciplina; </Estuda>.
```

Neste caso, *Disciplina* estaria definida no DTD ou algum documento acessível a partir dele, como em:

```
<!ENTITY Disciplina  "Engenharia de Software" >
```

Entidades precisam ter um nome único. Quando for preciso usar o conteúdo de uma entidade em outra, basta referenciá-la pelo seu nome.

Comentários: No espaço entre "<!--" e "-->" pode ser inserido qualquer tipo de caracter menos a cadeia "--". A esses caracteres são denominados comentários, e eles não farão parte do conteúdo do documento DTD. Exemplo: `<!-- Este é um comentário -->`. Um comentário pode ser colocado em qualquer lugar do documento.

Instruções de processamento: Instruções de processamento (IPs) são uma forma alternativa de fornecer informações a uma aplicação. São como comentários, ou seja, não fazem parte do texto do documento. É tarefa do processador XML passar a informação para a aplicação correspondente. Um exemplo para a inclusão de *scripts* escritos na linguagem *php* (Php3, 1999) para processamento de documentos hipertexto seria:

```
<?php
echo "<HTML> Hello World!
"?>.
```

Seções de marcação: Uma seção marcada com *CDATA* instrui o *parser* para ignorar todos os caracteres reservados da linguagem XML daquela seção. Utiliza-se o colchete angular ("<") para analisar uma expressão, por exemplo, `x + y < z + t`, em uma seção *CDATA*, o *parser* considera-o como um sinal de comparação, e não um colchete angular de marcação.

Existem vários *parsers* XML para auxiliar na manipulação de documentos e DTDs XML: na sua verificação (análise léxica) e validação (análise sintática). Estes *parsers* podem ser encontrados gratuitamente na Web, como o SAX (*Simple API for XML*) (Megginson, 1998), que auxilia na implementação de aplicações escritas na linguagem Java.

Especificamente, este trabalho explora o padrão XML na autoria de aplicações para a Web para promover o intercâmbio entre elas. De modo geral, as aplicações que utilizam XML na definição dos documentos estruturados suportados aproveitam-se das seguintes vantagens:

- **Facilidade de intercâmbio**: Não é necessário que aplicações de terceiros tenham conhecimento da estrutura de uma base de dados em particular. É suficiente ter acesso a uma especificação XML que define a sintaxe da informação para poder trocar e transferir informações do seu conteúdo.
- **Processamento customizado**: Aplicações distintas ou módulos de aplicações distribuídas podem apresentar as informações recebidas de acordo com a preferência e/ou necessidade do usuário local.
- **Pesquisa com maior eficiência**: Permite-se que aplicações possam fazer buscas mais eficientes em documentos que estejam formalizados, estruturados, padronizados e/ou hierarquizados.
- **Tráfego reduzido**: O intercâmbio de documentos estruturados permite o processamento local da informação pela aplicação possibilitando uma redução da quantidade de dados transmitidos pela rede.

2.4.2. Outras recomendações associadas a XML

O W3C tem coordenado a promoção de vários outros formatos e padrões para a Web, com o objetivo de suportar o intercâmbio de informações entre aplicações dos mais variados interesses. Nesse sentido, várias recomendações associadas a XML têm sido desenvolvidas. Alguns exemplos importantes são:

- *XSL - Extensible Stylesheet Language*) (XSL, 2000): É uma linguagem para a definição de *Style Sheets*, que são especificações relacionadas ao formato de apresentação de um documento XML. Por exemplo, é possível o associar ao elemento `disciplina` o elemento de apresentação `parágrafo`. Isso permite transformar um documento XML em outro no qual informações referentes à apresentação estão incluídas.
- *XLink - XML Linking Language* (DeRose et al., 2000): Especifica construtores que podem ser inseridos em documentos XML para descrever ligações entre objetos. Objetos podem ser quaisquer porções de dados, e as ligações podem ser internas ou externas aos documentos, uni ou multidirecionais, entre outras características.

- *Xpointer - XML Pointer Language* (DeRose et al., 1999): Xpointer especifica construtores que suportam endereçamento dentro da estrutura interna de documentos XML como, por exemplo, o terceiro parágrafo da segunda seção do quarto capítulo.

Várias outras recomendações estão concluídas ou em fase de elaboração. Percebe-se que, de posse desse tipo de especificações, o processamento e o intercâmbio de informações na Web se dará de maneira muito mais eficiente e poderosa. Assim, cresce a importância do projeto de aplicações em consonância com as recomendações do W3C em geral, e com XML em particular.

2.5. CSS e XSL

Um documento XML não tem recursos para especificar informações a respeito da forma de apresentação de seu conteúdo. Para isso, CSS (*Cascading Style Sheets*) e XSL (*Extensible Style Language*) podem ser utilizadas para permitir que informação de apresentação possa ser associada ao mesmo. XSL, em particular, foi especificado utilizando a própria linguagem XML.

CSS é uma linguagem declarativa (descreve o relacionamento entre variáveis em termos de funções e regras de inferência) proposta em uma recomendação do W3C (*World Wide Web Consortium*) (CSS, 1998). Esta linguagem especifica, na forma de declarações, como associar estilo a elementos de um DTD como, por exemplo, associar informações de tipo e cor de fonte aos elementos de um documento HTML. O efeito do estilo afeta o conteúdo do elemento e também os elementos hierarquicamente abaixo dele. Por exemplo, se o estilo do elemento <BODY> estiver com letra de tamanho 14 e cor preta, todo o conteúdo deste elemento e de todos os outros elementos que aparecerem aninhados a ele, como <H1>, <H2> e <H3>, também apresentarão o mesmo estilo, a não ser que para um destes elementos sejam definidos outros estilos.

XSL também é uma linguagem recomendada pela W3C (XSL, 2000), a qual permite a definição que regras de transformação sejam aplicadas aos elementos XML. Essas transformações podem ou não corresponder à inclusão de estilos de apresentação como, por exemplo, HTML, TeX, RTF e PostScript.

Um arquivo XSL possui uma série de regras, chamadas de *templates*. Um documento XSL como o da Figura 3 é aplicado a um documento XML da Figura 2, tomando-o como valor de entrada. Este é um exemplo de conversão de um documento XML para um documento HTML utilizando as especificações de um documento XSL. O documento HTML (Figura 4) gerado tem o conteúdo do documento XML e o formato de apresentação HTML do documento XSL. O comando:

```
<xsl:template match="/Recipe">
```

indica que a todo elemento <Recipe> encontrado no documento XML, será aplicada a estrutura de apresentação definida no documento XSL da Figura 3. Já o comando:

```
<xsl:value-of select="Qty"/>
```

seleciona o conteúdo do elemento <Qty> do documento XML para compor o documento HTML. Assim, é produzida uma nova estrutura como saída, com informações de apresentação HTML.

```
<?xml version="1.0"?>
<Recipe>
<Item> Lime Gelatin </Item>
<Qty> 500 </Qty>
<Unit> g </Unit>
</Recipe>
```

Figura 2 - XML simplificado para receita de bolo (Johnson, 1999)

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/Recipe">
<HTML>
<HEAD>
<BODY>
<H1>
 <xsl:value-of select="Qty"/>
</H1>
<H2>
<xsl:value-of select="Unit"/>
</H2>
<H3>
<xsl:value-of select="Item/>
</H3>
</BODY>
</HEAD>
</HTML>
</xsl:template>
```

Figura 3 - Arquivo XSL para conversão de um documento XML para HTML (Johnson, 1999)

O resultado final da transformação de um documento XML em documento HTML é mostrado na Figura 4:

```
<HTML>
 <HEAD>
<BODY>
<H1> 500  </H1>
<H2> g </H2>
<H3> Lime Gelatin  </H3>
</BODY>
</HEAD>
</HTML>
```

Figura 4 - Documento HTML resultante (Johnson, 1999)

A Figura 5 ilustra o processamento de um documento XML utilizando um documento XSL com formato de apresentação HTML para a Web. O resultado do processamento é apresentado às aplicações na Internet.

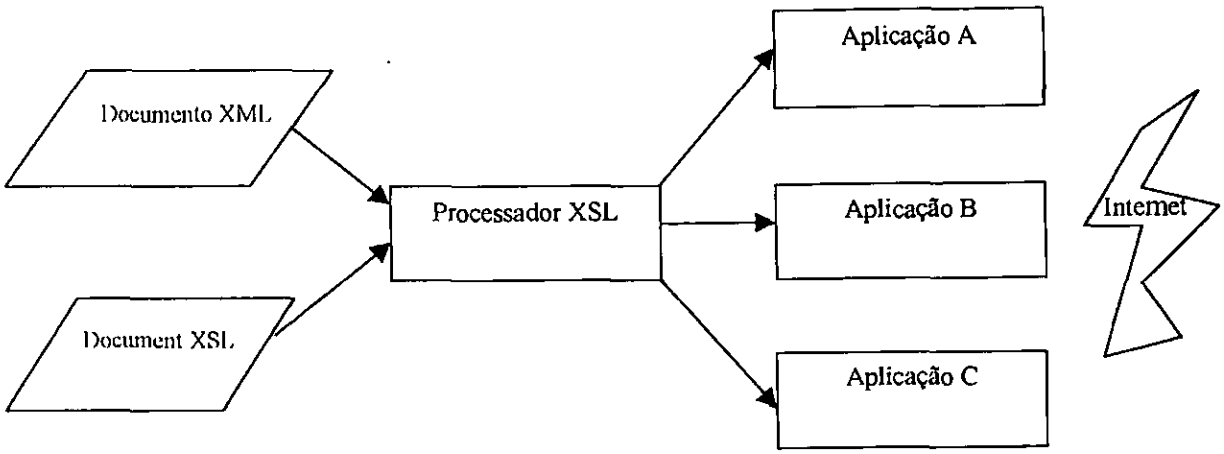

Figura 5 - Processador XSL combinando arquivos XML e XSL (Pimentel, 1999)

Existem várias iniciativas para o desenvolvimento de processadores XSL visando facilitar a manipulação de documentos XML. Entre elas, estão o processador XSL da Microsoft (XSLMic, 1999) e da IBM (XSLIbm, 1999).

No contexto deste trabalho, a utilização da linguagem XSL e de um processador XSL na apresentação dos documentos XML foram motivados por:

- Com XSL as aplicações não precisam processar as informações no servidor. As tarefas de processamento e entrega do documento HTML ao cliente são feitas pelo *browser*.
- As aplicações podem fazer uso de processadores genéricos XSL que processam automaticamente elementos, atributos e conteúdo de um documento XML.
- É permitido às aplicações ordenar e filtrar respostas ao cliente.
- As aplicações podem fazer uso de funções e condições definidas pelo programador.

2.6. Manipulação e Apresentação de Documentos XML com Java

As APIs (*Aplication Programming Interface*) SAX (*Simple API for XML*) e DOM (*Document Object Model*) são recursos que podem ser utilizados por desenvolvedores Java que precisam manipular e apresentar documentos XML suportados por suas aplicações. Esses recursos, comentados a seguir, são os mais explorados na literatura para estes propósitos.

2.6.1. Utilização de SAX e DOM

Uma alternativa para manipular e apresentar documentos XML é utilizando-os como objetos ou tratando-os como uma seqüência de eventos. Isso porque, uma vez que os elementos de um documento estruturado estão aninhados, estes podem ser representados como uma estrutura de árvore (os objetos são os nós da árvore) ou uma seqüência ordenada de eventos. Exemplos: a Figura 7(a) apresenta a hierarquia do documento HTML da Figura 6 em forma de nós de uma estrutura de árvore: o elemento <TBODY> é filho de <TABLE>, o elemento <Over the River, Charlie> é filho do elemento <TD>, que por sua vez, é filho à esquerda do elemento <TR> que, por sua vez, é filho à direita do elemento <TBODY>. Já a Figura 7(b)

apresenta os elementos HTML da Figura 7 como uma seqüência ordenada de eventos: 1) `start table`, 2) `start element:tbody`, 3) `start element:tr`, e assim sucessivamente.

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
 </TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

Figura 6 - Estrutura HTML utilizado no exemplo (DOM, 1998a)

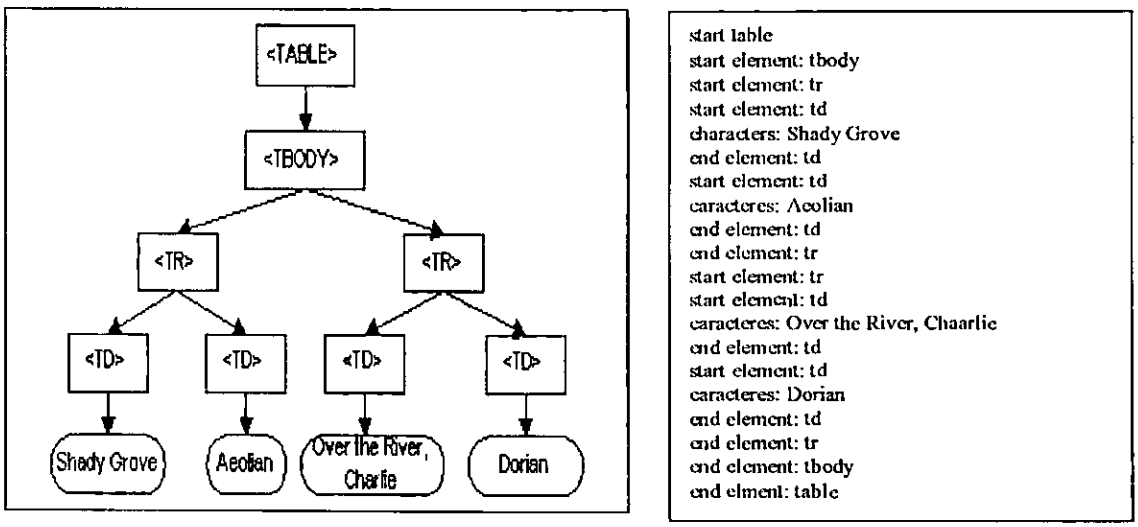

Figura 7 - a) Baseada em árvore (API DOM) (DOM, 1998) **b) Baseada em evento (API SAX)**

A conversão de um documento XML em uma estrutura de dados em árvore ou em uma seqüência ordenada de eventos, como exemplificado anteriormente, pode ser realizada através da linguagem Java. Neste caso, as APIs DOM e SAX oferecem pacotes (classes Java) que auxiliam o programador a manipular documentos XML em sua aplicação.

DOM (DOM, 1998b) auxilia na definição da estrutura dos documentos, representando-os como uma árvore de objetos, e também define funções utilizadas para acessar e manipular os objetos. Com o uso de DOM, programadores podem transformar um documento XML em um documento HTML utilizando as especificações XSL. O objetivo de DOM é fornecer uma interface de programação padrão que possa ser usada em uma grande variedade de ambientes e aplicações. DOM é projetado para ser usado em qualquer linguagem de programação, não sendo especifica para a linguagem Java, e foi especificado em Corba IDL (*Interface Definition Language*).

A API SAX representa um documento XML através da seqüência ordenada de eventos que ocorrem. Os *parsers* SAX são programas que percorrem um documento XML ativando métodos

que tratam de cada caso, como, por exemplo, início, fim e conteúdo de elemento, e início e fim de documento.

As aplicações AulaML e C2000ML que serão apresentadas nesta dissertação utilizam o processador XSL da Microsoft, processador este embutido no *browser* Explorer 5.0. Outros *browsers* existentes, como o Netscape, ainda não interpretam automaticamente documentos XML. Uma outra versão das aplicações AulaML e C2000ML foi construída utilizando os recursos SAX e DOM para a apresentação dos documentos XML suportados pelas aplicações (funcionando com um *plug-in* ao *browser*). Desta forma, as aplicações não ficam dependentes de um *browser* em particular.

2.6.2. Limitações

As limitações da utilização da linguagem XSL são que atualmente, poucos *browsers* possuem um processador já embutido para processar um documento XSL, e nem todas as funções já definidas pelo *draft* do W3C estão presentes nestes processadores XSL como, por exemplo, funções *scripts* e de geração de eventos. Já o recurso SAX apresenta limitações por ter sido desenvolvido apenas para a linguagem Java, enquanto que DOM limita-se pelo uso excessivo de memória.

2.7. Considerações Finais

A linguagem HTML fez muito sucesso por sua simplicidade e pela independência de plataforma, porém não é suficiente para a especificação e o suporte à variedade de aplicações da Web. Assim, tornou-se necessário especificar uma meta-linguagem que provesse tal suporte, o que demandou a criação da meta-linguagem XML. Com XML, o autor de documentos estruturados pode criar sua linguagem de marcação própria, de modo a satisfazer as necessidades específicas de suas aplicações. A criação dessa linguagem de marcação própria é feita através de declarações em um documento DTD XML.

CSS pode ser utilizada para permitir que informações de apresentação sejam associadas a documentos XML. XSL permite que regras de transformação sejam aplicadas aos elementos XML, podendo ou não corresponder à inclusão de estilos de apresentação, por exemplo, HTML, TeX, RTF e PostScript.

O estudo e a exploração da literatura relevante ao contexto de documentos estruturados se fizeram necessários para melhor entendimento e compreensão dos padrões associados à formalização dos mesmos. De modo particular a este trabalho, o padrão XML foi utilizado pelo *xRot* no apoio à construção de aplicações que se preocupam com o intercâmbio de informações.

No contexto deste trabalho as especificações DTD, XML e XSL foram exploradas pela arquitetura *ArgGDE* e utilizadas pelo roteiro *xRot* no apoio a autoria das aplicações para a Internet. Com a geração de documentos XML proposto pelo *xRot*, as aplicações podem fazer uso dos recursos Java, XSL, CSS, e das APIs DOM (Wood et al., 2000) e SAX (Megginson, 1998) para manipulação dos seus documentos. Este trabalho optou pela utilização de Java, XSL e CSS como recursos do roteiro proposto para a manipulação e apresentação de documentos XML. O roteiro *xRot* e a arquitetura *ArgGDE* são apresentados no próximo capítulo.

3. xRot: UM ROTEIRO PARA APOIAR AS ETAPAS DE DEFINIÇÃO, GERAÇÃO E APRESENTAÇÃO DE DOCUMENTOS ESTRUTURADOS MANIPULADOS POR APLICAÇÕES PARA A INTERNET

3.1. Considerações Iniciais

Com o objetivo de apoiar o desenvolvimento de aplicações para a Web que promovem o intercâmbio de informações através da utilização do padrão XML (*Extensible MarkUp Language*), foram definidos neste trabalho o roteiro *xRot* e sua arquitetura subjacente, *ArqGDE – Arquitetura para Geração de Documentos Estruturados*. O roteiro apóia as etapas de definição, geração e apresentação de documentos estruturados manipulados por aplicações para a Internet. A *ArqGDE* suporta as aplicações desenvolvidas com o *xRot*. Como estudos de caso da utilização do roteiro, foram desenvolvidas as aplicações AulaML e C2000ML apresentadas nos próximos capítulos.

A próxima seção discute uma arquitetura típica de aplicações existentes hoje na Web e apresenta a *ArqGDE*. A *ArqGDE* estende a arquitetura típica explorando a utilização de especificações XML (na forma de documentos DTD) e XSL na geração dos documentos por parte das aplicações. O roteiro *xRot* é apresentado na Seção 3.3. O capítulo finaliza discutindo alguns trabalhos relacionados. É importante ressaltar que o roteiro *xRot* e a arquitetura *ArqGDE* foram descritos em (Pimentel et al., 2000a).

3.2. Uma Arquitetura para Suporte à Geração de Documentos Estruturados na WWW

No ambiente da WWW, que obedece a arquitetura cliente-servidor, usuários solicitam, através de *browsers*-cliente, informações disponibilizadas por servidores que suportam o protocolo HTTP[3]. Na maioria das vezes, usuários solicitam informações específicas entre aquelas disponibilizadas no servidor e que não estão disponíveis diretamente como arquivos estáticos estando, por exemplo, armazenadas em banco de dados — uma outra abordagem quanto à solicitação de informações por um usuário seria por documentos estáticos existentes no servidor. No primeiro caso, a informação a ser apresentada ao usuário pode ser gerada dinamicamente por um programa ativado pelo servidor: o servidor repassa ao programa os parâmetros enviados pelo usuário e aguarda a geração, pelo programa, do documento a ser repassado como resposta ao usuário. Neste caso, o servidor e o programa por ele ativado devem se comunicar de modo apropriado, utilizando a especificação CGI[4], para que as informações enviadas pelo usuário sejam repassadas ao programa e o documento resultante, por sua vez, repassado ao usuário. Já no segundo caso, servir a informação (arquivos estáticos) existente é responsabilidade única do servidor[5].

A Figura 8 ilustra o primeiro caso, que corresponde a uma arquitetura típica de aplicações na Internet. Nessa figura:

- **(A)** ilustra a requisição enviada pelo *browser* (cliente) ao servidor WWW;

[3] Hypertext Transfer Protocol (HTTP, 1999).
[4] Common Gateway Interface (CGI, 1999).
[5] Por implmentar o protocolo HTTP de modo geral, e a função definida pelo método GET em particular

- **(B)** ilustra a ativação, pelo servidor, de programas de acordo com a interface CGI;
- **(C)** representa o fato de que estes programas muitas vezes realizam consultas a banco de dados associados à aplicação;
- **(D)** ilustra a passagem ao servidor dos documentos HTML gerados dinamicamente pelos programas;
- **(E)** representa o envio dos documentos pelo servidor ao *browser* cliente.

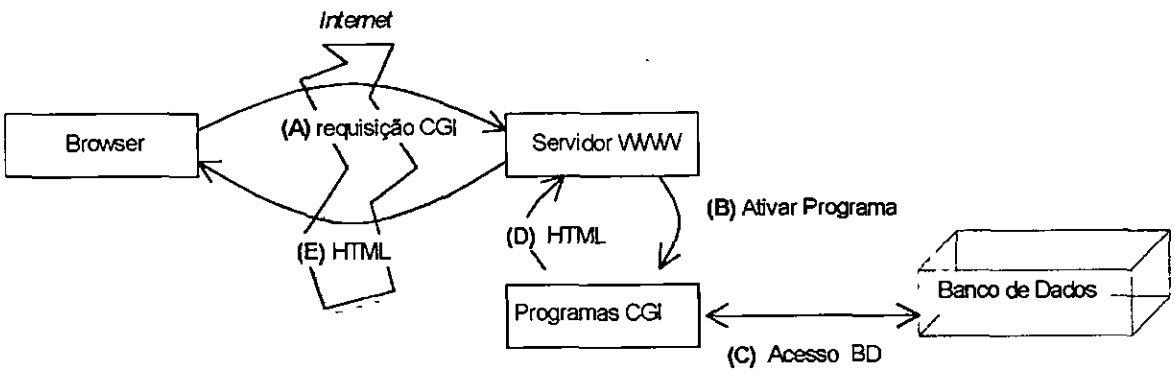

Figura 8 - Arquitetura típica de aplicações na Internet (Pimentel et al., 2000a).

Já a Figura 9 apresenta a *Arquitetura para Geração de Documentos Estruturados – ArqGDE*. Essa arquitetura estende a arquitetura típica explorando a utilização de especificações XML (na forma de documentos DTD) e XSL na geração dos documentos por parte das aplicações. A *ArqGDE* é subjacente às aplicações geradas com o auxílio do roteiro *xRot*. Na figura:

- **(A)** representa a requisição enviada pelo *browser* (cliente) ao servidor WWW;
- **(B)** representa a ativação, pelo servidor, de programas de acordo com a interface CGI;
- **(C)** indica a seleção, pela aplicação, de um entre vários DTDs suportados;
- **(D)** indica a utilização do DTD apropriado pelo programa;
- similarmente, **(E)** e **(F)** indicam a seleção e utilização, respectivamente, de uma entre várias *style sheets* XSL.
- como no caso da arquitetura típica, a aplicação pode utilizar os recursos de um gerenciador de banco de dados, interação essa ilustrada na Figura 9 por **(G)**, para obtenção da informação a ser repassada ao cliente.
- **(H)** indica que a aplicação combina a especificação dada no DTD com a informação obtida junto ao banco de dados e a *style sheet* XSL apropriada para a produção do documento XML — que são repassados ao servidor WWW.
- finalmente, ***I*** indica que essas informações são passadas ao *browser* (cliente) de modo transparente pelo servidor — isto é, o servidor não dá tratamento especial aos dados enviados.

Portanto, no contexto da *ArqGDE*, é função do cliente processar os documentos XML e XSL recebidos. De fato, os *browsers* mais recentes trazem embutidos ou podem ser estendidos com processadores XML e XSL que permitem apresentar a informação correspondente de forma transparente ao usuário.

A arquitetura *ArqGDE* proposta diferencia-se da arquitetura típica por explorar o uso de documentos DTD, XML e XSL. Dessa forma, aplicações que seguem essa arquitetura podem utilizar os documentos XML manipulados para intercâmbio.

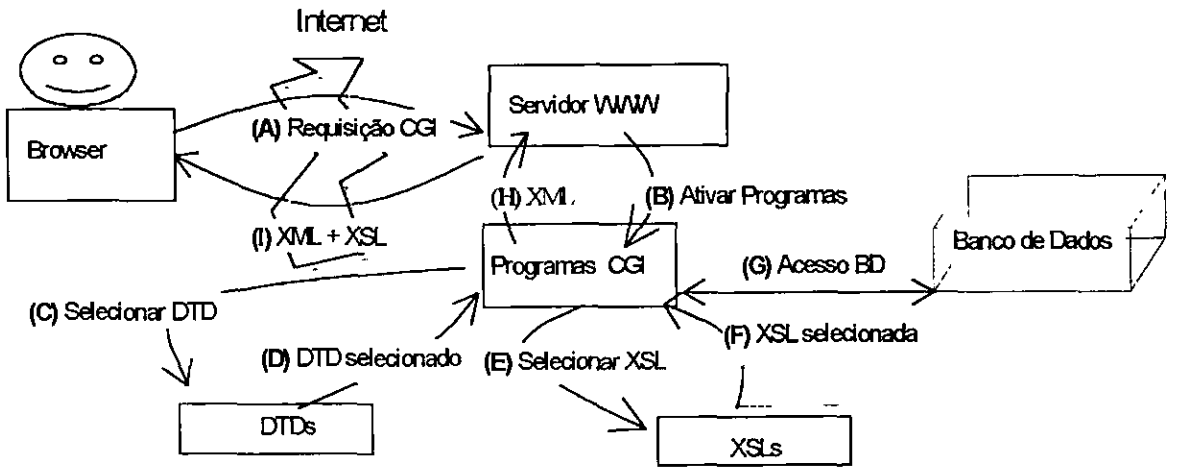

Figura 9 - Extensão da arquitetura típica - *ArqGDE* (Pimentel et al., 2000a).

Um dos passos mais importantes apresentados na Figura 9 é aquele ilustrado por (**H**), no qual a aplicação combina a especificação do DTD com a informação obtida junto ao banco de dados para a produção do documento XML com a *style sheet* XSL apropriada (passos (**C**), (**D**), (**E**), (**F**) e (**G**) da arquitetura *ArqGDE*).

Para apoiar o desenvolvimento de aplicações para a Internet de acordo com a *ArqGDE*, foi proposto o *xRot — Roteiro para Apoiar a Definição, Geração e Apresentação de Documentos Estruturados Manipulados por Aplicações na Internet*, detalhado na próxima seção.

3.3. xRot – Roteiro para Apoiar a Definição, Geração e Apresentação de Documentos Estruturados Manipulados por Aplicações na Internet

O roteiro *xRot* é composto de 4 fases, executados nesta ordem:

- **FASE 1: definição** dos documentos XML que devem ser gerados pela aplicação.
- **FASE 2: criação** de documentos DTD correspondentes aos documentos XML definidos na Fase 1.
- **FASE 3: criação** de documentos XSL com as declarações de transformação dos documentos XML — por exemplo, para transformá-los em HTML, RTF, PostScript, etc.

- **FASE 4: construção** de programas computacionais para **geração** dos documentos XML definidos na Fase 1 — responsáveis por implementar, especificamente, o passo **(H)** da *ArqGDE*.

Os programas correspondentes à fase 4 são criados utilizando-se o algoritmo *GERAXML*, que recebe como parâmetros um documento DTD, um documento XSL e valores correspondentes à solicitação de serviços feita pelo cliente (*browser*), ou seja, o resultado da consulta à base de dados — informações essas obtidas com os passos **(C)**, **(D)**, **(E)**, **(F)** e **(G)** da *ArqGDE*.

A Figura 10 apresenta um diagrama das fases do roteiro *xRot*.

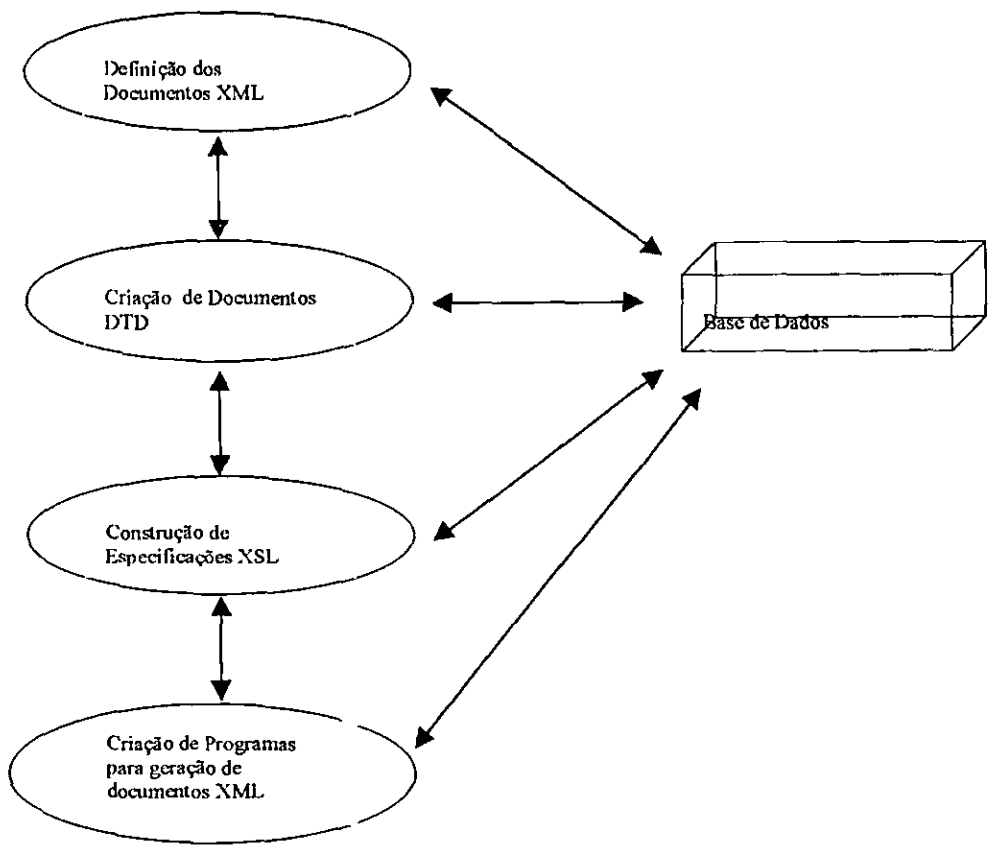

Figura 10 - Diagrama das Fases do roteiro *xRot*

Nessa figura:

a) as 4 fases estão indicadas nas 4 elipses à esquerda da figura; as setas entre as fases indicam a seqüência em que as fases são executadas, e que a mudança em uma fase pode provocar mudanças na(s) fase(s) anterior(es) e/ou posterior(es);

b) a referência a um sistema gerenciador de banco de dados na coluna da direita indica que ocorrendo uma mudança na estrutura do banco de dados pode também sugerir modificações nas fases do roteiro.

Cada uma das 4 fases é detalhada a seguir.

3.3.1. FASE 1: Definição de Documentos XML

As aplicações que correspondem à arquitetura típica ilustrada na Figura 8, de modo geral, oferecem diversos tipos de serviços ao usuário, serviços esses que muitas vezes implicam na busca ou atualização de informações manipuladas pela aplicação do lado do servidor. Associadas a esses serviços, são realizadas consultas à base de dados da aplicação; com os resultados específicos de cada consulta, é gerado um documento que é então repassado ao usuário para processamento ou apresentação.

A FASE 1 do *xRot* estabelece que, para cada tipo de documento resultante dos serviços oferecidos pela aplicação, deve ser definido um documento XML que corresponda a essa instância. Como exemplo, considere o caso de uma aplicação que disponibiliza três diferentes formas de material didático no servidor: documentos contendo teoria, exercícios propostos e exercícios resolvidos. Neste caso, três tipos diferentes de instâncias do banco de dados são gerados e, conseqüentemente, devem ser definidos três tipos diferentes de documentos XML correspondentes.

3.3.2. FASE 2: Criação de Documentos DTD

Definidos os diferentes tipos de documentos XML oferecidos pela aplicação na fase anterior, nesta fase são especificados os documentos DTD correspondentes — os quais fornecem uma especificação formal da estrutura dos documentos XML.

Os DTDs são definidos com base no conjunto de tabelas do banco de dados correspondente aos elementos relacionados à composição do documento XML. (Bos, 1997) demonstra que é possível representar toda uma base de dados, suas tabelas e campos como uma hierarquia de profundidade 3, tendo como nível 0 a base de dados com suas tabelas, nível 1 uma tabela com seus campos, e nível 2 os campos com seus respectivos tipos. No caso da representação da base de dados como um todo, a hierarquia funciona como segue: uma base de dados consiste de uma série de tabelas, que por sua vez consistem de uma série de campos e que, por sua vez, consistem do tipo do campo. No caso dos DTDs a serem gerados nesta fase, apesar deles corresponderem às porções das informações contidas no banco de dados, o procedimento equivalente para a definição da hierarquia correspondente ao documento pode ser utilizado.

O modelo de dados conceitual da aplicação também é utilizado na criação dos documentos DTD, uma vez que representa o relacionamento entre as entidades envolvidas. Nos DTDs, os relacionamentos são definidos de uma forma particular e, assim, a seguinte extensão do procedimento sugerido por (Bos, 1997) pode ser utilizada: 1) a todo relacionamento N:N indicado pelo modelo de dados conceitual da aplicação, o elemento correspondente no DTD é definido como uma composição dos elementos referentes às entidades envolvidas no relacionamento, bem como qualquer atributo exclusivo do relacionamento, e 2) a todo elemento participante de um relacionamento 1:N do lado "*1*" indicado pelo modelo de dados conceitual da aplicação, o elemento correspondente no DTD é definido como uma composição do outro elemento referente a entidade envolvida no relacionamento do lado "*N*". Além disso, a cardinalidade dos elementos que compõem os elementos dos dois casos é definida de acordo com a cardinalidade dada pelo relacionamento — 0:1, 0:N, 1:1 e 1:N correspondem, respectivamente, a *entidade?*, *entidade**, *entidade* e *entidade*$^+$. Para tal, relacionamentos N:N devem ser especificados pelos seus relacionamentos 1:N e N:1 correspondentes.

Como exemplo, seja o caso de uma aplicação de controle acadêmico que oferece, entre outros, os seguintes serviços:

- Informar a um aluno as disciplinas em que está matriculado no período corrente;
- Informar ao aluno seu *histórico escolar* contendo a relação de disciplinas cursadas e notas obtidas.

Analisando o caso de apresentar ao aluno informações das disciplinas do período, o desenvolvedor da aplicação identifica que as informações a serem apresentadas envolvem as entidades *Aluno* e *Disciplina* e o relacionamento *Matrícula* entre elas: um *aluno* pode se *matricular* em no mínimo 1 e no máximo N *disciplinas,* e em uma *disciplina* estão *matriculados* no mínimo 1 e no máximo N *alunos.*

Nesse caso, as tabelas do banco de dados da aplicação de controle acadêmico que representam as entidades *Aluno, Disciplina* e *Matricula* são definidas por:

Aluno = {numeroaluno, nomealuno, enderecoaluno},
Disciplina = {iddisciplina, nomedisciplina, numerocreditos} e
Matricula = {numeroaluno, iddisciplina, nota}.

A partir dessas tabelas, método similar ao indicado por (Bos, 1997) é utilizado na definição dos elementos *Aluno* e *Disciplina* da seguinte forma: às tabelas *Aluno* e *Disciplina* são associados elementos cujos atributos são as próprias colunas das tabelas — como indicado na Figura 11a.

A tabela *Matrícula* também é utilizada na definição de seu elemento associado. Entretanto, como o modelo de dados conceitual da aplicação indica que *Matrícula* é um relacionamento, a definição do elemento *Matrícula* é feita como uma composição dos elementos correspondentes às entidades envolvidas no relacionamento, bem como qualquer atributo exclusivo do relacionamento. Além disso, a cardinalidade dos elementos que compõem o elemento do relacionamento é definida de acordo com a cardinalidade dada pelo relacionamento. Nesse caso especial em que a cardinalidade do relacionamento é *N:N*, o relacionamento deve ser partido, e devem ser especificados dois relacionamentos: o relacionamento *Matricular-se* com cardinalidade **1:N** definido pela composição *Matricular-se* (aluno, disciplina, nota)+ e o relacionamento *Matriculado* com cardinalidade **N:1** definido pela composição *Matriculado* (aluno, disciplina, nota)+. Os elementos *Aluno* e *Disciplina* correspondem às entidades envolvidas no relacionamento e o último elemento é exclusivo do relacionamento. Os elementos *matricular-se* e *matriculado* são apresentados no documento DTD MatriculaML da Figura 11b.

```
<!ELEMENT aluno >
<!ATTLIST aluno
                    numeroaluno    ID        #REQUIRED
                    nomealuno      CDATA     #REQUIRED
                    enderecoaluno  CDATA     #REQUIRED>

<!ELEMENT disciplina >
<!ATTLIST disciplina
                    iddisciplina     ID  #REQUIRED
                    nomedisciplina   CDATA  #REQUIRED
                    numerocreditos   CDATA  #REQUIRED >
```

```
<!ELEMENT matricular-se (aluno, disciplina, nota)+ >
<!ELEMENT matriculado (aluno, disciplina, nota)+ >
<!ELEMENT aluno >
<!ATTLIST aluno
                    numeroaluno    ID        #REQUIRED
                    nomealuno      CDATA     #REQUIRED
                    enderecoaluno  CDATA     >
<!ELEMENT disciplina >
<!ATTLIST disciplina
                    iddisciplina     ID        #REQUIRED
                    nomedisciplina   CDATA     #REQUIRED
                    numerocredito    CDATA     #REQUIRED >
<!ELEMENT nota    PCDATA                                 >
```

Figura 11 - a) Documento DTD AlunoML e b) Documento DTD MatriculaML

Ao final desta fase deve ter sido definido um DTD correspondente a cada documento XML produzido pela aplicação.

3.3.3. FASE 3: Criação de documentos XSL

Esta fase corresponde à definição de *style sheets* responsáveis pela transformação de documentos XML em documentos a serem manipulados pelo cliente. Assim, se o objetivo é a apresentação, no *browser* do usuário, de documentos HTML, as *style sheets* definidas apresentam as declarações que, quando processadas por um processador XSL, transformam o documento XML no documento HTML esperado pelo cliente. Por outro lado, se o cliente é uma aplicação associada a um serviço de impressão, a *style sheet* deve especificar a transformação do documento XML em um documento RTF ou PostScript, por exemplo. Como exemplo complementar, se o cliente corresponde a uma aplicação interessada no documento XML propriamente dito — como o objetivo único de intercâmbio — nenhuma *style sheet* precisa ser definida nesta fase.

Uma *style sheet* XSL contém um conjunto de declarações que determinam como um documento XML deve ser transformado. Utilizando o exemplo da aplicação que deve gerar um documento XML com informações da matrícula de um aluno, a Figura 12a apresenta um documento XML de matrícula — essa figura contém, no início, a especificação de uma *style sheet* que deve ser utilizada em sua apresentação.

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.int
ermidia.icmc.sc.usp.br/~d
fpires/xsl">
<matricular-se>
<aluno idaluno='19422585'
nomealuno='Paulo'
endereco='São Paulo'/>
<disciplina
iddisciplina='sce-126'
nomedisciplina='ICC1'
numerocreditos='8'/>
<nota> 8.0 </nota>
<aluno idaluno='19422585'
nomealuno='Paulo'
endereco='São Paulo'/>
<disciplina
iddisciplina='sce-127'
nomedisciplina='ICC2'
numerocreditos='8'/>
<nota> 6.0 </nota>
<aluno idaluno='19422585'
nomealuno='Paulo'
endereco='São Paulo'/>
<disciplina
iddisciplina='sce-136'
nomedisciplina='ED1'
numerocreditos='8'/>
<nota> 8.0 </nota>
<matricular-se>
```

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/W
D-xsl">
<xsl:template match="/matricular-
se">
<HTML>
<HEAD>
<BODY>
<H1>
Nome do Aluno: <xsl:value-of
select="aluno/@nomealuno"/>
</H1>
<H2>
Numero de Matricula do Aluno:
<xsl:value-of
select='aluno/@numeroaluno'/>
</H2>
<H3>
Disciplinas Matriculas:
<xsl:for-each
select="disciplina">
<P><xsl:value-of
select="@nomedisciplina"/><BR>
</xsl:for-each>
</H3>
</BODY>
</HEAD>
</HTML>
</xsl:template>
```

```
<HTML>
<HEAD>
<BODY>
<H1>Nome do
Aluno:
Paulo</H1>
<H2>Numero de
Matricula do
Aluno:
19422585</H2>
<H3>Disciplinas
Matriculadas:
<P>ICC1<BR>
<P>ICC2<BR>
<P>ED1<BR>
</H3>
</BODY>
</HEAD>
</HTML>
```

Figura 12 - (a) documento XML, (b) documento XSL para transformação em HTML e (c) documento HTML resultante

A Figura 12b apresenta a *style sheet* XSL referenciada, a qual contém especificações de transformação do documento XML da Figura 12a para HTML. Nessa figura, a declaração:

```
<xsl:template match="/Aluno">
```

indica que a todo elemento *<Aluno>* encontrado no documento XML são aplicadas as declarações existentes nesse próprio template — isto é, são aplicadas as declarações encontradas entre este ponto e o `</xsl:template>` correspondente. Como outro exemplo, a declaração:

```
<xsl:value-of select="nomealuno"/>
```

especifica que sua ocorrência deve ser substituída pelo conteúdo do elemento *<nomealuno>* do documento XML. A Figura 12c apresenta o resultado da transformação do documento XML da Figura 12a em documento HTML — de acordo com as especificações dadas no documento da Figura 12b.

Alguns dos *browsers* mais recentes incluem um processador XSL que, recebendo do servidor o documento ilustrado na Figura 12a, busca o documento XSL especificado e realiza a transformação correspondente. Nesse cenário, a aplicação serve ao *browser* cliente um documento XML que, processado pelo *browser*, é apresentado como HTML ao usuário — processo esse realizado de modo transparente ao usuário.

Ao final desta fase, uma especificação XSL deve ter sido criada para cada possível forma de apresentação dos diferentes documentos XML produzidos pela aplicação. Isso significa que, para um mesmo documento XML (e seu DTD), mais de uma especificação XSL pode ser produzida. Por exemplo, podem ser geradas *style sheets* responsáveis pela transformação do documento XML em HTML, RTF e PostScript.

3.3.4. FASE 4: Criação de Programas para Geração de Documentos XML

Programas computacionais podem ser escritos com o objetivo de gerar documentos estruturados XML com conteúdo de uma base de dados e a partir da análise de um documento DTD. Esses programas correspondem à seqüência de passos de ***C*** a ***H*** da arquitetura *ArqGDE*. Nesta fase, os passos necessários para a construção desses programas são detalhados. Para melhor ilustrar esses passos, é considerado o caso da construção de um programa na linguagem Java, o qual utiliza *JDBC*[6] para comunicar-se com um gerenciador de banco de dados *mySQL*[7].

As principais atividades do programa a ser gerado são:

1. Importação de pacotes (*packets*) da API Java para que o programa tenha recursos para: manipular vetores (*java.util.**), executar comandos SQL (*java.sql.**) e conectar-se a um servidor de banco de dados (*gwe.org*), como exemplo.

2. Estabelecimento de conexão com banco de dados, por exemplo:

   Connection con=DriverManager.getConnection(jdbc:mysql://fce.cc.gatech.edu:3306/c2000, "username", "password");

[6] Java Database Connection: http://java.sun.com/products/jdbc/index.html
[7] Disponível em http://www.mysql.com

3. Obtenção dos valores passados ao programa Java de acordo com a interface CGI, por exemplo, identificador de um curso e forma de apresentação do documento XML:

   String CursoID = argv[0]; String idPresent = argv[1];

4. O conteúdo dos elementos do documento XML é buscado na base de dados através dos métodos executeQuery e getString ou getInt das classes Java stmt e ResultSet, respectivamente, que compõem o pacote java.sql importado no Passo 1. Um exemplo da estrutura de busca é apresentado:

   String queryConsulta = "string compondo uma consulta ao banco de dados"
   ResultSet rsConsulta = stmt.executeQuery(queryConsulta);
   String NomeAluno = rsConsulta.getString(1); //primeiro campo do conjunto de tuplas resultante
   System.out.println("<aluno nomealuno=\"NomeAluno\"/> ");

   Nesse exemplo, a composição da consulta a ser especificada em *queryConsulta* é de particular importância. É nesse momento que a aplicação define qual dos DTDs é o utilizado, juntamente com os demais parâmetros fornecidos pelo usuário (por exemplo, o número de identificação do aluno).

   Assim, do resultado da consulta é extraído o conteúdo da string *nomeAluno*, que é então colocado como valor do atributo *nomealuno* do elemento *<aluno>*.

5. A construção do documento XML é realizada utilizando-se o algoritmo GERAXML() ilustrado na Figura 13 e discutido a seguir.

6. Fechamento de conexão remota da aplicação com o servidor de banco de dados.

```
FUNCAO GERAXML (refDTD, idXSL, Ps)
INICIO
        SE ELEMENTO_RAIZ
                ENTAO       ESCREVA "<?xml-stylesheet type=\"text/xsl\" href=\"nome_XSL.xsl\" ?>"
                        // código para que um documento XML possa ser interpretado por um documento XSL
                        ESCREVA "<!DOCTYPE aulaml SYSTEM \"nome_DTD.dtd\">"
                        // código necessário para indicar a qual documento DTD o documento XML conforma
        FIMSE
        ESCREVA "<ELEMENTO";  // o valor de ELEMENTO deve ser buscado com um consulta ao banco de dados,
        podendo ter  ou não        // condições Ps
        ENQUANTO (ATRIBUTO) FACA
                ESCREVA "ATRIBUTO="VALOR_ATRIBUTO"";  // o valor de ATRIBUTO deve ser buscado com um
                consulta ao banco                      // de dados, podendo ter ou não condições Ps
        FIMENQUANTO
        SE (ELEMENTO = EMPTY)
                ENTAO ESCREVA "/>";
        SENAO ESCREVA ">";
              ENQUANTO (SUBELEMENTO) FACA
                 SE (SUBELEMENTO OU SUBELEMENTO?)
                     ENTAO GERAXML(SUBELEMENTO)
                 SENAO SE (SUBELEMENTO* OU SUBELEMENTO+)
                           ENTAO ENQUANTO (SUBELEMENTO) FACA
                                        GERAXML(SUBELEMENTO)
                                 FIMENQUANTO
                       FIMSE
                 FIMSE
              FIMENQUANTO
         ESCREVA </ELEMENTO>;
         FIMSE
FIM
```

Figura 13 - GERAML: Algoritmo para geração de documentos estruturados XML

A atividade 5 corresponde a um dos passos importantes da *ArqGDE* (ilustrado por ***H*** na Figura 9). Essa atividade inclui a implementação do algoritmo *GERAXML* apresentado na Figura 13, que indica os passos a serem executados para que a aplicação, combinando a especificação do DTD com a informação obtida no banco de dados, realize a produção do documento XML associado a *style sheet* XSL apropriada.

O algoritmo *GERAXML* recebe como parâmetros o documento DTD, a referência a um documento XSL, bem como valores correspondentes à solicitação de serviços por parte do usuário (indicado por *Ps* na lista de parâmetros). As estruturas de condição e de repetição do algoritmo foram baseadas nas regras de um documento DTD. O algoritmo *GERAXML* recebe, inicialmente, o elemento raiz do documento DTD — nesse caso, a referência para a *style sheet* é também processada. Os demais elementos definidos pela hierarquia do DTD são processados de maneira recursiva. As variáveis VALOR_ATRIBUTO e VALOR_ELEMENTO representam o conteúdo dos atributos e elementos, respectivamente, do documento XML. Esse conteúdo é buscado a partir do banco de dados da aplicação como mostrado na Atividade 4. As buscas podem ser condicionadas aos valores dos parâmetros *Ps* associados aos usuários.

3.4. Trabalhos Relacionados

Foram feitos levantamentos bibliográficos a fim de verificar trabalhos que estudavam soluções para os problemas de intercâmbio entre sistemas distintos e ainda, que utilizassem para isso, recursos semelhantes ao explorado neste trabalho. Trabalhos reportados na literatura, como os de (Royappa, 1999) e (Suzuki et. al., 1998) discutidos a seguir, utilizam, para solução de problemas específicos, recursos semelhantes ao explorados neste trabalho. Entretanto, o trabalho aqui reportado contribui também com a proposta do roteiro *xRot*, da generalização dos processos de definição de DTDs e de geração automática dos documentos XML manipulados.

(Royappa, 1999) apresenta uma arquitetura para a implementação de aplicações na Web em geral e, em particular, aquelas que disponibilizam na Web catálogos de restaurantes que são gerados dinamicamente como resposta de uma consulta do usuário. A arquitetura explora os recursos XML e XSL. Um aspecto discutido no desenvolvimento dessas aplicações foi a dificuldade em apresentar os catálogos já que cada restaurante tem uma interface de catálogo diferente. A solução foi a utilização de XSL. Outro aspecto levantado é a limitação e a dificuldade em realizar consultas complexas em uma base de dados relacional que armazena as informações dos catálogos. A solução para este problema foi resolvida com a utilização do padrão XML, construindo um sistema de arquivos formado por documentos XML e uma linguagem de consulta tipo XML-QL (Deutsch et al., 1998) para consultar esses documentos.

Com a tentativa de favorecer o intercâmbio de documentos de um projeto de software que possui um grupo de desenvolvedores distribuídos, (Suzuki et. al., 1998) propõem a UXF (*UML Exchange Format*), um formato para intercâmbio de documentos criados a partir de modelos UML. Também é apresentada uma arquitetura em camadas que utiliza clientes e servidores UXF e CORBA (OMG, 2000) com o objetivo de solucionar problemas relacionados à distribuição de documentos XML na Web.

Abordagens mais tradicionais de apoio ao desenvolvimento de aplicações para a Web normalmente não suportam diretamente o intercâmbio de documentos entre diferentes aplicações. (Cerqueira et al., 1999) apresentam a ferramenta HOOT. Esta ferramenta integra características de hipermídia e banco de dados orientado a objetos para solucionar problemas durante a implementação e manutenção de aplicações projetadas com métodos tradicionais de

modelagem de aplicações hipermídia. Já (Pizzol et al., 1999) propõem um *framework* orientado a objeto, denominado OOHDM-Java, para implementação de aplicações hipermídia complexas modeladas com o método OOHDM. Nos dois casos, a proposta tem subjacente uma arquitetura que estende a arquitetura típica da Web para suportar os módulos específicos de estrutura hipermídia oferecida às aplicações — como por exemplo, suporte à navegação. Em ambos os casos, a arquitetura a eles subjacente pode ser estendida para suportar as especificações de documentos — XML, DTDs e *style sheets* XSL — proposta pela *ArqGDE* — favorecendo assim o intercâmbio de documentos entre aplicações distintas.

3.5. Considerações Finais

Motivado em apoiar o desenvolvimento de aplicações para a Web que se preocupam com o intercâmbio das informações, este trabalho propôs o roteiro *xRot*, que orienta a utilização de recursos sugeridos pelo W3C, em especial ao padrão XML, nas etapas de definição, geração e apresentação de documentos estruturados manipulados por aplicações para a Internet.

O roteiro *xRot* define 4 fases para apoiar a autoria de aplicações para a Internet: **1) definição de documentos XML, 2) criação de documentos DTD, 3) criação de documentos XSL** e **4) criação de programas computacionais para geração de documentos XML**. Assim, aplicações construídas com o *xRot* podem utilizar os documentos XML manipulados para intercâmbio.

4. A APLICAÇÃO AULAML

4.1. Considerações Iniciais

A aplicação AulaML foi desenvolvida neste trabalho como estudo de caso da utilização do roteiro *xRot*. O roteiro apoiou a autoria da aplicação nas etapas de definição, geração e apresentação dos documentos estruturados suportados pela mesma na Web.

Este capítulo segue introduzindo a aplicação AulaML através de um diagrama de atividades da aplicação. O capítulo continua mostrando como a aplicação AulaML foi desenvolvida utilizando o roteiro *xRot*. Em seguida, são apresentados dois recursos da aplicação.

4.2. A aplicação AulaML

AulaML é utilizada no ambiente da WWW e foi inicialmente proposta em (Pimentel et al. 1999). Através desta aplicação, professores podem criar *quadros de aulas* contendo teoria e exercícios que, posteriormente, são disponibilizados na Web para que alunos da disciplina possam estudar e resolver exercícios. Foram projetadas duas visões: a do professor e a do aluno.

O professor pode criar *Cursos*, *Aulas*, *Quadros*, e associá-los entre si. Um *Quadro* é composto de uma parte *Teórica* (*Parágrafos* e *Listas*) e outra *Prática* (com exercícios de questões *Múltipla Escolha* e *Verdadeiro-Falso*).

O aluno pode visualizar os *Quadros* de uma determinada *Aula* na Web, dentre 4 opções disponíveis, escolhendo se deseja consultar somente *Teoria*, ou apenas *Testes*, ou somente o *Gabarito* dos exercícios (esta opção é protegida por senha), ou ainda todas as opções existentes (*Teoria*, *Testes* e *Gabarito*). Na consulta da parte prática dos *Quadros*, o aluno pode resolver exercícios, obtendo do sistema o resultado.

A tela inicial da aplicação AulaML na Web é apresentada na Figura 14. Nela, encontram-se opções para: criação de *Cursos*, criação de *Aulas*, criação de *Quadros*, associação de *Cursos* e *Aulas* (com o objetivo de viabilizar o reuso de *Quadros*) e consulta de *Quadros* de uma *Aula*. Como a ferramenta funciona no ambiente da WWW, as opções da tela inicial são apresentadas ao usuário na forma de ligações.

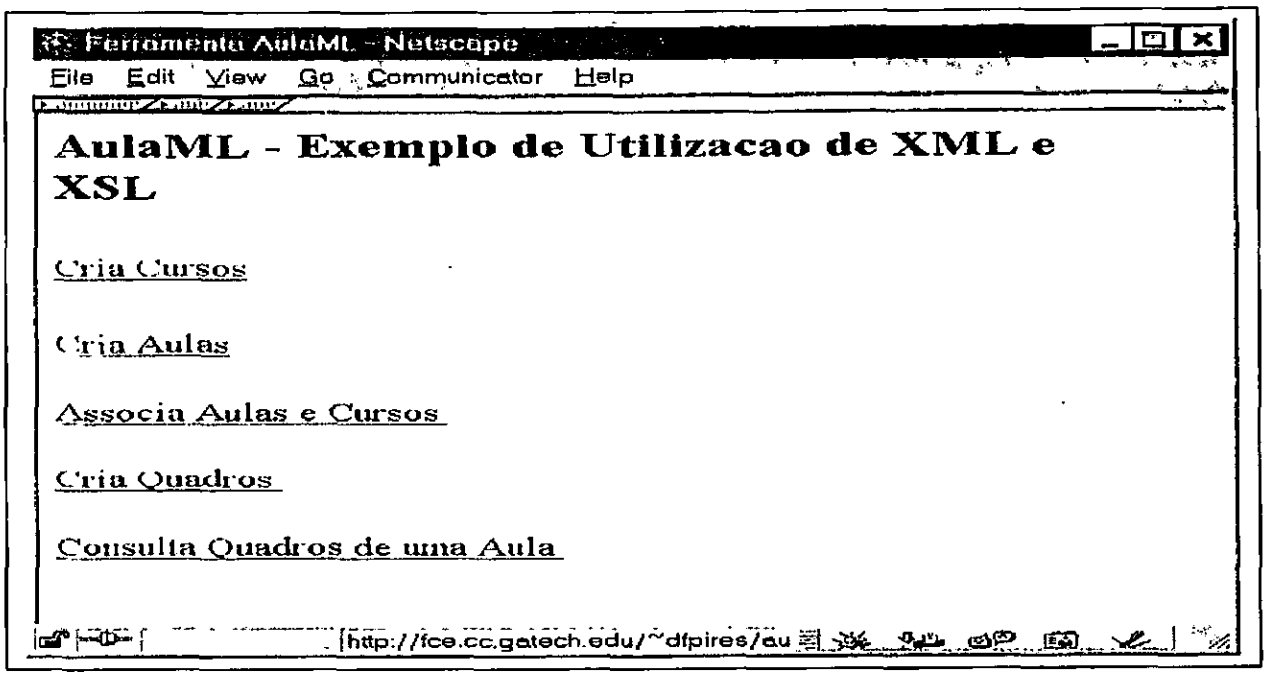

Figura 14 - Tela inicial da aplicação AulaML na WWW

A estrutura do ambiente foi originada na definição de uma especificação DTD apresentado em (Pimentel et al., 1999). Um resumo das principais atividades da aplicação, ilustrando o seu funcionamento, é apresentado na Figura 15.

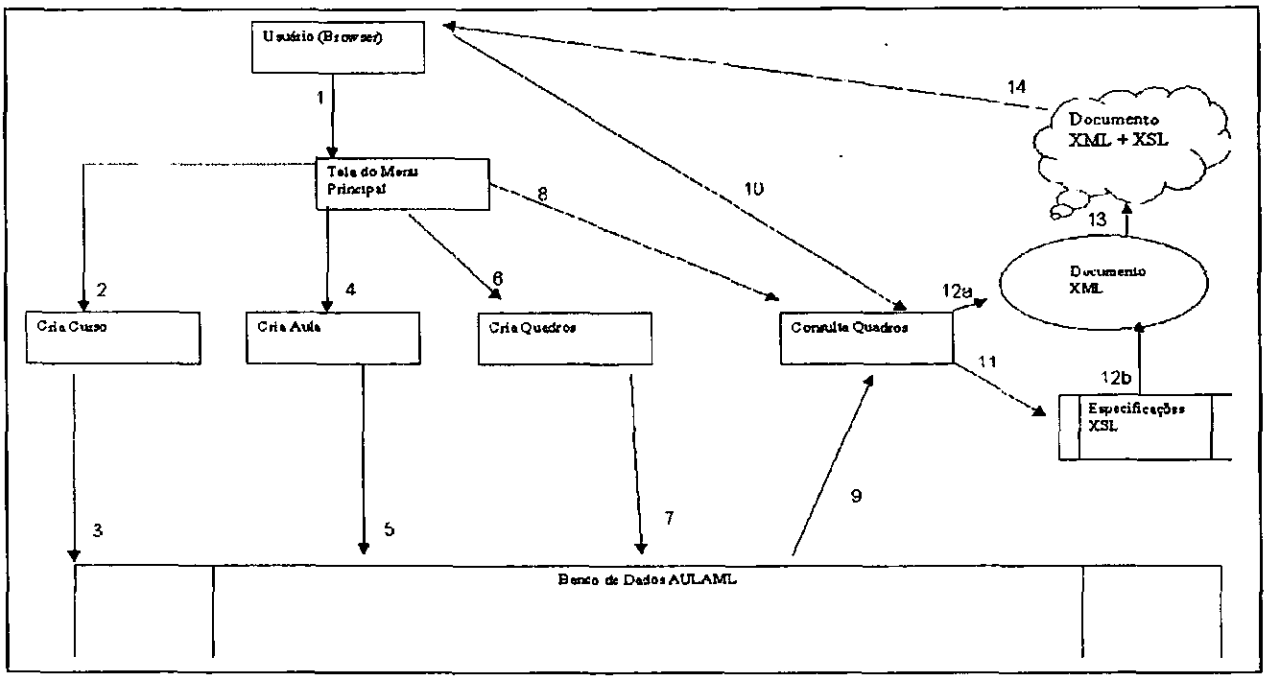

Figura 15 - Resumo das atividades da Aplicação AulaML

As atividades indicadas nessa figura são:

1. Usuário, utilizando um *browser* no ambiente da WWW, faz uma escolha dentre as opções existentes na tela inicial da aplicação AulaML.
2. Usuário professor cria um *Curso*
3. *Curso* é armazenado no Banco de Dados.
4. Usuário professor cria uma *Aula*
5. A *Aula* é armazenada no banco de dados.
6. Usuário professor cria *Quadros* para uma determinada *Aula* e *Curso*. Estes *Quadros* são, posteriormente, disponibilizados aos alunos na Web.
7. Os *Quadros* são armazenados no banco de dados
8. Usuário aluno escolhe consultar *Quadros* criados para uma determinada *Aula* e *Curso*.
9. O conteúdo necessário para a criação do documento XML é recuperado no banco de dados por um programa computacional no servidor.
10. Usuário aluno informa qual a melhor forma e estilo de visualização dentre 4 opções disponíveis (somente *Teoria*, apenas *Testes*, somente *Gabarito*, e todas as opções juntas).
11. Programa computacional consulta arquivo XSL que contém a especificação necessária para gerar informações dos *Quadros* de acordo com a escolha do usuário.
12. (a) O documento XML é gerado dinamicamente pelo programa no servidor (b) é função também do programa recuperar arquivo XSL.
13. Processador XSL no *browser* do cliente processa documento XML.
14. Documento é apresentado ao usuário aluno com interface do documento XSL (HTML) e conteúdo do documento XML.

A seguir, os passos realizados para a autoria da aplicação seguindo o roteiro *xRot* são explicados.

4.3. Desenvolvimento da Aplicação AulaML utilizando o xRot

Para o desenvolvimento da aplicação AulaML, os seguintes passos foram seguidos:

4.3.1. Definição de Documentos XML

De acordo com o roteiro *xRot*, é necessário definir um tipo de documento XML para cada tipo de documento resultante dos serviços oferecidos pela aplicação. Na aplicação AulaML é disponibilizado apenas um serviço ao usuário: apresentação no *browser* de informações da *Aula* de um determinado *Curso*. Este serviço resulta em apenas um tipo de documento XML.

4.3.2. Criação de Documentos DTD

Definido o documento XML, é necessário criar o documento DTD para a formalização do mesmo. Segundo o *xRot*, a criação de um documento DTD correspondente a um documento XML é baseada no conjunto de tabelas do banco de dados e no modelo conceitual da aplicação.

O serviço de apresentação no *browser* de informações da *Aula* de um determinado *Curso* envolve as entidades *Curso*, *Aluno*, *Quadro*, *Texto*, *Teste* e *Index* e o relacionamento entre elas, como *Aula_Curso_Quadro*. O método similar ao de (Bos, 1997) foi utilizado na definição dos

elementos DTD *Curso*, *Aluno*, *Quadro*, *Texto*, *Teste*, *Index* e *Aula_Curso_Quadro* a partir das tabelas do banco de dados da aplicação AulaML que representam as entidades da aplicação.

A Figura 16 apresenta o modelo conceitual da aplicação ilustrando o relacionamento entre essas entidades. Um *Curso* pode possuir várias *Aulas*. Uma *Aula* pode estar em vários *Cursos* e possuir vários *Quadros*, que por sua vez, pode estar em mais de uma *Aula*. Um *Quadro* possui um *Texto*, e pode ou não conter um *Teste* e um *Index*. Nessa figura é possível perceber que o elemento *Aluno_Curso_Quadro* é um relacionamento. De acordo com extensão do procedimento sugerido por (Bos, 1997) no *xRot*, a definição do elemento *Aluno_Curso_Quadro* é feita como uma composição dos elementos correspondentes às entidades envolvidas no relacionamento, bem como qualquer atributo exclusivo do relacionamento. A figura mostra também que o elemento *Quadro* está envolvido em um relacionamento 1:N do lado "*1*". Também de acordo com o procedimento no *xRot*, a definição do elemento *Quadro* é feita como uma composição do outro elemento referente à entidade envolvida no relacionamento do lado "*N*". Nesses dois casos, a cardinalidade dos elementos que compõem os elementos *Aula_Curso_Quadro* e *Quadro* é definida de acordo com a cardinalidade dada pelo relacionamento.

O relacionamento ternário **N:N:N** entre os elementos *Aula*, *Curso* e *Quadro* foi quebrado em três partes, e especificados três relacionamentos: o relacionamento *Aula_Curso_Quadro* com cardinalidade **1:N:N** definido pela composição *Aula_Curso_Quadro* (aula, curso, quadro)+, o relacionamento *Curso_Quadro_Aula* com cardinalidade **1:N:N** definido pela composição *Curso_Quadro_Aula* (curso, quadro, aula)+ e o relacionamento *Quadro_Aula_Curso* com cardinalidade **1:N:N** definido pela composição *Quadro_Aula_Curso* (quadro, aula, curso)+. Como o relacionamento *Aula_Curso_Quadro* (aula, curso, quadro)+ era o que mais interessava dentre os outros dois especificamente para o serviço da aplicação AulaML, ele foi o único definido no documento DTD AulaML_1 da Figura 17. Além disso, esse relacionamento foi definido com uma alteração: *Aula* agora é o nome do relacionamento ao invés de fazer parte da composição do relacionamento, ficando assim: *Aula* (curso, quadro)+. Isso foi feito para facilitar a geração dos documentos XML associados. Esse elemento faz parte do documento DTD AulaML_1.

O relacionamento binário **1:N** entre os elementos *Quadro* e *Texto*, *Quadro* e *Teste*, e *Quadro* e *Index* indica a cardinalidade dos elementos que compõem o elemento Quadro. De acordo com a segundo fase do roteiro xRot, a cardinalidade é definida de acordo com a cardinalidade dada pelo relacionamento — 0:1, 0:N, 1:1 e 1:N que correspondem, respectivamente, a *entidade?*, *entidade**, *entidade* e *entidade*$^+$. Assim, o elemento *Quadro* fica assim definido: *Quadro* (texto, index?, teste?). Esse elemento também faz parte do documento DTD AulaML_1.

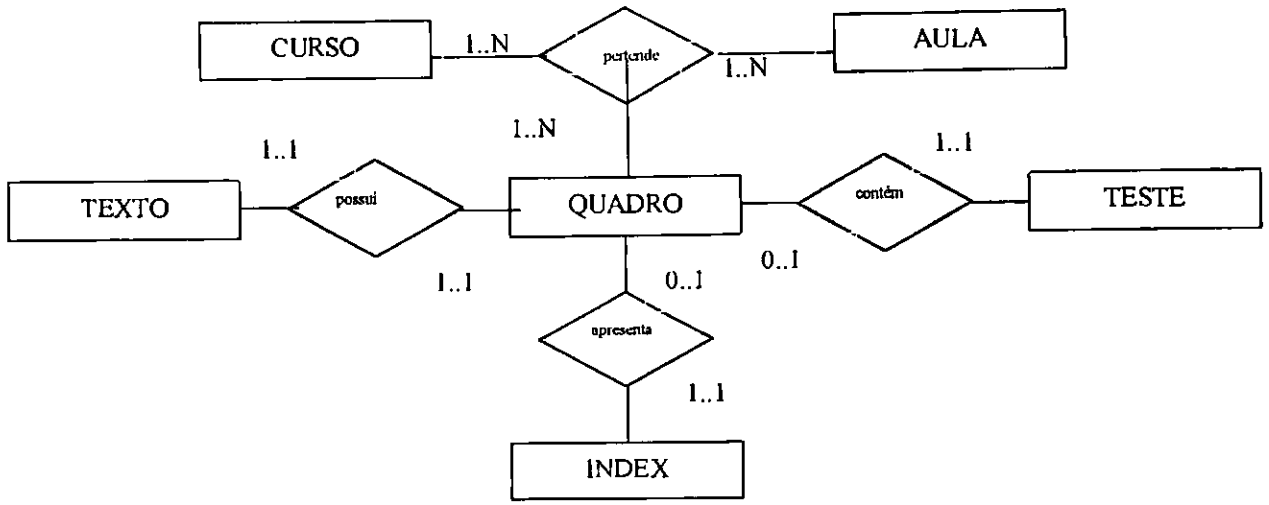

Figura 16 - Modelo Conceitual da aplicação AulaML

Dessa forma, foi definido o documento DTD AulaML_1 que corresponde à formalização dos documentos XML gerados na aplicação.

```
<!ELEMENT aula (curso, quadro)+                                          >
<!ATTLIST aula
                id ID #REQUIRED
                prof CDATA #REQUIRED
                titulo CDATA #REQUIRED                                   >
<!ELEMENT curso EMPTY                                                    >
<!ATTLIST curso id CDATA #REQUIRED                                       >
<!ATTLIST curso nome CDATA #IMPLIED                                      >
<!ELEMENT quadro (texto, index?, teste?)                                 >
<!ATTLIST quadro
                id ID #REQUIRED
                tipo (teoria|exemplo|exercicio) "teoria"                 >
<!ELEMENT texto (ul|p)+                                                  >
<!ELEMENT ul (li)+                                                       >
<!ELEMENT li (#PCDATA|index)*                                            >
<!ELEMENT p (#PCDATA|index)*                                             >
<!ELEMENT index EMPTY                                                    >
<!ATTLIST index termo CDATA #REQUIRED                                    >
<!ELEMENT teste (verdfalso|multiplo)+                                    >
<!ELEMENT verdfalso (li)                                                 >
<!ATTLIST verdfalso resp (V|F) #REQUIRED                                 >
<!ELEMENT multiplo (pergunta,ul)                                         >
<!ATTLIST multiplo resp CDATA #REQUIRED                                  >
<!ELEMENT pergunta (#PCDATA)                                             >
```

Figura 17 - Documento DTD AulaML_1 para a aplicação AulaML (Pimentel et. al., 1999)

4.3.3. Construção de Especificações XSL

Para o documento DTD AulaML_1 foram construídas 4 *style sheets* XSL. Essas 4 *style sheets* fornecem 4 visualizações diferentes do conteúdo de uma aula. O documento da Figura 18 ilustra

um exemplo: o documento gera informações apenas da parte prática de uma *Aula*. Nessa figura, a declaração:

```
<xsl:template match="/Quadro">
```

indica que a todo elemento <*Quadro*> encontrado no documento XML são aplicadas as declarações existentes nesse próprio *template* — isto é, são aplicadas as declarações encontradas entre este ponto e o `</xsl:template>` correspondente. Como outro exemplo, a declaração:

```
<xsl:value-of select="./teste/multiplo/pergunta" "/>
```

especifica que sua ocorrência deve ser substituída pelo conteúdo do elemento <*pergunta*> do documento XML.

A Figura 19 ilustra como o documento XML da Figura 20, gerado neste caso, aparece para o usuário na Web de acordo com especificação XSL da Figura 18. Outras 3 *style sheets* foram definidas também para o documento DTD AulaML_1: visualização apenas da parte teórica de uma *Aula*, somente do gabarito das questões, ou ainda, todas as opções existentes (parte teórica, parte prática, e gabarito). As *style sheets* XSL construídas para a AulaML encontram-se no Anexo A.

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<STYLE type="text/css">BODY {font-family: serif;} H1, H2, H3 {font-family: Arial, Helvetica, sans-serif;}
H1 {color: blue; text-decoration: underline;} H2 {color: green;} P {margin-left: 3em;} SUP {font-size: x-small;
color: #666666;} DIV.endnote P {text-indent: -1em; color: #333333;} .link {text-decoration:
underline;}</STYLE>
<HEAD> <TITLE>
Lecture: <xsl:value-of select="aulaml/@titulo" />
</TITLE> </HEAD>
<BODY> <H1> Lecture: <xsl:value-of select="aulaml/@titulo" />
</H1> <h2> Prof.: <xsl:value-of select="aulaml/@prof" />
</h2> <h3> <center>Exercise's List</center> </h3>
<xsl:apply-templates select="aulaml/quadro" />
<hr />
</BODY>
</HTML>
</xsl:template>
<xsl:template match="quadro">
<xsl:for-each select="."> <hr /> <h2> Topic:
<xsl:value-of select="./index/@termo" />
</h2> <P /> <P /> <I>T/F:</I> <B>
<xsl:value-of select="./teste/verdfalso/li" /> </B>
<FORM name="questionnaire" ACTION="testverify.php3" METHOD="POST"> <BR />
<INPUT TYPE="text" NAME="truefalse" />
<INPUT TYPE="hidden" NAME="aulaID">
<xsl:attribute name="value">
<xsl:value-of select="../@id" />
</xsl:attribute> </INPUT> <INPUT TYPE="hidden" NAME="cursoID">
<xsl:attribute name="value">
<xsl:value-of select="../curso/@id" />
</xsl:attribute> </INPUT>
<INPUT TYPE="hidden" NAME="quadroID">
<xsl:attribute name="value">
<xsl:value-of select="./@id" />
</xsl:attribute> </INPUT>
<INPUT TYPE="hidden" NAME="TFResp">
<xsl:attribute name="value">
<xsl:value-of select="./teste/verdfalso/@resp" />
</xsl:attribute> </INPUT> <P /> <I>Q:</I> <B>
<xsl:value-of select="./teste/multiplo/pergunta" /> </B> <BR />
<xsl:for-each select="./teste/multiplo/ul">
<xsl:eval>create(0)</xsl:eval><xsl:for-each select="./alternativa">
<INPUT TYPE="radio" name="multiple"> <xsl:attribute name="value">
<xsl:eval>create(1)</xsl:eval> </xsl:attribute> <xsl:value-of select="." />
</INPUT> <BR /> <INPUT TYPE="hidden" NAME="multipleResp">
<xsl:attribute name="value"> <xsl:value-of select="../../@resp" /> </xsl:attribute> </INPUT>
</xsl:for-each> </xsl:for-each> <PRE> <input type="submit" value="Post" />
</PRE> </FORM> <xsl:apply-templates /> </xsl:for-each> </xsl:template>
<xsl:script> <![CDATA[
      function create(aux) {
        if (aux == 0) {
           aux1 = 0; }
```

Figura 18 – Style Sheet XSL para apresentação de exercícios dos quadros de uma aula

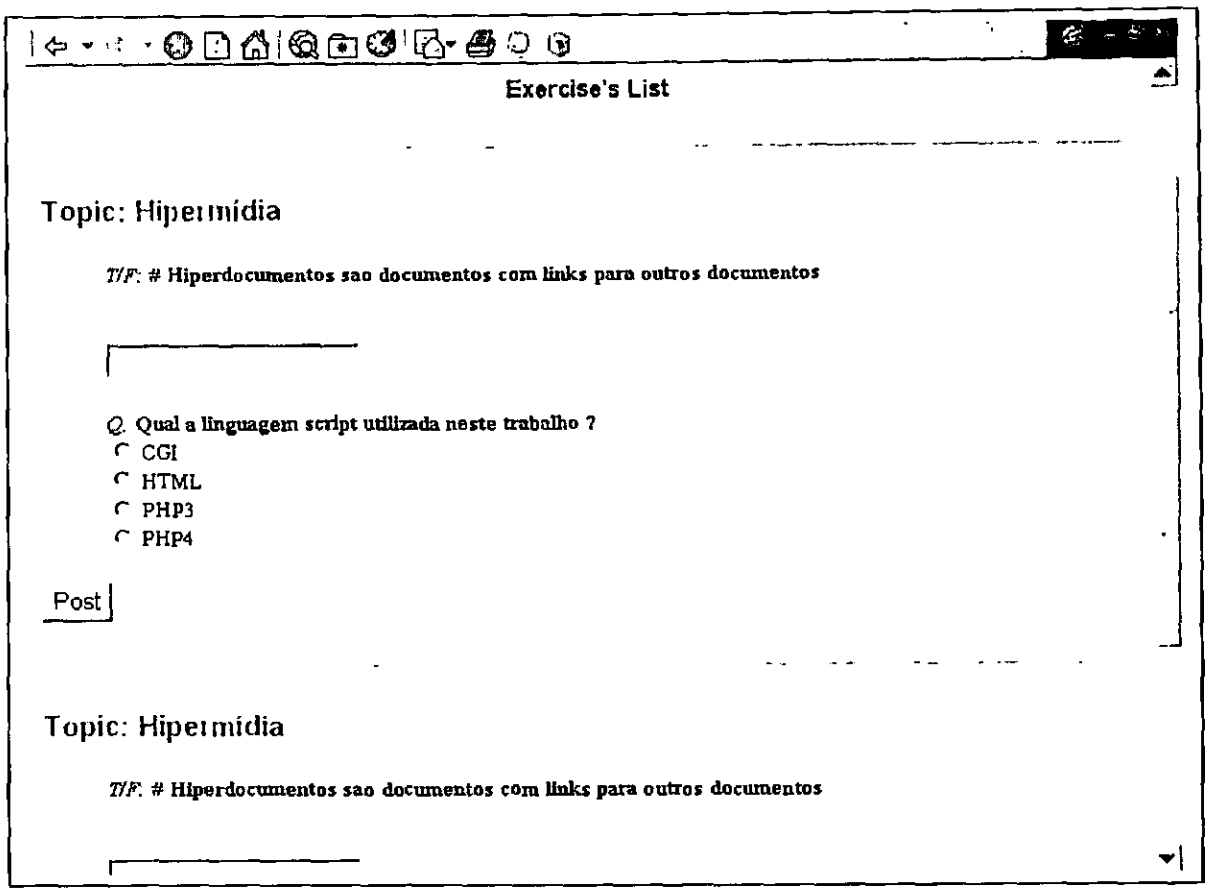

Figura 19 - Tela de visualização do documento XML da Figura 20

Definidos os documentos XML, DTD e XSL, é necessário construir programas para geração, *sob demanda*, dos documentos XML.

4.3.4. Criação de Programas para Geração de Documentos XML

A aplicação AulaML foi desenvolvida sob a arquitetura *ArqGDE* e segue a seqüência de passos de (**A**) a (**I**) daquela arquitetura para a geração dos seus documentos estruturados XML a partir de programas computacionais Java. Além disso, o programa Java desenvolvido segue as mesmas atividades definidas pelo roteiro *xRot* na sua quarta fase como, por exemplo, a importação de pacotes da API Java, o estabelecimento de conexão remota com banco de dados e, principalmente, a utilização do algoritmo recursivo GERAXML para auxiliar na geração dos documentos XML da aplicação.

A atividade de obtenção dos valores do *browser* e passagem desses valores ao programa Java de acordo com a interface CGI, atividade 3 do programa referente à quarta fase do *xRot*, é mostrada na declaração a seguir:

```
String CursoID = SCE; String AulaID = SCE229; idPresent =
        aulaml-tudo.xsl; idDTD = AulaML_1.dtd
```

Essa atividade, passos **A** e **B** da arquitetura, faz com que o programa Java receba os parâmetros associados à geração do documento XML e repasse ao algoritmo GERAXML o elemento DTD

guardado na variável *idDTD* (AulaML_1.dtd), a *style sheet* armazenada na variável *idPresent* (aulaml-tudo.xsl) e os parâmetros associados à escolha do usuário armazenados nas variáveis *CursoID* e *AulaID* (SCE e SCE229, respectivamente). De posse desses valores, o algoritmo GERAXML gera o documento XML apropriado, passos de (**C**) a (**H**). Finalmente, o documento XML é devolvido ao *browser* (passo **I**), e processado utilizando a *style sheet* aulaml-tudo.xsl para apresentação ao usuário. A Figura 20 ilustra o documento XML gerado pelo programa Java neste caso.

Assim, seguindo o roteiro *xRot* para a autoria da aplicação AulaML, foram construídos os documentos estruturados DTD, XML e XSL associados à aplicação. Os programas Java utilizam o algoritmo GERAXML para geração dos documentos XML e fazem acesso ao banco de dados MySql (MySql, 1999) utilizando recursos *JDBC (Java DataBase Connectivity)*. A linguagem *script* utilizada para geração da porção interativa da aplicação foi PHP (Php3, 1999).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="aulaml-tudo.xsl" ?>
<!DOCTYPE aulaml SYSTEM "aulaml.dtd">
<aula id="sce229" prof="Daniel Pires" titulo="Introd. Compiladores">
<curso id="sce" nome="Ciencia da Computacao"/>
<quadro id="sce229-01" tipo="THEORIA">
<texto>
<p> Modelando Documentos Estruturados para Aplicacoes</p>
<ul><li> A ferramenta AulaML</li><li> O roteiro proposto</li></ul>
</texto>
<index termo="Hipermídia"/>
<teste>
<verdfalso resp = "Valor nao Disponivel">
<li># Hiperdocumentos sao documentos com links para outros documentos</li>
</verdfalso>
<multiplo resp = "Valor nao disponivel">
<pergunta>Qual a linguagem script utilizada neste trabalho ?</pergunta>
<ul><alternativa> CGI</alternativa><alternativa> HTML</alternativa><alternativa> PHP3</alternativa>
<alternativa> PHP4</alternativa>
</ul>
</multiplo>
</teste>
</quadro>
<quadro id="sce229-02" tipo="THEORIA">
<texto>
<p> Armazenamento de Documentos Estruturados</p>
<ul><li> O roteiro proposto</li>
</ul>
</texto>
<index termo="Hipermídia2"/>
<teste>
<verdfalso resp = "Valor nao Disponivel">
<li># Documentos estruturados são documentos hierarquicamente aninhados /li>
</verdfalso>
<multiplo resp = "Valor nao disponivel">
<pergunta>Qual a linguagem utilizada neste trabalho ?</pergunta>
<ul>
<alternativa> C
</alternativa>
<alternativa> Pascal
</alternativa>
<alternativa> Java
</alternativa>
<alternativa> Perl</alternativa>
</ul>
</multiplo>
</teste>
</quadro>
</aula>
```

Figura 20 - Exemplo de um documento XML gerado em AulaML

4.4. Recursos da Aplicação AulaML

As atividades de definição, geração e apresentação de documentos estruturados para a construção da AulaML de acordo com o *xRot* possibilitaram a criação dos seguintes recursos:

4.4.1. Reutilização de Quadros de uma Aula

A aplicação AulaML permite que um professor, ou crie seus próprios quadros, ou reutilize quadros de uma aula diferente, mesmo esta pertencendo a cursos também diferentes. A reutilização é feita através de associações. A Figura 21 apresenta um exemplo. O quadro SCE22901 pertencente a aula SCE229 do curso SCE está sendo associado a aula ADM01 do curso ADM. Para que o professor chegue a esta tela de associação na aplicação, ele deve escolher a opção *Consulta Quadros de uma Aula* na tela inicial e, em seguida, selecionar *Reutilizar Quadros*.

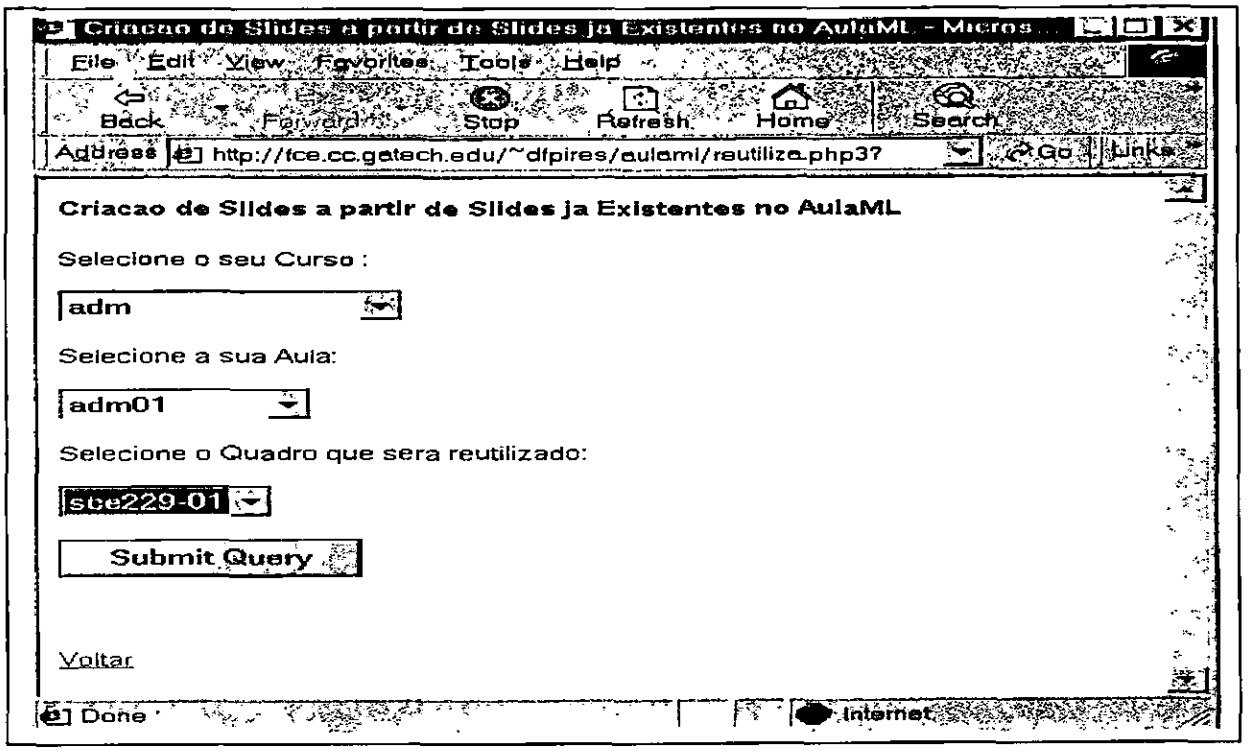

Figura 21 - Tela de reutilização de Quadros na AulaML

Caso o professor deseje criar seus próprios quadros, deve escolher a opção *Compor um novo Quadro* depois de ter passado pela tela inicial. As Figuras 22 e 23 ilustram a tela de criação de um novo quadro.

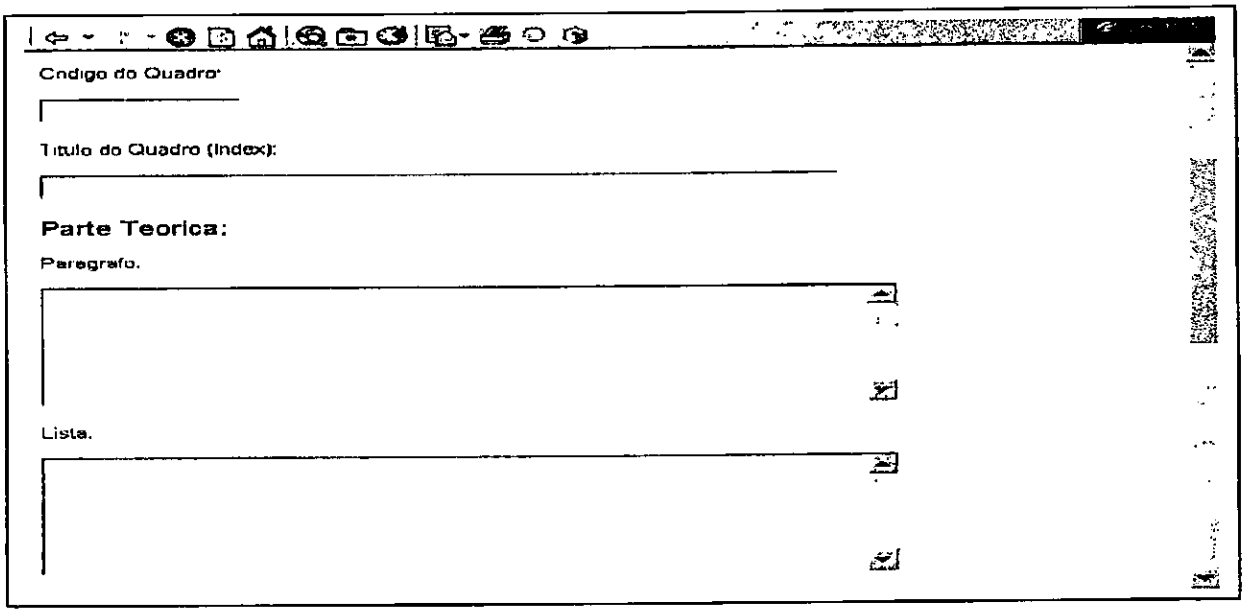

Figura 22 - Tela de criação de um novo Quadro na Aplicação AulaML (parte teórica)

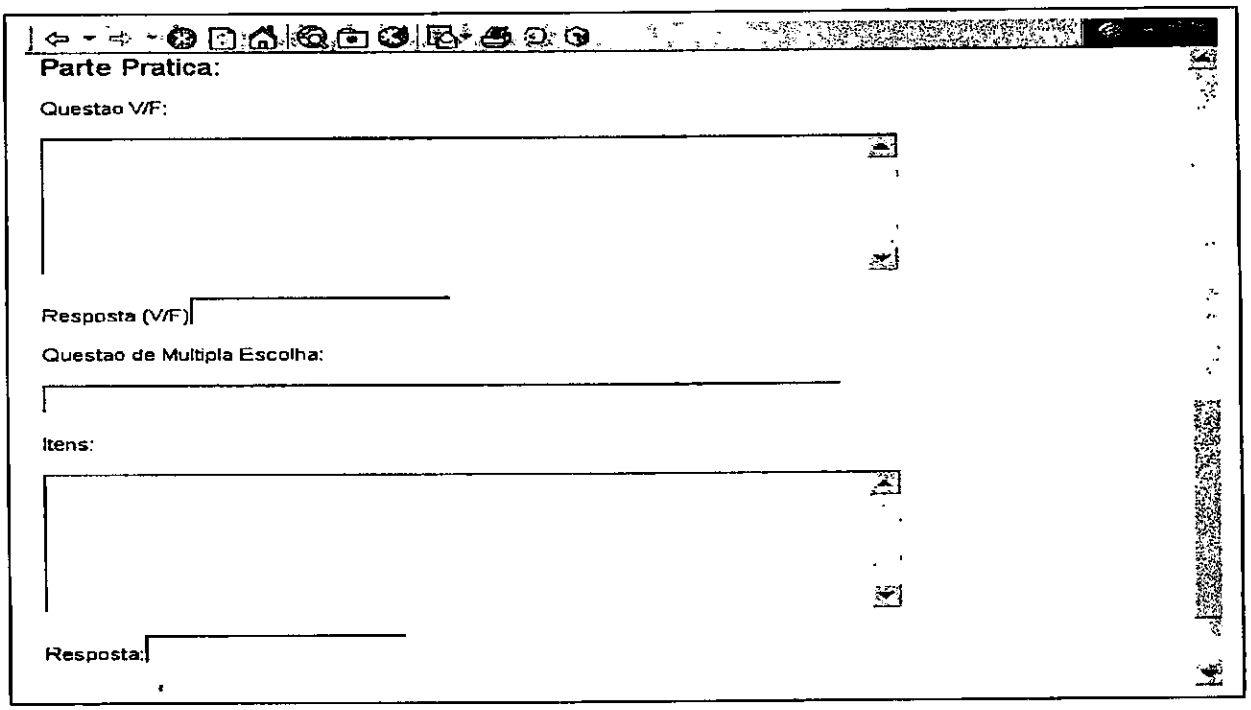

Figura 23 - Tela de criação de um novo Quadro na Aplicação AulaML (parte prática)

4.4.2. Apresentação de Quadros de uma Aula

Durante a consulta dos quadros de uma aula, é possível escolher a maneira como estes são apresentados: visualização apenas da parte teórica (listas e parágrafos), somente da parte prática (questões de múltipla escolha e verdadeiro-falso), apenas do gabarito das questões, ou ainda, de todo conteúdo existente (parte teórica, parte prática, e gabarito). Para cada uma das opções foi

especificado um documento XSL para o documento DTD AulaML_1. A Figura 24 apresenta uma tela onde o aluno escolhe um tipo de opção (Teoria, Teste, Gabarito, Tudo) para visualizar os quadros de uma aula.

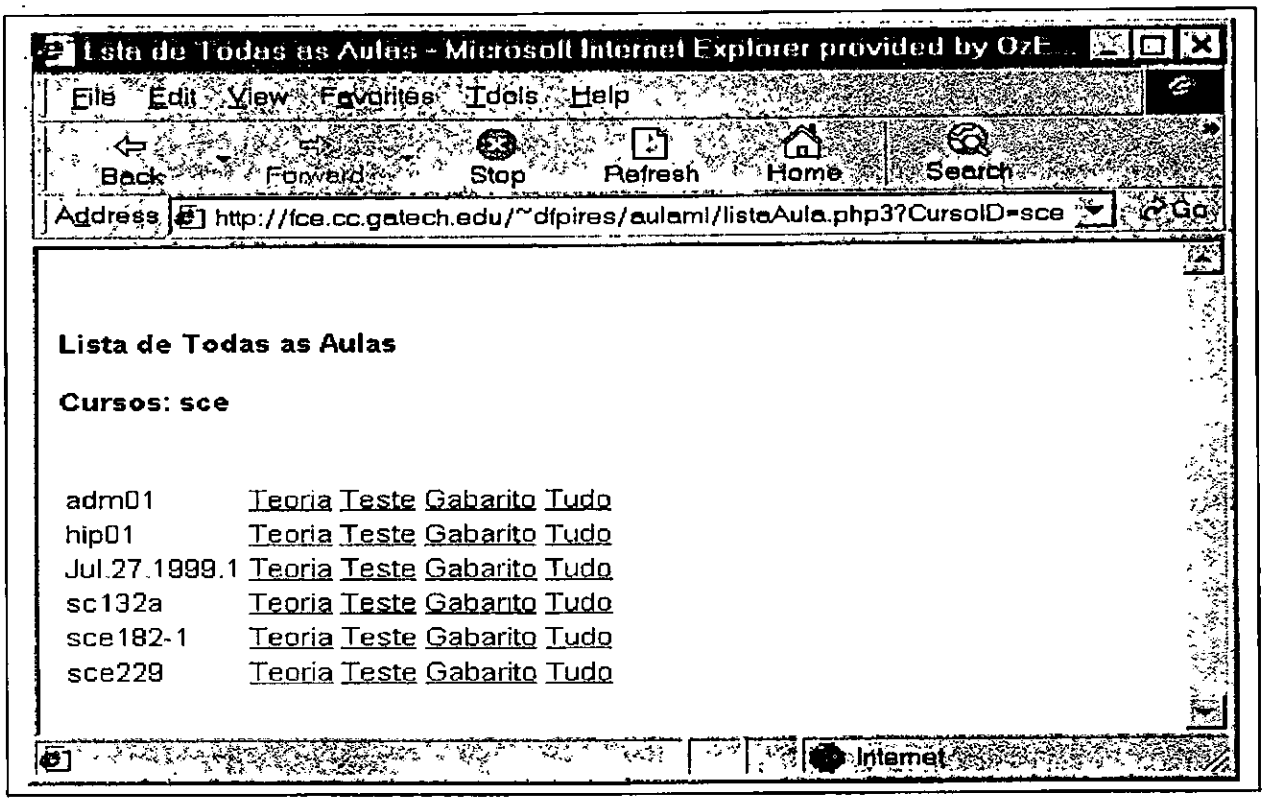

Figura 24 - Tela com opções de visualização de quadros de uma aula

4.5. Considerações Finais

A aplicação AulaML foi desenvolvida neste trabalho como estudo de caso da utilização do roteiro *xRot*. Esta aplicação oferece recursos como reutilização de quadros de aulas diferentes, mesmo estas pertencendo a cursos também diferentes, e opções de visualização dos quadros de uma aula utilizando documentos DTD, XML e XSL. Desta forma, a aplicação pode utilizar os documentos XML para intercâmbio.

Na aplicação AulaML, foi utilizado o processador XSL da Microsoft, já embutido no *browser* Explorer 5.0, e CSS para a apresentação dos documentos XML da aplicação. Outros *browsers* existentes, como o Netscape, ainda não possuem embutido um processador XSL para apresentação de documentos XML.

5. A APLICAÇÃO C2000ML

5.1. Considerações Iniciais

O Projeto Classroom 2000[8] (Abowd, 1999) está sendo desenvolvido no *Georgia Institute of Technology*, Atlanta, E.U.A, desde 1995.

A aplicação C2000ML foi desenvolvida neste trabalho explorando a infra-estrutura do projeto Classroom 2000 como estudo de caso da utilização do roteiro *xRot*.

Durante uma visita de 2 meses ao *Georgia Institute of Technology*, o autor deste trabalho conheceu o projeto com o objetivo de avaliar se as fases do *xRot* poderiam ser aplicadas a uma aplicação diferente da aplicação AulaML — a aplicação AulaML foi desenvolvida com o objetivo principal de ser um estudo de caso do *xRot*, não sendo utilizado ainda na prática. O Projeto Classroom 2000 é uma aplicação utilizada pelos alunos daquele Instituto. O roteiro *xRot* auxiliou na autoria da aplicação C2000ML, apoiando nas etapas de definição, geração e apresentação de documentos estruturados suportados pela mesma na Web.

O capítulo apresenta inicialmente uma visão geral do projeto Classroom 2000. Em seguida, é detalhado o desenvolvimento da aplicação C2000ML com base no roteiro *xRot*. Ainda, são apresentados dois serviços fornecidos ao projeto Classroom 2000 pela aplicação C2000ML.

5.2. O Contexto do Projeto Classroom 2000

O projeto Classroom 2000 possui uma infra-estrutura capaz de dar suporte para que aulas apresentadas em salas de aula convencionais sejam capturadas (áudio, vídeo e anotações em uma lousa eletrônica) automaticamente. A sala possui câmeras, microfones e uma lousa eletrônica que, juntamente a um conjunto de aplicações cliente-servidor, gravam automaticamente o conteúdo capturado de uma aula. Como resultado são produzidos, também automaticamente, documentos hipermídia que são disponibilizados na Web. Essa infra-estrutura é tratada como um ambiente de autoria hipermídia (Pimentel et al., 2000b). Com a disponibilidade dos documentos gerados automaticamente, é possível que professor, aluno ou monitor acrescentem novas informações às já existentes, como anotações e considerações discutidas pelo grupo fora do ambiente de sala de aula. O objetivo mais geral é que os documentos, assim disponibilizados, possam auxiliar os alunos em suas atividades de estudos e revisões.

Alguns termos no contexto do projeto Classroom 2000 que são utilizados nesta seção:

- *Handwriting*: A informação escrita pelo professor na lousa eletrônica é utilizada para gerar uma imagem ou *slide*. Após a aula, o professor pode associar a cada um desses slides uma transcrição de seu conteúdo, denominada *Handwriting information*.

- *Lecturer's Note*: Após a aula, o professor pode associar a cada um dos slides criados quaisquer anotações adicionais, denominadas *Lecturer's Note*.

- *CoWeb*: Quando existe necessidade de se criar um debate cooperativo sobre uma determinada aula, ou mesmo um slide em especial, é possível fazer ligações (*links*) do projeto C2000 para o ambiente de discussão cooperativa chamada *CoWeb*, onde as

[8] www.cc.gatech.edu/fce/c2000.

discussões são realizadas. Desta forma, um aluno pode acessar as ligações, dirigir-se ao ambiente e visualizar o que tem se discutido sobre determinado assunto.

A Figura 25 a seguir apresenta uma tela aonde o usuário visualiza informações de uma aula capturada do projeto Classroom 2000. Nesta figura, o *frame* à esquerda apresenta uma linha do tempo correspondente à duração de uma aula capturada. Nela, estão indicados os eventos que ocorreram durante a aula: passagem de um slide para outro (navegação que o professor fez entre os slides da aula) e *sites* visitados pelo professor durante a aula. Clicando na linha do tempo do frame à esquerda é possível assistir ou ouvir o que aconteceu na sala de aula no exato momento escolhido, com áudio e vídeo sincronizados (o software RealPlayer é utilizado neste caso para apresentação das mídias). O *frame* à direita apresenta outras informações capturadas de uma aula: figuras e *handwritings* dos slides, *lecturer's notes* da aula e dos slides e ligações (*links*) criados para elementos *CoWeb*.

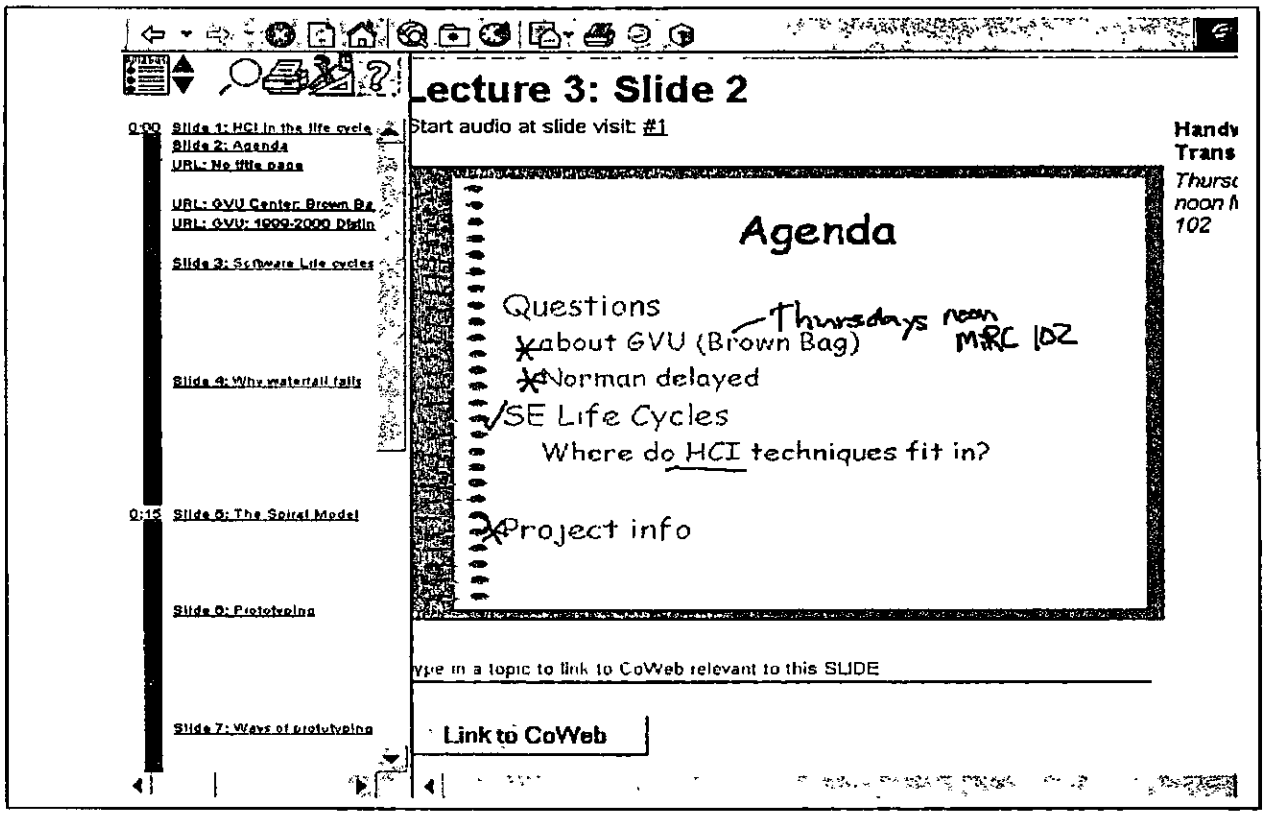

Figura 25 - Documento hipermídia do projeto Classroom 2000

5.3. Desenvolvimento da C2000ML utilizando o xRot

Os seguintes passos foram seguidos para o desenvolvimento da aplicação C2000ML:

5.3.1. Definição de Documentos XML

De acordo com o roteiro xRot, é necessário definir um tipo de documento XML para cada tipo de documento resultante dos serviços oferecidos pela aplicação. Na aplicação C2000ML são disponibilizados ao usuário 4 serviços: 1) apresentação no *browser* do usuário de uma aula

capturada, 2) apresentação da opção de editar informações de *Handwriting* e *Lecturer's Note*, 3) apresentação de todos os *slides* para a escolha daquele a ser editado, e 4) apresentação da opção de editar apenas um *slide* de cada vez com opção de navegação entre eles. Os 4 serviços resultam em 4 diferentes tipos de documento XML.

5.3.2. Criação de Documentos DTD

Definidos os documentos XML, é necessário criar os documentos DTD para a formalização dos mesmos. Segundo o *xRot*, a criação dos documentos DTD correspondentes aos documentos XML baseia-se nas tabelas do banco de dados e no modelo conceitual da aplicação.

Como foram criados 4 documentos DTD nesta aplicação, será ilustrado nesta seção apenas 1 deles já que os passos para a criação dos outros 3 documentos DTD são bastante semelhantes. O documento DTD escolhido foi referente ao primeiro serviço da aplicação C2000ML (apresentação no *browser* do usuário de uma aula capturada).

O serviço de apresentação no *browser* de informações de uma aula capturada envolve as entidades *Course*, *Lecture*, *CoWeb*, *Slides*, *Handwriting*, *LectureEdit*, *Session* e o relacionamento entre elas. O método similar ao de (Bos, 1997) foi utilizado na definição dos elementos DTD correspondentes a essas entidades a partir das tabelas do banco de dados da aplicação C2000ML que representam as entidades da aplicação.

A Figura 26 apresenta o modelo conceitual do projeto Classroom 2000 ilustrando o relacionamento entre essas entidades. Um *Course* pode possuir várias *Lecture*, uma *Lecture* pode conter vários *Slides* e várias ligações para a entidade *CoWeb*. Os *Slides*, por sua vez, podem apresentar *Handwritings*, possuir várias ligações para a entidade *CoWeb*, conter *Lecurer's Note*, e possuir *Session*. Nessa figura, é possível perceber que os elementos *Course*, *Lecture* e *Slide* estão envolvidos em um tipo de relacionamento 1:N do lado "*1*". De acordo com extensão do procedimento proposto em (Bos, 1997) no *xRot*, a definição desses elementos é feita como uma composição do outro elemento referente à entidade envolvida no relacionamento do lado "*N*". Além disso, a cardinalidade dos elementos que compõem os elementos *Course*, *Lecture* e *Slide* é definida de acordo com a cardinalidade dada pelo relacionamento.

O relacionamento binário **1:N** entre os elementos *Course* e *Lecture* indica a cardinalidade do elemento *Lecture* que compõe o elemento *Course*. De acordo com o procedimento da segundo fase do roteiro *xRot*, a cardinalidade é definida de acordo com a cardinalidade dada pelo relacionamento — 0:1, 0:N, 1:1 e 1:N que correspondem, respectivamente, a *entidade?*, *entidade**, *entidade* e *entidade*$^+$. Assim, o elemento *Course* fica assim definido: *Course* (*Lecture*+). Esse elemento faz parte do documento DTD C2000_1 da Figura 27. Já o relacionamento binário **1:N** entre os elementos *Lecture*, *Slide* e *Coweb* indica a cardinalidade dos elementos *Slide* e *Coweb* que compõem o elemento *Lecture*. Assim, o elemento *Lecture* fica assim definido: *Lecture* (*Slide*+, *Coweb**). Esse elemento também faz parte do documento DTD C2000_1. Finalizando, o relacionamento binário **1:N** entre os elementos *Slide*, *Images*, *Handwriting*, *Coweb*, *Session* e *LectureEdit* indica a cardinalidade dos elementos *Images*, *Handwriting*, *Coweb*, *Session* e *LectureEdit* que compõem o elemento *Slide*. Assim, o elemento *Slide* fica assim definido: *Slide* (*Images**, *Handwriting**, *Coweb**, *Session** e *LectureEdit**). Esse elemento também faz parte do documento DTD C2000_1.

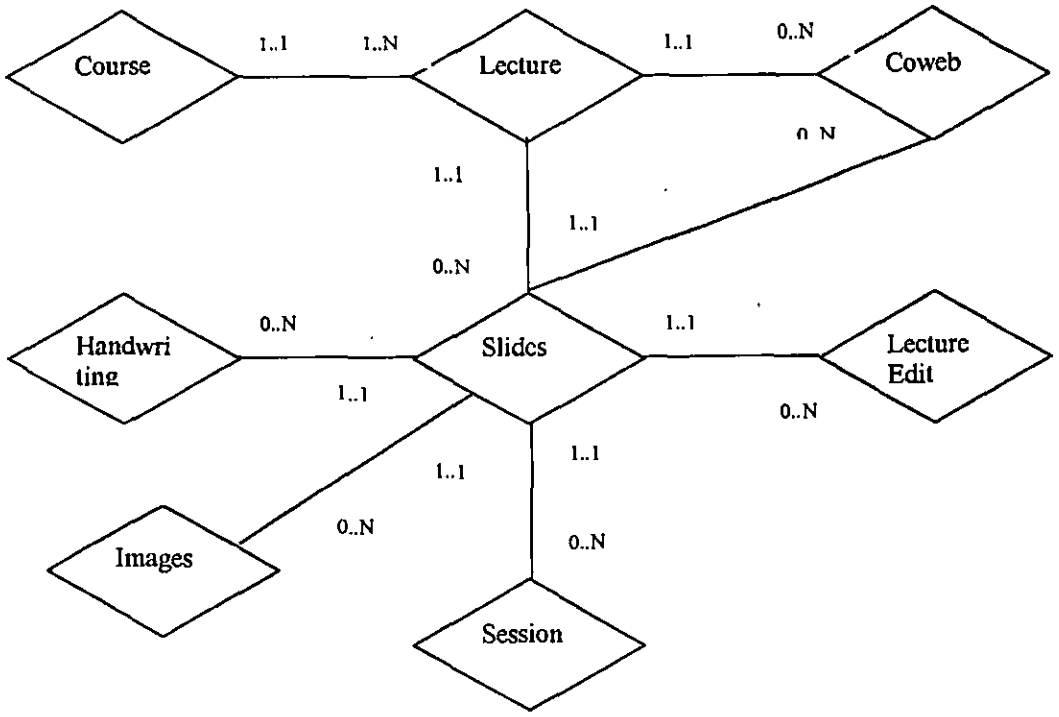

Figura 26 - Modelo de Dados do projeto Classroom 2000

Dessa forma, foi definido o documento DTD C2000ML_1 que corresponde à formalização dos seus respectivos documentos XML gerados na aplicação. Os outros 3 documentos DTD: C2000_2ML do segundo serviço, C2000_3ML do terceiro serviço e C2000_4ML do quarto serviço são apresentados no Anexo C.

```
<!ELEMENT COURSE (LECTURE+)                                                    >
<!ATTLIST COURSE
          COURSEID      ID          #REQUIRED
          COURSECODE    CDATA       #REQUIRED
          COURSENAME    CDATA       #REQUIRED
          INSTRUCTOR    CDATA       #REQUIRED                                >
<!ELEMENT LECTURE (SLIDE+, COWEB*)                                             >
<!ATTLIST LECTURE
          LECTUREID     ID          #REQUIRED
          DATE          CDATA       #REQUIRED
          LECTURETITLE  CDATA       #REQUIRED
          STARTTIME     CDATA       #REQUIRED
          ENDTIME       CDATA       #REQUIRED
          PART          CDATA       #REQUIRED
          AUDIOFILE     CDATA       #REQUIRED
          VIDEOFILE     CDATA       #REQUIRED                                >
<!ELEMENT SLIDE (IMAGES*, HANDWRITING*, COWEB*, SESSION*, LECTUREEDIT*)        >
<!ATTLIST SLIDE
          SLIDEID       ID          #REQUIRED
          SLIDENUMBER   CDATA       #REQUIRED                                >
<!ELEMENT IMAGES        EMPTY                                                  >
<!ATTLIST IMAGES
          SRC           CDATA       #REQUIRED                                >
<!ELEMENT HANDWRITNG (RECOGNIZEDTEXT)                                          >
<!ATTLIST HANDWRITING
          HANDWRITINGID ID          #REQUIRED                                >
<!ELEMENT RECOGNIZEDTEXT (#PCDATA)                                             >
<!ELEMENT COWEB (SUBJECT, URL)                                                 >
<!ATTLIST COWEB
          COWEBID       ID          #REQUIRED                                >
<!ELEMENT SUBJECT       (#PCDATA)                                              >
<!ELEMENT URL           (#PCDATA)                                              >
<!ELEMENT SESSION EMPTY                                                        >
<!ATTLIST SESSION
          SESSIONID     ID          #REQUIRED
          TIMEOFFSET    CDATA       #REQUIRED
          OPERATION     CDATA       #REQUIRED                                >
<!ELEMENT LECTUREEDIT (ANNOTATION)                                             >
<!ATTLIST LECTUREEDIT
          LECTUREID     ID          #REQUIRED                                >
<!ELEMENT ANNOTATION    (#PCDATA)                                              >
<!ELEMENT COWEBINFO (PAGETITLE, URL)                                           >
<!ATTLIST COWEBINFO
          COWEBINFOID   ID          #REQUIRED                                >
<!ELEMENT PAGETITLE     (#PCDATA)                                              >
```

Figura 27 - Documento DTD C2000_1ML

5.3.3. Construção de Especificações XSL

A Figura 28 apresenta, no plano de fundo, uma tela com todas as informações correspondentes a uma aula capturada e, no plano da frente, apenas as informações de *Handwriting* associadas. Essas informações são geradas com duas *style sheets* diferentes. O documento XSL da Figura 29 apresenta o exemplo correspondente à visualização da informação de *Handwriting*. Nessa figura, a declaração:

```
<xsl:value-of select=".course/lecture/@lecturetitle"/>
```

especifica que sua ocorrência deve ser substituída pelo conteúdo do atributo *lecturetitle* do elemento <*lecture*> do documento XML.

No total, 5 *style sheets* foram definidas para o DTD C2000_1ML (visualização apenas de Handwriting como apresentado, visualização apenas de *Lecturer's Note*, somente de ligações

para a entidade *Coweb*, apenas figuras dos slides capturados durante a aula, ou ainda, todas as opções existentes (*Handwritings*, *Lecturer's Note*, *Coweb* e *Figuras*)); outras 4 *style sheets* foram definidas para os outros 3 DTDs gerados para a aplicação. A associação entre as *style sheets* e os DTDs são explicados mais adiante. As *style sheets* XSL não apresentadas nesta seção encontram-se no Anexo A.

Definidos os documentos XML, DTD e XSL, é necessário construir programas para geração, *sob demanda*, dos documentos XML.

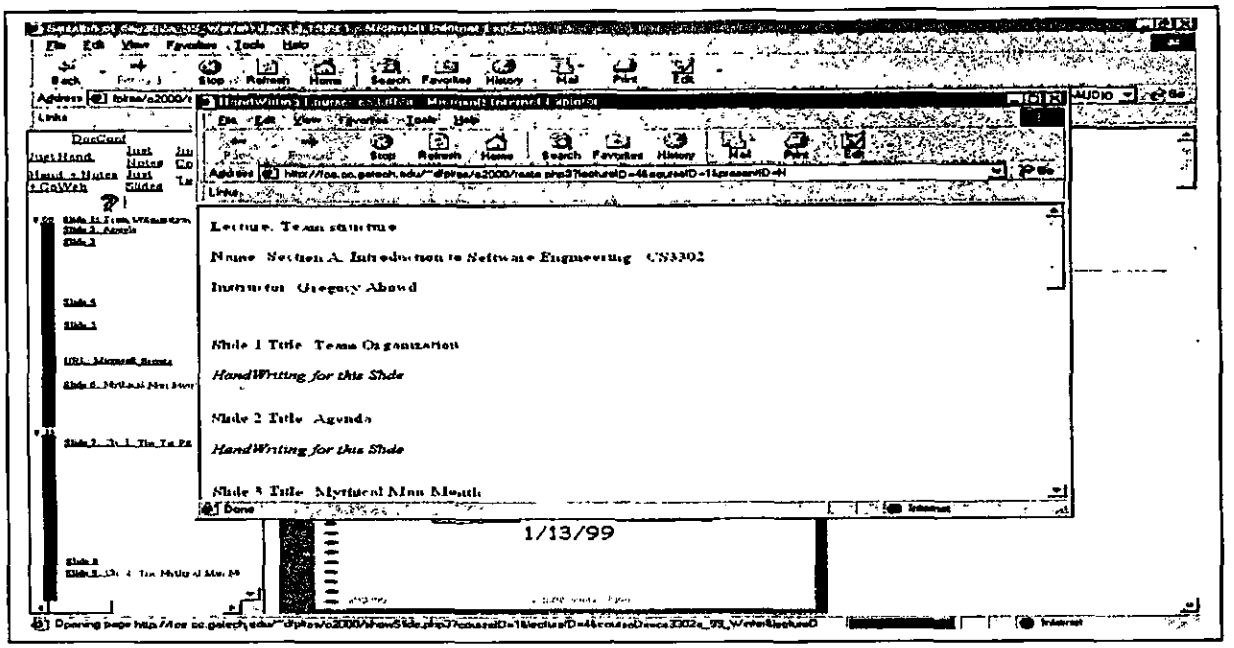

Figura 28 - Tela de visualização do documento XML

```
<?xml version="1.0 encoding="ISO-10646-UCS-2"
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-
<xsl:template
 <HTML>
  <STYLE type="text/css">    BODY {font-family:
   H1, H2, H3 {font-family: Times, Helvetica, sans-serif; font-
   H1 {color: blue;}    H2 {color: navy;}    P {margin-left:
   SUP {font-size: x-small; color: #666666;}    DIV.endnote P {text-indent: -1em; color:
   .link {text-decoration:

  <HEAD>    <TITLE>        HandWriting Course: <xsl:value-of
   </TITLE>  </HEAD>
   <h1> Lecture: <xsl:value-of select="course/lecture/@lecturetitle" />
      Name: <xsl:value-of select="course/@coursename"/>
      Instructor: <xsl:value-of select="course/@instructor"/>
   </h1>    <xsl:apply-templates select="course/lecture/slide"/>
  </BODY> </HTML></xsl:template><xsl:template
 <xsl:for-each select="."> <h2>    Slide <xsl:value-of
      Title: <xsl:value-of select="./@title"/>    </h2>    <I> HandWriting for this
   <xsl:apply-templates /> </xsl:for-each> </xsl:template> <xsl:template
 <xsl:for-each                <P/><xsl:apply-templates
 </xsl:for-each> </xsl:template> <xsl:template match="recognizedtext"> <xsl:for-each
                    <xsl:value-of select="." />
 </xsl:for-each>
```

Figura 29 - *Style sheet* XSL para apresentação de *Handwritings* de uma *Lecture*

5.3.4. Criação de Programas para Geração de Documentos XML

A aplicação C2000ML foi desenvolvida, semelhante à aplicação AulaML, também sob a arquitetura *ArqGDE* e segue a seqüência de passos de (**A**) a (**I**) daquela arquitetura para a geração dos seus documentos estruturados XML a partir de programas computacionais Java. As mesmas atividades de um programa computacional definidas pelo roteiro *xRot* na quarta fase foram seguidas pelos programas Java desenvolvidos nessa aplicação como, por exemplo, a importação de pacotes da API Java, o estabelecimento de conexão remota com banco de dados e, principalmente, a utilização do algoritmo recursivo GERAXML para auxiliar na geração dos documentos XML da aplicação. De modo particular a essa aplicação, 4 programa Java foram desenvolvidos para a geração de 4 tipos diferentes de documentos XML.

A atividade de obtenção dos valores passados ao programa Java de acordo com a interface CGI é mostrada na declaração a seguir:

```
String CourseID = Cs3302-a; String LectureID = 22; idPresent =
                Slide.xsl; idDTD = C2000ML_2.dtd
```

Essa atividade, passos **A** e **B** da arquitetura, faz com que o programa Java receba os parâmetros associados à geração do documento XML e repasse ao algoritmo GERAXML o elemento DTD guardado na variável *idDTD* (C2000ML_2.dtd), a *style sheet* armazenada na variável *idPresent* (Slide.xsl) e os parâmetros associados à escolha do usuário armazenados nas variáveis *CourseID* e *LectureID* (Cs3302-a e 22, respectivamente). De posse desses valores, o algoritmo GERAXML gera o documento XML apropriado, passos de (**C**) a (**H**). Finalmente, o documento XML é devolvido ao *browser* (passo **I**), e processado utilizando a *style sheet* Slide.xsl para apresentação ao usuário. A Figura 30 ilustra o documento XML gerado pelo programa Java neste caso. O documento XML contém informações de *Course*, *Lecturer's Note* e quais os *Slides* existentes para a *Lecture*.

Assim, seguindo o roteiro *xRot* para a autoria da aplicação C2000ML, foram construídos os documentos estruturados DTD, XML e XSL associados à aplicação. Os programas Java utilizam o algoritmo GERAXML para geração dos documentos XML e fazem acesso ao banco de dados MySql (MySql, 1999) utilizando recursos *JDBC (Java DataBase Connectivity)*. A linguagem *script* utilizada para geração da porção interativa da aplicação foi PHP (Php3, 1999).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="Slide.xsl" ?>
<!DOCTYPE C2000ML SYSTEM "c2000ML.dtd">
<course id="1" coursecode="cs3302a" coursename="Section A: Introduction to Software Engineering - CS3302" instructor="Gregory Abowd">
<lecture id="1" date="Jan.06.1999" lecturetitle="Introduction" starttime="13:06:13" endtime="13:56:49" part="1" audiofile="Jan.06.1999.1.ra"
videofile="Jan.06.1999.1.rm">
<slide id="521" number="1" update="1" title="">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/1.gif"/>
</slide><slide id="522" number="2" update="1" title="Introductions">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/2.gif"/>
</slide><slide id="523" number="3" update="1" title="Introductions">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/3.gif"/>
</slide><slide id="524" number="4" update="1" title="Introductions">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/4.gif"/>
</slide><slide id="525" number="5" update="0" title="Now it's your turn">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/5.gif"/>
</slide><slide id="526" number="6" update="0" title="Your thoughts">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/6.gif"/>
</slide><slide id="527" number="7" update="0" title="full life-cycle of activities">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/7.gif"/>
</slide><slide id="528" number="8" update="0" title="Course Objectives">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/8.gif"/>
</slide><slide id="529" number="9" update="0" title="Failure not always a bad thing">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/9.gif"/>
</slide><slide id="530" number="10" update="0" title="Exams">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/10.gif"/>
</slide><slide id="531" number="11" update="0" title="Projects">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/11.gif"/>
</slide><slide id="532" number="12" update="0" title="Project structure">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/12.gif"/>
</slide><slide id="533" number="13" update="0" title="Project advice">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/13.gif"/>
</slide><slide id="534" number="14" update="1" title="Capture of live lecture for later review">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/14.gif"/>
</lecture></course>
```

Figura 30 - Documento XML com informações dos Slides de uma *Lecute* no C2000ML

5.4. Recursos Adicionados ao Projeto Classroom 2000

As atividades de definição, geração e apresentação de documentos estruturados para a construção da C2000ML de acordo com o *xRot* possibilitaram a criação dos seguintes recursos no projeto Classroom 2000:

- **possibilitar ao aluno escolher o tipo de visualização de uma aula capturada .** Este recurso já estava implementado no projeto Classroom 2000 quando foi iniciado o desenvolvimento da aplicação C2000ML. O aluno que consulta o material capturado automaticamente em sala de aula pode querer visualizar apenas as figuras dos slides, ou somente as ligações de discussão que foram criadas, ou então somente *Handwritings* de um slide, ou mesmo apenas *Lecturer's Note* de slides. A Figura 31 mostra a parte da interface original alterada (*frame* à esquerda) para que o aluno pudesse visualizar uma entidade de cada vez. Essa figura apresenta ainda, um *frame* à direita com todas as informações de uma só vez (figuras dos slides, *Handwritings*, *Lecturer's Note* e ligações para a entidade *CoWeb*).

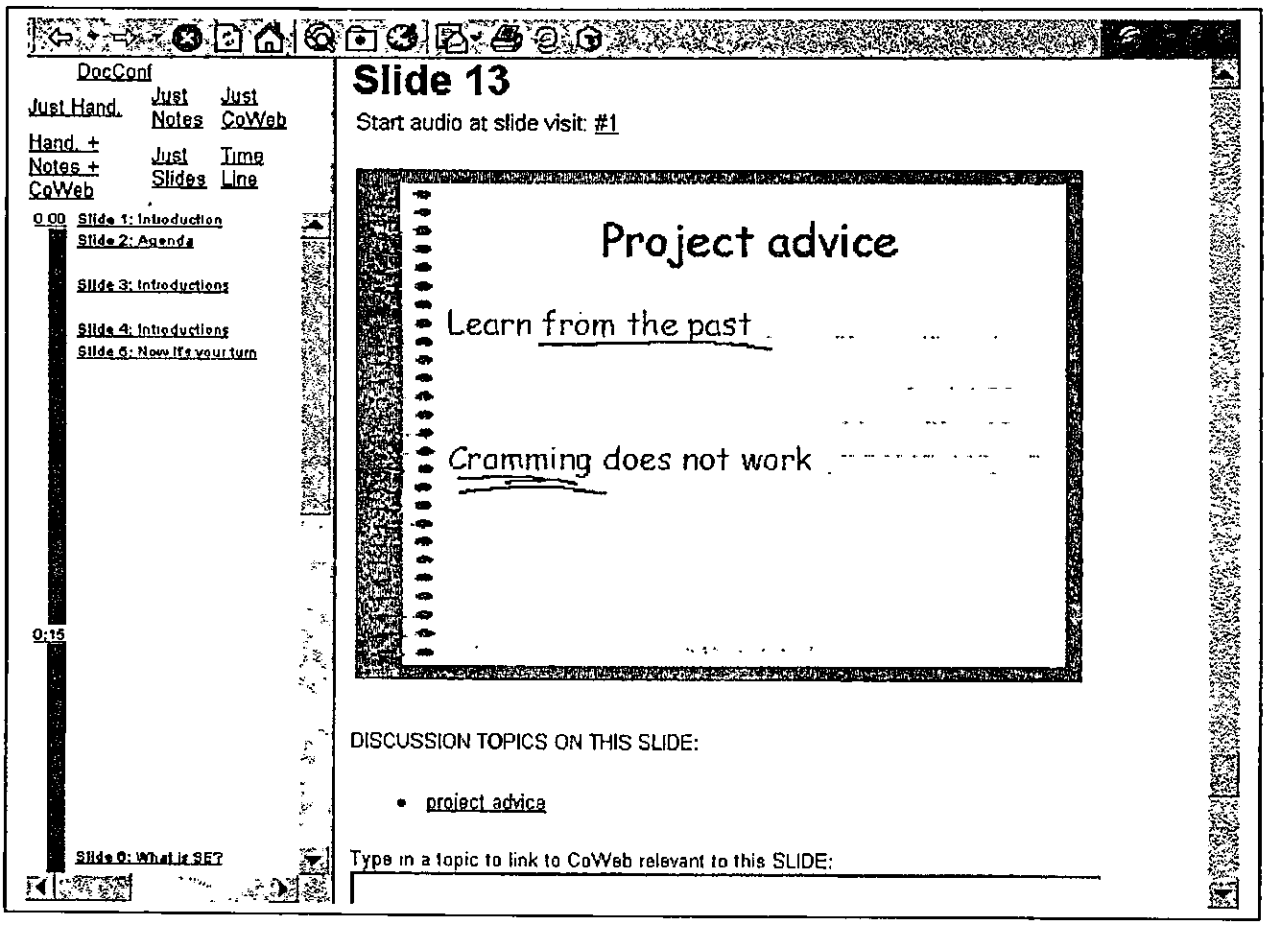

Figura 31 - Tela alterada (*frame* à esquerda) com as ligações de opções de visualização disponíveis

- **novas alternativas para editar informações após uma aula capturada.** Este recurso ainda não tinha sido desenvolvido no projeto Classroom 2000 quando do início do desenvolvimento da aplicação C2000ML. Uma das vantagens que o projeto Classroom 2000 oferece é a possibilidade de editar informações de slides após a captura automática de uma aula, como, por exemplo, *Handwritings*, *Lecturer's Note* e ligações para a *CoWeb*. Mas, muitas vezes, é necessário editar as informações apenas de uma destas entidades (evitando processamento desnecessário), ou então de todas elas, mas somente de um único slide. A Figura 32 mostra a parte da interface original do projeto alterada (*frame* à esquerda) com ligações para editar informações apenas de *Handwritings* ou apenas de *Lecturer's Note*, ou

ainda, editar informações de todas entidades, mas de apenas um único slide. Essa figura apresenta ainda, um *frame* à direita com possibilidade de editar informações de todos os slides, com todas as entidades, ao mesmo tempo.

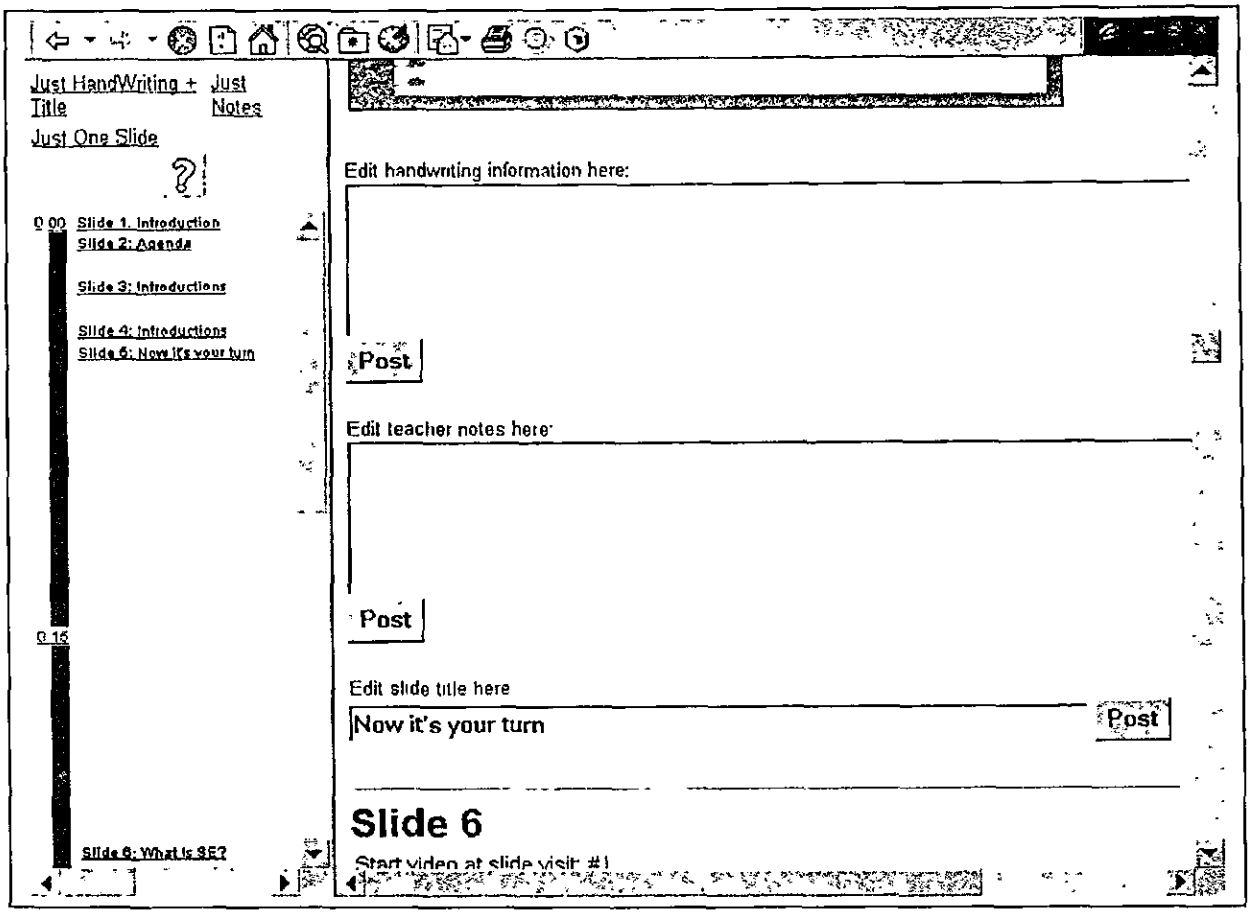

Figura 32 - Tela alterada com as ligações de opções de edição (aumentar ou alterar) disponíveis

A interface original com as modificações apresentadas nas duas figuras anteriores localiza-se em uma área no College of Computing utilizada para testes. As interfaces do projeto que realmente são utilizadas pelos alunos e professores do GeorgiaTech ainda não foram modificadas.

A seguir, nas próximas duas subseções, esses dois recursos são detalhados, indicando-se quais documentos XML, DTD e XSL foram criados para fornecer tais recursos.

5.4.1. Opções de Visualização de uma Aula Capturada

Com o objetivo de permitir que o aluno tenha opções de visualizar as aulas capturadas foram construídas 5 *style sheets* XSL. Esses documentos foram definidos para o documento DTD C2000_1ML. O documento da Figura 43 do Anexo A apresenta um exemplo. Essa *style sheet* XSL é utilizada para gerar ao usuário informações de figuras dos slides que compuseram a aula. A Figura 33 mostra o conteúdo do documento XML sendo apresentado ao usuário na Web seguindo o estilo da especificação XSL. A Figura 30 já apresentada anteriormente ilustra parte do documento XML gerado por um programa Java neste caso. 4 *style sheets* foram definidas com funcionalidade equivalente: visualização apenas de *Lecturer's Note*, apresentação somente de ligações para elementos *CoWeb*, apresentação somente de *Handwritings* e, por último, uma

style sheet que permite visualizar todas as opções existentes (*Handwritings*, *Lecturer's Note*, *Coweb* e figuras).

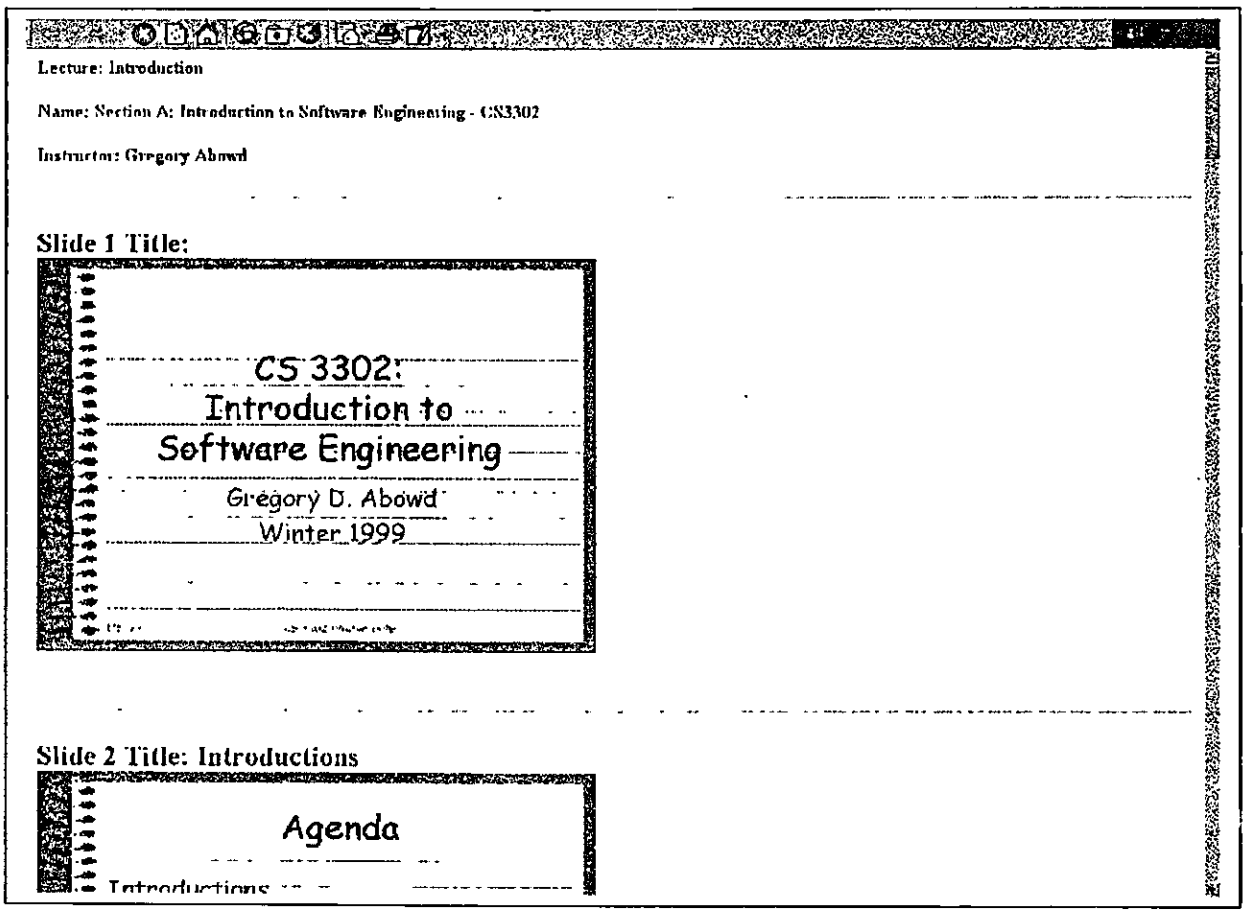

Figura 33 - Conteúdo do documento XML apresentado ao aluno

É importante ressaltar que os elementos que compõem o documento XML gerado são sempre os mesmos, independente do tipo de visualização do aluno. O que muda no documento XML é apenas o conteúdo dentro das *tags* e a linha que indica qual documento XSL será utilizado para transformar o documento XML em HTML (até o documento DTD referenciado é o mesmo).

5.4.2. Novas Alternativas para Editar Informações Após uma Aula Capturada

Para a criação de interfaces permitindo que professores e monitores tenham novas alternativas para editarem informações de um slide após uma aula já finalizada foram construídas 4 *style sheets*.

O documento C2000_2ML (Figura 47 do Anexo C) foi criado para que o professor ou monitor possa optar por editar apenas uma entidade de cada vez dos slides de uma aula. A Figura 44 do Anexo A apresenta uma *style sheet* XSL criada para esse documento DTD no caso de se editar apenas *Lecturer's Note* de um slide. As Figuras 34 e 35 mostram a interface do usuário de acordo com especificação XSL e documento XML gerado, respectivamente, para esse caso. Outro documento XSL foi construído também para o documento DTD C2000_2ML, só que para a opção de editar apenas *handwritings* de um slide.

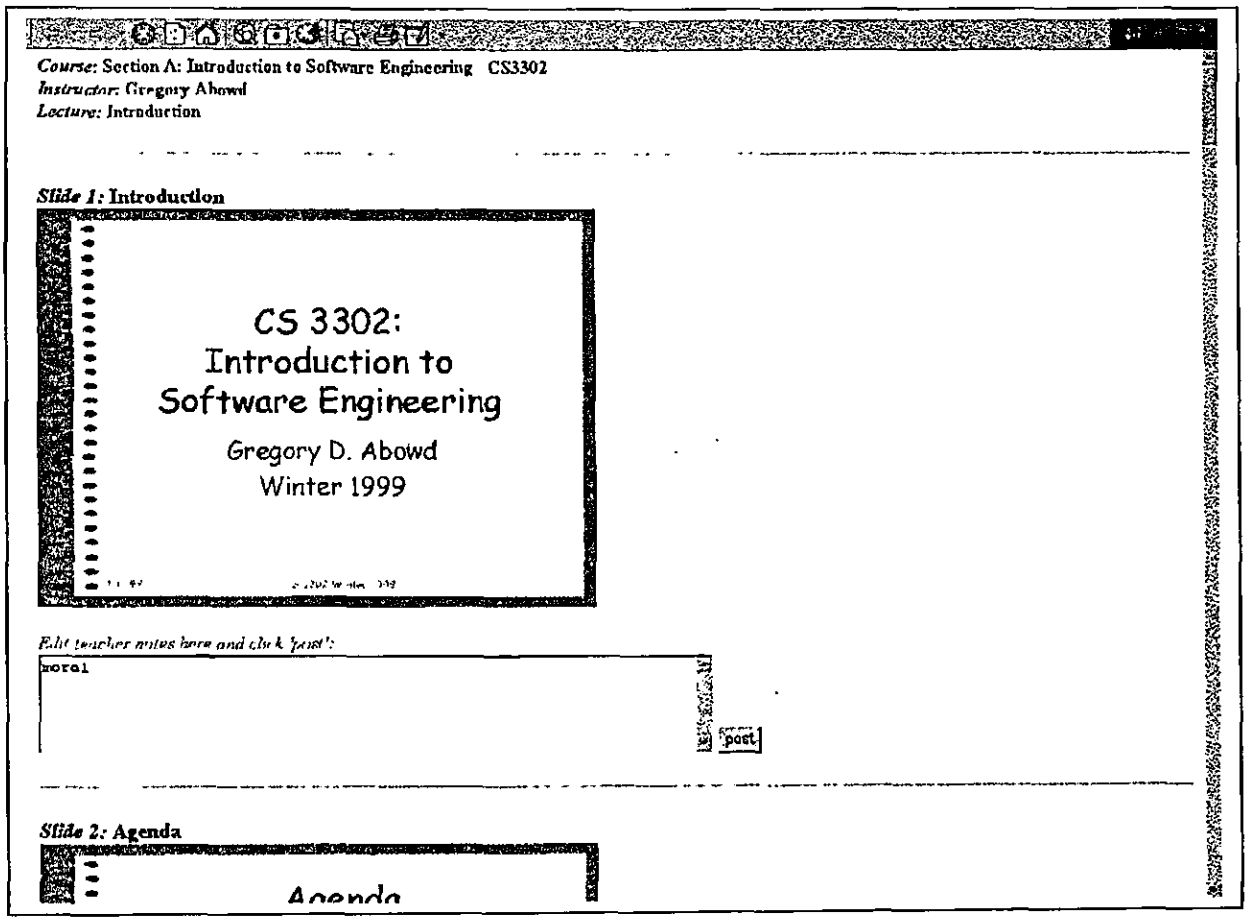

Figura 34 - Interface do usuário para alterar apenas *Lecturer's Note* dos slides

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="EditNote.xsl" ?>
<!DOCTYPE C2000ML SYSTEM "C2000_2ML.dtd">
<course id="1" coursecode="cs3302a" coursename="Section A: Introduction to Software Engineering - CS3302" instructor="Gregory
Abowd">
<lecture id="1" date="Jan.06.1999" lecturetitle="Introduction" starttime="13:06:13" endtime="13:56:49" part="1"
audiofile="Jan.06.1999.1.ra" videofile="Jan.06.1999.1.rm">
<slide id="521" number="1" title="Introduction" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/1.gif"/>
<handwriting id="12" update="1">
<recognizedtext>dfpires: testing
dfpires: testing slide gg
again
</recognizedtext>
</handwriting>
<lectureedit id="1" update="1">
<annotation>moral</annotation>
</lectureedit>
</slide>
<slide id="522" number="2" title="Agenda" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/2.gif"/>
<handwriting id="13" update="1">
<recognizedtext>dfpires: testing hw for Slide 2
</recognizedtext>
</handwriting>
<lectureedit id="1" update="1">
<annotation>dfpires: notes information</annotation>
</lectureedit>
</slide>
<slide id="523" number="3" title="Introductions" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/3.gif"/>
<handwriting id="14" update="1">
<recognizedtext>dfpires: testing EH for Slide 3
dfpires: again
</recognizedtext>
</handwriting>
<lectureedit update="0">
</lectureedit>
</slide>
<slide id="524" number="4" title="Introductions" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/4.gif"/>
<handwriting id="15" update="1">
<recognizedtext>testing again and again and again
</recognizedtext>
</handwriting>
<lectureedit id="1" update="1">
<annotation>dfpires: notes test for slide 4</annotation>
</lectureedit>
</slide>
<slide id="525" number="5" title="Now it's your turn" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/5.gif"/>
<handwriting update="0">
</handwriting>
<lectureedit update="0">
</lectureedit>
</slide>
<slide id="526" number="6" title="What is SE?" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/6.gif"/>
<handwriting update="0">
</handwriting>
<lectureedit update="0">
</lectureedit>
</slide>
<slide id="527" number="7" title="What is SE?" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/7.gif"/>
<handwriting update="0">
</handwriting>
<lectureedit update="0">
</lectureedit>
</slide>
</lecture>
</course>
```

Figura 35 - Documento XML gerado quando usuário escolhe editar apenas *Lecturer's Note*

Para o caso de editar informações referentes a um único slide foi definido o documento DTD C2000_3ML. Para este DTD foi construído a *style sheet* da Figura 45 do Anexo A. Este documento XSL apresenta todas as figuras de slides (com tamanho diminuído) de uma aula para que o usuário possa escolher o slide que será utilizado. As Figuras 36 e 37 mostram a tela que aparece ao usuário neste caso e o documento XML gerado, respectivamente. Depois de escolhido o slide, é apresentada uma interface que: permite a edição das informações (*handwritings*, *lecturer's note* e título) do slide escolhido, e cria ligações de navegação para os demais slides, anteriores ou posteriores ao slide escolhido, caso o usuário deseje alterar outro slide. Para este caso, foi criado o documento DTD C2000_4ML. A style sheet da Figura 46 do Anexo A foi construída para esse DTD . As Figuras 38 e 39 apresentam, respectivamente, a interface do usuário para esta situação e o documento XML gerado.

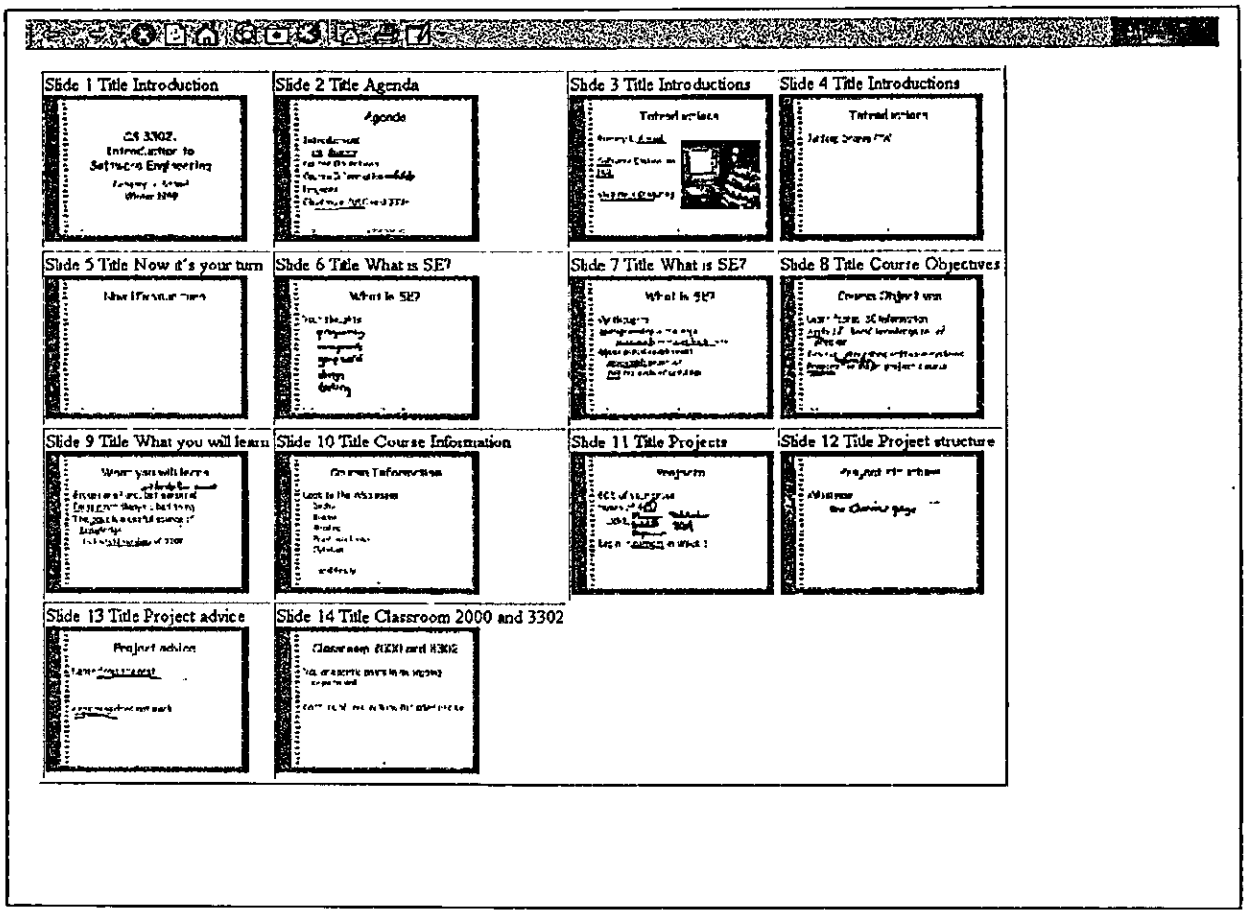

Figura 36 - Tela apresentada ao usuário contendo todos os *Slides* de uma aula

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="EditSelectSlides.xsl" ?>
<!DOCTYPE C2000ML SYSTEM "c2000ML.dtd">
<course id="1" coursecode="cs3302a" coursename="Section A: Introduction to Software Engineering -
CS3302" instructor="Gregory Abowd">
<lecture id="1" date="Jan.06.1999" lecturetitle="Introduction" starttime="13:06:13" endtime="13:56:49"
part="1" audiofile="Jan.06.1999.1.ra" videofile="Jan.06.1999.1.rm">
<slide id="521" number="1" title="Introduction" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/1-tiny.gif"/>
</slide>
<slide id="522" number="2" title="Agenda" updatetitle="1" table="3">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/2-tiny.gif"/>
</slide>
<slide id="523" number="3" title="Introductions" updatetitle="1" table="2">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/3-tiny.gif"/>
</slide>
<slide id="524" number="4" title="Introductions" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/4-tiny.gif"/>
</slide>
<slide id="525" number="5" title="Now it's your turn" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/5-tiny.gif"/>
</slide>
<slide id="526" number="6" title="What is SE?" updatetitle="1" table="3">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/6-tiny.gif"/>
</slide>
<slide id="527" number="7" title="What is SE?" updatetitle="1" table="2">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/7-tiny.gif"/>
</slide>
<slide id="528" number="8" title="Course Objectives" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/8-tiny.gif"/>
</slide>
<slide id="529" number="9" title="What you will learn" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/9-tiny.gif"/>
</slide>
<slide id="530" number="10" title="Course Information" updatetitle="1" table="3">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/10-tiny.gif"/>
</slide>
<slide id="531" number="11" title="Projects" updatetitle="1" table="2">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/11-tiny.gif"/>
</slide>
<slide id="532" number="12" title="Project structure" updatetitle="1" table="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/12-tiny.gif"/>
</slide>
<slide id="533" number="13" title="Project advice" updatetitle="1" table="0">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/13-tiny.gif"/>
</slide>
<slide id="534" number="14" title="Classroom 2000 and 3302" updatetitle="1" table="3">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/14-tiny.gif"/>
</slide>
</lecture>
</course>
```

Figura 37 - Documento XML para escolha de um slide a ser alterado

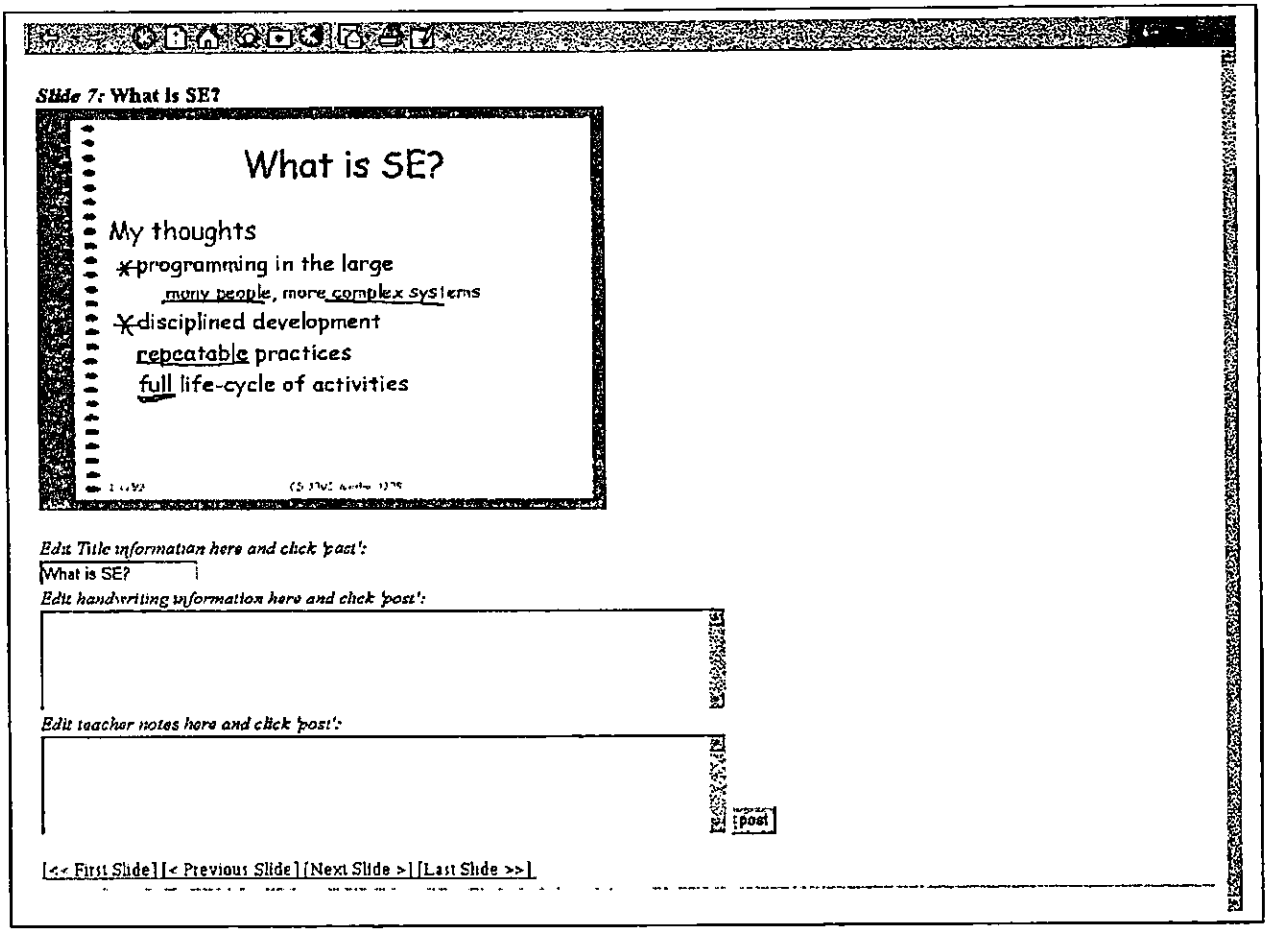

Figura 38 - Interface para alterações apenas em um slide com ligações para navegação entre os demais slides

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="EditOneSlide.xsl" ?>
<!DOCTYPE C2000ML SYSTEM "c2000ML.dtd">
<course id="1" coursecode="cs3302a" coursename="Section A: Introduction to Software Engineering -
CS3302" instructor="Gregory Abowd">
<lecture id="1" date="Jan.06.1999" lecturetitle="Introduction" starttime="13:06:13" endtime="13:56:49"
part="1" audiofile="Jan.06.1999.1.ra" videofile="Jan.06.1999.1.rm">
<slide id="527" number="7" title="What is SE?" updatetitle="1">
<imagens src="http://c2000.cc.gatech.edu/zenpad/classes/cs3302a_99_Winter/Jan.6.1999.1/gif/7.gif"/>
<navigation prevSlideNumber="6" nextSlideNumber="8" firstSlideNumber="1" lastSlideNumber="14"/>
<handwriting update="0">
</handwriting>
<lectureedit updateNote="0">
</lectureedit>
</slide>
</lecture>
</course>
```

Figura 39 - Documento XML para criar ou aumentar informações de apenas um único slide

5.5. Considerações Finais

A aplicação C2000ML foi desenvolvida neste trabalho como um novo estudo de caso do roteiro *xRot*. O Projeto Classroom 2000 possibilitou que o aluno deste trabalho pudesse explorar sua infra-estrutura para adquirir experiências na manipulação de documentos estruturados DTD e XML, e especificações XSL. Por outro lado, a aplicação C2000ML possibilitou disponibilizar novos recursos no ambiente do projeto Classroom 2000, como: a possibilidade de o aluno escolher o tipo de visualização das informações de uma aula capturada e as novas alternativas para aumentar informações de *Handwritings*, *Lecturer's Note* e título de um slide. Apesar do primeiro recurso apresentado já existir no projeto, este foi desenvolvido de uma maneira diferente, utilizando e criando documentos DTD, XML e XSL. Dessa forma, a aplicação pode utilizar os documentos XML para intercâmbio.

Da mesma maneira que a aplicação AulaML, a aplicação C2000ML utiliza o processador XSL da Microsoft, já embutido no *browser* Explorer 5.0, para a apresentação e manipulação dos documentos XML da aplicação através de especificações XSL e CSS.

6. CONCLUSÃO

6.1. Considerações Iniciais

O trabalho correspondente a esta dissertação foi motivado por apoiar o desenvolvimento de aplicações para a Web que se preocupam com o intercâmbio das informações através da utilização do padrão XML. Foi proposto o roteiro *xRot*, que orienta o desenvolvimento das aplicações para a utilização de recursos sugeridos pelo W3C — XML, DTD e XSL, nas etapas de definição, geração e apresentação de documentos estruturados manipulados por aplicações para a Internet.

6.2. Contribuições

Com a criação do roteiro proposto *xRot* e de sua arquitetura subjacente *ARGDE*, este trabalho contribuiu para a área de Hipermídia apoiando o desenvolvimento de aplicações para a Internet que manipulam documentos estruturados e que preocupam com o intercâmbio de informações.

Contribuições específicas, além da definição do roteiro *xRot* e da arquitetura *ARQGDE*, são (a) a extensão do processo sugerido por (Bos, 1997) para a criação de DTDs a partir de bancos de dados relacionais e modelos conceituais das aplicações e (b) a definição do algoritmo GERAXML.

Com a implementação das aplicações AulaML e C2000ML, experiências no desenvolvimento de aplicações que manipulam documentos estruturados na Web, a partir das recomendações da W3C, como XML e XSL, puderam ser reportadas. Essas experiências também ajudaram na criação e aperfeiçoamento do roteiro *xRot*. O roteiro pode ser utilizado para a autoria de outras aplicações para a Internet, com contexto diferente ou similar das aplicações construídas neste trabalho.

Este trabalho também contribuiu, através da criação da aplicação C2000ML, com o projeto Classroom 2000, uma vez que permitiu a produção de documentos padronizados por parte da infra-estrutura do projeto.

6.3. Trabalhos Futuros

Várias atividades foram desenvolvidas neste trabalho — criação do *xRot* e da *ARGDE* e construção das aplicações AulaML e C2000ML — e, como resultado, vários aspectos podem ser explorados na realização de trabalhos futuros.

Um modelo mais genérico do roteiro *xRot* pode ser desenvolvido de modo que não apenas o padrão XML e especificações DTD e XSL sejam utilizados pelas aplicações. A flexibilidade na escolha de qual padrão e quais especificações seria utilizada pela aplicação de acordo com sua necessidade, por exemplo, XMLSchema (Fallside, 2000) em substituição a DTD, e SAX e DOM em substituição a XSL. Também um modelo mais genérico da arquitetura *ArqGDE* pode ser construído para que as aplicações não fiquem limitadas à utilização da especificação CGI e suporte a banco de dados. Outras soluções para o desenvolvimento das aplicações seriam a utilização de Corba OMG (Atwater, 2000) ou Servlets Java (substituindo CGI) e sistema de arquivos (substituindo banco de dados).

Uma extensão do *xRot* deve ser desenvolvida para adicionar aspectos de navegação aos documentos definidos e criados de uma aplicação para Internet. Para isto, a definição e criação dos documentos poderiam fazer uso dos recursos XPointer (DeRose et al., 1999) e XLink (DeRose et al., 2000). Como estudo de caso dessa extensão do *xRot*, o C2000ML poderia ser estendido de modo a suportar o ambiente de autoria hipermídia do projeto Classroom 2000 (Pimentel et al., 2000b), e depois avaliado. Também uma alteração no *xRot* pode ser realizada para a substituição dos documentos DTD para a utilização de XMLSchema (Fallside, 2000), uma futura recomendação do W3C para formalização da estrutura dos documentos XML.

Uma formalização do processo de extensão do trabalho de (Bos, 1997) deve ser realizada na forma de um algoritmo GeraDTD para criação de documentos DTDs a partir do modelo conceitual e banco de dados relacional da aplicação.

A aplicação AulaML pode ser estendida para incluir novos recursos, entre eles: prover a correção automática de exercícios de múltipla escolha e questões de verdadeiro-falso, prover interfaces alternativas para as funcionalidades já implementadas, e integrar um ambiente de trabalho cooperativo para que alunos possam discutir tópicos específicos ao acessar a ferramenta. A construção desses recursos utilizando o roteiro *xRot* e documentos XML, XSL e DTD pode servir de avaliação do roteiro, verificando a existência de limitações e necessidade de aperfeiçoamento no mesmo.

Durante a utilização, pelos alunos, da aplicação C2000ML no ambiente do projeto Classroom 2000, seria viável um trabalho de análise quanto à velocidade de processamento. A versão original (projeto Classroom 2000) e a versão desenvolvida com documentos padronizados (C2000ML) seriam comparadas com o objetivo de verificar qual das duas tem menor tempo de processamento, tanto no lado do cliente, como do servidor.

Uma outra versão das aplicações AulaML e C2000ML poderia ser construída utilizando os recursos das APIs SAX e DOM na apresentação dos documentos XML em substituição ao processador XSL, já que nem todos *browsers* possuem embutido um processador XSL. Durante essa construção, seria viável analisar se o *xRot* suportaria essa mudança, e caso negativo, quais as modificações necessárias para o suporte. Também como trabalho futuro, seria importante desenvolver uma análise comparativa de desempenho entre as versões desenvolvidas com SAX e DOM.

REFERÊNCIAS BIBLIOGRÁFICAS

(Abowd, 1999) Abowd, Gregory. Classroom C2000: An Experiment with the Instrumentation of a Living Educational Environment. IBM Systems Journal. Special Issue on Pervasive Computing. v. 38, n. 4, p. 508-530, Outubro de 1999.

(Atwater, 2000) Atwater, Majel. The OMG's site for CORBA and UML Success Stories. [on line]. Disponível na Internet em: http://www.corba.org 2000.

(Baru et. al., 1999a) Baru, Chaitan, Gupta, Amarnath, Ludascher, Bertram, Marciano, Richard, Papakonstantinou, Yannis, Velikhov, Pavel, Chu, Vincent. XML-Based Information Mediation with MIX. IN: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data. p. 597 – 599, 1999.

(Baru et. al., 1999b) Baru, Chaitanya, Chu, Vincent, Gupta, Amarnath, Ludascher, Bertram, Marciano, Richard, Papakonstantinou, Yannis, Velikhov, Pavel. XML-Based Information Mediation for Digital Libraries. IN: Proceedings of the Fourth ACM Conference on Digital Libraries. p. 214 – 215, 1999.

(Bertino et. al., 1999) Bertino, Elisa, Castano, Silvana, Ferrari, Elena, Mesiti, Marco. IN: Proceedings of the Second International Workshop on Web Information and Data Management. p. 22 – 27, 1999.

(Bos, 1997) Bos, Bert. XML Representation of a Relational DataBase. [online]. Disponível na Internet em: http://www.w3.org/XML/RDB.html. 1997.

(Bosak, 1997) Bosak, Jon. XML, JAVA, and the Future of the Web. World Wide Web: XML Principles, Tools and Techniques. v. 2, n. 4, p. 219-227, Setembro de 1997.

(Bray et al., 1997a) Bray, Tim, Paoli, Jean, McQueen, C.M.Sperberg. Extensible MarkUp Language. XML Principles, Tools and Techniques. v. 2, n. 4, p. 29-66, Setembro de 1997.

(Bray et al., 1997b) Bray, Tim, DeRose, Steve. Extensible MarkUp Language Part 2. Linking. XML Principles, Tools and Techniques. v. 2, n. 4, p. 67-82, Setembro de 1997.

(CGI, 1999) CGI: Common Gateway Interface. [online]. Disponível na Internet: http://www.w3c.org/CGI. Outubro, 1999.

(Ciancarini et. al., 1999) Ciancarini, Paolo, Vitali, Fabio, Mascolo, Cecilia. Managing Complex Documents Over the WWW: A Case Study for XML. IEEE Transactions on Knowledge and Data Engineering. v. 11, n. 4, Agosto de 1999.

(Citr, 1997) Canadian Institute for Telecommunication Research. 1997. [online]. Disponível na Internet em: www.citr.uwaterloo.ca/

(Connolly et al., 1997) Connolly, Dan, Khare, Rohit, Rifkin, Adam. The Evolution of Web Documents - The Ascent of XML: XML Principles, Tools and Techniques. v. 2, n. 4, p. 119-228, Setembro de 1997.

(CSS, 1998) Learning CSS. [online]. Disponível na Internet em: www.w3.org/Style/CSS 1998.

(Deutsch et al., 1998) Deutsch, Alin, Fernandez, Mary, Florescu, Daniela, Levy, Alon, Suciu, Dan. XML-QL: A Query Language for XML. [online]. Disponível na Internet em: Http://www.w3.org/TR/NOTE-xml-ql/ 1998.

(DHTML, 1998) DHTML Resources Page. 1998. [online]. Disponível na Internet em: www.hotwired.com/webmonkey/dynamic_html/?tw=dynamic.html 1998.

(DOM, 1998) What is the Document Object Model? [online]. Disponível na Internet em: http://www.w3.org/TR/REC-DOM-Level-1/introduction.html 1998.

(Ferrandez, 1997) Ferrandez, Teresa. Thesis Submitted for the Degree of Master of Science in Computing Engineering. University of Dalarna. Orlando, Florida, EUA, 1997.

(Garzotto et al., 1993) Garzotto, Franca, Paolini, Paolo, Schwabe, Daniel. HDM - A Model-Based Approach to Hypertext Application Design. ACM Transaction on Information Systems. v.11, n. 1, p. 1-26, 1993.

(Herwijnen, 1994) Herwijnen, Eric Van. Practical SGML. Kluwer Academic Publishers. 1994.

(HTTP, 1999) Hypertext Transfer Protocol - HTTP/1.1. [online]. Disponível na Internet: http://www.w3.org/Protocols/rfc2616/rfc2616.html 1999.

(Isakowitz et al., 1995) Isakowitz, Tomás, Stohr, Edward A., Balasubramanian, P. RMM: A Methodology for Strucutured Hypermedia Design. Communication of the ACM. v. 38, n. 8, p. 34-44, 1995.

(ISO, 1986) ISO/IEC IS 8879. Information Processing - Text and Office Systems – Standard Generalised MarkUp Language (SGML). 1986.

(ISO, 1992) ISO/IEC 88744 Hypermedia/Time-Based Structuring Language – HyTime. 1992.

(Johnson, 1999) Johnson, Mark. XML for the Absolute Beginner. [online]. Disponível na Internet em: http://www.javaworld.com/javaworld/jw-04-1999/jw-04-xml-p.html 1999.

(Macleod, 1990) Macleod, Ian A. Storage and Retrieval of Structured Documents. Information Processing & Management. v. 26, n. 2, p. 197-208, 1990.

(Macleod, 1991) Macleod, Ian A. A Query Language for Retrieving Information from Hierarchic Text Structures. The Computer Journal. v. 34, n. 3, p. 254-264, 1991.

(Megginson, 1998) Megginson, David. SAX: The Simple API for Java. [online]. Disponível na Internet em: http://www.megginson.com/SAX/index.html 1999.

(Mm, 1999) MM MySQL JDBC Drivers. [online]. Disponível na Internet em: http://www.worldserver.com/mm.mysql/ . 1999.

(MySql, 1999) Página Oficial na Internet do Sistema Gerenciador de Banco de Dados MySQL. [online]. Disponível na Internet em: http://www.mysql.org. 1999.

(OMG, 2000) OMG News & Info. [on line]. Disponível na Internet em: http://www.omg.org/corba. 2000.

(Php3, 1999) PHP: Hypertext Preprocessor. [online]. Disponível na Internet em: http://www.php.net .1999.

(Pimentel et al., 1997) Pimentel, M. G. C., Leiva, W. D., Chorfi, M. A., Silva, R. F., Nitta, F. A., Faria, G. B. Projeto e Desenvolvimento de Ferramentas para Manipulação de Hiperdocumentos em Ambientes Abertos Distribuídos. IN: III Workshop em Sistemas Multimídia e Hipermídia. São Carlos. Maio de 1997. p. 227-238.

(Pimentel et at., 1998) Pimentel, M. G. C., Santos, J. B. J., Fortes, R. P. M. "Supporting Structured Multimedia Documents in the World Wide Web". JUCS - Journal of Universal Computer Science. Spring-Verlag. v. 4, n. 11, p. 825-838. 1998.

(Pimentel et al., 1999) Pimentel, M.G.C., Teixeira, C.A.C., Pinto, C. C. – Hiperdocumentos Estruturados na WWW: Teoria e Prática. IN: XVIII Jornada de Atualização em Informática. JAI99 –SBC. Julho de 1999. p. 367-424.

(Pimentel et al., 2000a) Pimentel, M.G.C., Pires, D.F., Teixeira, C.A.C. xRot: Um Roteiro para Autoria de Aplicações Interoperáveis para a Web. IN: Simpósio Brasileiro de Multimídia e Hipermídia, Natal - RN, 14 a 16 de Junho de 2000, pp. ainda não definido.

(Pimentel et al., 2000b) Pimentel, M.G.C., Abowd, Gregory, Yoshihide Ishiguro. IN: Proceedings of ACM Hypertext' 2000. Maio de 2000, pp. ainda não definido.

(PostgreSql, 1998) Página Oficial na Internet do Sistema Gerenciador de Banco de Dados PostgreSql. [online]. Disponível na Internet em: www.postgresql.org. 1998.

(Rada et al., 1998) Rada, R., Cargill, C., Klensin, J. Consensus and the Web. CACM. v. 41, n.7, p. 17-22, 1998.

(Royappa, 1999) Royappa, Andrew V. Implementing Catalog Clearinghouses with XML and XSL. IN: Proceedings of the 1999. ACM Symposium on Applied Computing. 1999. p. 616–621.

(Santos, 1998) Santos, J. B. J. Ferramentas para Autoria de Material Didático no Aambiente WWW. Dissertação de Mestrado. ICMC-USP. Agosto de 1998.

(Schwabe et al., 1996) Schwabe, D., Rossi, G., Barbosa, S. D. J. Systematic Hypermedia Application Design with OOHDM. IN: Proccedings of Hypertext 1996. Washington. p. 116 – 128.

(Sun, 1999) The Source for Java Technology. [online]. Disponível na Internet em: www.javasoft.com. 1999.

(Suzuki et. al., 1998) Suzuki, Junichi, Yamamoto, Yoshikazu. Managing the Software Design Documents with XML. IN: Proceedings on the Sixteenth Annual International Conference on Computer Documentation. 1998. p. 127–136.

(Teorey et al., 1986) Teorey, Toby J., Yang, Donging, Fry, James P. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. Computing Surveys – ACM. v. 18, n. 2, Junho de 1986.

(XSL, 2000) Learning XSL. [online]. Disponível na Internet em: www.w3.org/Style/XSL. 2000.

(XSLIbm, 1999) IBM AlphaWorks. [online]. Disponível na Internet em: http://www.alphaworks.ibm.com. 1999.

(XSLMic, 1999) Getting Started with XSL. XSLGuide. [online]. Disponível na Internet em: http://msdn.microsoft.com/xml/XSLGuide/xsl-overview.asp. 1999.

(Walh, 1997) Walh, Norman. A Guide to XML. XML Principles, Tools and Techniques. v. 2, n. 4, p. 97 - 107. Setembro de 1997.

(Wood et al., 2000) Wood, Lauren, Hegaret, Philippe Le. Document Object Model (DOM). [online]. Disponível na Internet em: http://www.w3.org/DOM 2000.

APÊNDICE A – *STYLE SHEETS* XSL

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<STYLE type="text/css">BODY {font-family: serif;} H1, H2, H3 {font-family: Arial, Helvetica, sans-serif;}
H1 {color: blue; text-decoration: underline;} H2 {color: green;} P {margin-left: 3em;} SUP {font-size: x-small;
color: #666666;} DIV.endnote P {text-indent: -1em; color: #333333;} .link {text-decoration:
underline;}</STYLE> <HEAD> <TITLE>
Lecture: <xsl:value-of select="aulaml/@titulo" />
</TITLE> </HEAD> <BODY> <H1> Lecture:
<xsl:value-of select="aulaml/@titulo" />
</H1> <h2> Prof.:
<xsl:value-of select="aulaml/@prof" />
</h2> <h3> <center>Exercise's List</center> </h3>
<xsl:apply-templates select="aulaml/quadro" />
<hr /> </BODY> </HTML>
</xsl:template> <xsl:template match="quadro">
<xsl:for-each select="."> <hr /> <h2> Topic:
<xsl:value-of select="./index/@termo" />
</h2> <P /> <P /> <I>T/F:</I> <B>
<xsl:value-of select="./teste/verdfalso/li" /> </B>
<FORM name="questionnaire" ACTION="testverify.php3" METHOD="POST"> <BR />
<INPUT TYPE="text" NAME="trucfalse" />
<INPUT TYPE="hidden" NAME="aulaID">
<xsl:attribute name="value">
<xsl:value-of select="../@id" />
</xsl:attribute> </INPUT> <INPUT TYPE="hidden" NAME="cursoID">
<xsl:attribute name="value">
<xsl:value-of select="../curso/@id" />
</xsl:attribute> </INPUT>
<INPUT TYPE="hidden" NAME="quadroID">
<xsl:attribute name="value">
<xsl:value-of select="./@id" />
</xsl:attribute> </INPUT>
<INPUT TYPE="hidden" NAME="TFResp">
<xsl:attribute name="value">
<xsl:value-of select="./teste/verdfalso/@resp" />
</xsl:attribute> </INPUT> <P /> <I>Q:</I> <B>
<xsl:value-of select="./teste/multiplo/pergunta" /> </B> <BR />
<xsl:for-each select="./teste/multiplo/ul">
<xsl:eval>create(0)</xsl:eval><xsl:for-each select="./alternativa">
<INPUT TYPE="radio" name="multiple"> <xsl:attribute name="value">
<xsl:eval>create(1)</xsl:eval> </xsl:attribute> <xsl:value-of select="." />
</INPUT> <BR /> <INPUT TYPE="hidden" NAME="multipleResp">
<xsl:attribute name="value"> <xsl:value-of select="../../@resp" /> </xsl:attribute> </INPUT>
</xsl:for-each> </xsl:for-each> <PRE> <input type="submit" value="Post" />
</PRE> </FORM> <xsl:apply-templates /> </xsl:for-each> </xsl:template>
<xsl:script> <![CDATA[
      function create(aux) {
        if (aux == 0) {
           aux1 = 0; }
        else { aux1 = (aux1 + 1);
               return aux1;
             }
```

Figura 40 - Apresenta apenas a parte prática dos quadros de uma aula na AulaML

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML> <HEAD> <TITLE> LECTURE: <xsl:value-of select="aulaml/@titulo" />
</TITLE> </HEAD> <BODY> <h1> Lecture: <xsl:value-of select="aulaml/@titulo" />
</h1> <h3> Prof.: <xsl:value-of select="aulaml/@prof" /> </h3> <UL> <I>
Valid for Courses: <xsl:for-each select="aulaml/curso"> <LI> <xsl:value-of select="@cod" />
<xsl:value-of select="@nome" /> </LI> </xsl:for-each> </I> </UL>
<xsl:apply-templates select="aulaml/quadro" /> <hr /></BODY> </HTML>
</xsl:template>
<xsl:template match="quadro"> <xsl:for-each select="."> <hr /> <I> Type: <xsl:value-of select="./@tipo" />
</I> <h2> Topic: <xsl:value-of select="./index/@termo" />
</h2> <xsl:apply-templates /> </xsl:for-each> </xsl:template>
<xsl:template match="item">
<xsl:for-each select="."> <P /> <xsl:apply-templates /> </xsl:for-each>
</xsl:template>
<xsl:template match="texto"> <xsl:for-each select=".">
<P /> <xsl:apply-templates /> </xsl:for-each> </xsl:template>
<xsl:template match="p">
<xsl:for-each select="."> <P> <xsl:value-of select="." /> </P>
</xsl:for-each> </xsl:template>
<xsl:template match="ul"> <xsl:for-each select=".">
<UL> <xsl:apply-templates /> </UL>
</xsl:for-each> </xsl:template>
<xsl:template match="li">
<xsl:for-each select=".">
<LI>
<xsl:value-of select="." />
</LI>
</xsl:for-each>
</xsl:template>
<xsl:template match="teste"> <p>
<h3>Test your knowledge</h3>
</p> <xsl:for-each select=".">
<P /><xsl:apply-templates />
</xsl:for-each> </xsl:template>
<xsl:template match="verdfalso">
<I>True or False?</I>
<xsl:apply-templates /> </xsl:template>
<xsl:template match="multiplo">
<P /> <I>Question:</I>
<xsl:value-of select="./pergunta" />
<xsl:for-each select="./ul"> <OL>
<xsl:apply-templates /> </OL>
</xsl:for-each> </xsl:template>
</xsl:stylesheet>
```

Figura 41 - Apresenta teoria, prática e gabarito da AulaML

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
 <HTML>
  <STYLE type="text/css">
   BODY {font-family: serif;}
   H1, H2, H3 {font-family: Times, Helvetica, sans-serif; font-size:small;}
   H1 {color: blue;}
   H2 {color: navy;}
   P {margin-left: 3cm;}
   SUP {font-size: x-small; color: #666666;}
   DIV.endnote P {text-indent: -1cm; color: #333333;}
   .link {text-decoration: underline;}
  </STYLE>
  <HEAD>
   <TITLE>
   HandWriting Course: <xsl:value-of select="course/@coursecode"/>
   </TITLE>
  </HEAD>
  <BODY>
   <h1> Lecture: <xsl:value-of select="course/lecture/@lecturetitle" /> <BR></BR>
      Name: <xsl:value-of select="course/@coursename"/> <BR></BR>
      Instructor: <xsl:value-of select="course/@instructor"/> <BR></BR>
   </h1>
   <xsl:apply-templates select="course/lecture/slide"/>
   <hr/>
  </BODY>
 </HTML>
</xsl:template>
<xsl:template match="slide">
 <xsl:for-each select="."> 
   <hr/>
          <h2>
      Slide <xsl:value-of select="./@number"/>
      Title: <xsl:value-of select="./@title"/>
     </h2>
   <I> HandWriting for this Slide </I>
   <xsl:apply-templates />
 </xsl:for-each>
</xsl:template>
<xsl:template match="handwriting">
 <xsl:for-each select="."> 
          <P/><xsl:apply-templates />
 </xsl:for-each>
</xsl:template>
<xsl:template match="recognizedtext">
 <xsl:for-each select="."> 
     <TR><TD>
          <xsl:value-of select="." />
     </TD></TR>
 </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figura 42 - Opção de escrever apenas *Handwriting*

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<STYLE type="text/css">BODY {font-family: serif;} H1 {color: blue;font-size:small;} H2 {color: navy;} P
{margin-left: 3em;} SUP {font-size: x-small; color: #666666;} DIV.endnote P {text-indent: -1em; color:
#333333;} .link {text-decoration: underline;}</STYLE>
<HEAD>
<TITLE>
Course's Slide:
<xsl:value-of select="course/@coursecode" />
</TITLE>
</HEAD>
<BODY>
<H1>
Lecture:
<xsl:value-of select="course/lecture/@lecturetitle" />
<BR />
Name:
<xsl:value-of select="course/@coursename" />
<BR />
Instructor:
<xsl:value-of select="course/@instructor" />
<BR />
</H1>
<xsl:apply-templates select="course/lecture/slide" />
<hr />
</BODY>
</HTML>
</xsl:template>
<xsl:template match="slide">
<xsl:for-each select=".">
<hr />
<h2>
Slide
<xsl:value-of select="./@number" />
Title:
<xsl:value-of select="./@title" />
<BR />
<IMG>
<xsl:attribute name="SRC">
<xsl:value-of select="./imagens/@src" />
</xsl:attribute>
</IMG>
</h2>
<BR />
<xsl:apply-templates />
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figura 43 - Apresentação apenas de figuras de slides

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<STYLE type="text/css">BODY {font-family: serif;} H1 {color: blue;font-size:small;} H2 {color: navy;font-
size:normal;} P {margin-left: 3em;} SUP {font-size: x-small; color: #666666;} DIV.endnote P {text-indent: -
1em; color: #333333;} .link {text-decoration: underline;}</STYLE>
<HEAD> <TITLE>
Course Code: <xsl:value-of select="course/@coursecode" />
</TITLE> </HEAD> <BODY> <H1>
<I>Course:</I>
 <xsl:value-of select="course/@coursename" /> <BR />
<I>Instructor:</I>
 <xsl:value-of select="course/@instructor" /> <BR />
<I>Lecture:</I>
 <xsl:value-of select="course/lecture/@lecturetitle" />
</H1> <xsl:apply-templates select="course/lecture/slide" />
<hr /> </BODY> </HTML> </xsl:template>
<xsl:template match="slide">
<xsl:for-each select="."> <hr /> <h3> <I>
Slide <xsl:value-of select="./@number" /> :
 </I> <xsl:value-of select="./@title" />
<BR /> <IMG>
<xsl:attribute name="SRC">
<xsl:value-of select="./imagens/@src" />
</xsl:attribute>
</IMG> </h3> <FONT COLOR="green">
<FORM NAME="inkEdit" ACTION="edit/postAnnotation.php3" METHOD="post">
<I>Edit teacher notes here and click 'post':</I>
<BR /> <INPUT TYPE="hidden" NAME="lectureID">
<xsl:attribute name="value"> <xsl:value-of select="../@id" />
</xsl:attribute>
</INPUT>
<INPUT TYPE="hidden" NAME="slideNumber">
<xsl:attribute name="value">
<xsl:value-of select="./@number" />
</xsl:attribute>
</INPUT>
<INPUT TYPE="hidden" NAME="update">
<xsl:attribute name="value">
<xsl:value-of select="lectureedit/@update" />
</xsl:attribute>
</INPUT>
<TEXTAREA NAME="TextArea" WRAP="hard" COLS="70" ROWS="5">
<xsl:for-each select="lectureedit/annotation">
<xsl:value-of select="." />
</xsl:for-each>
<xsl:apply-templates />
</TEXTAREA>
<INPUT TYPE="submit" VALUE="post" />
</FORM>
</FONT>
<xsl:apply-templates />
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figura 44 - Atualizar apenas *Lecturer's Note* de slides

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML> <HEAD> <TITLE> Course's Slide:
<xsl:value-of select="course/@coursecode" />
</TITLE> </HEAD> <BODY> <TABLE BORDER="2">
<xsl:for-each select="course/lecture/slide">
<xsl:if test="@table[.=0]">
<TR /> <TD> Slide <xsl:value-of select="./@number" />
Title <xsl:value-of select="./@title" />
<BR /> <A> <xsl:attribute name="HREF">
testeEditOneSlide.php3?courseID=
<xsl:value-of select="../../@id" />
<xsl:eval>create(this)</xsl:eval>
lectureID= <xsl:value-of select="../@id" />
<xsl:eval>create(this)</xsl:eval>
presentID=EOS
<xsl:eval>create(this)</xsl:eval>
curSlideNumber=
<xsl:value-of select="./@number" />
</xsl:attribute>
<IMG> <xsl:attribute name="SRC">
<xsl:value-of select="./imagens/@src" />
</xsl:attribute> <xsl:attribute name="WIDTH">170</xsl:attribute>
<xsl:attribute name="WEIGHT">170</xsl:attribute>
<xsl:attribute name="ALIGN">center</xsl:attribute>
</IMG></A> </TD> </xsl:if>
<xsl:if test="@table[.!=0]">
<TD> Slide
<xsl:value-of select="./@number" />
Title <xsl:value-of select="./@title" />
<BR /> <A>
<xsl:attribute name="HREF">
testeEditOneSlide.php3?courseID=
<xsl:value-of select="../../@id" />
<xsl:eval>create(this)</xsl:eval>
lectureID=
<xsl:value-of select="../@id" />
<xsl:eval>create(this)</xsl:eval>
presentID=EOS <xsl:eval>create(this)</xsl:eval>
curSlideNumber= <xsl:value-of select="./@number" />
</xsl:attribute> <IMG> <xsl:attribute name="SRC">
<xsl:value-of select="./imagens/@src" />
</xsl:attribute> <xsl:attribute name="WIDTH">170</xsl:attribute>
<xsl:attribute name="WEIGHT">170</xsl:attribute>
<xsl:attribute name="ALIGN">center</xsl:attribute>
</IMG> </A> </TD> </xsl:if> </xsl:for-each> </TABLE>
</BODY> </HTML> </xsl:template>
<xsl:script>
<![CDATA[
      function create(e) {
        link = '&';
        return link;
        }

]]>
</xsl:script> </xsl:stylesheet>
```

Figura 45 - Apresentação de todos slides (miniaturas) para escolha de alteração

```
<?xml version="1.0" encoding="ISO-10646-UCS-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
<STYLE type="text/css">BODY {font-family: serif;} H1 {color: blue;font-size:small;} H2 {color:
navy;font-size:normal;} P {margin-left: 3cm;} SUP {font-size: x-small; color: #666666;} DIV.endnote P
{text-indent: -1em; color: #333333;} .link {text-decoration: underline;}</STYLE>
<HEAD>
<TITLE>
Course Code:
<xsl:value-of select="course/@coursecode" />
</TITLE>
</HEAD>
<BODY>
<H1>
<I>Course:</I>
<xsl:value-of select="course/@coursename" />
<BR />
<I>Instructor:</I>
<xsl:value-of select="course/@instructor" />
<BR />
<I>Lecture:</I>
<xsl:value-of select="course/lecture/@lecturetitle" />
</H1>
<xsl:apply-templates select="course/lecture/slide" />
<hr />
</BODY>
</HTML>
</xsl:template>
<xsl:template match="slide">
<xsl:for-each select=".">
<hr />
<h3>
<I>
Slide
<xsl:value-of select="./@number" />
:
</I>
<xsl:value-of select="./@title" />
<BR />
<IMG>
<xsl:attribute name="SRC">
<xsl:value-of select="./imagens/@src" />
</xsl:attribute>
</IMG>
</h3>
<FONT COLOR="blue">
<FORM NAME="inkEdit" ACTION="edit/postInkEdit.php3" METHOD="post">
<I>Edit Title information here and click 'post':</I>
<BR />
<INPUT TYPE="hidden" NAME="updatetitle">
<xsl:attribute name="value">
<xsl:value-of select="./@updatetitle" />
</xsl:attribute>
</INPUT>
<INPUT TYPE="text" NAME="titleText">
<xsl:attribute name="value">
```

Figura 46 - Alterar informações de apenas um slide e ligações de navegação entre os demais

APÊNDICE B – ESTRUTURA DE PROGRAMAS JAVA DE GERAM DOCUMENTOS XML — FASE 4 DO ROTEIRO xRot

1. Importação de pacotes (*packets*) da API Java para que o programa tenha recursos para: manipular vetores (*java.util.**), executar comandos SQL (*java.sql.**) e conectar a uma base de dados localizada remotamente (*gwe.org*)

2. Carregamento de *driver* JDBC para fazer conexão da aplicação localizada no cliente com banco de dados remoto no servidor. Exemplo:

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

3. Estabelecimento de conexão remoto da aplicação com banco de dados no servidor. Exemplo:

```
Connection con = DriverManager.getConnection(jdbc:mysql://fce.cc.gatech.edu:3306/c2000l,
"username", "password");
```

4. Criação de uma declaração (*statement*) da conexão estabelecida necessária para consultas ao banco de dados. Exemplo:

```
Statement stmt = con.createStatement();
```

5. Quando o programa Java é executado no *browser* por um programa PHP, como:

```
<?php require("config.inc.php3");
$programa = "$javaCommand nome_programa_java \"$CursoID\" \"$AulaID\"
\"$idPresent\"";
passthru($programa); ?>,
```

é preciso obter os valores passados como parâmetro. Exemplo:

```
String CursoID = "" + argv[0] + "";
String idPresent = argv[2];
```

6. Para que um documento XML possa ser interpretado por um documento XSL, é necessário que seja indicado no documento XML o documento XSL. O documento XSL que será colocado depende de como o usuário que utiliza a aplicação deseja visualizar o documento XML. Exemplo:

```
if (idPresent.equals("ALL")) {
    System.out.println("<?xml-stylesheet type=\"text/xsl\" href=\"aulaml-tudo.xsl\" ?>");}
if (idPresent.equals("ANSWERS")) {
    System.out.println("<?xml-stylesheet type=\"text/xsl\" href=\"aulaml-gabarito .xsl\" ?>"); }
```

7. Para que um documento XML possa ser verificado e validado durante a sua criação e também durante a sua interpretação por um documento XSL, é necessário que se indique no documento XML o documento DTD. Exemplo:

```
System.out.println("<!DOCTYPE aulaml SYSTEM \"aulaml.dtd\">");
```

8. O conteúdo dos elementos que serão criados no documento XML é buscado a partir da base de dados remota através dos métodos *executeQuery* e *getString* ou *getInt* das classes Java *stmt* e *ResultSet*, respectivamente, que compõem o pacote *java.sql* importado no Passo 1. Toda busca possui esta estrutura:

```
String queryConsulta = "string compondo uma consulta ao banco de dados"
ResultSet rsConsulta = stmt.executeQuery(queryConsulta);
String NomeAluno = rsConsulta.getString(1); //primeiro campo do conjunto de tuplas resultante
System.out.println("<aluno nomealuno="NomeAluno"/>");
```

Após recuperar o resultado da consulta, o conteúdo da *string NomeAluno* é colocado como valor do atributo *nomealuno* do elemento ALUNO.

Os passos de 9 a 13 demandam maior atenção e cuidado. O documento DTD, que define a estrutura hierárquica do documento, bem como o tipo de conteúdo que cada um dos elementos do documento XML pode receber, deve ser bem analisado e estudado. Os seguintes aspectos devem ser considerados:

9. Se o documento DTD define que o elemento AULAML deve ter os atributos como:

```
<!ATTLIST aulaml
        id ID #REQUIRED
        prof CDATA #REQUIRED
        titulo CDATA #REQUIRED>
        idade CDATA >
```

na criação do documento XML pelo programa Java o elemento AULAML deve ser criado como:

```
System.out.println("<aulaml id=\""conteudo_id"\" prof=\""conteudo_prof"\"
titulo=\""conteudo_titulo" idade=\""conteudo_idade"\">");
```

sendo que o atributo idade pode ou não aparecer.

10. Se o documento DTD define o elemento AULAML como:

```
<!ELEMENT aulaml EMPTY>
<!ATTLIST aulaml
            id ID #REQUIRED
```

então este deve ser criado e fechado como:

```
System.out.println("<aulaml id=\""conteudo_id"\" />");
```

11. Se o documento DTD define os elementos AULAML, CURSO e QUADRO como:

```
<!ELEMENT aulaml (curso, quadro+)>
<!ATTLIST aulaml
        id ID #REQUIRED
<!ELEMENT curso EMPTY>
```

```
- !ATTLIST curso
        cod CDATA #REQUIRED>
<!ELEMENT quadro EMPTY>
<!ATTLIST quadro
        id ID #REQUIRED
```

então esse elemento deve ser criado e depois fechado apenas depois que os elementos CURSO e QUADRO forem, necessariamente, criados e fechados, nesta ordem. Uma estrutura de laço repetitivo deve ser criada para que, a cada elemento Quadro de uma AulaML encontrado, uma abertura e fechamento do elemento Quadro deve ser feita. Exemplo:

```
// Obtenção das informações da aula
   String queryAula = "SELECT AulaID FROM mgp_Aula where
AulaID = "+ AulaID;
   ResultSet rsAula = stmt.executeQuery(queryAula);
   ResultSetMetaData rsmdAula = rsAula.getMetaData();
   String IdentAula = rsAula.getString(1);
   System.out.println("<aulaml id=\""+IdentAula+"\">");
// Obtenção das informações do curso.
   String queryCurso = "SELECT CursoID FROM mgp_Curso where (mgp_Curso.CursoID =
"+ CursoID + ")";
   ResultSet rsCurso = stmt.executeQuery(queryCurso);
   ResultSetMetaData rsmdCurso = rsCurso.getMetaData();
   while (rsCurso.next())
     {
      String idCurso = rsCurso.getString(1);
            System.out.println("<curso id=\""+idCurso+"\" />");
     }
     // Obtenção das informações de Quadro
     // Cria vetor para o Quadro
   String queryID = "SELECT mgp_Quadro.QuadroID FROM mgp_Quadro,
mgp_Curso_Aula_Quadro where (CursoID = "+CursoID+") and
(mgp_Curso_Aula_Quadro.AulaID = " +AulaID+") and (mgp_Curso_Aula_Quadro.QuadroID
= mgp_Quadro.QuadroID)";
   ResultSet rsID = stmt.executeQuery(queryID);
   ResultSetMetaData rsmdID = rsID.getMetaData();
   Vector VQID = new Vector(qtdeQ); // cria o vetor para os quadros
   Vector VQTitulo = new Vector(qtdeQ);
   while (rsID.next())
     {
      String idID = rsID.getString(1);
      VQID.addElement(new String(idID));
     }
   rsID.close();
  // como podem existir vários Quadros para um elemento aulaml, uma                //
estrutura de laço de repetição foi criada
  for(int q=0; q <= qtdeQ - 1 ; q++) // loop do quadro
    { String idQuadro = "'" + (String) VQID.elementAt(q) + "'";
      System.out.println("<quadro id=\""+idQuadro+"\" />");
```

```
        }
System.out.println("</aulaml>");
```

12. Se o documento DTD define os elementos AULAML, CURSO e QUADRO como:

```
<!ELEMENT aulaml (curso, quadro*)>
<!ATTLIST aulaml
        id ID #REQUIRED
<!ELEMENT curso EMPTY>
<!ATTLIST curso
        cod CDATA #REQUIRED>
<!ELEMENT quadro EMPTY>
<!ATTLIST quadro
        id ID #REQUIRED
```

então esse elemento deve ser criado e depois fechado apenas depois que os elementos CURSO e QUADRO, este último se existir, forem, necessariamente, criados e fechados, nesta ordem. Uma contagem deve ser realizada para verificar a existência ou não do elemento QUADRO. Exemplo:

```
// Obtenção das informações da aula
   String queryAula = "SELECT AulaID FROM mgp_Aula where
AulaID = " + AulaID;
   ResultSet rsAula = stmt.executeQuery(queryAula);
   ResultSetMetaData rsmdAula = rsAula.getMetaData();
   String IdentAula = rsAula.getString(1);
   System.out.println("<aulaml id=\""+IdentAula+"\">");
// Obtenção das informações do curso.
   String queryCurso = "SELECT CursoID FROM mgp_Curso where (mgp_Curso.CursoID =
"+ CursoID+")";
   ResultSet rsCurso = stmt.executeQuery(queryCurso);
   ResultSetMetaData rsmdCurso = rsCurso.getMetaData();
   while (rsCurso.next())
     {
      String idCurso = rsCurso.getString(1);
            System.out.println("<curso id=\""+ idCurso+"\" />");
     }
    // Obtenção das informações de Quadro
// Numero de Quadros de uma aula
   String queryqq = "SELECT count(*) as contaQuadro FROM mgp_Curso_Aula_Quadro
where (CursoID = "+CursoID+") and (AulaID = "+ AulaID+")";
   ResultSet rsqq = stmt.executeQuery(queryqq);
   ResultSetMetaData rsmdqq = rsqq.getMetaData();
   int qtdeQ = rsqq.getInt(1);
   If (qtdeQ > 0) {
    // Cria vetor para o Quadro
   String queryID = "SELECT mgp_Quadro.QuadroID FROM mgp_Quadro,
mgp_Curso_Aula_Quadro where (CursoID = "+CursoID+") and
(mgp_Curso_Aula_Quadro.AulaID = " +AulaID+") and (mgp_Curso_Aula_Quadro.QuadroID
= mgp_Quadro.QuadroID)";
   ResultSet rsID = stmt.executeQuery(queryID);
```

```
  ResultSetMetaData rsmdID = rsID.getMetaData();
  Vector VQID = new Vector(qtdeQ); // cria o vetor para os quadros
  Vector VQTitulo = new Vector(qtdeQ);
  while (rsID.next())
    {
     String idID = rsID.getString(1);
     VQID.addElement(new String(idID));
    }
  rsID.close();
 // como podem existir vários Quadros para um elemento aulaml, uma                    //
estrutura de  laço de repetição foi criada
  for(int q=0; q <= qtdeQ - 1 ; q++) // loop do quadro
    { String idQuadro = "" + (String) VQID.elementAt(q) + "";
     System.out.println("<quadro id=\"" + idQuadro+ "\" />");
          }
        } // fim do if do qtdeQ
System.out.println("</aulaml>");
```

13. Se o documento DTD define os elementos AULAML, CURSO e QUADRO como:

```
<!ELEMENT aulaml (curso, quadro?)>
<!ATTLIST aulaml
        id ID #REQUIRED
<!ELEMENT curso EMPTY>
<!ATTLIST curso
        cod CDATA #REQUIRED>
<!ELEMENT quadro EMPTY>
<!ATTLIST quadro
        id ID #REQUIRED
```

então esse elemento deve ser criado e depois fechado apenas depois que os elementos CURSO e QUADRO, este último se existir, forem, necessariamente, criados e fechados, nesta ordem. Uma contagem deve ser realizada para verificar a existência ou não do elemento QUADRO. A diferença do Caso E) para este caso é que se o elemento QUADRO aparecer, é apenas uma única vez. Exemplo:

```
// Obtenção das informações da aula
   String queryAula = "SELECT AulaID FROM mgp_Aula where
AulaID = "+ AulaID;
   ResultSet rsAula = stmt.executeQuery(queryAula);
   ResultSetMetaData rsmdAula = rsAula.getMetaData();
   String IdentAula = rsAula.getString(1);
   System.out.println("<aulaml id=\"" + IdentAula+ "\" >");
// Obtenção das informações do curso.
   String queryCurso = "SELECT CursoID FROM mgp_Curso where (mgp_Curso.CursoID =
"+ CursoID+")";
   ResultSet rsCurso = stmt.executeQuery(queryCurso);
   ResultSetMetaData rsmdCurso = rsCurso.getMetaData();
   while (rsCurso.next())
     {
```

```
    String idCurso = rsCurso.getString(1);
          System.out.println("<curso id=\""+idCurso+"\" />");
    }
    // Obtenção das informações de Quadro
// Numero de Quadros de uma aula
   String queryqq = "SELECT count(*) as contaQuadro FROM mgp_Curso_Aula_Quadro
where (CursoID = "+CursoID+") and (AulaID = "+ AulaID+")";
   ResultSet rsqq = stmt.executeQuery(queryqq);
   ResultSetMetaData rsmdqq = rsqq.getMetaData();
   int qtdeQ = rsqq.getInt(1);
   If (qtdeQ > 0) {
    // Cria vetor para o Quadro
   String queryID = "SELECT mgp_Quadro.QuadroID FROM mgp_Quadro,
mgp_Curso_Aula_Quadro where (CursoID = "+CursoID+") and
(mgp_Curso_Aula_Quadro.AulaID = " +AulaID+") and (mgp_Curso_Aula_Quadro.QuadroID
= mgp_Quadro.QuadroID)";
   ResultSet rsID = stmt.executeQuery(queryID);
   ResultSetMetaData rsmdID = rsID.getMetaData();
   Vector VQID = new Vector(qtdeQ); // cria o vetor para os quadros
   Vector VQTitulo = new Vector(qtdeQ);
   while (rsID.next())
     {
      String idID = rsID.getString(1);
      VQID.addElement(new String(idID));
     }
   rsID.close();
  // como podem existir vários Quadros para um elemento aulaml, uma                    //
estrutura de laço de repetição foi criada
  for(int q=0; q <= qtdeQ - 1 ; q++) // loop do quadro
    { String idQuadro = "" + (String) VQID.elementAt(q) + "";
      System.out.println("<quadro id=\""+idQuadro + "\" />");
          }
        } // fim do if do qtdeQ
System.out.println("</aulaml>");
```

14. Como foi criada uma conexão (*con*) e uma *statement* (*stmt*) para que a aplicação pudesse fazer consultas ao banco de dados remoto, é necessário o fechamento das mesmas, como:

```
stmt.close();

con.close();
```

APÊNDICE C – DOCUMENTOS DTD DA APLICAÇÃO C2000ML

```
<!ELEMENT COURSE (LECTURE+)                                          >
<!ATTLIST COURSE
          COURSEID      ID        #REQUIRED
          COURSECODE    CDATA     #REQUIRED
          COURSENAME    CDATA     #REQUIRED
          INSTRUCTOR    CDATA     #REQUIRED                          >
                                                                     >
<!ELEMENT LECTURE (SLIDE+)
<!ATTLIST LECTURE
          LECTUREID     ID        #REQUIRED
          DATE          CDATA     #REQUIRED
          LECTURETITLE  CDATA     #REQUIRED
          STARTTIME     CDATA     #REQUIRED
          ENDTIME       CDATA     #REQUIRED
          PART          CDATA     #REQUIRED
          AUDIOFILE     CDATA     #REQUIRED
          VIDEOFILE     CDATA     #REQUIRED                          >
<!ELEMENT SLIDE (IMAGES*, HANDWRITING*, LECTUREEDIT*)                >
<!ATTLIST SLIDE
          SLIDEID       ID        #REQUIRED
          SLIDENUMBER   CDATA     #REQUIRED                          >
<!ELEMENT IMAGES        EMPTY                                        >
<!ATTLIST IMAGES
          SRC           CDATA     #REQUIRED                          >
<!ELEMENT HANDWRITNG (RECOGNIZEDTEXT)                                >
<!ATTLIST HANDWRITING
          HANDWRITINGID ID        #REQUIRED                          >
<!ELEMENT RECOGNIZEDTEXT (#PCDATA)                                   >
<!ELEMENT LECTUREEDIT (ANNOTATION)                                   >
<!ATTLIST LECTUREEDIT
          LECTUREID     ID        #REQUIRED                          >
<!ELEMENT ANNOTATION    (#PCDATA)                                    >
```

Figura 47 - Documento DTD C2000_2ML

```
<!ELEMENT COURSE (LECTURE+)                                          >
<!ATTLIST COURSE
          COURSEID      ID        #REQUIRED
          COURSECODE    CDATA     #REQUIRED
          COURSENAME    CDATA     #REQUIRED
          INSTRUCTOR    CDATA     #REQUIRED                          >
<!ELEMENT LECTURE (SLIDE+)                                           >
<!ATTLIST LECTURE
          LECTUREID     ID        #REQUIRED
          DATE          CDATA     #REQUIRED
          LECTURETITLE  CDATA     #REQUIRED
          STARTTIME     CDATA     #REQUIRED
          ENDTIME       CDATA     #REQUIRED
          PART          CDATA     #REQUIRED
          AUDIOFILE     CDATA     #REQUIRED
          VIDEOFILE     CDATA     #REQUIRED                          >
<!ELEMENT SLIDE (IMAGES*)                                            >
<!ATTLIST SLIDE
          SLIDEID       ID        #REQUIRED
          SLIDENUMBER   CDATA     #REQUIRED                          >
<!ELEMENT IMAGES        EMPTY                                        >
<!ATTLIST IMAGES
          SRC           CDATA     #REQUIRED                          >
```

Figura 48 - Documento DTD C2000_3ML

```
<!ELEMENT COURSE (LECTURE+)                                                    >
<!ATTLIST COURSE
          COURSEID      ID        #REQUIRED
          COURSECODE    CDATA     #REQUIRED
          COURSENAME    CDATA     #REQUIRED
          INSTRUCTOR    CDATA     #REQUIRED                                    >
<!ELEMENT LECTURE (SLIDE+)                                                     >
<!ATTLIST LECTURE
          LECTUREID     ID        #REQUIRED
          DATE          CDATA     #REQUIRED
          LECTURETITLE  CDATA     #REQUIRED
          STARTTIME     CDATA     #REQUIRED
          ENDTIME       CDATA     #REQUIRED
          PART          CDATA     #REQUIRED
          AUDIOFILE     CDATA     #REQUIRED
          VIDEOFILE     CDATA     #REQUIRED                                    >
<!ELEMENT SLIDE (IMAGES*, NAVIGATION*, HANDWRITING*, LECTUREEDIT*)             >
<!ATTLIST SLIDE
          SLIDEID       ID        #REQUIRED
          SLIDENUMBER   CDATA     #REQUIRED                                    >
<!ELEMENT IMAGES        EMPTY                                                  >
<!ATTLIST IMAGES
          SRC           CDATA     #REQUIRED                                    >
<!ELEMENT NAVIGATION    EMPTY                                                  >
<!ATTLIST NAVIGATION
          PREVSLIDENUMBER   CDATA   #REQUIRED                                  >
          NEXTSLIDENUMBER   CDATA   #REQUIRED                                  >
          FIRSTSLIDENUMBER  CDATA   #REQUIRED                                  >
          LASTSLIDENUMBER   CDATA   #REQUIRED                                  >
<!ELEMENT HANDWRITNG (RECOGNIZEDTEXT)                                          >
<!ATTLIST HANDWRITING
          HANDWRITINGID     ID      #REQUIRED                                  >
<!ELEMENT RECOGNIZEDTEXT  (#PCDATA)                                            >
<!ELEMENT LECTUREEDIT (ANNOTATION)                                             >
<!ATTLIST LECTUREEDIT
          LECTUREID     ID        #REQUIRED                                    >
<!ELEMENT ANNOTATION    (#PCDATA)                                              >
```

Figura 49 - Documento DTD C2000_4ML