ULBRA
CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Credenciado pelo Decreto de 06/07/2000 - D.O.U. nº 130 de 07/07/2000

Emilio Mario Wieczorek

Sistema de Gerenciamento de Documentos Jurídicos utilizando XML, DOM e a Plataforma .NET

Palmas

2004

ULBRA
CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Credenciado pelo Decreto de 06/07/2000 - D.O.U. nº 130 de 07/07/2000

Emilio Mario Wieczorek

Sistema de Gerenciamento de Documentos Jurídicos utilizando XML, DOM e a Plataforma .NET

"Monografia apresentada como requisito parcial da disciplina Prática em Sistemas de Informação II (TCC) do Curso de Sistemas de Informação, orientada pela Professora Parcilene Fernandes de Brito".

Palmas

2004

EMILIO MARIO WIECZOREK

SISTEMA DE GERENCIAMENTO DE DOCUMENTOS JURÍDICOS UTILIZANDO XML, DOM E A PLATAFORMA .NET

"Monografia apresentada como requisito parcial da disciplina Prática em Sistemas de Informação II do Curso de Sistemas de Informação, orientada pela Professora Parcilene Fernandes de Brito".

Aprovada em 15 de dezembro de 2004

BANCA EXAMINADORA

_____

Profª. Parcilene Fernandes de Brito - Orientadora
Centro Universitário Luterano de Palmas

_____

Profª. Thereza P. Padilha
Centro Universitário Luterano de Palmas

_____

Prof.ª Madianita Bogo
Centro Universitário Luterano de Palmas

Palmas
2004

AGRADECIMENTOS

Agradeço a Deus, por me dar apoio quando necessito; a meus familiares e minha namorada, por sempre me incentivarem na realização de meus sonhos e a minha professora orientadora que muito contribuiu para a conclusão desta monografia.

DEDICATÓRIA

Dedico esta monografia a Deus, a meus familiares e minha namorada, que sempre estiveram comigo para me auxiliar nos momentos mais difíceis de minha vida.

RESUMO

Este trabalho apresenta os passos realizados no desenvolvimento de um aplicativo para gerenciamento de documentos jurídicos, utilizando arquivos XML como a estrutura de armazenamento das informações dos documentos. A plataforma .NET e a API DOM são utilizados para processamento dos dados dos formulários e armazenamento destes em arquivos XML. Desta forma, é possível, ainda, validar o conteúdo de cada documento jurídico de acordo com uma definição de tipo de documento, DTD, criada para descrever a estrutura (elementos e relação entre estes) de cada documento.

Palavras-chave: XML, ASP.NET, documentos, jurídicos, sistema.

ABSTRACT

This work presents the steps realized to develop an application for legal documents management, using XML files the structure of storage of the information of those documents. The .Net framework and the API DOM are used to process form data and to store these data in XML files. By that way, also is possible to validate the content of each legal document, according with a document type definition, DTD, created to describe the structure (elements and the relation between them) of each document.

Key-Word: XML, ASP.NET, documents, legal, system.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Aplications Programming Interface*
DOM	*Document Object Model*
XML	*Extensible Markup Language*
PHP	*Hypertext Preprocessor*
JSP	*Java Server Pages*
SGML	*Standard Generalized Markup Language*
IIS	*Internet Informaton Service*
HTML	*HiperText Markup Language*
MSIL	*Microsoft Intermediate Language*
DTD	*Document Type Definition*
CLR	*Common Language Runtime*
XSL	*Extensible Stylesheet Language*
XSLT	*Extensible Stylesheet Language Transformations*
CLS	*Common Language Specification*
MSXML	*Microsoft XML*
HTTP	*Simple API for XML*
W3C	*World Wide Web Consortium*

1 Introdução

A área jurídica gera diariamente centenas de documentos (contratos, convênios, etc.) que muitas vezes não são sequer armazenados em um sistema computacional, ficando estes expostos a ação do tempo em armários e pastas, sujeitos a serem perdidos a qualquer momento. Formas de armazenamento e manipulação de tais informações tornam-se essenciais para garantir a integridade destes documentos jurídicos no futuro.

Hoje em dia, as tecnologias mais interessantes para se trabalhar com esta forma de documentos são tecnologias voltadas para a *web*, pois estas facilitam a integração de dados espalhado pelo globo, além de possuírem maior versatilidade, como o XML, que vem se desenvolvendo significativamente e já tem sido utilizado como padrão para transmissão de dados na *web* (SHUI, 2001).

Neste trabalho optou-se por fazer a junção das tecnologias XML, DOM e .NET para desenvolver um sistema para armazenamento e manipulação de documentos jurídicos.

Logicamente, existem diversas tecnologias que podem ser utilizadas para o desenvolvimento de um sistema para controle de documentos jurídicos, mas as tecnologias citadas acima possibilitam, dentre outras coisas, que os dados do sistema possam ser integrados com outros aplicativos, desde que estes possam converter seus dados no padrão XML aceito pelo sistema proposto.

Através destas características levantadas, podemos dizer que este trabalho tem como objetivo o desenvolvimento de um sistema de armazenamento e manipulação de documentos jurídicos. Para isso, foi utilizada a linguagem XML para a estruturação dos documentos e a API DOM para a sua manipulação na plataforma .NET, através da utilização do ASP.NET.

Este trabalho está organizado da seguinte forma: na seção 1 tem-se a introdução; a seção 2 é reservada para a revisão de literatura, oaboradnado o XML, a DTD, a linguagem

de consulat a documento XML *Xpath*, a plataforma .NET, mais especificamente falando sobre o ASP.NET e o C#, além da interface DOM; a seção 3 trata do material e métodos utilizados para o desenvolvimento do trabalho; a seção 4 aborda os resultados e discussões; na seção 5 são mostradas as conclusões deste estudo e a seção 6 é destinado às referências bibliográficas.

2 Revisão de Literatura

Neste capítulo serão abordadas as tecnologias utilizadas no desenvolvimento do sistema, exemplificando os principais recursos utilizados de cada tecnologia, além de funcionalidades gerais das mesmas, incluindo seu histórico.

2.1 XML

XML é a linguagem de marcação de dados (*meta-markup language*) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. Segundo Abiteboul (2000), o XML também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via Internet, e difere da HTML em três grandes aspectos:

- Novas marcações podem ser definidas a vontade;
- Estruturas podem ser aninhadas a profundidades arbitrárias;
- Um documento XML pode conter uma descrição opcional de sua gramática.

O XML permite que os usuários definam novas marcas para indicar estrutura. Diferente da HTML, um documento XML não fornece instruções sobre como deve ser exibido, sendo esta informação incluída separadamente em uma folha de estilo. Folhas de estilo, são utilizadas para traduzir dados XML para HTML, podendo as páginas HTML resultantes ser exibidas por navegadores padrão (ABITEBOUL, 2000). Podem ser utilizadas também linguagens de programação para a *Web*, como ASP, PHP, ASP.NET entre outras, para fazer a tradução de XML para HTML.

Em sua forma básica, XML é uma sintaxe para transmissão de dados, e como tal, é bem provável que se torne um dos padrões principais para a troca de dados na *Web*

(KADE, 2001). Para uma organização ou grupo de usuários, XML permite uma especificação que facilita a troca de dados e a reutilização por muitas aplicações. Segundo Shui (SHUI, 2001), acredita-se que a XML, ao tornar-se um padrão universalmente estabelecido, irá:

- Possibilitar a publicação eletrônica internacionalmente independente (suporte unicode e multilingüe);
- Permitir a definição de protocolos independentes de plataforma para o intercâmbio de dados;
- Disponibilizar e fornecer informação para agentes do usuário em uma forma que possibilite o processamento automático após o recebimento dos dados;
- Facilitar o desenvolvimento de *software* para manipular informação especializada distribuída através da Web;
- Facilitar às pessoas o processamento de dados utilizando *software* gratuito ou de baixo custo;
- Permitir às pessoas a apresentação da informação da forma que elas desejem, através do controle de formulários de estilo (*style sheet control*);
- Facilitar o fornecimento de meta-dados - dados sobre a informação, e auxiliar os produtores e consumidores de informação a encontrar uns aos outros.

2.1.1 O Aspecto do XML

O XML se parece bastante com o HTML, já que ambos são derivados da mesma fonte, a SGML (linguagem de marcação generalizada padrão). Como um documento HTML, um documento-fonte XML é constituído de elementos XML, cada um dos quais com uma marca de início e outra de fim (NATANYA; KIRK, 2000). As informações existentes entre as marcas são chamadas de conteúdo. Entretanto, diferentemente do HTML, o XML permite o uso de um conjunto ilimitado de marcas que indicam o significado dos dados e não a aparência dos mesmos. A figura 1 especifica que o texto usuário deve aparecer em negrito. Porém a marca de elemento XML da figura 2 especifica que as mesmas informações são um usuário de fato. Fica totalmente a cargo do criador do

documento XML determinar quais marcas são usadas e que conteúdo é colocado entre essas marcas.

```
<B>Usuário</B>
```

Figura 1: Marca de elemento HTML

```
<Usuario>Usuário Teste</Usuario>
```

Figura 2: Marca de elemento XML

No HTML, o desenvolvedor tem de escolher a partir de uma lista predeterminada de marcas, sendo que estas marcas somente servem para "classificar" de que forma o conteúdo será exibido, não sendo usadas para delimitar conteúdos, enquanto que no XML, as marcas são utilizadas para delimitar (armazenar) conteúdos. A seção a seguir irá demonstrar os componentes do XML utilizados neste trabalho.

2.1.2 Componentes XML

Nas seções a seguir, são demonstrados os elementos XML que foram utilizados neste estudo.

2.1.2.1 Elementos

Elemento é algo que descreve um dado (NATANYA; KIRK, 2000). Um elemento é diferente de uma marca usada no HTML, pois ela realmente descreve a marca e não o conteúdo. Por exemplo, a marca **<B>** no HTML descreve como o texto deve ser marcado. Por outro lado, um elemento é um aplicativo totalmente formado. Ele especifica como manipular os dados contidos entre as marcas de início e de fim.

No XML, os elementos são módulos para armazenamento de dados. No HTML, as marcas são apenas indicadores de onde algo deve mudar em termos de exibição. Cada documento XML possui um elemento principal que contém ou armazena todos os dados de todo o documento (EVANS; KAMANNA; MUELLER, 2003). O exemplo da figura 3 mostra exatamente o que isto significa.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<usuarios>
      <usuario id='1'>
            <nome>Emilio Mario</nome>
            <sobrenome>Wieczorek</sobrenome>
            <login>emw</login>
            <senha>123</senha>
            <setor>Coordenação</setor>
      </usuario>
</usuarios>
```

Figura 3: Exemplo de documento XML

Como se pode observar pela figura 3, o elemento <usuarios></usuarios> **"**armazena" todo o conteúdo do documento, sendo que todos os demais elementos como, por exemplo, <usuario></usuario>, são subelementos aninhados sob o elemento <usuarios>**.** Mais especificamente, o elemento <usuarios> é o pai, e o restante dos elementos são os descendentes. Todos os elementos mostrados na figura 3 criam a estrutura do "conjunto" de usuários, sendo que nenhum deles irá criar a sua aparência, pois isso é deixado para as marcas HTML usadas na folha de estilos que deverá ser atribuída a esse documento.

Os elementos podem conter uma série de tipos diferentes de conteúdo, inclusive:

- Dados de caracteres como textos;
- Outros elementos, chamados subelementos ou filhos;
- Seções CDATA**,** que são seções da DTD que incluem dados literais.

Os elementos são declarados dentro do documento XML ou então em uma DTD, que será vista em uma seção posterior.

2.1.2.2 Atributos

Atributos são, basicamente, simples fontes de informações adicionais sobre um elemento, e podem ser especificados e atribuídos dentro da DTD ou na marca de abertura de um elemento (NATANYA; KIRK, 2000). Na figura 3, tem-se o exemplo de um atributo id, que fará o papel de índice do usuário, fazendo assim a sua indexação.

Se fosse efetuada uma busca no documento da figura 3, e fosse procurado pelo usuário, seria mais fácil se o este fosse procurado através de seu atributo, pois não seria necessário percorrer todos os filhos do elemento usuário pesquisar por algum outro dado,

como *login*, por exemplo, o que consumiria mais processamento da máquina utilizada na pesquisa.

Para se utilizar o atributo em um documento XML, tem-se que declarar o mesmo dentro do documento XML ou então na DTD que valida este documento (NATANYA; KIRK, 2000).

2.1.2.3 Entidades

Uma entidade é qualquer unidade de dado caractere que pode ser referenciada em um documento XML, podendo ser de dois tipos: a entidade genérica, que são aquelas amplamente utilizadas no HTML para especificação de caracteres como o "e" comercial (&), acentos, <> (que são usados para definir marcas), ou qualquer caractere diferente dos básicos; e a entidade paramétrica, sendo utilizadas para agrupar elementos que repetem muito no documento XML (EVANS; KAMANNA; MUELLER, 2003).

O dado caractere, que é uma seção de texto, pode ser um dos seguintes tipos:

- Um caractere reservado que não pode ser colocado dentro de um documento XML;
- Um grupo de dados caracteres que não se quer ficar digitando a todo momento.

Basicamente, uma entidade é utilizada para reaproveitamento de declarações, minimizando o montante de trabalho para construção de uma DTD, além de minimizar erros.

2.2 DTD

Uma DTD é a estrutura que fornece todas as ferramentas usadas para criar um documento XML. Em outros termos, a DTD é um arquivo que fornece um conjunto de regras para o documento XML, definindo as instruções e a estrutura do mesmo, além de informar quais elementos serão usados ao longo do documento. Em suma, é a base a partir da qual os documentos XML são criados (NATANYA; KIRK, 2000).

Uma DTD pode ser de dois tipos: interna e externa.

A DTD interna está localizada no próprio documento XML, fazendo com que o mesmo se torne auto-suficiente (NATANYA; KIRK, 2000), como demonstrado pela figura 4.

```
<?xml version="1.0"?>
<!DOCTYPE usuarios [
  <!ELEMENT usuarios (usuario)>
  <!ELEMENT usuario (nome, login, senha)>
  <!ELEMENT nome (#PCDATA)>
  <!ELEMENT login (#PCDATA)>
  <!ELEMENT senha (#PCDATA)>
]>
<usuarios>
  <usuario>
    <nome>Emilio Mario</nome>
    <login>emw</login>
    <senha>Reminder</senha>
  </usuarios>
</usuario>
```

Figura 4: Exemplo de uma DTD interna

A figura 4 demonstra a utilização de uma DTD interna ao arquivo XML. Neste caso, o arquivo XML já está sendo validadopor ele próprio, pois sua DTD está "acoplada". Neste caso, a DTD utilizada neste arquivo XML não poderia ser relacionada a outro documento XML.

A DTD externa está em um arquivo próprio, possibilitando que diversos documentos XML utilizem a mesma DTD (NATANYA; KIRK, 2000), como mostrado pela figura 5.

```
<!ELEMENT usuarios (usuario)>
<!ELEMENT usuario (nome, login, senha)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT login (#PCDATA)>
<!ELEMENT senha (#PCDATA)>
```

Figura 5: Exemplo de uma DTD externa

Como observado pela figura 5, vê-se que a DTD está em um arquivo separado, e para poder ser utilizado em algum arquivo XML, o mesmo deverá conter a especificação DOCTYPE demonstrada pela figura 6.

```
<!DOCTYPE usuarios  SYSTEM "usuarios.dtd">
```

Figura 6: Especificação DOCTYPE que deverá ser inserida em um documento XML que referenciar uma DTD externa.

A utilização da DTD interna é mais interessante para validar um documento específico. Por outro lado, a utilização da DTD externa torna-se mais interessante quando é necessário validar vários documentos XML baseados na mesma estrutura.

Uma DTD é formada por "partes", e estas partes podem ser descritas da seguinte forma:

1) Entidades: as entidades especificam arquivos ou dados de caracteres adicionais que possam estar incluídos no conteúdo do documento (NATANYA; KIRK, 2000).

2) Elementos: elementos especificam a estrutura do documento XML e lhe dão a flexibilidade para definir o documento para as suas necessidades específicas (NATANYA; KIRK, 2000). Além de ter de declarar o elemento, deve-se especificar o seu conteúdo (tipo). O conteúdo do elemento especifica que informação pode aparecer dentro do elemento (EVANS; KAMANNA; MUELLER, 2003). Existem vários tipos de conteúdo de elementos que podem ser utilizados, como por exemplo:

- Dados de caracteres analisados sintaticamente (#PCDATA), que contém somente caracteres (texto) e nenhum outro elemento adicional, sendo o modelo de conteúdo mais simples que um elemento pode conter, como por exemplo: <!ELEMENT nome (#PCDATA)>;
- Outros elementos, que permitem o elemento seja formado por um ou mias elementos, como por exemplo: `<!ELEMENT usuario (nome, login, senha)>;`
- Palavras-chave específicas, que servem para definir de forma melhor o conteúdo do elemento, como por exemplo: `<!ELEMENT usuario ANY>`. A palavra ANY aqui representada indica que o elemento pode conter um conteúdo #PCDATA ou qualquer outro, como um outro elemento.
- Existem outros tipos de conteúdos de elementos, mas estes não serão tratados neste trabalho, pois o foco no mesmo não está voltado para isto.

3) Atributos: segundo (NATANYA; KIRK, 2000), os atributos ajudam a descrever exatamente o que são os elementos, o tipo de informações que devem ser inseridas neles e a ordem na qual as informações devem ser colocadas. O XML fornece três tipos diferentes de atributos:

- Atributos String, que é utilizado para criar atributos que possibilitam ao usuário definir qualquer valor para o atributo;

- Atributos contendo tokens, que se refere a sete tipos de atributos definidos que desempenham um papel específico em documentos XML e que devem ter um determinado tipo de valor, como, por exemplo, o atributo ID, que fornece um identificador exclusivo para o elemento dentro do documento XML; e
- Atributos enumerados, que possibilita ao atributo incluir uma lista predefinida de opções como seu valor.

4) Instruções de Processamento: as instruções de processamento informam ao processador XML que versão do XML o processador XML está compilando, e é importantíssimo que o documento XML possua esta instrução, pois somente através desta instrução é que o processador XML poderá saber quais funcionalidades este arquivo XML possui.

2.3 Xpath

Através do *Xpath* que as consultas dos documentos XML do sistema são executadas. Segundo Wahlin (2003), *Xpath* define um termo denominado "Passos de Localização", que resume como a linguagem *Xpath* funciona de verdade. Para se chegar ao local de um certo nó, é preciso listar os passos até o mesmo, utilizando-se de uma expressão ao padrão *Xpath*.

Assim, pode-se dizer que *Xpath* é uma linguagem para endereçar partes de documentos XML, criando uma árvore de nós a partir de um documento XML, sendo que estes nós podem ser dos tipos elementos, atributos ou textos.

Segundo Valentine e Minnick (2001), *Xpath* possui esse nome porque utiliza caminhos tipo URI para acessar as diferentes partes dos documentos XML.

O resultado de uma expressão *Xpath* é sempre um objeto dos tipos conjunto de nós; booleano; número ou string. Segundo Valentine e Minnick (2001), existem três etapas para a localização de nós:

- Eixo: especifica o relacionamento entre os nós que são selecionados através do nó de contexto (nó atual que a expressão está verificando);
- Teste de nó: especifica o tipo de nó e o nome dos nós a serem selecionados;

- Seleção de Predicados: filtram o conjunto de nós selecionados pelo eixo e pelo teste de nó. Esta etapa é opcional;

A figura abaixo demonstra um exemplo de consulta *Xpath* utilizando as três etapas abordadas acima:

```
child::p[position()=3]
```

Figura 7: Exemplo de consulta Xpath

Como pode ser observado na figura 7, é realizada uma consulta em todos os filhos do nó de contexto, selecionando o terceiro elemento p.

É importante salientar que os processadores *Xpath* precisam oferecer suporte a certas funções para a filtragem de conjunto de nós. Estas funções podem ser vistas no site da W3C (http://www.w3c.org/TR/xpath#section-Node-Set-Functions).

Como *Xpath* pode ser uma linguagem com muito texto, foram "criadas" várias maneiras de se abreviar expressões de *Xpath*, como utilizando o símbolo de arroba (@) no lugar de atribute::. (VALENTINE; MINNICK, 2001)

2.4 DOM

DOM é uma especificação para desenvolvimento de uma interface independente de plataforma e linguagem para a estrutura e conteúdo de documentos XML e HTML (MCGRATH, 1999). O DOM busca fornecer um modelo de padrão único de como são organizados os "objetos" que formam os documentos XML e HTML, além de padronizar uma interface para navegação e processamento de documentos.

Devido ao DOM ser uma especificação independente de plataforma e de linguagem, é utilizado o `Object Management Groups IDL` (OMG IDL) para expressar seus tipos de objeto. Desta forma, o DOM pode ser utilizado em qualquer plataforma, pois o OMG IDL trabalha com uma definição de linguagem ISO, mais especificamente a ISO 14750 (MCGRATH, 1999).

Como dito acima, o DOM possui vários Objetos, dentre os quais os que mais se destacam são:

- Objeto Nó (`Node`): os nós são unidos em relação pai/filho. Um nó pode conter nenhum ou vários nós-filhos, e cada nó, com exceção ao nó raiz possui um nó pai;

- Objeto Elemento (`Element`): representam elementos do documento XML (ou HTML) de origem. Um nó de elemento possui uma lista de atributos;
- Objeto Documento (`Document`): é o objeto de nível mais alto, possuindo propriedades como fornecimento de acesso a DTD's e retorno do nó raiz do documento; e
- Objeto Lista de Nós (`NodeList`): fornece uma estrutura de dados para armazenar acessar uma lista de nós (somente leitura).

O DOM possui alguns componentes específicos para se trabalhar com XML, além de funcionalidades adicionais, para permitir que todos os aspectos de um documento XML possam ser representados. Os componentes XML do DOM adicionam suporte para:

- Declarações de Tipo de Documento (DTD);
- Entidades;
- Seções CDATA; e
- Seções Condicionais.

Com o uso do DOM, existe o benefício de uma interface homogênea tanto para HTML como para XML, sendo possível que aplicações possam ser movidas para qualquer plataforma, sem que exista alteração de código.

2.5 Plataforma .NET (.NET Framework)

A plataforma .NET surgiu em Julho de 2000, sendo criada para ser um padrão de desenvolvimento para *web*, possibilitando a utilização de diferentes linguagens de programação na construção de *softwares*, sendo as linguagens de maior destaque o C#, VB.NET, Jscript.NET e ASP.NET (REILLY, 2000).

A proposta desta plataforma é proporcionar um ambiente de desenvolvimento avançado, disponibilizando uma grande quantidade de recursos para uso dos desenvolvedores. Para isso, a Microsoft unificou todas as suas soluções de desenvolvimento nessa nova plataforma, além de melhorar bastante os recursos oferecidos. Pode-se dizer que o .NET *Framework* disponibiliza um ambiente de desenvolvimento multi-plataforma, multi-linguagem, orientada a objetos, e com uma grande e eficiente biblioteca de classes (GUIMARÃES, 2004).

A plataforma de desenvolvimento .NET contém uma biblioteca de classes nativas, que incluem classes de acesso e manipulação de dados, classes que contém objetos visuais (*Windows forms*), classes de manipulação de informações transmitidas pela WEB (ASP.NET), classes de acesso ao sistema, entre outros.

Nas subseções a seguir estarão sendo apresentados os "componentes" da plataforma .NET mais relevantes para esse trabalho.

2.5.1 MSIL – Microsoft Intermediate Language

O conceito de abstração, utilizado para ocultar a complexidade de sistemas oferecendo ao usuário uma interface simples, é extremamente importante na engenharia de software. O *bytecode* gerado pelo compilador do Java é uma forma de abstração que permite que o programa seja executado em diferentes sistemas operacionais, desde que a máquina virtual do Java (JVM) esteja instalada (REILLY, 2000).

A MSIL é a forma de abstração encontrada pela Microsoft. A MSIL é semelhante ao *bytecode* do Java, suportando orientação a objetos, herança, polimorfismo e exceções. Além destes oferece também o suporte a múltiplas linguagens.

2.5.2 CLR – Commom Language Runtime

O CLR tem a função de gerenciar e executar os programas, realizando tarefas como incorporação de objetos, verificações de segurança, execução, gerenciamento de memória e coleta de lixo de todos os códigos escritos em qualquer uma das linguagens utilizadas na plataforma .NET. O CLR está dividido em três sub-componentes (REILLY, 2000):

1. CTS (*Common Type Specification*), que especifica os tipos que uma linguagem tem que suportar para que o CLR possa compilar e executar um programa escrito nessa linguagem;
2. CLS (*Common Language Specification*), que é o conjunto de diretivas que devem ser seguidas por uma aplicação ou implementação .NET para ser compatível com a plataforma;
3. motor de execução, que gerencia as referências a objetos, coleta de lixo, verificação de acesso, métodos seguros e não seguros.

2.5.3 JIT – Compilador de Tempo Real

Uma aplicação .NET diferencia-se das aplicações tradicionais somente no método de compilação. Ao invés de criar um código executável baseado nas instruções de máquina da arquitetura em que está sendo compilado, a plataforma .NET utiliza um conjunto de classes e métodos próprios da plataforma igual para todas as linguagens suportadas. Desse modo existe uma integração entre as linguagens sendo possível derivar uma classe C# de um objeto escrito em VB .NET (REILLY, 2000).

O arquivo em linguagem intermediária recebe o nome de `assembly`, e na plataforma .NET é entendido como uma coleção de um ou mais arquivos com tipos e recursos que formam uma unidade funcional lógica contendo dados especiais denominados manifesto. O manifesto define que arquivos fazem parte do programa, que permissões são necessárias para ser executado, qual a versão do `assembly` e outras informações.

Existem quatro tipos de `assembly` na plataforma .NET (REILLY, 2000):

1. Estáticos: são os arquivos criados com o compilador específico da linguagem utilizada;
2. Dinâmicos: são os arquivos criados diamicamente através das classes localizadas no `namespace System.Reflection`;
3. Privados: são assembly estáticos usados em uma aplicação específica;
4. Públicos ou compartilhados: são assembly estáticos que tem um único nome público e pode ser usado por qualquer aplicação.

A execução do `assembly` se dá através de um compilador de tempo real (JIT), responsável por converter as instruções em MSIL contidas no arquivo em instruções da arquitetura alvo. O funcionamento, de forma básica, dos *JITers* ocorre da seguinte maneira:

- Quando uma classe é carregada, um ponteiro para cada método dessa classe é criado;
- Na execução de um desses métodos, o JIT compila apenas o trecho relativo ao método, criando um ponteiro para o código nativo gerado;
- Esse novo ponteiro permanecerá em *cache* e as chamadas subseqüentes ao mesmo método utilizarão o código já compilado.

2.6 ASP .NET

O ASP.NET, sucessor do ASP, é um dos elementos chave da nova plataforma de desenvolvimento .NET (GUIMARÃES, 2004). É um *framework* para desenvolvimento de aplicações *web* completo e extensível que introduz um novo modelo de programação para criar Aplicações Web e XML *Web Services*. Uma das principais diferenças do novo modelo de programação do ASP.NET para o seu predecessor é sua arquitetura baseada em componentes.

Nas seções a seguir serão demonstrados algumas características do ASP.NET e alguns de seus componentes utilizados neste estudo.

2.6.1 Processamento de uma Requisição http em uma aplicação ASP.NET

O modelo de processamento do ASP.NET é baseado em eventos (PAINE, 2001). Desta forma, o servidor aguarda algo acontecer no cliente, e quando algo acontece, o servidor toma "medidas" para realizar alguma funcionalidade.

Basicamente o que acontece é o seguinte: o HTTP *runtime* manipula a requisição de um cliente, direciona esta requisição para que os componentes a processem, e depois gera a resposta para a aplicação cliente.

Os passos mais importantes do processamento são:

- Requisição HTTP é recebida pelo IIS;
- O IIS redireciona a requisição para o ASP.NET IS API (o *runtime* do ASP.NET serve como uma "ponte" entre a aplicação *web* e o IIS);
- Todo o restante do processamento da requisição é tratado pelo *runtime* do ASP.NET, sendo que este é também o responsável por inicializar os objetos responsáveis pelo processamento da requisição.

Os componentes mais importantes neste processo são:

- `HttpContext`: contém informações sobre a requisição que está sendo processada no momento (fornece acesso ao objeto que representa a requisição "`HttpRequest`" e ao objeto que representa a resposta "`HttpReponse`"). Além destas informações (serviços), este objeto possui outros serviços que estão

disponíveis durante o processamento da requisição, como: estado da aplicação, estado da seção e *cach*.

- `IHttpModule`: contém a implementação de vários módulos, como pós-processamento e módulos HTTP;
- `IHttpHandler`: instancia uma hierarquia de objetos de servidor e componentes de interface de usuários (como os `Server Controls`).

2.6.2 Web Server Controls

Web Server Controls são *tag's* especiais do ASP.NET que são compreendidas somente pelo servidor. Como ocorrem com controles HTML (*HTML Server Controls*), estes também são compilados no servidor e requerem um atributo `runat="server"` para funcionar. Entretanto, os *Web Server Controls* não "mapeiam" somente os elementos existentes no HTML e podem representar elementos mais complexos (W3SCHOOLS, 2003). A sintaxe para criar um *Web Server Control* é mostrada na figura 8:

```
<asp:nome_controle id="id" runat="server" />
```

Figura 8: Sintaxe de um Web Server Control ASP.NET

Nas seções a seguir, serão demonstrados alguns *Web Server Controls* existentes no ASP.NET. Não serão mostrados todos os *Web Server Controls* porque este não é o foco deste trabalho, mas a partir da referência utilizada, será possível visualizar as características da maioria dos controles.

2.6.2.1 *Button Control*

O *Button Control* é utilizado para acionar um botão de envio. O botão de envio pode ser um botão para submeter/enviar ou um botão de comando. Por padrão, este controle "gera" um botão de envio. As principais propriedades deste *Web Control* estão mostradas na tabela 1.

Tabela 1: Propriedades do Button Control (W3SCHOOLS, 2003)

Propriedade	Descrição
CausesValidation	Por default, uma página é validade quando um Button Control é pressionado. Para impedir que uma página seja validada ao se pressionar o controle, ajuste esta propriedade para "false"
CommandArgument	Informação adicional sobre o comando "executar" do controle
CommandName	O comando do controle é associado a um comando de evento
Id	Fornece uma identificação única par o controle
OnClick	Indica o nome do método que é acionado quando o controle é pressionado
Runat	Especifica que o controle é um Server Control. Deve sempre conter o valor "server"
Text	Possui o texto do controle que é mostrado ao cliente (ex. Enviar)

A figura 9 mostra um exemplo da utilização do `Control Button`:

```
<script  runat="server">
  public submit(Source As Object, e As EventArgs)
   {
    button1.Text="Você me Pressionou!"
   }
</script>
<html>
<body>
<form runat="server">
 <asp:Button id="button1" Text="Pressione-me!" runat="server"
OnClick="submit"/>
</form>
</body>
</html>
```

Figura 9: Exemplo de utilização do Control Button

Como pode ser observado na figura 9, o *Control Button* `button1`, quando precionado, executa o método `submit`, que irá alterar o atributo `text` do `button1`.

2.6.2.2 *TextBox Control*

O *TextBox Control* é utilizado para criar uma caixa de texto onde o usuário digita algum texto de entrada. A tabela 2 demonstra as principais propriedades do controle *TextBox*.

Tabela 2: Propriedades do TextBox Control (W3SCHOOLS, 2003)

Propriedade	**Descrição**
AutoPostBack	Valor booleano que especifica se o controle será enviado para o servidor quando os índices mudarem ou não. O valor default é falso
Columns	A largura do controle
Id	Fornece uma identificação única par o controle
MaxLength	O número máximo de caracteres que podem ser inseridos no controle
OnTextChanged	O nome do método que deve ser executado quando o valor contido no controle for alterado
Rows	A altura do controle (usado somente se TextMode="MultiLine")
Runat	Especifica que o controle é um Server Control. Deve sempre conter o valor "server"
Text	O conteúdo do controle
TextMode	Pode possuir três valores: SingleLine: cria um controle com somente uma linha MultiLine: cria um controle com várias linhas Password: cria um controle com somente uma linha e mascara o valor digitado pelo usuário O valor padrão é SingleLine
Wrap	Um valor booleano que indica se os índices do controle deve ou não ser envolvidos

2.6.2.3 *Label Control*

O *Control Label* é utilizado para mostrar algum texto em uma página. O texto a ser mostrado é programável. As principais propriedades deste *Web Control* estão mostradas na tabela 3.

Tabela 3: Propriedades do Label Conttrol (W3SCHOOLS, 2003)

Propriedade	**Descrição**
Id	Fornece uma identificação única par o controle
Runat	Especifica que o controle é um Server Control. Deve sempre conter o valor "server"
Text	O texto que será mostrado pelo Controle

A figura 10 exibe um exemplo da utilização dos controles `TextBox` e `Label`

```
<script  runat="server">
  public submit(sender As Object, e As EventArgs)
   {
    lbl1.Text="Seu nome é " & txt1.Text
```

```
  }
</script>
<html>
<body>
<form runat="server">
  Enter your name:
  <asp:TextBox id="txt1" runat="server" />
  <asp:Button OnClick="submit" Text="Submit" runat="server" />
  <p><asp:Label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Figura 10: Exemplo de utilização dos controles TextBox e Label

Como pode ser observado na figura 10, quando *o Control Button* é acionado, este executa o método `submit`, que irá atribuir ao *Control Label* `lbl1` no atributo `text` o valor que está atribuído ao atributo `text` do *Text Control* `txt1`.

2.6.2.4 *ListBox Control*

O *ListBox Control* é utilizado para criar uma lista *Drop-Down* de simples ou múltipla seleção. Cada item selecionável no *ListBox Control* é definido por um elemento `ListIten`. A tabela 4 demonstra as principais propriedades do controle *ListBox*.

Tabela 4: Propriedades do LisBox Control (W3SCHOOLS, 2003)

Propriedade	**Descrição**
AutoPostBack	Valor booleano que especifica se o controle será enviado para o servidor quando os índices mudarem ou não. O valor default é falso
BorderColor	Especifica a cor da borda da lista Drop-Down
BorderStyle	Especifica o estilo da borda da lista Drop-Down
BorderWidth	Especifica a largura da borda da lista Drop-Down
DataSource	A origem dos dados que está se utilizando
DataTextField	Um campo na origem dos dados que deve ser mostrado
DataValueField	Um campo na origem dos dados de que especifica o valor de cada Ítem selecionável na lista drop-down
Id	Fornece uma identificação única par o controle
OnSelectedIndexChanged	O número máximo de caracteres que podem ser inseridos no controle
OnTextChanged	O nome do método que deve ser executado quando o índice do item selecionado alterado
Rows	A altura do controle
Runat	Especifica que o controle é um Server Control. Deve sempre conter o valor “server”
SelectionMode	Permite únicas ou múltiplas seleções. Valores Possíveis: single e multiple. O valor padrão é single

2.6.3 Validation Server Controls

Estes controles são utilizados para validar dados de um controle utilizado para entrada de dados. Caso os dados informados em um controle que esteja utilizando o *Validation Control* não sejam validados por este, é informada uma mensagem de erro ao usuário.

Nas seções a seguir, serão demonstrados alguns *Validation Server Controls* existentes no ASP.NET. Não serão mostrados todos os *Validation Server Controls*.

2.6.3.1 *CompareValidator*

Este controle compara os valores de um controle de entrada de dados (`TextBox Control`, por exemplo) com outro controle de entrada ou um valor fixo. A tabela 5 demonstra as principais propriedades do *CompareValidator*.

Tabela 5: Propriedades do CompareValidator (W3SCHOOLS, 2003)

Propriedade	**Descrição**
ControlToCompare	O ID do controle com o qual comparar os valores
Operator	A comparação a fazer
ValueToCompare	Um valor constante com o qual comparar

2.6.3.2 *CustomValidator*

Este controle valida o conteúdo dos controles de entrada através de uma rotina de validação (feita pelo desenvolvedor) do lado do servidor ou do lado do cliente. A tabela 6 demonstra as principais propriedades do *CustomValidator*.

Tabela 6: Propriedades do CustomValidator (W3SCHOOLS, 2003)

Propriedade	**Descrição**
ClientValidationFunction	A função do lado do cliente que deve ser utilizada para validar o controle específico

2.6.3.3 *RegularExpressionValidator*

Este controle compara os valores de um controle de entrada para uma expressão regular, ou seja, verifica se o conteúdo do controle de entrada é válido para uma expressão (teste) definida pelo desenvolvedor, como por exemplo, validação de CPF. A tabela abaixo demonstra as principais propriedades do *RegularExpressionValidator*.

Propriedade	**Descrição**
ValidationExpression	Uma expressão regular para validar o conteúdo do controle observado

Tabela 7: Propriedades do RegularExpressionValidator (W3SCHOOLS, 2003)

2.7 C#

A linguagem de programação C# deriva do C e C++, simplificando (e modernizando) o C++ nas áreas de classes, espaço de nomes, sobrecarga de métodos e gerenciamento de exceções (WILLE, 2001). Muito da complexidade do C++ foi removida do C#, como a utilização de ponteiros, a fim de tornar o C# mais fácil de ser utilizado e menos propenso a erros.

Segundo Wille (WILLE, 2001), o C# é uma linguagem:

- Simples: possui diversos recursos; por padrão se trabalha com código gerenciado; não existem as redundâncias encontradas no C++, como diferentes tipos de caracteres, dentre outras;
- Moderna: além de ser projetada para ser a linguagem principal de escrita de aplicativos .NET, possui recursos que estavam indisponíveis em C++ implementados, além de possuir maior segurança que seus antecessores;
- Orientada a Objetos: não existem mais funções globais, variáveis ou constantes, tudo deve ser encapsulado dentro de uma classe, tornando o código C# mais legível, reduzindo conflitos de nomeação, além de possuir suporte a herança, polimorfismo, encapsulamento e abstração;
- Type-safe: o C#, juntamente com coletores de lixo, implementa a segurança de tipo mais estrita, impedindo que erros graves, como, por exemplo, atribuir um ponteiro int* para um double*;
- Compatível: é possível acessar API's diferentes, através do CLS, além de objetos COM mais antigos; e

- Flexível: possibilita a incorporação de aplicações C# em aplicações já existentes, em C, por exemplo.

Os recursos citados fazem do C# uma linguagem fácil de aprender e de usar, robusta e com boa performance. Em conjunto com os demais recursos da arquitetura .NET, o C# é a linguagem ideal para a criação de uma nova categoria de programas que aproveitam as oportunidades trazidas pela Internet (SANT'ANA, 2004).

A figura 11 demonstra a sintaxe utilizada do C#.

```
public XmlElement criarElementoXml(XmlDocument pai, string nomeElemento)
{
  return pai.CreateElement(nomeElemento);
}
public void insereValorElementoXml(XmlElement nomeElemento, string valor)
{
  nomeElemento.InnerText = valor;
}
  public void adicionarElementoXml(XmlElement pai, XmlElement filho)
{
  pai.AppendChild(filho);
}
```

Figura 11: Exemplo de Sintaxe Utilizada no C#

A figura 11 demonstra alguns métodos na linguagem C# que são utilizados para a manipulação de documentos XML.

2.8 Implementação DOM na Plataforma .NET

Na seção 2.4 deste trabalho foram demonstradas algumas características da API DOM. Nesta seção será demonstrado como funciona a API DOM na plataforma .NET através do *Microsoft XML Core Services* (MSXML 4.0), sendo baseado no DOM *Level* 2 CORE da W3C (MSDN, 2004).

As interfaces/objetos de MSXML incluem as extensões da Microsoft pata suportar espaços identificadores, tipos de dados, esquemas XML, XSLT, operações de transformações XSL (XSLT), carregamento assíncrono e gravação de documentos (EVANS, KAMANNA, MUELLER, 2003). A capacidade de fornecer extensões na mesma API permite que os programadores trabalhem com uma única API consistente para transformações e processamento de documentos. Na tabela 8, é mostrada a listagem das

interfaces fundamentais, enquanto que na tabela 9 tem-se a listagem das interfaces estendidas, tanto da W3C, quanto da MSXML 4.0.

Tabela 8: Interfaces fundamentais da API DOM (EVANS, KAMANNA, MUELLER, 2003)

Interface W3C	**Interface MSXML**	**Descrição**
Nó	IXMLDOMNode	Representa um único nó na árvore documentos; a interface base para acessar dados no modelo de objetos XML. Tipos de nós válidos são definidos nas constantes enumeradas XML DOM. IXMLNode inclui suporte para tipos de dados, espaços identificadores, definições de tipos de documentos (DTD's) e esquema XML
Documento	DOMDocument	Representa o nó superior da árvore XML DOM
	IXMLDocument2	Extensão de DOMDocument, que suporta armazenagem em cachê de esquemas, validação no tempo de execussão e uma forma de ativar o suporte para XML Path Language (XPath)
Implementação DOM	IXMLDOMImplementation	Fornece métodos independentes de qualquer instância em particular de DOM. Útil para descobrir se uma versão específica da implementação do analisador MSXML suporta um recurso especificado
Fragmento de Documento	IXMLDOMDocument Fragment	Representa um objeto leve que é útil para as operações de inserção em árvore
Lista de nós	IXMLDOMNodeList	Suporta operações de acesso indexado e iteração na coleção viva de IXMLDOMNode
Elemento	IXMLDOMElement	Representa o objeto do elemento
NodeMap Nomeado	IXMLDOMNamed NodeMap	Fornece iteração e acesso pelo nome à coleção de atributos. IXMLDOMNameNedeMap inclui suporte para espaços identificadores
Attr	IXMLDOMAtribute	Representa um atributo do IXMLDOMElement. Os valores padrão e válidos para o atributo são definidos em uma DTD ou esquema
CharacterData	IXMLDOMCharacterData	Fornece métodos de manipulação de texto usados por diversos objetos
Texto	IXMLDOMText	Representa o conteúdo do texto de um elemento ou atributo
Comentário	IXMLDOMComment	Representa o conteúdo de um comentário

		XML

Tabela 9: Interfaces estendidas da API DOM (EVANS, KAMANNA, MUELLER, 2003)

Interface W3C	**Interface MSXML**	**Descrição**
Seção CDATA	IXMLDOMCDATASection	Coloca entre aspas ou escapa blocos de texto de modo que o texto não seja interpretado como linguagem de marcação
Tipo de Documento	IXMLDOMDocumentType	Contém as informações associadas com a declaração do tipo de documento
Notação	IXMLDOMNotation	Contém uma notação declarada na DTD ou no esquema
Entidade	IXMLDOMEntity	Representa uma entidade analisada ou não no documento XML
Referência de Entidade	IXMLDOMEntityReference	Representa um nó de referência de entidade
Instrução de Processamento	IXMLProcessingInstruction	Representa uma instrução de processamento, que XML define para manter informações específicas do processador no texto do documento

As interfaces/objetos listados na tabela 10 incluem as extensões da Microsoft para suportar espaços identificadores, tipos de dados, esquemas XML, XSL e operações de transformações XSL (XSLT).

Tabela 10: Extensões da API DOM feitas pela Microsoft (EVANS, KAMANNA, MUELLER, 2003)

Interface MSXML	**Descrição**
XMLSchemaCache	Representa um conjunto de URI's de espaços identificadores. Usado pelas propriedades de espaços identificadores e esquemas em IXMLDOMDocument2
IXMLDOMSchema Collection	Representa um objeto SchemaCache
IXSLProcessor	Usado para transformações em folhas de estilo compiladas
IXSLTemplate	Representa uma folha de estiloXSL armazenada em cachê
IXSLDOMSelection	Representa a Lista de nós que combinam com um certo padrão XSL ou expressão XML Path Language (XPath)
IXTLRuntime	Implementa métodos que podem ser chamados de folhas de estilo XSLT

A implementação do DOM na plataforma .NET e suas funcionalidades básicas estão na classe abstrata `XmlNode`, que representa um elemento na árvore de documento XML e pode ser utilizada para navegar nos nós filhos e nó pai, bem como editar e excluir

dados. A classe `XmlDocument` estende a classe `XmlNode` e permite realizar operações no arquivo XML inteiro, como carregar e salvar arquivos. Há várias outras classes no DOM que são derivações da classe `XmlNode`, como `XmlElement` e `XmlAtribute` (PAINE, 2001).

Nas seções seguintes serão apresentadas as classes, suas propriedades e métodos mais importantes da implementação do DOM na plataforma .NET que foram utilizados neste trabalho. É interessante informar que as classes aqui representadas não serão exemplificadas, pois o mesmo será demonstrado na seção 4 que mostra a implementação do sistema aqui desenvolvido.

2.8.1 A classe XmlDocument (IXMLDOMDocument)

A classe `XmlDocument` é a implementação da interface *Document* da W3C. Esta classe herda da classe `XMLNode` e fornece funcionalidades como o carregamento de documentos XML na estrutura DOM e a criação de documentos XML. Sem esta classe, as outras classes não teriam acesso ao DOM de um documento XML. A tabela 11 e a tabela 12 listam as propriedades e os métodos, respectivamente, que foram utilizados no desenvolver deste trabalho.

Tabela 11: Propriedades da classe `XmlDocument` (PAINE, 2001)

Nome da Propriedade	Descrição
DocumentElement	Recupera o elemento root do documento XML como um objeto XmlElement
ChildNodes	Obtém todos os filhos do nó atual
DocumentType	Obtém a declaração <!DOCTYPE>, se existe uma como objeto XmlDocumentType
FirstChild	Obtém o primeiro filho do nó atual
InnerText	Obtém ou define os valores concatenados do nó e de todos os seus filhos
Item	Retorna o documento filho especificado. Este é o indexador em C# para a classe XmlDocument
Value	Obtém ou define o valor do nó atual

Tabela 12: Métodos da Classe `XmlDocument` (PAINE, 2001)

Nome do Método	Descrição
AppendChild	Adiciona o nó especificado ao final da lista de filhos do nó atual
CreateAtribute	Cria um nó XmlAtribute com um nome especificado

CreateDocumentType	Cria um objeto XmlDocumentType
CreateElement	Cria um XmlElement vazio. Usa a propriedade Value para especificar dados de texto para o elemento ou anexar um elemento XmlTextNode como o filho do elemento
CreateNode	Cria um objeto XmlNode com o XmlNodeType especificado
CreateXmlDeclaration	Cria um objeto XmlDeclaration com os valores especificados. Aceita parâmetros para versão, codificação e autônomo
Load	Carrega os dados XML. As fontes par os dados XML incluem um URI, um fluxo, um leitor de texto e um XmlReader
RemoveChild	Remove o nó filho especificado associado com o nó atual
Save	Salva o documento XML para o local especificado
SelectNodes	Seleciona uma XmlNodeList de nós que corresponde à expressão XPath especificada
SelecSingleNode	Seleciona um XmlNode que corresponde à expressão XPath especificada. Se mais de uma correspondência for encontrada, será retornada apenas a primeira correspondência
ToString	Retorna uma string que representa o nó atual

2.8.2 A classe XmlNodeList (IXMLDOMNodeList)

A classe `XmlNodeList` é uma implementação da interface `NodeList` do W3C. Esta classe representa uma coleção ordenada de nós que pode ser alterada. A propriedade `ChildNodes` do `XmlDocument` retorna um objeto *NodeList*. A tabela 13 e a tabela 14 listam as propriedades e os métodos, respectivamente, que foram utilizados no desenvolver deste trabalho.

Tabela 13: Propriedades da Classe `XmlNodeList` (PAINE, 2001)

Nome da Propriedade	**Descrição**
Count	Obtém o número de nós da lista
ItemOf	Obtém o nó correspondente ao índice especificado. Em C#, esta propriedade é o indexador para a classe XmlNodeList

Tabela 14: Métodos da Classe `XmlNodeList` (PAINE, 2001)

Nome do Método	**Descrição**
Item	Obtém um nó na coleção correspondente ao índice especificado

2.8.3 A classe XmlElement (IXMLDOMElement)

A classe `XmlElement` é uma implementação da interface *Element* do W3C. Esta classe representa um elemento em um documento XML. A tabela 15 lista as propriedades utilizadas no desenvolver deste trabalho.

Tabela 15: Propriedades da Classe `XmlElement` (PAINE, 2001)

Nome da Propriedade	**Descrição**
Atributes (anula XMLNode::Atributes)	Obtém uma XmlAtributeCollection contendo os atributos deste nó
ItemOf	Obtém o nó correspondente ao índice especificado. Em C#, esta propriedade é o indexador para a classe XmlNodeList

Este capítulo apresentou os itens teóricos necessários para a realização do trabalho, cujos detalhes da implementação serão especificados no capítulo 4, após uma apresentação do material e métodos utilizados para a concretização deste trabalho.

3 Material e Métodos

Neste capítulo serão descritos os materiais e os métodos que foram utilizados para o desenvolvimento do mesmo.

3.1 Material

Para o desenvolvimento deste trabalho foram utilizados os seguintes materiais:

- Microcomputador com Processador Intel Pentium 4 com clock de 1.6 GHZ, 256 MB de memória RAM e HD com capacidade para 40 GB;
- Macromedia Dreamweaver MX 2004: através deste software foi feita a codificação da interface do sistema e a codificação de algumas classes da camada de lógica de negócio, além da criação/edição de DTD's e documentos XML;
- Notepad: para a visualização rápida de fragmentos de código ASP.NET. C# e XML;
- Macromedia FireWoks MX 2004: através deste software foram desenvolvidas as imagens que são vistas na interface do sistema;
- Microsoft ASP.NET Web Matrix: neste software foram desenvolvidos alguns códigos web em ASP.NET;
- Microsoft .NET Framework: utilizado para o desenvolvimento de aplicações que utilizaram a plataforma .NET, como o ASP.NET, o C#.
- Microsoft Visual Studio.NET: utilizado para a implementação de códigos C# e para a consulta a biblioteca MSDN;
- Rational Rose 2000: neste software foi realizada a modelagem do sistema;

- Adobe Acrobat Reader: para leitura de referências bibliográficas.

3.2 Métodos

Os métodos utilizados para o desenvolvimento deste trabalho foram os seguintes:

- Internet: na Internet foram buscadas referências bibliográficas para serem aplicadas a este trabalho, principalmente as que detalhavam a plataforma .NET da Microsoft;
- Livros: foram utilizados livros para o aprendizado das tecnologias utilizadas neste trabalho, principalmente ASP.NET e XML.
- Reuniões com a professora orientadora para definir caminhos a serem seguidos e verificar o andamento do trabalho.

O trabalho foi realizado da seguinte forma:

- definição do domínio do trabalho (sistema);
- estudo dos documentos que o sistema deverá gerenciar;
- estudo das linguagens e componentes utilizados no desenvolvimento do sistema;
- elaboração da modelagem do sistema;
- elaboração da estrutura do sistema (definição de DTD)
- desenvolvimento, em C#, das classes de controle do Sistema, como as classes `GerenviadorDocumentos`, `GerenciadorUsuarios` e `UtilXML`;
- desenvolvimento da interface do sistema através da utilização de C# e ASP.NET
- realização de testes para verificar o funcionamento do sistema, a fim de corrigir eventuais falhas.

4 Resultados e Discussão

Neste capítulo serão demonstrados os resultados provenientes do desenvolvimento do sistema de gerenciamento de documentos jurídicos.

O sistema tem como objetivo a manipulação (criação, alteração, exclusão) de documentos jurídicos, através da plataforma .NET e da utilização de documentos XML, sendo que, para isto, foi realizada uma estruturação eficiente dos dados, através da utilização de DTD's.

Além do objetivo citado acima, outro objetivo foi o estudo do funcionamento da API DOM e da plataforma .NET, para que fosse possível verificar como a plataforma .NET trabalha com documentos XML através da API DOM.

Os documentos, de quaisquer tipos (convênios, contratos, etc.), são armazenados em arquivos XML diferentes. Como os documentos são interpretados como arquivos distintos, cada arquivo XML possui as informações pertinentes a um único documento jurídico. Neste trabalho foi implementado o módulo relativo a documentos jurídicos do tipo "Contrato de Estágio".

A seguir serão demonstradas as características do sistema, como a modelagem, a interface e a codificação do mesmo.

4.1 Modelagem do Sistema

A modelagem do sistema foi baseada no domínio do trabalho e na elaboração de módulos que conseguissem gerenciar o sistema, como a utilização de usuários para realizar as transações do sistema.

Além disso, foi necessária a criação de um descritor de documentos das classes que armazenariam as informações em memória antes do envio destes para os documentos

XML, e as classes de gerenciamento, que possuem métodos para trabalhar com os documentos XML, como é o caso da classe `GerenciarDocumentos`, mostrada na figura 13.

Nas seções aseguir estarão sendo demonstrados o diagrama de classes e a estrutura do sistema.

4.1.1 Diagrama de Classes

O diagrama de classes do sistema pode ser observado na da figura 12.

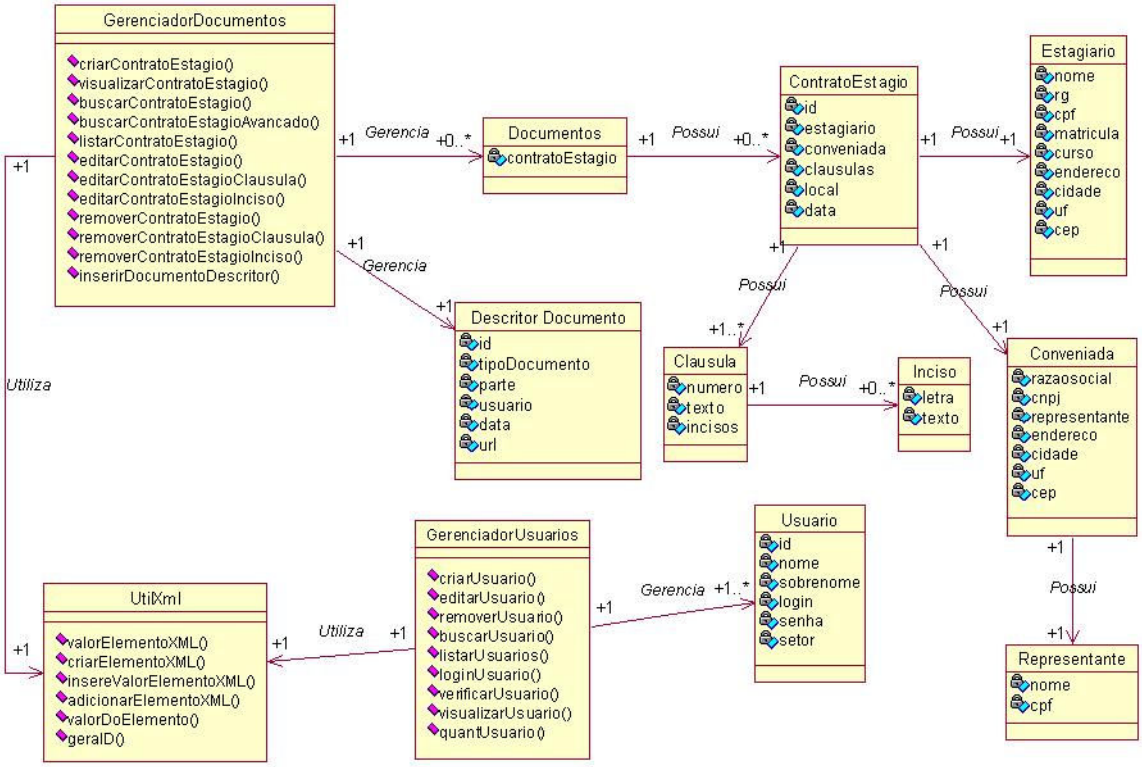

Figura 12: Diagrama de Classes do Sistema SisJurXML

Como o diagrama representado pela figura 11 mostra, o sistema possui uma classe `GerenciadorDocumentos` que possui os métodos necessários para o gerenciamento dos documentos jurídicos que serão manipulados pelo sistema.

O sistema também possui uma classe `GerenciadorUsuarios` que é responsável pelo gerenciamento dos usuários do sistema, e uma classe `UtilXML` que possui métodos comum às outras duas classes, para evitar assim a redundância de métodos nas classes.

Além destas classes de controle, o diagrama mostra as classes geradas através da análise dos documentos jurídicos que serão manipulados pelo sistema, como a classe `ContratoEstagio`, que possui as informações referentes a um contrato de estágio e a classe `Estagiario` que contém as informações do estagiário que estará no contrato de estágio.

Existem ainda as classes que possuirão dados de controle do sistema, como a classe `Usuarios` que possui as informações sobre os usuários do sistema e a classe `DescritorDocumentos`, que possui informações sobre todos os documentos jurídicos presentes no sistema.

4.1.2 Diagramas de Seqüência

Nesta seção serão demonstrados os diagramas de seqüência mais importantes do sistema.

a) Criar Contrato Estágio

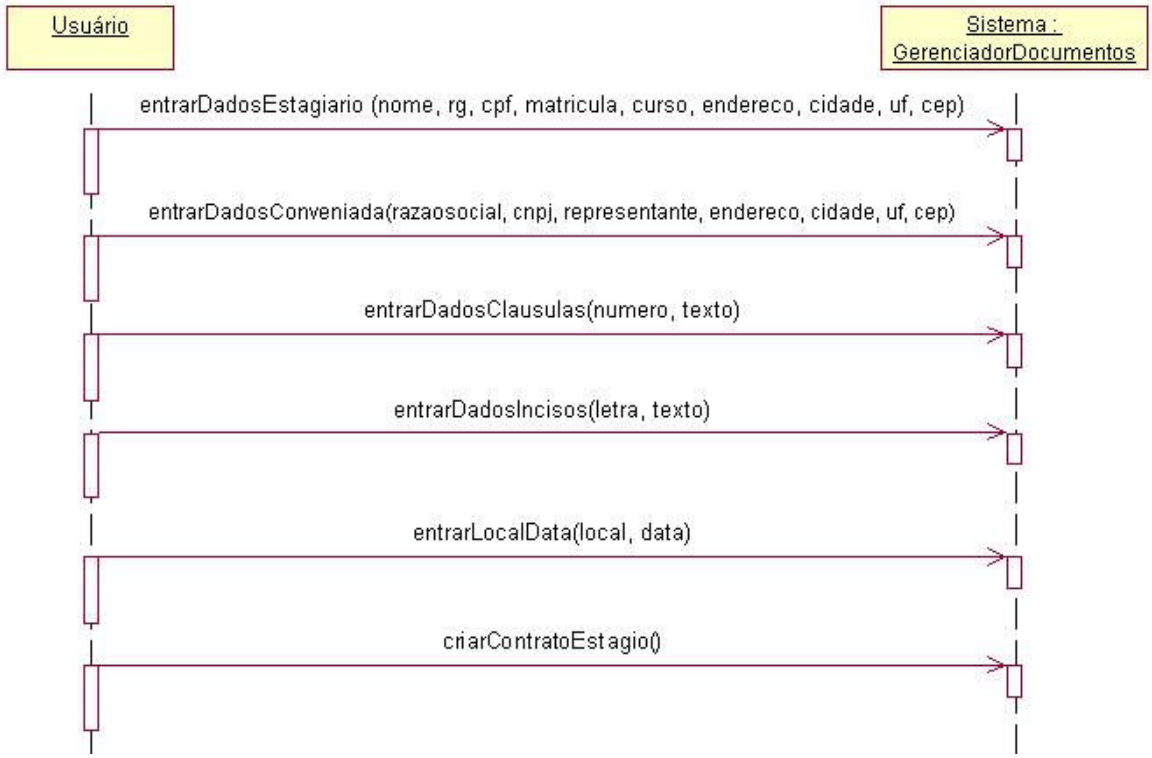

Figura 13: Diagrama de seqüência Criar Contrato Estágio

Observando a figura 13, pode-se visualizar que para se criar um contrato de estágio no sistema, tem-se que informar uma série de dados, divididas por grupo (estagiário, conveniada, cláusulas, etc), antes de o contrato ser criado realmente, o que impede que um contrato seja criado de forma incoerente (com falta de informações requeridas).

b) Visualizar Contrato Estágio

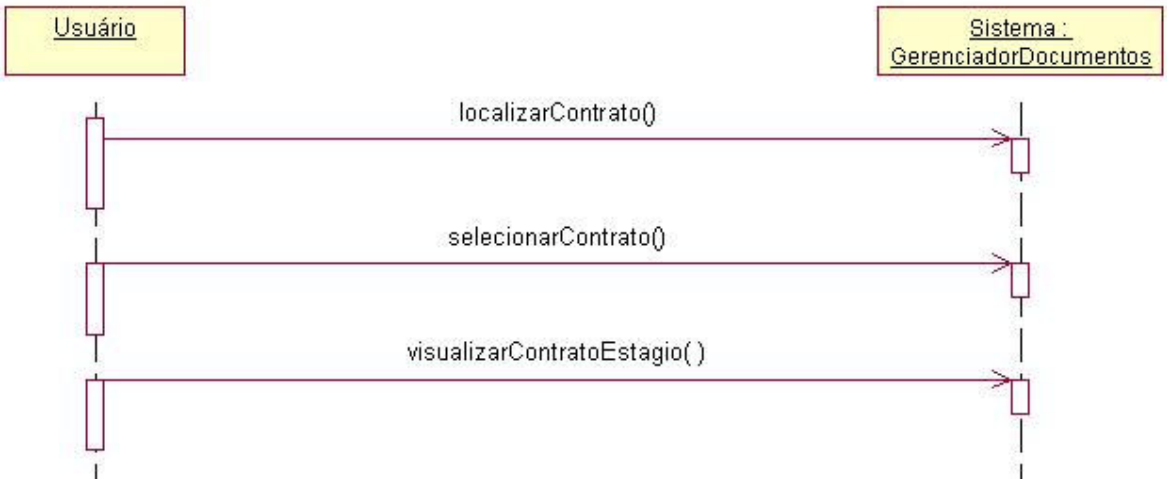

Figura 14: Diagrama de seqüência Visualizar Contrato Estágio

Como pode ser observado na figura 14, é necessário primeiramente que o usuário localize o contrato ao qual queira visualizar, selecione tal contrato e após isto é que o usuário pode visualizar o contrato de estágio pretendido.

c) Editar Contrato Estágio

Figura 15: Diagrama de seqüência Editar Contrato Estágio

Observando a figura 15, percebe-se que antes do usuário aplicar as alterações necessárias no contrato de estágio pretendido, tem que localizar o contrato no sistema, e após aplicar as alterações o contrato é salvo novamente no sistema.

d) Inserir Documento Descritor

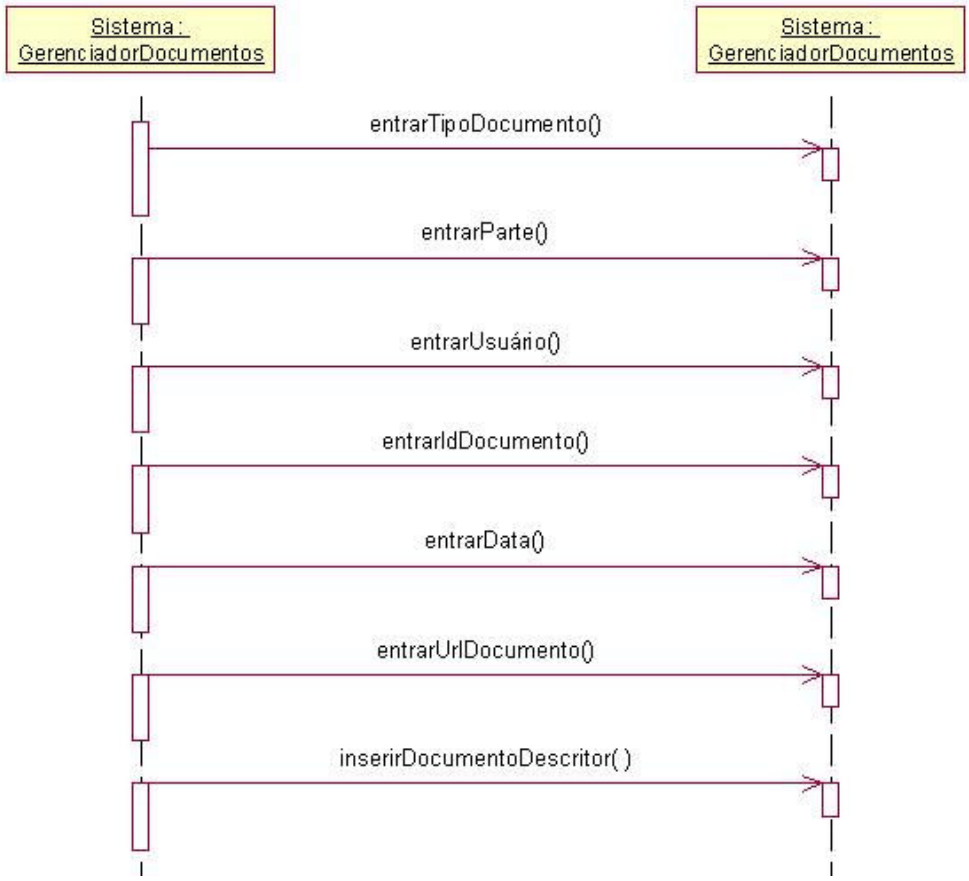

Figura 16: Diagrama de seqüência Inserir Documento Descritor

Como é possível observar na figura 16, o descritor de documentos possui informações do documento que acaba de ser inserido no sistema, o que vai facilitar a busca dos documentos armazenados. Vale ressaltar que o descritor de documentos só é criado após a inserção de todos os dados que este necessita, evitando assim incoerências no sistema. Outro ponto importante que pode ser observado na figura é que estas "transformações" são realizadas apenas pelo sistema, sem a presença do usuário.

e) Criar Usuário

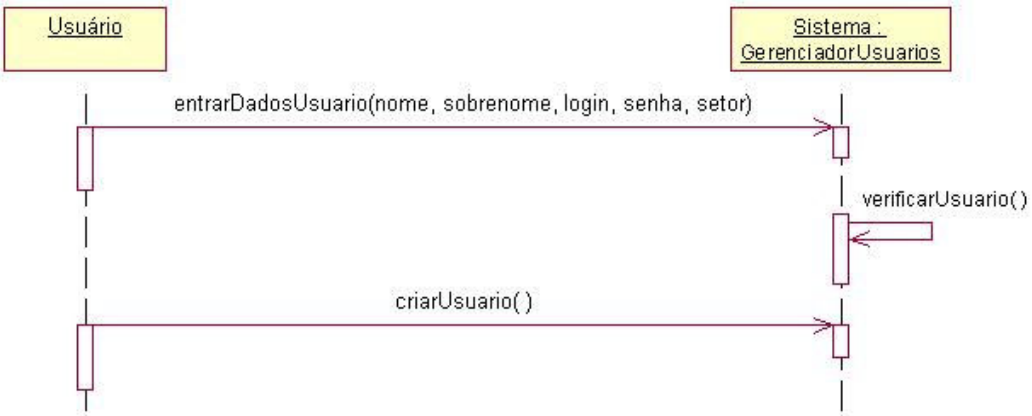

Figura 17: Diagrama de seqüência Criar Usuário

Na figura 17, observa-se que o usuário informa ao sistema os dados do novo usuário, a partir da verificação de existência. Após isso, o novo usuário é cadastrado no sistema e, em seguida, é efetivada sua criação.

f) Excluir Usuário

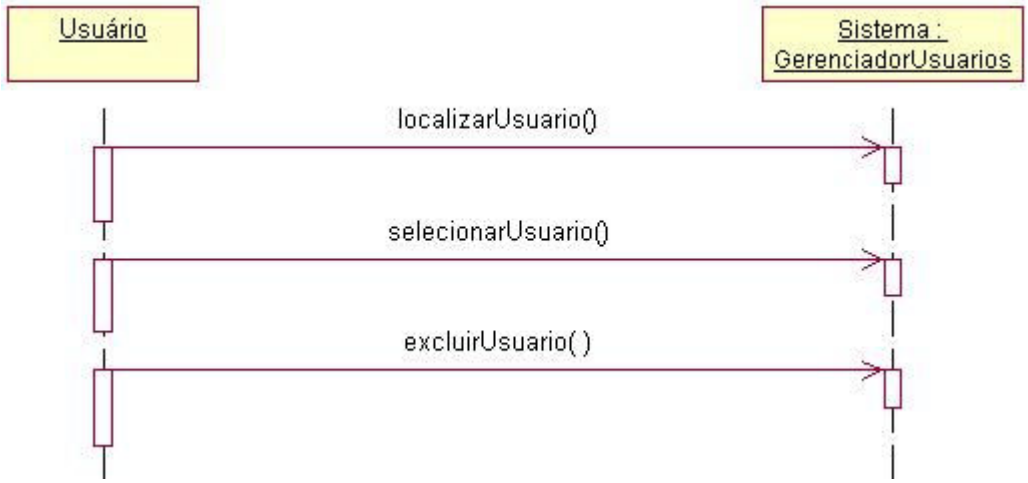

Figura 18: Diagrama de seqüência Excluir Usuário

Como pode ser observado na figura 18, o usuário localiza e seleciona o usuário que deseja excluir e após estes passos exclui o usuário do sistema.

g) Login Usuário

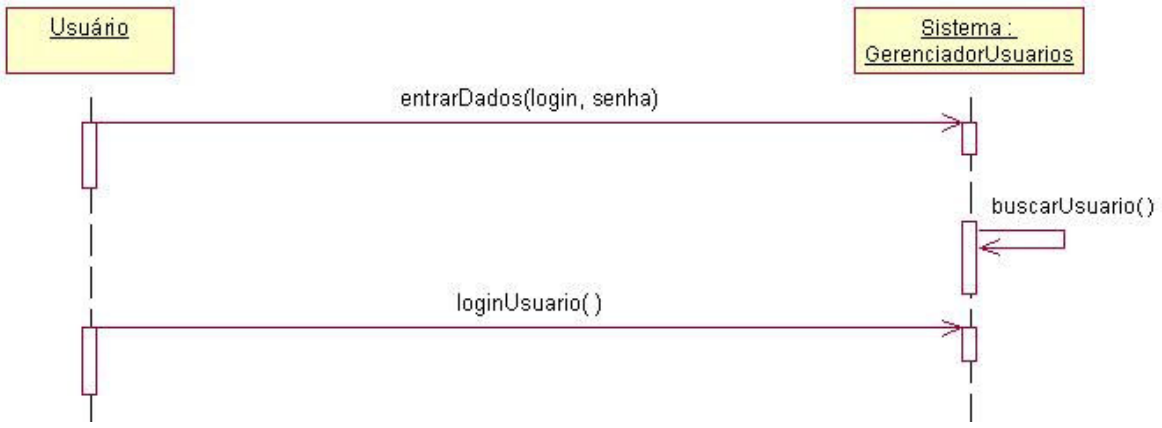

Figura 19: Diagrama de seqüência Login Usuário

Como pode ser observado na figura 19, o usuário entra com seus dados, o sistema faz uma busca dos usuários cadastrados através dos dados informados e, em caso afirmativo, o usuário efetiva o *logon* no sistema.

h) Criar Elemento XML

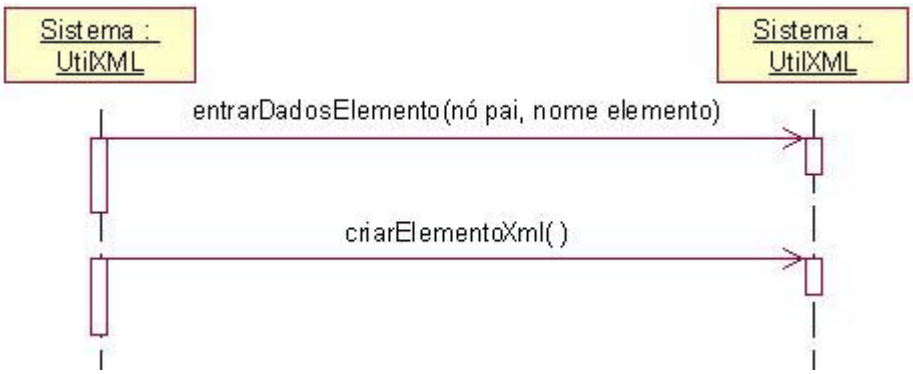

Figura 20: Diagrama de seqüência Criar Elemento XML

Na figura 20, pode-se observar que o sistema espera os dados do elemento que deve ser criado, e só após o recebimento destes dados é que o elemento XML é criado.

4.1.3 Estrutura do Sistema

A partir dos modelos dos documentos jurídicos (contratos, convênios), foi definida a estrutura dos documentos XML de armazenamento. Na figura 21 é apresentada a DTD de um documento do tipo “Contrato de Estagio”. Nela, é especificado cada elemento que o documento possui, sua multiplicidade e até a definição de possíveis atributos.

```
<!ELEMENT contrato (id, estagiario, conveniada, clausula+, local, data)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT estagiario (nome, rg, cpf, matricula, curso, endereco, cidade,
uf, cep)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT rg (#PCDATA)>
<!ELEMENT cpf (#PCDATA)>
<!ELEMENT matricula (#PCDATA)>
<!ELEMENT curso (#PCDATA)>
<!ELEMENT endereco (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
<!ELEMENT uf (#PCDATA)>
<!ELEMENT cep (#PCDATA)>
<!ELEMENT conveniada (razaosocial, cnpj, representante, endereco, cidade,
uf, cep)>
<!ELEMENT razaosocial (#PCDATA)>
<!ELEMENT cnpj (#PCDATA)>
<!ELEMENT representante (nome, cpf)>
<!ELEMENT clausula (numero, texto, inciso*)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT texto (#PCDATA)>
<!ELEMENT inciso (letra, texto)>
<!ELEMENT letra (#PCDATA)>
<!ELEMENT local (#PCDATA)>
<!ELEMENT data (#PCDATA)>
```

Figura 21: DTD do documento jurídico “Contrato de Estágio”

A estrutura prevê como elemento raiz a representação de um contrato, formada por um identificador, um estagiário, uma instituição conveniada, um conjunto de cláusulas, um local e uma data. O `id` é o identificador do contrato, e serve para identificá-lo no sistema. O estagiário é umas das partes do contrato e possui um conjunto de elementos para identificá-lo (`nome, rg, cpf, matricula, curso, endereço, cidade, uf, cep`). A conveniada é a outra parte do contrato, e possui um conjunto de elementos para identificá-la (`razaosocial, cnpj, representante, endereço, cidade, uf, cep`). Cláusulas, representada por um item, são constituídas por um ou mais elementos do tipo `clausula`, bem como dos elementos `numero`, `texto`, e `inciso`. Uma cláusula possui um conjunto próprio de elementos que armazena a descrição da cláusula e ainda a possibilidade de conter, ou não, incisos. Os incisos contêm uma `letra` e um `texto` para identificá-los.

Para o controle dos usuários e dos documentos que fazem parte do sistema, não foi necessário a construção de DTD’s, visto que estes dados serão manipulados apenas pelo sistema, não havendo possibilidade destes dados serem trabalhados de forma inconsistente, além de que os mesmos seguem as especificações contidas no modelo de classes.

4.2 Arquitetura do Sistema

O sistema está dividido em camadas, ou seja, existem camadas responsáveis por cada "parte" do sistema. Nas seções abaixo serão mostradas as codificações utilizadas em cada uma destas camadas.

4.2.1 Camada de Apresentação

Esta camada compreende a parte da interface do sistema, que nesse caso, foi desenvolvida em ASP.NET, utilizando os controles descritos na seção 2.4.1.

Nas seções a seguir, estão sendo apresentadas algumas interfaces do sistema, juntamente com codificação das chamadas a métodos das camadas inferiores. Os eventos de validação de formulários e os `Web Server Controls` (`Control Button`, `Control TextBox`, etc.) não serão exemplificados, pois o funcionamento dos mesmos já foi demonstrado no capítulo 2 de Revisão de Literatura.

4.2.1.1 Efetuando Login no Sistema

Nesta interface, o usuário irá digitar o seu login e sua senha para poder ter acesso ao sistema. A figura 22 apresenta a interface descrita nesta seção.

Figura 22: Página de Acesso ao Sistema

Como pode ser observado na figura 22, é solicitado ao usuário o seu login e sua senha e, após a entrada de dados, este deve pressionar o botão `Entrar` para acessar o Sistema. A figura 23 demonstra os eventos acionados pelo botão (`Control Button`), quando este é pressionado.

```
1:  protected void Logar(Object Src, EventArgs E)
2:  {
3:  string caminho = Server.MapPath(@"../dados/controle/usuarios.xml");
4:  string login = Login.Text;
5:  string senha = Senha.Text;
6:  GerenciadorUsuarios gu = new GerenciadorUsuarios();
7:  Usuario usuario = gu.loginUsuario(login, senha, caminho);
8:  if(usuario == null)
9:  lblMensagem.Text = "Usuário e/ou Senha Incorreto!";
10: else{
11: Session["Usuario"] = usuario;
12: Response.Redirect("default.aspx");
13: }
14: }
```

Figura 23: Método Logar executado pelo Control Button na página de acesso do sistema

Como pode ser observado no código apresentado pela figura 23, o método `Logar` irá chamar o método `loginUsuario` da classe `GerenciadorUsuarios` (linha 7), que irá verificar se o `login` e a senha do usuário estão corretas. Em caso afirmativo (linhas11 e 12), será atribuído a uma variável de seção o valor do objeto `usuario` e o usuário será redirecionado para a página *default.aspx* do sistema. Caso o usuário não esteja cadastrado, o sistema irá informar que o login e/ou senha está incorreto e redirecionará o usuário para a página login.aspx, impedindo a utilização do sistema.

4.2.1.2 Criando um Novo Contrato de Estágio

Nesta página (interface), o usuário irá criar um novo contrato de estágio. Devido ao contrato de estágio possuir vários campos, a solicitação dos mesmos ao usuário foi dividida da seguinte maneira: primeiro são solicitados os dados do estagiário, em seguida os dados da conveniada, logo após as cláusulas, caso exista inciso na cláusula este é solicitado, e em seguida são inseridos o local e a data do contrato de estágio, finalizando assim a criação do mesmo. A Figura 18 apresenta o formulário que solicita os dados do estagiário.

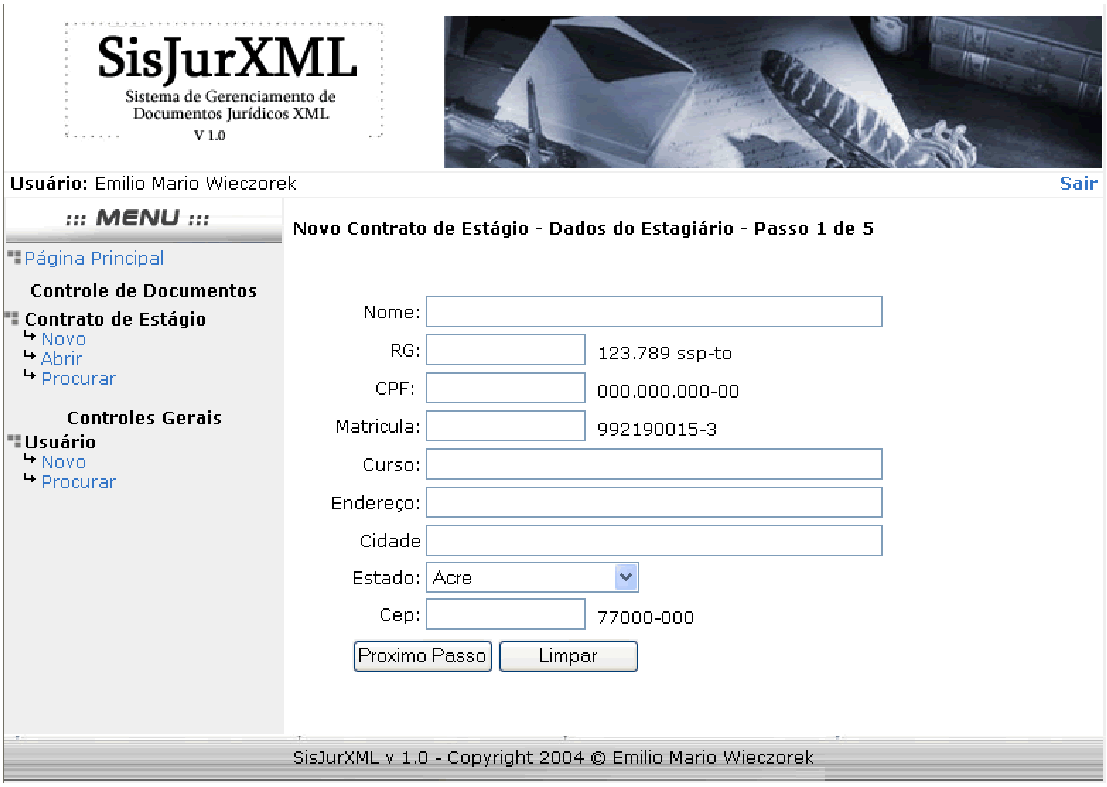

Figura 24: Tela para cadastro das informações do estagiário de um novo Contrato de Estágio

A validação dos campos solicitados nesta fase do cadastro do contrato de estágio são mostrados na figura 24, juntamente com o evento "disparado" pelo botão `Próximo Passo`.

```
1:  protected void Proximo(Object Src, EventArgs E)
2:  {
3:  ContratoEstagio contrato = new ContratoEstagio();
4:  contrato.estagiario = new Estagiario();
5:  contrato.estagiario.nome = NomeEstagiario.Text;
6:  contrato.estagiario.rg = RgEstagiario.Text;
7:  contrato.estagiario.cpf = CpfEstagiario.Text;
8:  contrato.estagiario.matricula = MatriculaEstagiario.Text;
9:  contrato.estagiario.curso = CursoEstagiario.Text;
10: contrato.estagiario.endereco = EndResEstagiario.Text;
11: contrato.estagiario.cidade = CidadeEstagiario.Text;
12: contrato.estagiario.uf =
ufEstagiario.Items[ufEstagiario.SelectedIndex].Value;
13: contrato.estagiario.cep = CepEstagiario.Text;
14: Session["Contrato"] = contrato;
15: Response.Redirect("novoContratoEstagio2.aspx");
16: }
```

Figura 25: Método (evento) executado quando o botão próximo passo é disparado

Como pode ser observado na figura 25, quando o método `Proximo` é executado, este cria um objeto `ContratoEstagio` (linha 3), que por sua vez instancia um objeto `Estagiario` (linha 4), e após isto, faz com que o objeto `ContratoEstagio` armazene as informações do estagiário (linhas 5 a 13). Após os dados do estagiário terem sido transferidos para o objeto `ContratoEstagio`, este é inserido em uma variável de sessão chamada "Contrato" (linha 14), e por fim o usuário é redirecionado para a próxima tela de cadastro do contrato (linha 15). Isto é repetido em todos os formulários que recebem os dados que vão formar o contrato, sempre alterando o Objeto que será inserido no mesmo (`estagiário, conveniada, cláusulas, incisos, local data`).

4.2.1.2 Encerrando um Novo Contrato de Estágio

Após o usuário inserir todas as informações referentes ao novo contrato de estágio, este é redirecionado para uma página que irá "criar" o documento XML referente ao contrato. A figura 20 mostra a tela que indica para o usuário que o contrato de estágio foi criado com sucesso.

Figura 26: Tela que mostra ao usuário que o contrato de estágio foi criado

A tela apresentada na figura 26 mostra ao usuário que o contrato de estágio foi criado com sucesso, informa alguns dados para o usuário, como `Id` do contrato, parte envolvida (neste caso o estagiário), o tipo de documento que foi criado e a data de criação do mesmo, e mostra o botão "Visualizar", para que o usuário possa visualizar o contrato que acabou de ser criado para imprimi-lo ou para fazer alguma outra transação. A figura 27 apresenta o método que foi chamado nesta página para efetuar a criação do documento XML do contrato de estágio.

```
1:  protected void Page_Load(Object Src, EventArgs E)
2:  {
3:  if(Session["Contrato"] != null){
4:  Contrato contrato = (Contrato)Session["Contrato"];
5:  GerenciadorDocumentos gc = new GerenciadorDocumentos();
6:  string dtdpath =
Server.MapPath(@"../dados/contrato_estagio/dtd/contrato_estagio.dtd");
7:  string filename =
erver.MapPath(@"../dados/contrato_estagio/xml/"+contrato.id++".xml");
8:  bool retorno = gc.criarContrato(contrato, dtdpath, filename);
9: if(retorno){
10: descritor.tipo = "Contrato de Estágio";
11: descritor.id = contrato.id;
12: descritor.parte = contrato.estagiario.nome;
13: descritor.data = DateTime.Now.ToString();
14: descritor.usuario = Session["Usuario"].ToString();
15: descritor.url = filename;
16: string urlDescritor =
S7rverMaPath(@"../dados/controle/descritor.xml");
18: retorno = gc.inserirDocumentosDescritor(descritor, urlDescritor);
19: status.Text = "Contrato de Estágio Cadastrado com Sucesso!!";
20: idContrato.Text = contrato.id;
21: Session["Contrato"] = null;
22: }
23: else
24: status.Text = Contrato de Estágio não Cadastrado";
25: }
26: }
```

Figura 27: Método Page_Load que está sendo utilizado para criar o contrato de estágio

É utilizado o método `Page_Load` (linha 1) que é o método executado pelo ASP.NET toda vez que a página é carregada. Observa-se que o método primeiramente verifica se a variável de sessão "Contrato" possui algum valor (linha 4) e, em caso afirmativo, chama o método `criarContrato` da classe `GerenciadorDocumentos` (linha 8), enviando para o método o objeto contrato, o local onde a DTD que irá validar o documento está localizada (linha 6) e o local onde o arquivo deve ser gravado (linha7). É

interessante ressaltar aqui que todos os documentos XML, que armazenam um contrato de estágio criado pelo sistema, possuem o nome igual ao `id` do mesmo. Isso foi utilizado para facilitar as buscas que serão realizadas internamente pelo sistema.

Após gravar as informações do contrato de estágio em um documento XML, o método `Page_load` ainda chama o método `inserirDocumentosDescritor` (linha 18), fazendo com que as informações do contrato de estágio sejam gravadas no arquivo descritor.xml, para que o sistema possa gerenciar o arquivo XML do contrato de estágio gerado. Antes de informar ao usuário se o documento foi criado ou não (linha 23), o sistema limpa o objeto contrato, a fim de limpar a memória do sistema.

4.2.2 Camada de Lógica de Negócio

Esta camada compreende os objetos que realizam o gerenciamento dos dados, ou seja, é a camada responsável por alimentar tanto a camada de apresentação, vista acima, quanto a camada de acesso a dados que será demonstrada na seção 4.2.3. Nas seções a seguir, serão mostrados alguns dos métodos utilizados na codificação da camada de gerenciamento dos dados.

4.2.2.1 Classe GerenciadorUsuarios

Nesta classe, temos todos os métodos responsáveis pelo controle do usuário, desde a inserção de um novo usuário, até a exclusão de um usuário já cadastrado no sistema. Para facilitar a integração com a camada de aplicação, esta classe recebeu um apelido (*namespace*), de mesmo nome. A figura 28 mostra a declaração desta classe.

```
1:   public class GerenciadorUsuarios
2:   {
3:     public XmlDocument novo = null;
4:     public GerenciadorUsuarios()
5:     {
6:
7:     }
8:     .
9:     .
10:    .
11:  }
```

Figura 28: Declaração (escopo) da classe GerenciadorUsuarios

Como é observado na figura 28, na linha um (1) tem-se a declaração da classe, na linha 3 tem-se uma variável pública do tipo `XmlDocument` que será utilizada para gerenciar a manipulação do documento `usuarios.xml` que contém as informações do usuário, na linha 4 é mostrado o construtor desta classe, e entre as linhas 8 e 10, tem-se os métodos que serão demonstrados separadamente abaixo (representados por ponto).

A seguir serão apresentados os métodos mais importantes desta classe.

Método criarUsuario()

Este método é responsável por fazer a inserção de um novo usuário no sistema, ou seja, para criar um novo nó usuário no documento XML que armazena as informações dos usuários.A figura 29 ilustra a codificação deste método, realizada em C#.

```
1:  public bool criarUsuario(Usuario usuario, string url)
2:  {
3:  bool status = this.verificarUsuario(usuario, url);
4:  if(status){
5:  this.novo = new XmlDocument();
6:  novo.Load(url);
7:  XmlElement raiz = novo.DocumentElement;
8:  XmlElement novoUsuario = this.criarElementoXml(this.novo, "usuario");
9:  this.adicionarElementoXml(raiz, novoUsuario);
10: usuario.Attributes["id"].Value = usuario.id;
11: XmlElement nome = this.criarElementoXml(this.novo, "nome");
12: this.insereValorElementoXml(nome, usuario.nome);
13: this.adicionarElementoXml(novoUsuario, nome);
14: XmlElement sobrenome = this.criarElementoXml(this.novo, "sobrenome");
15: this.insereValorElementoXml(sobrenome, usuario.sobrenome);
16: this.adicionarElementoXml(novoUsuario, sobrenome);
17: XmlElement login = this.criarElementoXml(this.novo, "login");
18: this.insereValorElementoXml(login, usuario.login);
19: this.adicionarElementoXml(novoUsuario, login);
20: XmlElement senha = this.criarElementoXml(this.novo, "senha");
21: this.insereValorElementoXml(senha, usuario.senha);
22: this.adicionarElementoXml(novoUsuario, senha);
23: XmlElement setor = this.criarElementoXml(this.novo, "setor");
24: this.insereValorElementoXml(setor, usuario.setor);
25: this.adicionarElementoXml(novoUsuario, setor);
26: this.novo.Save(url);
27: return true;
28: }else
29: return false;
30: }
```

Figura 29: Método criarUsuario()

Como pode ser observado através da figura 29, este método recebe dois parâmetros, um do tipo `Usuario`, que possui as informações do usuário que se deseja inserir e o outro do tipo *string*, que possui o caminho do arquivo em que o novo usuário deve ser inserido, retornando um valor *booleano*, como é demonstrado na linha 1. Isto significa que o método irá retornar verdadeiro (*true*) caso a inserção do novo usuário tenha ocorrido normalmente e falso (*false*) em casos em que não seja possível adicionar um novo elemento. Abaixo estão sendo detalhados os passos mais importantes do código:

Linha 1: chamada do método `criarUsuario();`

Linha 3: atribuía variável status do tipo `bool` o valor da verificação do usuário, informando o `login` para o método `verificarUsuario()`, que será visto mais adiante;

Linha 4: se valor da variável `status` for verdadeiro, inicia a criação do usuário;

Linha 5: indica que a variável `novo`, do tipo `XmlDocument`, irá ser um novo documento XML;

Linha 6: atribui a este novo documento o documento XML passado como parâmetro através da variável `url`;

Linha 7: cria um elemento `raiz` e atribui a este o valor da raiz do documento ML;

Linhas 8 e 9: inicia a inserção do novo usuário no documento XML, criando um novo nó usuário e inserido o mesmo ao documento `usuarios.xml`;

Linha 10: adiciona o atributo `id` ao usuário;

Linhas 11 a 25: cria os elementos que devem conter as informações do usuário (`nome, sobrenome, login, senha, setor`), insere o valor referente a cada um, e adiciona estes elementos ao nó `usuario`;

Linha 26: sobrepõe o arquivo passado como parâmetro através da variável `url`, agora contendo as informações do novo usuário;

Linha 27: retorna verdadeiro (*true*) para a aplicação que solicitou a execução do método;

Linha 28: caso a verificação da linha 3 retorne falso (*false*), o método retorna falso para a aplicação que o solicitou, informando que não foi possível criar o novo usuário.

Método verificarUsuario()

Este método é responsável por verificar a existência de um determinado usuário no documento XML que armazena as informações dos usuários do sistema. A figura 30 ilustra a codificação deste método, realizada em C#.

```
1:  public bool verificarUsuario(Usuario usuario, string url)
2:  {
3:  XmlDocument doc = new XmlDocument();
4:  doc.Load(url);
5:  XmlNode login =
doc.DocumentElement.SelectSingleNode("/usuarios/usuario[login='" +
usuario.login + "']");
6:  if (login != null)
7:  return false;
8:  else
9:  return true;
10: }
```

Figura 30: Método verificarUsuario

Como é observado através da figura 30 na linha 1 do código, este método recebe dois parâmetros, um do tipo `Usuario`, que contém os dados do usuário, e outro que contém o caminho onde deve ser efetuada a verificação do `login` do usuário, e retorna para a aplicação/método que o chamou o valor verdadeiro (caso o login do usuário não exista) e falso (caso o `login` do usuário exista). Abaixo estão sendo detalhados os passos mais importantes do código:

Linha 3: é criado um novo documento XML;

Linha 4: o documento XML é carregado em memória através do documento criado na linha 1 e do caminho que é informado na variável `url`;

Linha 5: é criada uma variável `login` do tipo `XmlNode`, que irá receber o resultado da consulta *XPath* efetuada nó usuário

Linhas 6 a 9: faz a verificação para ver se o `login` foi encontrado, e em caso afirmativo, ou seja, se o `login` existir, retorna falso, e em caso negativo (`login` não existe) retorna *true*.

Método loginUsuario()

Este método é responsável por verificar se um determinado usuário possui ou não permissão de acesso ao sistema, através da comparação do login e senha, informados ao sistema. A figura 31 ilustra a codificação deste método, realizada em C#.

```
1:  public Usuario loginUsuario(string login, string senha, string url)
2:  {
3:  XmlDocument doc = new XmlDocument();
4:  doc.Load(url);
5:  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");
6:  Usuario usuario = new Usuario();
7:  bool status = false;
8:  for(int i = 0; i <listaDeNos.Count ; i++){
9:  string vlogin =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
10: string vsenha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
11:  if((vlogin == login) && (vsenha == senha)){
12:  status = true;
13:  usuario.id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
14:  usuario.nome =
listaDeNos[i].SelectSingleNode("nome").FirstChild.Value.ToString();
15:  usuario.sobrenome =
listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value.ToString();
16:  usuario.login =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
17:  usuario.senha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
18:  usuario.setor =
listaDeNos[i].SelectSingleNode("setor").FirstChild.Value.ToString();
19:  }
20:  }
21:  if(status)
22:  return usuario;
23:  else
24:  return null;
25:  }
```

Figura 31: Método loginUsuario()

Observando a figura 31, linha 1, pode-se perceber que o método recebe três parâmetros (`login`, senha e `url`) que contém respectivamente o `login` informado, a senha informada, e o caminho do arquivo XML que contém as informações dos usuários. A seguir serão listadas as transações mais importantes que o método executa:

Linha 3: é criado um novo documento XML;

Linha 4: o documento XML é carregado em memória através do documento criado na linha 1 e do caminho que é informado na variável `url`;

Linha 5: é criado uma variável que vai armazenar uma lista contendo todos os nós do documento XML;

Linha 6: é criado um novo objeto `usuario` através da chamada a classe `Usuario`;

Linha 7: é criada uma variável para verificar se o usuário possui permissão ao sistema;

Linha 8 a 10: os nós do documento XML são percorridos, para que se possa verificar se o `login` e senha informados estão corretos;

Linha 11: é feita a verificação de `login` e `senha` para constatar se os mesmos estão corretos;

Linha 12: em caso positivo da verificação da linha acima, a variável é definida como *true*, ou seja, o usuário está autorizado a acessar o sistema;

Linhas 13 a 18: as informações contidas do documento XML referentes ao usuário que está acessando o sistema são atribuídas ao objeto usuário;

Linha 21: caso a variável possua o valor *true*, é enviado à aplicação o valor que está contido no objeto usuário (linha 22);

Linha 23: caso os dados de `login` e senha estejam incorretos, é retornado à aplicação o valor `null` (linha 24), informando que a senha ou `login` informados estão incorretos.

Método editarUsuario()

Este método é responsável por alterar as informações de um determinado usuário no arquivo XML que contém as informações dos usuários. A figura 32 ilustra a codificação deste método, realizada em C#.

```
1:  public bool editarUsuario(Usuario usuario, string url)
2:  {
3:  XmlDocument doc = new XmlDocument();
4:  doc.Load(url);
5:  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");
6:  bool status = false;
7:  for(int i = 0; i <listaDeNos.Count ; i++){
8:  string id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
9:  if(id == usuario.id){
```

```
10: listaDeNos[i].SelectSingleNode("nome").FirstChild.Value =
usuario.nome;
11: listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value =
usuario.sobrenome;
12: listaDeNos[i].SelectSingleNode("senha").FirstChild.Value =
usuario.senha;
13: listaDeNos[i].SelectSingleNode("setor").FirstChild.Value =
usuario.setor;
14: status = true;
15: }
16: }
17: doc.Save(url);
18: return status;
19: }
```

Figura 32: Método editarUsuario

Como pode ser visto na figura 32, linha 1, este método recebe como parâmetro um objeto do tipo `Usuario` e uma variável do tipo *string* que, respectivamente, contém os dados do usuário a ser editado e o caminho do arquivo XML onde as informações dos usuários são armazenados, retornando verdadeiro ou falso sobre o sucesso da operação. Abaixo estão listadas as transações mais importantes que o método executa:

Linha 3: é criado um novo documento XML;

Linha 4: o documento XML é carregado em memória através do documento criado na linha 1 e do caminho que é informado na variável `url`;

Linha 5: é criado uma variável que vai armazenar uma lista contendo todos os nós do documento XML;

Linha 6: é criada uma variável `status` para verificar se o usuário possui permissão ao sistema;

Linhas 7 e 8: os nós do documento XML são percorridos, até encontrar o `id` do usuário que se deseja efetuar as modificações;

Linha 9: é feita a verificação do `id` do usuário;

Linhas 10 a 13: os dados do usuário que são possíveis de ser alterados (`nome`, `sobrenome`, `senha`, `setor`) são atribuídos ao documento XML no usuário desejado;

Linhas 14: a variável `status` é definida como *true*, informando que as modificações forma efetuadas;

Linha 17: o arquivo XML é sobreposto, agora com as alterações processadas no usuário desejado;

Linha 18: é retornado à aplicação o *status* da operação (*true* ou *false*), informando se foi ou não possível alterar o usuário.

Método removerUsuario()

Este método possui a funcionalidade de excluir um usuário do sistema através da exclusão do mesmo do arquivo XML que detém as informações do usuário. A figura 33 ilustra a codificação deste método, realizada em C#.

```
1:  public bool removerUsuario(Usuario usuario, string url)
2:  {
3:  XmlDocument doc = new XmlDocument();
4:  doc.Load(url);
5:  XmlNode remUsuario =
doc.DocumentElement.SelectSingleNode("/usuarios/usuario[id='" +
usuario.id + "']");
6:  if(remUsuario != null ){
7:  remUsuario.RemoveChild(usuario);
8:  doc.Save(url);
9:  return true;
10: }else
11: return false;
12: }
```

Figura 33: Método removerUsuario

Como é verificado na figura 33, linha 1, o método recebe dois parâmetros, um objeto `Usuario` e uma variável do tipo *string* e retorna verdadeiro ou falso, de acordo com o andamento do método. Abaixo estão listadas as funcionalidades mais importantes do método.

Linha 3: é criado um novo documento XML;

Linha 4: o documento XML é carregado em memória através do documento criado na linha 1 e do caminho que é informado na variável `url`;

Linha 5: é criado uma variável que vai conter o nó resultante da pesquisa *XPath* utilizada, que neste caso busca o nó onde o `id` do usuário seja igual ao passado pelo objeto usuário;

Linha 6: é feita a verificação do `id` do usuário;

Linha 7: caso o usuário exista, condição da linha 6, é feita a remoção do nó filho usuário;

Linha 8: o documento XML informado na variável `url` é sobreposto, agora sem existir o nó usuário que foi excluído;

Linhas 9 a 11: o método retorna *true* caso a exclusão tenha sido efetuada com sucesso e *false* caso o `id` do usuário não tenha sido encontrado.

4.2.2.1 Classe GerenciadorDocumentos

Nesta classe, tem-se todos os métodos responsáveis pelo controle de documentos, desde a inserção de um novo documento, até a exclusão de um documento já existente no sistema. Para facilitar a integração com a camada de aplicação, esta classe recebeu um apelido (*namespace*), de mesmo nome. A figura 34 mostra a declaração desta classe:

```
1:   public class GerenciadorDocumentos
2:   {
3:     public XmlDocument novo = null;
4:     public GerenciadorDocumentos()
5:     {
6:
7:     }
8:     .
9:     .
10:    .
11:  }
```

Figura 34: Classe GerenciadorDocumentos

Observando na 34 acima, na linha 1 tem-se a declaração da classe, na linha 3 uma variável pública do tipo `XmlDocument` que será utilizada para gerenciar a manipulação do documento usuarios.xml que contém as informações do usuário, na linha 4 tem-se o construtor desta classe, e entre as linhas 8 e 10, tem-se os métodos que serão demonstrados separadamente a seguir (representados por ponto).

A seguir serão apresentados os métodos mais importantes desta classe, ressaltando que como são muitas informações para serem inseridas no documento XML, os métodos não serão mostrados por inteiro, e que caso exista alguma dúvida sobre o funcionamento do mesmo deverá ser verificado nos anexos deste trabalho, que contém a codificação na íntegra.

Método CriarContratoEstagio

Este método possui a funcionalidade de criar um novo contrato de estágio, através da criação de um documento XML que armazena as informações do referido contrato. A figura 35 ilustra a codificação deste método, realizada em C#.

```
1:  public bool criarContratoEstagio(ContratoEstagio contrato, string
dtdpath, string filename)
2:  {
3:  novo = new XmlDocument();
4:  XmlDeclaration xmldecl = novo.CreateXmlDeclaration("1.0", "UTF-8",
null);
5:  novo.AppendChild(xmldecl);
6:  XmlDocumentType doctype = novo.CreateDocumentType("contrato", null,
dtdpath, null);
7:  novo.AppendChild(doctype);
8:  XmlElement raiz = this.criarElementoXml(this.novo, "contrato");
9:  this.novo.AppendChild(raiz);
10: //Inserindo ID do Contrato
11: XmlElement idContrato = this.criarElementoXml(this.novo, "id");
12: this.insereValorElementoXml(idContrato, contrato.id);
13: this.adicionarElementoXml(raiz, idContrato);
14: .
15: .
16: .
17: XmlElement clausula;
18: XmlElement numeroClausula;
19: XmlElement textoClausula;
20: XmlElement inciso;
21: XmlElement letraInciso;
22: XmlElement textoInciso;
23: for(int i=0; i < contrato.clausulas.Length; i++)
24: {
25: clausula = this.criarElementoXml(this.novo, "clausula");
26: this.adicionarElementoXml(raiz, clausula);
27: numeroClausula = this.criarElementoXml(this.novo, "numero");
28: this.insereValorElementoXml(numeroClausula,
contrato.clausulas[i].numero);
29: this.adicionarElementoXml(clausula, numeroClausula);
30: textoClausula = this.criarElementoXml(this.novo, "texto");
31: this.insereValorElementoXml(textoClausula,
contrato.clausulas[i].texto);
32: this.adicionarElementoXml(clausula, textoClausula);
33: if(contrato.clausulas[i].inciso.Length > 0)
34: {
35: for(int j=0; j < contrato.clausulas[i].inciso.Length; j++)
36: {
37: inciso = this.criarElementoXml(this.novo, "inciso");
38: this.adicionarElementoXml(clausula, inciso);
39: letraInciso = this.criarElementoXml(this.novo, "letra");
40: this.insereValorElementoXml(letraInciso,
contrato.clausulas[i].inciso[i].letra);
41: this.adicionarElementoXml(inciso, letraInciso);
42: textoInciso = this.criarElementoXml(this.novo, "texto");
43: this.insereValorElementoXml(textoInciso,
contrato.clausulas[i].inciso[i].texto);
44: this.adicionarElementoXml(inciso, textoInciso);
```

```
45: }
46: }
47: }
48: XmlElement local = this.criarElementoXml(this.novo, "local");
49: this.insereValorElementoXml(local, contrato.local);
50: this.adicionarElementoXml(raiz, local);
51: XmlElement data = this.criarElementoXml(this.novo, "data");
52: this.insereValorElementoXml(data, contrato.data.ToString());
53: this.adicionarElementoXml(raiz, data);
54: this.novo.Save(filename);
55: return true;
56: }
```

Figura 35: Método criarContratoEstagio

Como pode ser observado na figura 35, nas linhas 14 a 16 foram colocados pontos representando parte do código do método que foi retirado desta figura, pois realizam transações semelhantes às que são observadas entre as linhas 11 e 13.

O método em questão recebe três parâmetros, um objeto `ContratoEstagio`, e duas *string's*. O objeto contrato possui as informações do contrato (`id, estagiário, conveniada, cláusulas, local, data`), enquanto que a primeira *string* possui o caminho da DTD que valida o documento XML que será gerado e a segunda contém o cominho onde o documento XML será armazenado. Abaixo estão sendo listadas as funcionalidades mais importantes deste método.

Linha 3: é criado um documento XML em memória;

Linhas 4 e 5: é criada e inserida no documento a declaração do documento, que irá conter as informações do documento, como versão, e codificação;

Linhas 6 e 7: é criada e inserida a declaração DOCTYPE no documento XML;

Linhas 8 e 9: é atribuído o nó raiz contrato ao documento XML;

Linhas 11 a 13: é criado, atribuído valor e adicionado ao documento XML o elemento `id`;

Linhas 14 a 16: são efetuados os procedimentos para inserir os valores do estagiário e da conveniada ao contrato, sendo efetuados da mesma forma como foi demonstrado nas linhas 11 a 13, somente não atribuindo valor aos nós pais `estagiario` e `conveniada`;

Linhas 17 a 22: são criadas as variáveis que vão receber os valores a serem inseridos nos filhos `clausula` e `inciso` do documento XML;

Linhas 23 a 32: o objeto `contrato` é percorrido enquanto existir cláusulas no mesmo, sendo feita a inserção dos valores em seus respectivos nó (caso exista mais de uma cláusula, serão inseridos vários nós cláusulas filhos no documento XML);

Linha 33: é verificado se existe inciso em cada cláusula que está sendo inserida;

Linha 35 a 44: caso a condição da linha anterior seja satisfeita, faz o mesmo que nas linhas 23 a 32, só que neste caso serão criados os nós incisos;

Linhas 48 a 54: são inseridos do documento XML os nós `local` e `data`, com seus respectivos valores que estão no objeto `contrato`;

Linha 55: o documento é salvo no local especificado pela variável `url` (é interessante ressaltar que o caminho já deve conter o nome do arquivo e sua extensão, que no nosso caso é .xml).

Método visualizarContratoEstagio

Este método possui a funcionalidade de visualizar um contrato de estágio através da informação do caminho do documento XML que o armazena. A figura 36 ilustra a codificação deste método, realizada em C#.

```
1:  public Contrato vsisualizarContratoEstagio(string url)
2:  {
3:  XmlDocument doc = new XmlDocument();
4:  doc.Load(url);
5:  XmlElement raiz = doc.DocumentElement;
6:  ContratoEstagio contrato = new ContratoEstagio();
7:  contrato.id = ValorDoElemento(raiz, "/contrato/estagiario/id");
8:  contrato.estagiario = new Estagiario();
9:
10: contrato.estagiario.nome = ValorDoElemento(raiz,
"/contrato/estagiario/nome");
11: contrato.estagiario.rg = ValorDoElemento(raiz,
"/contrato/estagiario/rg");
12: contrato.estagiario.cpf = ValorDoElemento(raiz,
"/contrato/estagiario/cpf");
13: contrato.estagiario.matricula =
ValorDoElemento(raiz,"4contrato/estagiario/matricula");
15: contrato.estagiario.curso = ValorDoElemento(raiz,
"/contrato/estagiario/curso");
16: contrato.estagiario.endereco = ValorDoElemento(raiz,
"/contrato/estagiario/endereco");
17: contrato.estagiario.cidade = ValorDoElemento(raiz,
"/contrato/estagiario/cidade");
18: contrato.estagiario.uf = ValorDoElemento(raiz,
"/contrato/estagiario/uf");
19: contrato.estagiario.cep = ValorDoElemento(raiz,
"/contrato/estagiario/cep");
20: .
21: .
22: .
23: XmlNodeList listaClausulas, listaIncisos;
```

```
24: XmlNode no = doc.DocumentElement;
25: listaClausulas = no.SelectNodes("/contrato/clausula");
26: contrato.clausulas = new Clausula[listaClausulas.Count];
27: for(int i = 0; i < listaClausulas.Count; i++)
28: {
29: contrato.clausulas[i].numero =
listaClausulas[i].SelectSingleNode("numero").FirstChild.Value.ToString();
30: contrato.clausulas[i].texto =
listaClausulas[i].SelectSingleNode("texto").FirstChild.Value.ToString();
31: listaIncisos = listaClausulas[i].SelectNodes("inciso");
32: if(listaClausulas[i].SelectSingleNode("inciso").FirstChild.Value !=
null){
33: for(int j = 0; j < listaIncisos.Count; j++){
34: contrato.clausulas[i].inciso[j] = new Inciso();
35: contrato.clausulas[i].inciso[j].letra =
listaIncisos[j].SelectSingleNode("letra").FirstChild.Value.ToString();
36: contrato.clausulas[i].inciso[j].texto =
listaIncisos[j].SelectSingleNode("texto").FirstChild.Value.ToString();
37: }
38: }
39: }
40: contrato.local = ValorDoElemento(raiz, "/contrato/conveniada/local");
41: contrato.data = ValorDoElemento(raiz, "/contrato/conveniada/data");
42: return contrato;
43: }
```

Figura 36: Método visualizarContratoEstagio

Conforme o código apresentado na figura 36 linha 1, este método recebe como parâmetro somente a `url` do arquivo onde este contrato está sendo armazenado, retornando um objeto `ContratoEstagio` caso o mesmo seja encontrado, ou `null`, informando que o mesmo não foi localizado ou não possui os dados corretos. Observa-se também que nas linhas 20 a 22 foram utilizados pontos, pois as operações que o método executa nesta parte do código é muito semelhante as que são demonstradas nas linhas 10 a 19, e podem ser vistas de forma integral nos anexos deste trabalho.

Abaixo serão mostrados os passos que este método executa:

Linha 3: é criado um documento XML em memória;

Linha 4: é atribuído a este novo documento o valor contido no documento XML que está localizado na variável `url`;

Linha 6: é criado um novo objeto `contrato`, que irá receber as informações do documento XML que está na memória;

Linhas 7 a 19: são atribuídos ao objeto `ContratoEstagio` os valores referentes que estão no documento XML;

linhas 23 a 39: são realizadas as transformações necessárias para atribuir ao objeto `contrato` os valores das cláusulas e dos incisos que existem do documento XML em questão;

Linhas 40 e 41: são atribuídos ao objeto `contrato` os valores dos elementos `local` e `data` que estão no documento XML;

Linha 42: é retornado à aplicação o objeto `contrato`;

Método InserirDocumentoDescritor

Este método possui a capacidade de inserir em um documento XML (`descritor.xml`), que descreve todos os documentos que são inseridos no sistema, os dados de cada documento jurídico que é suportado pelo sistema, a fim de facilitar o gerenciamento (busca, remoção, etc.) dos mesmos.

É importante salientar, que este método é responsável por criar um "arquivo de índices" para os documentos jurídicos armazenados no sistema. As informações que são inseridas no documento XML (`descritor.xml`), através deste método, são as informações que estarão disponíveis para buscar algum documento jurídico armazenado no sistema. Este método é executado automaticamente ao se encerrar o cadastro de um novo documento jurídico no sistema. A figura 37 esboça como o método seria acionado. A figura 38 ilustra a codificação deste método, realizada em C#.

Figura 37: Esboço do acionamento do método inserirDocumentoDescritor

Como é observado na figura 37, quando o usuário finaliza a criação de um contrato de estágio, o sistema cria um documento XML representando o contrato de estágio e insere no documento `descritor.xml` as informações sobre o contrato de estágio que acaba de ser criado. Na seção 4.2.3 será demonstrado que informações são armazenadas no documento XML do descritor de documentos.

```
1:  public bool inserirDocumentoDescritor(DescritorDocumento descritor,
string urlDescritor)
2:  {
3:  novo = new XmlDocument();
4:  novo.Load(urlDescritor);
5:  XmlElement raiz = novo.DocumentElement;
6:  XmlElement documento = this.criarElementoXml(this.novo, "documento");
7:  this.adicionarElementoXml(raiz, documento);
8:  Inserindo informações no Descritor
9:  XmlElement tipo = this.criarElementoXml(this.novo, "tipo");
10: this.insereValorElementoXml(tipo, descritor.tipo);
11: this.adicionarElementoXml(documento, tipo);
12: XmlElement id = this.criarElementoXml(this.novo, "id");
13: this.insereValorElementoXml(id, descritor.id);
14: this.adicionarElementoXml(documento, id);
15: XmlElement parte = this.criarElementoXml(this.novo, "parte");
16: this.insereValorElementoXml(parte, descritor.parte);
17: this.adicionarElementoXml(documento, parte);
17: XmlElement usuario = this.criarElementoXml(this.novo, "usuario");
18: this.insereValorElementoXml(usuario, descritor.usuario);
19: this.adicionarElementoXml(documento, usuario);
20: XmlElement data = this.criarElementoXml(this.novo, "data");
21: this.insereValorElementoXml(data, descritor.data);
22: this.adicionarElementoXml(documento, data);
23: XmlElement url = this.criarElementoXml(this.novo, "url");
24: this.insereValorElementoXml(url, descritor.url);
25: this.adicionarElementoXml(documento, url);
26: this.novo.Save(urlDescritor);
27: return true;
28: }
```

Figura 38: Método inserirDocumentoDescritor

Como podemos observar na figura 38, o método `inserirDescritorContrato` é muito semelhante ao método `criarUsuario`, descrito na seção 4.2.2.1, e por tal motivo, somente será descrito aqui, os parâmetros que o método recebe e o tipo retorno que o mesmo possui. Seguindo, então, podemos observar pela linha 1, que o método receberá um objeto `DescritorDocumento` e uma variável do tipo *string*, que pelos quais o método obtém os dados a serem inseridos no arquivo XML que descreve os documentos (`documento`, `tipoDocumento` (representado pelo

nó tipo do documento), `usuario`, `data`, `url`) e o caminho onde o arquivo do descritor se encontra, respectivamente.

4.2.2.1 Classe UtilXML

Nesta classe, tem-se os métodos que são comuns (genéricos) e que são utilizados pelas classes `GerenciadorUsuarios` e `GerenciadorContratos`. Os métodos criados nesta classe são métodos simples, que poderiam ser trabalhados de forma direta nas outras classes, mas foi definido que os mesmos seriam incorporados a uma classe genérica (de utilidades), a fim de melhorar o controle de erros do sistema, facilitar a reutilização de código, e facilitar futuras manutenções ao sistema.

Método criarElementoXml

Este método tem a capacidade de criar um novo elemento filho em um elemento pai. A figura 39 ilustra a codificação deste método, realizada em C#.

```
1: public XmlElement criarElementoXml(XmlDocument pai, string
nomeElemento)
2: {
3: return pai.CreateElement(nomeElemento);
4: }
```

Figura 39: Método CriarElementoXml

Na figura 39, linha 1, é observado que o método irá receber dois atributos, um nó pai e uma variável do tipo *string* com o nome do elemento que deve ser adicionado ao nó pai, e em seguida, na linha 3, o método retorna o novo elemento filho que foi criado ao elemento pai.

4.2.3 Camada de Acesso a Dados

Esta camada é responsável pelo acesso aos dados do sistema, e neste caso, a mesma é responsável pelo acesso aos documentos XML que contém as informações armazenadas.

Como pode ser observado na seção 4.2.2, que mostra a camada de Lógica de negócio, sempre que se trabalha com os documentos XML do sistema, o mesmo é carregado integralmente na memória, são feitas as transformações que devem ser efetuadas e, após isto, os documentos são salvos. Esta técnica é utilizada principalmente porque o trabalho foi realizado com o DOM, e este não provê métodos de acesso ao documento XML sem que haja o carregamento do mesmo em memória. (PAINE, 2001).

Além disso, fica mais fácil de se trabalhar com o documento quando já se tem uma "cópia" do mesmo na memória, pois é possível realizar as transformações necessárias no mesmo, sem que estas sejam atribuídas de forma automática, evitando assim que dados incoerentes sejam inseridos nos documentos XML.

O documento XML mais importante para a camada de acesso a dados é o `descritor.xml` (descritor de documentos), que armazena as informações mais importantes de cada documento jurídico armazenado no sistema, como `tipoDocumento`, `parte`, `id` (do documento), `data`, e `url`. Este documento servirá como um arquivo de índices para os documentos armazenados no sistema, pois é através deste documento que o usuário poderá fazer as buscas para encontrar um documento jurídico específico, escolhendo fazer esta busca a partir de algum dado que esteja disponível no descritor (`tipoDocumento`, `parte`, `id`, `data`, e `url`).

Sem o `descritor.xml`, a busca por um documento jurídico armazenado no sistema se tornaria muito cara, em termos de tempo e processamento, pois seria necessário carregar todos os documentos XML que armazenam os documentos do sistema na memória, um por um, até que fosse encontrado o documento que se está procurando, além de o sistema não ter em parte alguma o caminho (`url`) destes documentos armazenados.

Uma outra informação importante que deve ser colocada neste momento é que todos os documentos XML do sistema ficam armazenados fora do compartilhamento *web*, o que faz com que somente a aplicação (sistema) tenha acesso aos mesmos, impedindo que estes sejam manipulados sem a utilização do sistema.

O acesso ao documento XML no disco rígido da máquina que roda a aplicação, como dito anteriormente, é feito através de uma cópia do arquivo para a memória, evitando assim que o disco seja acessado a todo o instante, já que o documento XML em questão somente é enviado ao disco após o término de todas as transformações que ocorrem sobre o documento XML.

5 Conclusões

O desenvolvimento de um sistema baseado em XML, DOM e .NET é sem dúvida muito interessante, pois além de conhecer funcionalidades do XML, também foi possível descobrir e utilizar uma nova plataforma de desenvolvimento, no caso a plataforma .NET, através da utilização do ASP.NET e do C#.

O que mais dificultou a realização deste trabalho foi a falta de experiência em se trabalhar com XML e DOM e o não conhecimento das funcionalidades da plataforma .NET que tiveram que ser vistas no decorrer do trabalho.

Além de possibilitar espaço para projetos futuros envolvendo as tecnologias de desenvolvimento discutidas neste trabalho, o mesmo possibilitou que se tivesse uma outra visão sobre documentos XML. Ou seja, foi possível compreender, através de uma atividade prática, que o XML não serve apenas para descrever conteúdos tirados de fontes de dados e facilitar o intercâmbio dos mesmos, mas também para o armazenamento dessa informações.

O objetivo deste trabalho foi, além do desenvolvimento de uma ferramenta para criação de documentos jurídicos, a busca de formas para ampliar a capacidade de leitura, compreensão e manipulação de documentos jurídicos a partir de um formato flexível de armazenamento, possibilitando também o estudo de ferramentas antes desconhecidas, como o DOM, e a plataforma .NET. Os documentos gerados poderão ser trabalhados de forma independente da plataforma aplicada, através da adoção da linguagem XML, sendo que a manipulação dos documentos foi realizada através da implementação da API do DOM em .NET.

As contribuições deste trabalho estão relacionadas ao estudo da aplicação do DOM e do XML com a plataforma .NET, servindo de apoio para trabalhos futuros que venham a

implementar sistemas de forma semelhante à demonstrada neste trabalho e também a oferecer suporte para a continuação do sistema aqui apresentado.

Como trabalhos futuros podem ser inseridos mais módulos no sistema, para que o mesmo possa realizar o gerenciamento do maior número de documentos jurídicos possíveis, além de se tentar o desenvolvimento do mesmo utilizando-se de outras tecnologias, como o XML *Schema* ao invés da utilização DTD's; a melhora do sistema de buscas, podendo o usuário buscar por cláusulas e outras informações que não estejam contidas somente no descritor de documentos; a utilização de outras linguagens de programação para Web, como PHP ou JSP; além do estudo de linguagens de marcação, com origem no XML, próprias para documentos jurídicos.

Referências Bibliográficas

ABITEBOUL, Serge; BUNEMAN, Peter; SUCIU, Dan. **Gerenciando Dados na Web**. Campus: Rio de Janeiro, 2000.

EVANS, Kirk A.; KAMANNA, Ashwin; MUELLER, Joel. **XML e ASP.NET**. Rio de Janeiro: Editora Ciência Moderna, 2003.

GUIMARÃES, Guilherme. **Criando Componentes no ASP.NET**. Disponível em < http://www.imasters.com.br/web/conteudo/coluna_aspnet.php?codcoluna=2569>. Acesso em 15/04/2004.

KADE, Adrovane M.; HEUSER, Carlos A.. **Tendências em Linguagem de Consulta para documentos XML**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2001.

MCGRATH, Sean. XML: **Aplicações Práticas**. Rio de Janeiro: Campus, 1999.

MSDN. **NSDN Library**. Disponível em < http://msdn.microsoft.com/library/ >. Acesso em 25/06/2004.

NATANYA P. M.; KIRK Cheryl. **XML Black Book**. São Paulo: Makron Books, 2000.

REILLY, O'Reilly & Associates, Inc; **.Net Framework Essentials – Introducing the .Net Framework**. O'Reilly & Associates, Inc. 2000.

PAINE, Chris. **Aprenda em 21 dias ASP.NET**. Rio de Janeiro: Campus, 2001.

SANT'ANA, Mauro. **C#: A nova linguagem da arquitetura .NET**. Disponível em < http://www.linhadecodigo.com.br/artigos.asp?id_ac=86 >. Acesso em 07/04/2004.

SHUI, Willian M. **Utilizing Multiple Bioinformatic Information Sources: An XML Database pproach 2001 Bioinformatics Honours Thesis**. Sydney: University of New South Wales, 2001.

VALENTINE, Chelsea; MINNICK, Cris. **XHTML**. Rio de Janeiro: Campus, 2001.

W3 SCHOOLS. **ASP.NET Tutorial**. Disponível em < http://www.w3schools.com >. Acesso em 07/04/2004.

WAHLIN, Dan. **AML e ASP.NET para Desenvolvedores**. São Paulo: Pearson Education do Brasil, 2003.

Anexos

Anexo I – Modelo Utilizado para Mapear a Classe ContratoEstagio

TERMO DE COMPROMISSO DE ESTÁGIO

Pelo presente Termo de Compromisso de Estágio entre si celebram, de um lado como beneficiário e doravante denominado ESTAGIÁRIO:
Nome:_____
RG:_____ CPF:_____
Matrícula no Colégio:_____
Curso: _____
Endereço Resid.: _____
Cidade:_____ UF:___ CEP: _____

e de outro como CONVENIADA:

****

O CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS, Instituição de Ensino Superior, situada na Av. Teotônio Segurado- 1.501 Sul – Palmas/TO, CEP 77054-970, CGC/MF N.º 99999999/9999-99, neste ato representada por seu Diretor Geral, professor xxxxxxxxxxxx, portador do CPF/MF 999.999.999-99, doravante denominada INTERVENIENTE, celebram entre si este TERMO DE COMPROMISO DE ESTÁGIO – TCE, convencionando as cláusulas e condições seguintes:

1- O presente TERMO DE COMPROMISSO DE ESTÁGIO – TCE decorre do CONVÊNIO celebrado entre as partes, nos termos estabelecidos pelo Decreto n.º 8.7497/82, que regulamenta a Lei n.º 6.494/77, e tem por finalidade particularizar a relação jurídica especial existente entre ESTAGIÁRIO e a UNIDADE CONCEDENTE, não caracterizando nenhuma vinculação empregatícia.

2 - Na vigência do presente Termo o ESTAGIÁRIO estará segurado contra acidentes pessoais, pela INTERVENIENTE, através da Apólice **** da Seguradora **** .

3 - Ficam compromissadas entre as partes as seguintes condições básicas para realização do ESTÁGIO:

a) este TERMO DE COMPROMISSO DE ESTÁGIO – TCE terá vigência de ----- /---- /----- a ---/----/------, podendo ser denunciado a qualquer tempo, unilateralmente, mediante comunicação escrita ou ser prorrogado através da emissão de um TERMO ADITIVO.

b) as atividades de ESTÁGIO a serem cumpridas pelo ESTAGIÁRIO serão desenvolvidas no horário das ____ horas, totalizando_____hora/aulas semanais.

c) as atividades principais a serem desenvolvidas pelo ESTAGIÁRIO, deverão ter supervisão de um profissional especializado, que deverá ser designado pela CONVENENTE. Essas atividades terão caráter subsidiário e complementar, compatíveis com o Contexto Básico da Profissão ao qual o Curso se refere.

4 - No desenvolvimento do ESTÁGIO ora compromissado, caberá ao ESTÁGIÁRIO:
a) cumprir com todo o empenho e interesse toda programação estabelecida para seu ESTÁGIO.
b) observar e obedecer às normas internas da UNIDADE CONCEDENTE.
c) elaborar e entregar a INSTITUIÇÃO DE ENSINO, relatórios sobre seu ESTÁGIO na forma, prazo e padrões estabelecidos.

5 - Constituem motivos para a interrupção automática da vigência do presente TERMO DE COMPROMISSO DE ESTÁGIO:

a) - a conclusão ou abandono do curso e o trancamento da matrícula;
b) - o não cumprimento do convencionado neste TERMO DE COMPROMISSO DE ESTÁGIO, bem como do CONVÊNIO.

E, por estarem de inteiro e comum acordo com as condições e dizeres do CONVÊNIO e do decorrente TERMO DE COMPROMISSO DE ESTÁGIO – TCE, as partes assinam em 4 (quatro) vias de igual teor.

Palmas, **********2003

ESTAGIÁRIO

CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

****

Anexo II – Código Fonte da Classe GerenciadorUsuarios

```
using System;
using System.Xml;
using System.IO;
using System.Data;

namespace GerenciadorUsuarios
{
      public class GerenciadorUsuarios
      {

            public XmlDocument novo = null;

            public GerenciadorUsuarios()
            {

            }

            public bool criarUsuario(Usuario usuario, string url)
            {
                  //Verifica se existe usuário com mesmo Login Cadastrado
no sistema
                  bool status = this.verificarUsuario(usuario, url);
                  //Se não existir login já cadastrado, efetua a inserção
do usuário no doc XML
                  if(status){
                        this.novo = new XmlDocument();
                        novo.Load(url);
                        XmlElement raiz = novo.DocumentElement;
                        //Inserindo novo Usuario
                        XmlElement novoUsuario =
this.criarElementoXml(this.novo, "usuario");
                        this.adicionarElementoXml(raiz, novoUsuario);
                        //Inserindo informações do Usuário
                        XmlElement id = this.criarElementoXml(this.novo,
"id");
                        this.insereValorElementoXml(id, usuario.id);
                        this.adicionarElementoXml(novoUsuario, id);

                        XmlElement nome =
this.criarElementoXml(this.novo, "nome");
                        this.insereValorElementoXml(nome, usuario.nome);
                        this.adicionarElementoXml(novoUsuario, nome);

                        XmlElement sobrenome =
this.criarElementoXml(this.novo, "sobrenome");
                        this.insereValorElementoXml(sobrenome,
usuario.sobrenome);
                        this.adicionarElementoXml(novoUsuario,
sobrenome);

                        XmlElement login =
this.criarElementoXml(this.novo, "login");
                        this.insereValorElementoXml(login,
usuario.login);
                        this.adicionarElementoXml(novoUsuario, login);
```

```
                        XmlElement senha =
this.criarElementoXml(this.novo, "senha");
                        this.insereValorElementoXml(senha,
usuario.senha);
                        this.adicionarElementoXml(novoUsuario, senha);

                        XmlElement setor =
this.criarElementoXml(this.novo, "setor");
                        this.insereValorElementoXml(setor,
usuario.setor);
                        this.adicionarElementoXml(novoUsuario, setor);
                        //Salvando Usuário
                        this.novo.Save(url);
                        //confirma o cadastro do usuário
                        return true;
                  //se existir login já cadastrado retorna false
                  }else
                        return false;
            }

      public bool verificarUsuario(Usuario usuario, string url)
      {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNode login =
doc.DocumentElement.SelectSingleNode("/usuarios/usuario[login='" +
usuario.login + "']");
                  if (login != null)
                        return false;
                  else
                        return true;
            }

            public Usuario loginUsuario(string login, string senha,
string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");

                  Usuario usuario = new Usuario();
                  bool status = false;
                  string vlogin = null;
                  string vsenha = null;

                  for(int i = 0; i < listaDeNos.Count ; i++)
                  {
                        try
                        {
                              vlogin =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
                              vsenha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
                        }
                        catch(Exception e)
                        {
                        }
```

```
                        if((vlogin == login) && (vsenha == senha))
                        {
                              status = true;
                              usuario.id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
                              usuario.nome =
listaDeNos[i].SelectSingleNode("nome").FirstChild.Value.ToString();
                              usuario.sobrenome =
listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value.ToString();
                              usuario.login =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
                              usuario.senha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
                              usuario.setor =
listaDeNos[i].SelectSingleNode("setor").FirstChild.Value.ToString();
                        }
                  }


                  if(status)
                        return usuario;
                  else
                        return null;
            }

            public Usuario buscarUsuario(string busca, string tipo,
string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");

                  Usuario usuario = new Usuario();

                  bool status = false;

                  if(tipo == "1"){
                        for(int i = 0; i <listaDeNos.Count ; i++){
                              string login =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();

                              if(login == busca){
                                    status = true;
                                    usuario.id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
                                    usuario.nome =
listaDeNos[i].SelectSingleNode("nome").FirstChild.Value.ToString();
                                    usuario.sobrenome =
listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value.ToString();
                                    usuario.login =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
                                    usuario.senha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
                                    usuario.setor =
listaDeNos[i].SelectSingleNode("setor").FirstChild.Value.ToString();
                              }
                        }
```

```
                  }
                  if(status)
                        return usuario;
                  else
                        return null;
            }

            public Usuario visualizarUsuario(string id, string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");

                  Usuario usuario = new Usuario();

                  for(int i = 0; i <listaDeNos.Count ; i++){
                        string vid =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();

                        if(vid == id){
                              usuario.id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
                              usuario.nome =
listaDeNos[i].SelectSingleNode("nome").FirstChild.Value.ToString();
                              usuario.sobrenome =
listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value.ToString();
                              usuario.login =
listaDeNos[i].SelectSingleNode("login").FirstChild.Value.ToString();
                              usuario.senha =
listaDeNos[i].SelectSingleNode("senha").FirstChild.Value.ToString();
                              usuario.setor =
listaDeNos[i].SelectSingleNode("setor").FirstChild.Value.ToString();
                        }
                  }
                  return usuario;
            }

            public DataSet listarUsuarios(string url)
            {
                  DataSet user = new DataSet();
                  user.ReadXml(url);
                  return user;
            }

            public int quantUsuarios(string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList listaUsuarios =
doc.DocumentElement.SelectNodes("/usuarios/usuario");

                  return listaUsuarios.Count;
            }

            public bool editarUsuario(Usuario usuario, string url)
            {
                  XmlDocument doc = new XmlDocument();
```

```
                  doc.Load(url);

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/usuarios/usuario");

                  bool status = false;

                  for(int i = 0; i <listaDeNos.Count ; i++){
                        string id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();

                        if(id == usuario.id){
                              status = true;

      listaDeNos[i].SelectSingleNode("nome").FirstChild.Value =
usuario.nome;

      listaDeNos[i].SelectSingleNode("sobrenome").FirstChild.Value =
usuario.sobrenome;

      listaDeNos[i].SelectSingleNode("senha").FirstChild.Value =
usuario.senha;

      listaDeNos[i].SelectSingleNode("setor").FirstChild.Value =
usuario.setor;
                        }
                  }
                  doc.Save(url);
                  return status;
            }

            public bool removerUsuario(Usuario usuario, string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNode remUsuario =
doc.DocumentElement.SelectSingleNode("/usuarios/usuario[id='" +
usuario.id + "']");
                  XmlNode noParente = remUsuario.ParentNode;

                  if(remUsuario != null ){
                        noParente.RemoveChild(remUsuario);
                        doc.Save(url);
                        return true;
                  }else
                        return false;
            }
      }

      public class Usuario
      {
            public string id;
            public string nome;
            public string sobrenome;
            public string login;
            public string senha;
            public string setor;
      }
}
```

Anexo III – Código Fonte da Classe GerenciadorDocumentos

```
using System;
using System.Xml;
using System.IO;
using System.Data;

namespace GerenciadorDocumentos
{
      public class GerenciadorDocumentos
      {
            XmlDocument novo = null;

            public GerenciadorDocumentos()
            {

            }

            public bool criarContratoEstagio(Contrato contrato, string
dtdpath, string filename)
            {

                  novo = new XmlDocument();

                  // Cria a declaracao XML (version, encoding -- usar
UTF-8 para poder armazenar caracteres especiais)
                  XmlDeclaration xmldecl =
novo.CreateXmlDeclaration("1.0", "UTF-8", null);
                  novo.AppendChild(xmldecl);

                  //Criando novo Documento
                  XmlDocumentType doctype =
novo.CreateDocumentType("contrato", null, dtdpath, null);
                  novo.AppendChild(doctype);

                  //Inserindo o nó raiz CONTRATO no Documento
                  XmlElement raiz = this.criarElementoXml(this.novo,
"contrato");
                  this.novo.AppendChild(raiz);

                  //Inserindo ID do Contrato
                  XmlElement idContrato =
this.criarElementoXml(this.novo, "id");
                  this.insereValorElementoXml(idContrato, contrato.id);
                  this.adicionarElementoXml(raiz, idContrato);

                  //Inserindo o nó ESTAGIÁRIO
                  XmlElement estagiario =
this.criarElementoXml(this.novo, "estagiario");
                  this.adicionarElementoXml(raiz, estagiario);

                  //Inserindo os filhos de ESTAGIÁRIO
                  XmlElement nomeEstagiario =
this.criarElementoXml(this.novo, "nome");
```

```
                    this.insereValorElementoXml(nomeEstagiario,
contrato.estagiario.nome);
                    this.adicionarElementoXml(estagiario, nomeEstagiario);

                    XmlElement rgEstagiario =
this.criarElementoXml(this.novo, "rg");
                    this.insereValorElementoXml(rgEstagiario,
contrato.estagiario.rg);
                    this.adicionarElementoXml(estagiario, rgEstagiario);

                    XmlElement cpfEstagiario =
this.criarElementoXml(this.novo, "cpf");
                    this.insereValorElementoXml(cpfEstagiario,
contrato.estagiario.cpf);
                    this.adicionarElementoXml(estagiario, cpfEstagiario);

                    XmlElement matriculaEstagiario =
this.criarElementoXml(this.novo, "matricula");
                    this.insereValorElementoXml(matriculaEstagiario,
contrato.estagiario.matricula);
                    this.adicionarElementoXml(estagiario,
matriculaEstagiario);

                    XmlElement cursoEstagiario =
this.criarElementoXml(this.novo, "curso");
                    this.insereValorElementoXml(cursoEstagiario,
contrato.estagiario.curso);
                    this.adicionarElementoXml(estagiario, cursoEstagiario);

                    XmlElement enderecoEstagiario =
this.criarElementoXml(this.novo, "endereco");
                    this.insereValorElementoXml(enderecoEstagiario,
contrato.estagiario.endereco);
                    this.adicionarElementoXml(estagiario,
enderecoEstagiario);

                    XmlElement cidadeEstagiario =
this.criarElementoXml(this.novo, "cidade");
                    this.insereValorElementoXml(cidadeEstagiario,
contrato.estagiario.cidade);
                    this.adicionarElementoXml(estagiario,
cidadeEstagiario);

                    XmlElement ufEstagiario =
this.criarElementoXml(this.novo, "uf");
                    this.insereValorElementoXml(ufEstagiario,
contrato.estagiario.uf);
                    this.adicionarElementoXml(estagiario, ufEstagiario);

                    XmlElement cepEstagiario =
this.criarElementoXml(this.novo, "cep");
                    this.insereValorElementoXml(cepEstagiario,
contrato.estagiario.cep);
                    this.adicionarElementoXml(estagiario, cepEstagiario);

                    //Inserindo o nó CONVENIADA
                    XmlElement conveniada =
this.criarElementoXml(this.novo, "conveniada");
                    this.adicionarElementoXml(raiz, conveniada);
```

```
                    //Inserindo os filhos de CONVENIADA
                    XmlElement razaoSocial =
this.criarElementoXml(this.novo, "razaosocial");
                    this.insereValorElementoXml(razaoSocial,
contrato.conveniada.razaosocial);
                    this.adicionarElementoXml(conveniada, razaoSocial);

                    XmlElement cnpj = this.criarElementoXml(this.novo,
"cnpj");
                    this.insereValorElementoXml(cnpj,
contrato.conveniada.cnpj);
                    this.adicionarElementoXml(conveniada, cnpj);

                    XmlElement representanteConveniada =
this.criarElementoXml(this.novo, "representante");
                    this.adicionarElementoXml(conveniada,
representanteConveniada);

                    XmlElement nomeRepresentante =
this.criarElementoXml(this.novo, "nome");
                    this.insereValorElementoXml(nomeRepresentante,
contrato.conveniada.nome_representante);
                    this.adicionarElementoXml(representanteConveniada,
nomeRepresentante);

                    XmlElement cpfRepresentante =
this.criarElementoXml(this.novo, "cpf");
                    this.insereValorElementoXml(cpfRepresentante,
contrato.conveniada.cpf_representante);
                    this.adicionarElementoXml(representanteConveniada,
cpfRepresentante);

                    XmlElement enderecoConveniada =
this.criarElementoXml(this.novo, "endereco");
                    this.insereValorElementoXml(enderecoConveniada,
contrato.conveniada.endereco);
                    this.adicionarElementoXml(conveniada,
enderecoConveniada);

                    XmlElement cidadeConveniada =
this.criarElementoXml(this.novo, "cidade");
                    this.insereValorElementoXml(cidadeConveniada,
contrato.conveniada.cidade);
                    this.adicionarElementoXml(conveniada,
cidadeConveniada);

                    XmlElement ufConveniada =
this.criarElementoXml(this.novo, "uf");
                    this.insereValorElementoXml(ufConveniada,
contrato.conveniada.uf);
                    this.adicionarElementoXml(conveniada, ufConveniada);

                    XmlElement cepConveniada =
this.criarElementoXml(this.novo, "cep");
                    this.insereValorElementoXml(cepConveniada,
contrato.conveniada.cep);
                    this.adicionarElementoXml(conveniada, cepConveniada);

                    //Criando Clausulas e Incisos----------------------
----------------------------------------------------------|
```

```
                  XmlElement clausula;
                  XmlElement numeroClausula;
                  XmlElement textoClausula;
                  XmlElement inciso;
                  XmlElement letraInciso;
                  XmlElement textoInciso;


                  //inserindo os NÓS clausula
                  //aqui vai o FOR do Vetor de Cláusulas, o IF do vetor
de incisos e etc...
                  int i = 0;
                  int j = 0;
                  try{
                        for(i=0; i < contrato.clausulas.Length; i++)
                        {
                              if(contrato.clausulas[i].numero == "")
                                    break;

                              clausula = this.criarElementoXml(this.novo,
"clausula");
                              this.adicionarElementoXml(raiz, clausula);

                              numeroClausula =
this.criarElementoXml(this.novo, "numero");
                              this.insereValorElementoXml(numeroClausula,
contrato.clausulas[i].numero);
                              this.adicionarElementoXml(clausula,
numeroClausula);

                              textoClausula =
this.criarElementoXml(this.novo, "texto");
                              this.insereValorElementoXml(textoClausula,
contrato.clausulas[i].texto);
                              this.adicionarElementoXml(clausula,
textoClausula);

                              //IF inciso da Cláusula[i] estiver OK
Insere

                              for(j=0; j <
contrato.clausulas[i].inciso.Length; j++)
                              {
                                    try{

      if(contrato.clausulas[i].inciso[j].letra != ""){
                                                inciso =
this.criarElementoXml(this.novo, "inciso");

      this.adicionarElementoXml(clausula, inciso);

                                                letraInciso =
this.criarElementoXml(this.novo, "letra");

      this.insereValorElementoXml(letraInciso,
contrato.clausulas[i].inciso[j].letra);

      this.adicionarElementoXml(inciso, letraInciso);
```

```
                                                textoInciso =
this.criarElementoXml(this.novo, "texto");

      this.insereValorElementoXml(textoInciso,
contrato.clausulas[i].inciso[j].texto);

      this.adicionarElementoXml(inciso, textoInciso);
                                          }
                                    }catch(Exception e1) {}
                              }

                        }
                  }catch(Exception e) {}

                  //Criando Clausulas e Incisos-------------------------
-----------------------------------------------------------|

                  //Inserindo nós Local e Data
                  XmlElement local = this.criarElementoXml(this.novo,
"local");
                  this.insereValorElementoXml(local, contrato.local);
                  this.adicionarElementoXml(raiz, local);

                  XmlElement data = this.criarElementoXml(this.novo,
"data");
                  this.insereValorElementoXml(data,
contrato.data.ToString());
                  this.adicionarElementoXml(raiz, data);

                  //Salvando Documento XML
                  this.novo.Save(filename);

                  return true;
            }

            public bool inserirDocumentoDescritor(descritorContrato
descritor, string urlDescritor)
            {
                  novo = new XmlDocument();
                  novo.Load(urlDescritor);

                  XmlElement raiz = novo.DocumentElement;

                  //Inserindo informações no descritor de contratos
                  XmlElement documento = this.criarElementoXml(this.novo,
"documento");
                  this.adicionarElementoXml(raiz, documento);

                  //Inserindo informações no Descritor
                  XmlElement tipo = this.criarElementoXml(this.novo,
"tipo");
                  this.insereValorElementoXml(tipo, descritor.tipo);
                  this.adicionarElementoXml(documento, tipo);

                  XmlElement id = this.criarElementoXml(this.novo, "id");
                  this.insereValorElementoXml(id, descritor.id);
                  this.adicionarElementoXml(documento, id);

                  XmlElement parte = this.criarElementoXml(this.novo,
"parte");
```

```
                    this.insereValorElementoXml(parte, descritor.parte);
                    this.adicionarElementoXml(documento, parte);

                    XmlElement usuario = this.criarElementoXml(this.novo,
"usuario");
                    this.insereValorElementoXml(usuario,
descritor.usuario);
                    this.adicionarElementoXml(documento, usuario);

                    XmlElement data = this.criarElementoXml(this.novo,
"data");
                    this.insereValorElementoXml(data, descritor.data);
                    this.adicionarElementoXml(documento, data);

                    XmlElement url = this.criarElementoXml(this.novo,
"url");
                    this.insereValorElementoXml(url, descritor.url);
                    this.adicionarElementoXml(documento, url);

                    //Salvando Descritor
                    this.novo.Save(urlDescritor);

                    return true;
             }

             public Contrato vsisualizarContratoEstagio(string url)
             {
                    XmlDocument doc = new XmlDocument();
                    doc.Load(url);

                    XmlElement raiz = doc.DocumentElement;

                    XmlNodeList listaDeNos1 =
doc.DocumentElement.SelectNodes("/contrato");
                    XmlNodeList listaDeNos2 =
doc.DocumentElement.SelectNodes("/contrato/estagiario");
                    XmlNodeList listaDeNos3 =
doc.DocumentElement.SelectNodes("/contrato/conveniada");
                    XmlNodeList listaDeNos4 =
doc.DocumentElement.SelectNodes("/contrato/conveniada/representante");

                    int i = 0;
                    int j = 0;

                    //criando objeto contrato
                    Contrato contrato = new Contrato();

                    //Carregando ID
                    contrato.id =
listaDeNos1[i].SelectSingleNode("id").FirstChild.Value.ToString();
                    contrato.local =
listaDeNos1[i].SelectSingleNode("local").FirstChild.Value.ToString();
                    contrato.data =
listaDeNos1[i].SelectSingleNode("data").FirstChild.Value.ToString();

                    //carregando dados do Estagiário
                    contrato.estagiario = new Estagiario();

                    contrato.estagiario.nome =
listaDeNos2[i].SelectSingleNode("nome").FirstChild.Value.ToString();
```

```
                    contrato.estagiario.rg =
listaDeNos2[i].SelectSingleNode("rg").FirstChild.Value.ToString();
                    contrato.estagiario.cpf =
listaDeNos2[i].SelectSingleNode("cpf").FirstChild.Value.ToString();
                    contrato.estagiario.matricula =
listaDeNos2[i].SelectSingleNode("matricula").FirstChild.Value.ToString();
                    contrato.estagiario.curso =
listaDeNos2[i].SelectSingleNode("curso").FirstChild.Value.ToString();
                    contrato.estagiario.endereco =
listaDeNos2[i].SelectSingleNode("endereco").FirstChild.Value.ToString();
                    contrato.estagiario.cidade =
listaDeNos2[i].SelectSingleNode("cidade").FirstChild.Value.ToString();
                    contrato.estagiario.uf =
listaDeNos2[i].SelectSingleNode("uf").FirstChild.Value.ToString();
                    contrato.estagiario.cep =
listaDeNos2[i].SelectSingleNode("cep").FirstChild.Value.ToString();

                    //carregando dados da Conveniada
                    contrato.conveniada = new Conveniada();

                    contrato.conveniada.razaosocial =
listaDeNos3[i].SelectSingleNode("razaosocial").FirstChild.Value.ToString(
);
                    contrato.conveniada.cnpj =
listaDeNos3[i].SelectSingleNode("cnpj").FirstChild.Value.ToString();

                    contrato.conveniada.nome_representante =
listaDeNos4[i].SelectSingleNode("nome").FirstChild.Value.ToString();
                    contrato.conveniada.cpf_representante =
listaDeNos4[i].SelectSingleNode("cpf").FirstChild.Value.ToString();

                    contrato.conveniada.endereco =
listaDeNos3[i].SelectSingleNode("endereco").FirstChild.Value.ToString();
                    contrato.conveniada.cidade =
listaDeNos3[i].SelectSingleNode("cidade").FirstChild.Value.ToString();
                    contrato.conveniada.uf =
listaDeNos3[i].SelectSingleNode("uf").FirstChild.Value.ToString();
                    contrato.conveniada.cep =
listaDeNos3[i].SelectSingleNode("cep").FirstChild.Value.ToString();

                    XmlNodeList listaClausulas =
doc.DocumentElement.SelectNodes("/contrato/clausula");
                    XmlNodeList listaIncisos;


                    try{
                          for(i = 0; i < listaClausulas.Count; i++)
                          {
                                contrato.clausulas[i] = new Clausula();
                                contrato.clausulas[i].numero =
listaClausulas[i].SelectSingleNode("numero").FirstChild.Value.ToString();
                                contrato.clausulas[i].texto =
listaClausulas[i].SelectSingleNode("texto").FirstChild.Value.ToString();

                                listaIncisos =
listaClausulas[i].SelectNodes("inciso");

                                for(j = 0; j < listaIncisos.Count; j++){
                                      contrato.clausulas[i].inciso[j] = new
Inciso();
```

```
                                    contrato.clausulas[i].inciso[j].letra
= listaIncisos[j].SelectSingleNode("letra").FirstChild.Value.ToString();
                                    contrato.clausulas[i].inciso[j].texto
= listaIncisos[j].SelectSingleNode("texto").FirstChild.Value.ToString();
                              }

                        }
                  }catch(Exception e){ }

                  return contrato;
            }

            public string buscarContratoEstagio(string consulta, string
tipoBusca, string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlElement raiz = doc.DocumentElement;

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/descritor/documento");

                  descritorContrato descritor = new descritorContrato();

                  string turl = "";

                  if(tipoBusca == "3"){
                        for(int i = 0; i <listaDeNos.Count ; i++){

                              string id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();


                              if(id == consulta){
                                    turl = "if";
                                    /*
                                    descritor.tipo =
listaDeNos[i].SelectSingleNode("tipo").FirstChild.Value.ToString();
                                    descritor.id =
listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToString();
                                    descritor.usuario =
listaDeNos[i].SelectSingleNode("usuario").FirstChild.Value.ToString();
                                    descritor.data =
listaDeNos[i].SelectSingleNode("data").FirstChild.Value.ToString();
                                    */
                                    /*turl =
listaDeNos[i].SelectSingleNode("tipo").FirstChild.Value.ToString();
                                    turl =
turl+"<br>"+listaDeNos[i].SelectSingleNode("id").FirstChild.Value.ToStrin
g();
                                    turl =
turl+"<br>"+listaDeNos[i].SelectSingleNode("parte").FirstChild.Value.ToSt
ring();
                                    turl =
turl+"<br>"+listaDeNos[i].SelectSingleNode("usuario").FirstChild.Value.To
String();
```

```
                                    turl =
turl+"<br>"+listaDeNos[i].SelectSingleNode("data").FirstChild.Value.ToStr
ing();*/
                                    turl =
listaDeNos[i].SelectSingleNode("url").FirstChild.Value.ToString();
                              }
                        }
                  }
                  return turl;
            }

            public string buscarContratoEstagioAvancado(string consulta,
string tipoBusca, string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlElement raiz = doc.DocumentElement;

                  string retorno = null;
                  string xpath = null;

                  switch(tipoBusca)
                  {
                        case "1": {
                              xpath =
"/descritor/documento[contains(id,'" + consulta + "')]";
                        } break;
                        case "2":
                        {
                              xpath =
"/descritor/documento[contains(parte,'" + consulta + "')]";
                        } break;
                        case "3":
                        {
                              xpath =
"/descritor/documento[contains(tipo,'" + consulta + "')]";
                        } break;
                        case "4":
                        {
                              xpath =
"/descritor/documento[contains(data,'" + consulta + "')]";
                        } break;
                        case "5": {
                              xpath =
"/descritor/documento[contains(usuario,'" + consulta + "')]";
                        } break;

                  }

                  XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes(xpath);

                  try
                  {
                        for(int i = 0; i < listaDeNos.Count; i++)
                        {
                              retorno = retorno+" "+
                                    "<tr>"+
```

```
                                    "<td height='20'
align='left'>"+listaDeNos[i].SelectSingleNode("parte").FirstChild.Value+"
</td>"+
                                    "<td height='20'
align='left'>"+listaDeNos[i].SelectSingleNode("data").FirstChild.Value+"<
/td>"+
                                    "<td height='20' align='left'><a
href='visualizarContratoEstagio.aspx?cod="+listaDeNos[i].SelectSingleNode
("id").FirstChild.Value+"'>Visualizar</a></td>"+
                                    "</tr>";
                        }
                  }
                  catch(Exception e){}

                  return retorno;
            }
            public DataSet listarContratoEstagio(string url)
            {
                  DataSet documentos = new DataSet();
                  documentos.ReadXml(url);
                  return documentos;
            }

            public bool editarContratoEstagioClausula(string url, int
indice, string clausula)
            {
                  try
                  {
                        XmlDocument doc = new XmlDocument();
                        doc.Load(url);

                        XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/contrato/clausula");


      listaDeNos[indice].SelectSingleNode("texto").FirstChild.Value =
clausula;

                        doc.Save(url);

                        return true;

                  }
                  catch(Exception e)      { return false; }

                  return false;
            }

            public bool editarContratoEstagioInciso(string url, int
indiceClausula, int indiceInciso, string inciso)
            {
                  try
                  {
                        XmlDocument doc = new XmlDocument();
                        doc.Load(url);

                        XmlNodeList listaClausulas =
doc.DocumentElement.SelectNodes("/contrato/clausula");
```

```
                        XmlNodeList listaIncisos =
listaClausulas[indiceClausula].SelectNodes("inciso");


      listaIncisos[indiceInciso].SelectSingleNode("texto").FirstChild.Val
ue = inciso;

                        doc.Save(url);

                        return true;

                  }
                  catch(Exception e)      { return false; }

                  return false;
            }

            public bool removerContratoEstagioClausula(string url, int
indiceClausula)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList remClausula =
doc.DocumentElement.SelectNodes("/contrato/clausula");
                  XmlNode noParente =
remClausula[indiceClausula].ParentNode;

                  if(remClausula != null )
                  {

      noParente.RemoveChild(remClausula[indiceClausula]);
                        doc.Save(url);
                        return true;
                  }
                  else
                        return false;
            }

            public bool removerContratoEstagioInciso(string url, int
indiceClausula, int indiceInciso)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNodeList listaClausulas =
doc.DocumentElement.SelectNodes("/contrato/clausula");
                  XmlNodeList listaIncisos =
listaClausulas[indiceClausula].SelectNodes("inciso");
                  XmlNode noParente =
listaIncisos[indiceInciso].ParentNode;

                  if(listaIncisos != null )
                  {

      noParente.RemoveChild(listaIncisos[indiceInciso]);
                        doc.Save(url);
                        return true;
                  }
                  else
```

```
                        return false;
            }

            public bool editarContratoEstagio(Contrato contrato, string
url, string tipo)
            {

                  switch(tipo)
                  {
                        case "1":
                        {
                              XmlDocument doc = new XmlDocument();
                              doc.Load(url);

                              XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/contrato/estagiario");


      listaDeNos[0].SelectSingleNode("nome").FirstChild.Value =
contrato.estagiario.nome;

      listaDeNos[0].SelectSingleNode("rg").FirstChild.Value =
contrato.estagiario.rg;

      listaDeNos[0].SelectSingleNode("cpf").FirstChild.Value =
contrato.estagiario.cpf;

      listaDeNos[0].SelectSingleNode("matricula").FirstChild.Value =
contrato.estagiario.matricula;

      listaDeNos[0].SelectSingleNode("curso").FirstChild.Value =
contrato.estagiario.curso;

      listaDeNos[0].SelectSingleNode("endereco").FirstChild.Value =
contrato.estagiario.endereco;

      listaDeNos[0].SelectSingleNode("cidade").FirstChild.Value =
contrato.estagiario.cidade;

      listaDeNos[0].SelectSingleNode("uf").FirstChild.Value =
contrato.estagiario.uf;

      listaDeNos[0].SelectSingleNode("cep").FirstChild.Value =
contrato.estagiario.cep;

                              doc.Save(url);

                              XmlDocument descritor = new XmlDocument();

      descritor.Load("C:\\EMILIO\\TCC\\SisJurXML\\dados\\controle\\descri
tor.xml");

                              XmlNodeList nosDesc =
descritor.DocumentElement.SelectNodes("/descritor/documento");

                              for(int i=0; i < nosDesc.Count; i++)
                              {
                                    string id =
nosDesc[i].SelectSingleNode("id").FirstChild.Value.ToString();
```

```
                                    if(id == contrato.id)
                                    {

      nosDesc[i].SelectSingleNode("parte").FirstChild.Value =
contrato.estagiario.nome;

      nosDesc[i].SelectSingleNode("data").FirstChild.Value =
DateTime.Now.ToString();
                                    }
                              }

      descritor.Save("C:\\EMILIO\\TCC\\SisJurXML\\dados\\controle\\descri
tor.xml");

                              return true;
                        }break;

                        case "2":
                        {
                              XmlDocument doc = new XmlDocument();
                              doc.Load(url);

                              XmlNodeList listaDeNos =
doc.DocumentElement.SelectNodes("/contrato/conveniada");


      listaDeNos[0].SelectSingleNode("razaosocial").FirstChild.Value =
contrato.conveniada.razaosocial;

      listaDeNos[0].SelectSingleNode("cnpj").FirstChild.Value =
contrato.conveniada.cnpj;

      listaDeNos[0].SelectSingleNode("endereco").FirstChild.Value =
contrato.conveniada.endereco;

      listaDeNos[0].SelectSingleNode("cidade").FirstChild.Value =
contrato.conveniada.cidade;

      listaDeNos[0].SelectSingleNode("uf").FirstChild.Value =
contrato.conveniada.uf;

      listaDeNos[0].SelectSingleNode("cep").FirstChild.Value =
contrato.conveniada.cep;

                              listaDeNos =
doc.DocumentElement.SelectNodes("/contrato/conveniada/representante");


      listaDeNos[0].SelectSingleNode("nome").FirstChild.Value =
contrato.conveniada.nome_representante;

      listaDeNos[0].SelectSingleNode("cpf").FirstChild.Value =
contrato.conveniada.cpf_representante;

                              doc.Save(url);

                              return true;
                        }break;
```

```
                        default:
                        {

                        }break;
                  }

                  return false;
            }

            public bool removerContratoEstagio(string id, string url)
            {
                  XmlDocument doc = new XmlDocument();
                  doc.Load(url);

                  XmlNode remContrato =
doc.DocumentElement.SelectSingleNode("/descritor/documento[id='" + id +
"']");
                  XmlNode noParente = remContrato.ParentNode;

                  if(remContrato != null ){
                        noParente.RemoveChild(remContrato);
                        //remContrato.RemoveAll();
                        doc.Save(url);
                        return true;
                  }else
                        return false;
            }

      }

      public class Clausula
      {
            public Clausula(){}

            public string numero;
            public string texto;
            public Inciso[] inciso = new Inciso[100];
      }

      public class Contrato
      {
            public Estagiario estagiario;
            public Conveniada conveniada;
            public string id;
            public string local;
            public string data;
            public Clausula[] clausulas = new Clausula[100];
      }

      public class Estagiario
      {
            public string nome;
            public string rg;
            public string cpf;
            public string matricula;
            public string curso;
            public string endereco;
            public string cidade;
            public string uf;
            public string cep;
```

```
      }

      public class Conveniada
      {
            public string razaosocial;
            public string cnpj;
            public string nome_representante;
            public string cpf_representante;
            public string endereco;
            public string cidade;
            public string uf;
            public string cep;
      }

      public class Inciso
      {
            public string letra;
            public string texto;
      }

      public class descritorDocumentos
      {
            public string tipo;
            public string id;
            public string parte;
            public string usuario;
            public string data;
            public string url;
      }
}
```

Anexo IV – Código Fonte da Classe UtilXML

```
public string ValorDoElemento(XmlElement elemento, string xpath)
            {
            XmlElement eretorno = (XmlElement)
elemento.SelectSingleNode(xpath);
            string retorno = eretorno.FirstChild.Value.ToString();
            return retorno;
            }

            public XmlElement criarElementoXml(XmlDocument pai, string
nomeElemento)
            {
                  return pai.CreateElement(nomeElemento);
            }

            public void insereValorElementoXml(XmlElement nomeElemento,
string valor)
            {
                  nomeElemento.InnerText = valor;
            }

            public void adicionarElementoXml(XmlElement pai, XmlElement
filho)
            {
```

```
                pai.AppendChild(filho);
        }
    public Guid geraID()
        {
                Guid docID = Guid.NewGuid();
                return docID;
        }
```