INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

STEVAN DANIELLI COSTA

WXDC: UMA FERRAMENTA PARA IMPOSIÇÃO E VALIDAÇÃO DE RESTRIÇÕES DE INTEGRIDADE BASEADAS NA LINGUAGEM XDCL

Prof. Msc. Anubis Graciela de Moraes Rossetto

PASSO FUNDO, 2012

STEVAN DANIELLI COSTA

WXDC: UMA FERRAMENTA PARA IMPOSIÇÃO E VALIDAÇÃO DE RESTRIÇÕES DE INTEGRIDADE BASEADAS NA LINGUAGEM XDCL

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador (a): Prof. Msc. Anubis Graciela de Moraes Rossetto

PASSO FUNDO, 2012

STEVAN DANIELLI COSTA

WXDC: UMA FERRAMENTA PARA IMPOSIÇÃO E VALIDAÇÃO DE RESTRIÇÕES DE INTEGRIDADE BASEADAS NA LINGUAGEM XDCL

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Msc. Anubis Graciela de Moraes Rossetto

_____

Prof. Msc. Alexandre Tagliari Lazzaretti

_____

Prof. Msc. Evandro Miguel Kuskera

_____

Prof. Esp. Jorge Luis Boeira Bavaresco

_____

Prof. Msc. Evandro Miguel Kuskera

Coordenador do Curso

PASSO FUNDO, 2012

Aos meus pais, ao meu irmão
e à minha namorada
por sempre acreditarem e apoiarem
a minha caminhada.

AGRADECIMENTOS

Primeiramente agradeço à Deus, pela sabedoria e ajuda, não somente neste, mas em todos os momentos da minha vida em que tive dificuldades a serem superadas. Também agradeço aos meus pais, Alcemar Costa e Salete Danielli Costa, e ao meu irmão Murillo Danielli Costa, pelo apoio em todos os momentos, por estarem sempre do meu lado, mesmo nos piores momentos, acreditando no meu esforço e sempre dando a maior motivação e força para prosseguir. Agradeço também a minha namorada Monique Regina Wassem pela compreensão nos momentos de dificuldade, e nos momentos de alegria também, obrigado por ser minha melhor amiga, a pessoa a quem eu confio todos os meus segredos. Pai, Mãe, Mano, Amor, obrigado por tudo e eu amo vocês muito. Devo também agradecer à todos os professores do Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense, Campus Passo Fundo, especialmente ao meus orientadores, Alexandre Tagliari Lazzaretti e Anubis Graciela de Moraes Rossetto, muito obrigado pela confiança, e pelo apoio que me deram em todos os momentos deste trabalho e também ao longo de todo o curso. E por último, mas não menos especial, aos meus amigos e companheiros de caminhada, Alex Finatto Donassolo, Josias Graboski, Maurício Corvello e Héber Martins, o meu sincero muito obrigado, e que este final de curso não seja o final dessa nossa amizade.

“Não devemos ter medo dos confrontos...
até os planetas se chocam e do caos,
nascem as estrelas.”
Charles Chaplin

RESUMO

Este trabalho apresenta uma ferramenta web para a validação e imposição de restrições de integridade de domínio em documentos XML com base na linguagem XDCL. A ferramenta utiliza o *parser* XDCL para fazer a validação dos documentos. O *parser* foi implementado utilizando a tecnologia Java e as páginas da web foram desenvolvidas utilizando a tecnologia JSP juntamente com outras tecnologias relacionadas à web. A ferramenta apresentada tem como objetivo agilizar o processo de validação do documento XML quando é necessário impor restrições a um documento, sem a necessidade de conhecimento prévio da língua XDCL para a criação de documentos XDC. Como resultado, é possível fazer a geração do documento XDC e a validação de documentos XML com base no documento XDC.

Palavras-chave: XML; XDCL; XDC; Restrições de Integridade; Validação.

ABSTRACT

This work presents a web tool for validating and imposition of domain integrity constraints in the XML document based on language XDCL. This tool uses the XDCL parser to validate the document. The parser has been implemented using Java technology and web pages were developed using JSP technology together with other web-related technologies. The tool presented aims to streamline the process of validating the XML document when it is necessary to impose restrictions on a document without the need for prior knowledge of the language XDCL for creating documents XDC. As a result, it can generate the document XDC and validation of XML documents based on document XDC.

Key words: XML; XDCL; XDC; Integrity Constraints; Validation.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX - *Asynchronous Javascript and XML* - p.39

API - *Application Programming Interface* - p.34

ASP - *Active Server Page* - p.43

BNF - *Backus Naur Form* - p.21

CSS - *Cascading Style Sheets* - p.39

DHTML - *Dynamic HTML* - p.39

DOM - *Document Object Model* - p.34

DTD - *Document Type Definition* - p.11

HTML - *Hyper Text Markup Language* - p.41

HTTP - *Hypertext Transfer Protocol* - p.43

JSON - *JavaScript Object Notation* - p.43

JSP - *Java Server Page* - p.43

PHP - *Personal Home Page / Hypertext Preprocessor* - p.43

SAX - *Simple API for XML* - p.34

W3C - *World Wide Web Consortium* - p.13

WXDC - *Web XML Definition Constraints* - p.47

XDC - *XML Domain Constraints* - p.11

XDCL - *XML Domain Constraints Language* - p.11

XML - *Extensible Markup Language* - p.11

XPATH - *XML Path Language* - p.30

XSD - *XML Schema Definition* - p.11

SUMÁRIO

1 INTRODUÇÃO

A tecnologia XML vem a cada dia sendo mais utilizada nas trocas de dados via internet, e em muitos casos pode haver necessidade de armazenar os dados de documentos XML em algum local. Entretanto, no momento de se adicionar esses dados em um banco de dados, podem ocorrer muitos erros devido à inconsistência nos dados presentes no documento, pois o documento XML é escrito livremente pelo usuário, contando apenas com algumas restrições imposta por DTDs (*Data Type Definition*) e XSD (*XML Schema Definition*) e deixando de lado muitas outras restrições de grande importância.

As restrições de domínio são restrições que são utilizadas para garantir que os dados inseridos em um determinada coluna, pertençam a um determinado domínio, ou seja, que os valores ali contidos sejam válidos (SILVA; MARINALDO, 2000).

Visando a melhoria da segurança e integridade dos dados em documento XML, o trabalho proposto por Lazzaretti (2005) define um controle para tratar das restrições de domínio que não são suportadas pelas DTDs e XSDs. Esse controle é chamado XDC (*XML Domain Constraints*) e é responsável por analisar documentos XML com base em um documento XDC, que foi gerado a partir de uma linguagem chamada XDCL (*XML Domain Constraint Language*). É neste documento que estarão as restrições de integridade que o usuário deseja confrontar com o documento XML.

Com o desenvolvimento desse novo controle, tem-se a necessidade de implementar uma ferramenta gráfica para fazer a geração desses documentos XDCs de forma visual, abstraindo do usuário o conhecimento prévio da linguagem XDCL. Além do objetivo de abstração, tem-se o propósito de agilizar o processo de validação dos documentos, assim como trazer uma ferramenta para auxiliar na correção de muitos dos erros encontrados em documentos XML.

1.1 MOTIVAÇÃO

A motivação deste trabalho se dá pelo fato de que as restrições de integridade são essenciais para que os dados contidos nos documentos XML se tornem mais seguros e confiáveis e que possam ser armazenados em algum local de forma que possam ser usados no futuro. Lazzaretti (2005) propôs este controle em sua dissertação de mestrado, porém para fazer a validação o usuário deve ter um conhecimento prévio, tanto da linguagem XDCL como da estruturação de documento XML.

Com base nisso, este trabalho vem realizar o desenvolvimento de uma *interface* gráfica para suprir a necessidade deste controle, e também facilitar e agilizar a validação dos documentos XML, garantindo maior consistência nos dados.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver uma ferramenta web, baseando-se na linguagem XDCL, para fazer a imposição de restrições de integridade de domínio em documentos XML, com o propósito de agilizar e facilitar a validação e também assegurar que os dados que estão contidos neste documento estejam corretos e íntegros.

1.2.2 Objetivos específicos

- Criar documentos XML válidos, consistidos em relação a restrições de integridade de domínio, conforme a linguagem XDCL;
- Desenvolver uma ferramenta web para a imposição de restrições de integridade de domínio;
- Disponibilizar a ferramenta para uso da comunidade científica e acadêmica.

Este trabalho está dividido em capítulos, onde o capítulo 2 se refere aos conteúdos necessários para o entendimento de documentos XML e restrições de integridade. O capítulo 3 é destinado a explicação da ferramenta que foi desenvolvida neste trabalho. No capítulo 4 são apresentados estudos de casos que demonstram o funcionamento da ferramenta e os documentos gerados por ela. Por fim, tem-se as considerações finais do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo são abordados temas importantes para definição da ferramenta. São abordados XML, banco de dados e as restrições de integridade e a linguagem XDC proposta por Lazzaretti.

2.1 XML

Conforme Box, “A linguagem *Extensible Markup Language* (XML) é o resultado do trabalho de uma equipe de especialistas estabelecido em 1996 pelo W3C”.(BOX *apud* Bryan, 1998).

O documento XML é escrito em arquivos textos normais, com o propósito de estruturar, armazenar e transmitir dados pela internet ou também para que esses dados possam ser armazenados em algum lugar específico, como por exemplo, um banco de dados. XML é uma linguagem de marcação, ou seja, todos os dados contidos neste documento estarão delimitados por um elemento inicial e um elemento final, de maneira que essas marcas qualificam e delimitam o texto contido nesse intervalo (LAZZARETTI, 2005).

Os elementos definidos em um documento XML seguem uma estrutura hierárquica, onde um elemento vem após o outro e cada elemento pode ser classificado como *elemento simples* ou *elemento composto.* O elemento será do tipo simples quando dentro das suas delimitações não houver nenhum outro elemento e também se este não for composto por atributos. Quando um elemento possuir um atributo ou dentro das suas delimitações existirem um ou mais elementos, este pode ser chamado de elemento composto.

Na Figura 1, mostra-se um exemplo de documento XML. Nesse documento podem-se encontrar exemplos dos dois tipos de elementos citados anteriormente.

Figura 1 – Exemplo documento XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <dados>
3     <cidades>
4         <cidade id="1">
5             <nome>Passo Fundo</nome>
6             <estado>RS</estado>
7         </cidade>
8     </cidades>
9 </dados>
```

Fonte: Do Autor

Na linha número 1 do exemplo é mostrado o cabeçalho do documento XML, nele é determinada a versão da linguagem XML que será usada, atualmente a versão usada é a versão 1.0, e também é informado o tipo de codificação para os dados que estão contidos no documento.

Na segunda linha, é aberto o elemento raiz através do sinal de marcação <*dados*>, este elemento é fechado na linha de número 9. Todo e qualquer documento XML irá conter um elemento raiz, sendo esse sempre um elemento composto, pois dentro sempre haverá outros elementos afim de melhor qualificar os dados do documento.

Dentro do elemento raiz, existe o elemento *cidades*, sendo este um elemento composto, pois contém dentro dele mais elementos. No exemplo, existe na linha 4 a definição do elemento *cidade*, este elemento será composto não somente porque dentro dele há outros elementos, mas também por nele existir um atributo chamado *id*. Esse atributo é único, portanto, nesse documento somente o elemento *cidade* terá um atributo chamado *id*.

Como citado anteriormente, os elementos podem ser simples ou compostos, até agora foram vistos somente elementos compostos; agora serão apresentados dois exemplos de elementos simples. Na linha 5 e 6 existem elementos do tipo simples, pois dentro deles somente aparecem textos e também esses elementos não têm atributos, o que os torna elementos simples.

Quando se fala em validações em documentos XML, têm-se dois tipos de validações: a validação de documento bem formado e a de documento válido. Quando um documento for submetido à validação para saber se ele é bem formado, serão verificadas algumas exigências como:

- O documento deve conter apenas um elemento raiz;
- Todos os elementos devem ter um elemento inicial e um elemento final;
- A delimitação dos elementos é *case sensitive*;
- Os elementos devem estar aninhados;
- Os valores de todos os atributos devem estar entre aspas duplas ou simples;
- Os nomes dos elementos não podem começar com números ou caracteres de pontuação, não podem começar com “XML” e também não podem conter espaços (XML..., 2012).

Para verificar se o documento é válido existem os esquemas XML, “A W3C recomenda dois tipos de esquemas, visando organizar ou definir elementos e atributos em documentos XML. São eles a DTD e o XSD” (LAZZARETTI, 2005, p. 28).

Caso todos os itens sejam cumpridos e o documento XML esteja de acordo com as regras que a DTD ou o XSD impuseram, o documento estará, em relação à estrutura, bem formado e será um documento válido.

2.1.1 DTD (Document Type Definition)

Devido ao documento XML ser gerado livremente, existem alguns mecanismos que fazem com que os elementos estejam em uma determinada estrutura. Essa estrutura vem a seguir um padrão chamado *Document Type Definition*, Nardon define muito bem o que é DTD, afirmando que:

> Uma DTD descreve a estrutura do documento e, uma vez associada a um documento XML, é utilizada para validá-lo, garantindo que as *tags* estão sendo utilizadas de forma correta. A criação de uma DTD é feita através da descrição dos elementos e atributos que podem aparecer no documento XML, informando também se os elementos podem se repetir, que valores possíveis um atributo pode assumir e que elementos e atributos são opcionais. (2000, p. 3)

A partir dessa colocação, percebe-se que o padrão XML, apesar de ser um padrão aberto, consegue impor algumas restrições quanto a sua estrutura hierárquica de representação dos dados, definindo uma sequência de elementos e atributos que devem constar nos documentos, mas para que essa estrutura seja validada, o documento XML deve conter uma referência à DTD.

Na Figura 2 é apresentada uma possível DTD para o exemplo dado acima.

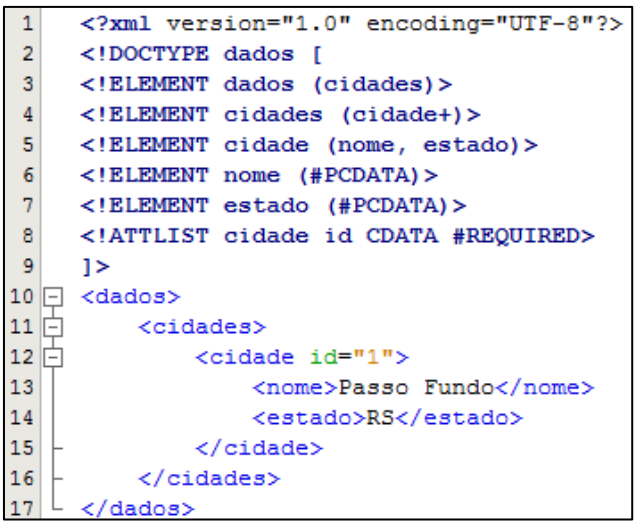
Figura 2 - Exemplo DTD

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!ELEMENT dados (cidades)>
3 <!ELEMENT cidades (cidade+)>
4 <!ELEMENT cidade (nome, estado)>
5 <!ELEMENT nome (#PCDATA)>
6 <!ELEMENT estado (#PCDATA)>
7 <!ATTLIST cidade id CDATA #REQUIRED>
```

Fonte: Do Autor

Para essa DTD ser utilizada em um documento XML ela deve ser referenciada no documento através da marcação <!DOCTYPE dados SYSTEM “DTD.dtd”>, sendo assim uma DTD Externa; porém a DTD pode ser descrita no próprio documento XML, chamada de DTD Interna.Um exemplo desse uso pode ser encontrado na Figura 3.

Figura 3 - Exemplo DTD Interna



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE dados [
3  <!ELEMENT dados (cidades)>
4  <!ELEMENT cidades (cidade+)>
5  <!ELEMENT cidade (nome, estado)>
6  <!ELEMENT nome (#PCDATA)>
7  <!ELEMENT estado (#PCDATA)>
8  <!ATTLIST cidade id CDATA #REQUIRED>
9  ]>
10 <dados>
11     <cidades>
12         <cidade id="1">
13             <nome>Passo Fundo</nome>
14             <estado>RS</estado>
15         </cidade>
16     </cidades>
17 </dados>
```

Fonte: Do Autor

Conforme Bradley (2004), a DTD pode ser composta por várias declarações de elementos e cada uma pode ser de diferentes tipos. Existem quatro tipos principais que são: ELEMENT (definição de elementos), ATTLIST (definição de atributos de elementos), ENTITY (declaração de entidades) e NOTATION (definição de tipos de dados).

Ainda existem mais dois controles que podem ser feitos nas DTD, que são: o controle de sequência e o controle de cardinalidade.

O Controle de sequência indica quais as sequências dos elementos e dos conteúdos e podem também definir a opção dos elementos e conteúdos, os “conectores” são os seguintes:

- Vírgula (,): especifica a ordem dos elementos, na Figura 3, na linha 5, o elemento estado deve aparecer logo após o elemento nome.
- Barra Vertical (|): especifica a condição OU, definindo que apenas um dos elementos entre a barra pode aparecer.

O controle de cardinalidade define qual a frequência que um elemento pode aparecer dentro do documento XML. Os “conectores” de cardinalidade podem ser:

- Sinal de Mais (+): O elemento que for seguido deste sinal, pode aparecer uma ou mais vezes dentro do documento, constituindo assim uma cardinalidade [1..N] (um para muitos). Tem-se um exemplo dessa cardinalidade na linha 4, da Figura 3, na declaração do elemento cidade.

Asterisco (*): O elemento que for seguido deste sinal, terá cardinalidade [0..N] (zero para muitos).

2.1.2 XSD (XML Schema Definition)

De acordo com (ANDERSON *et al*., 2000), (W3C) e (BRADLEY, 2004), devido à ausência de alguns controles nas DTDs, como não ser extensível, oferecer poucos tipos de dados, a W3C criou uma especificação chamada XSD ou *XML Schema Definition.* Os documentos XSD são constituídos de metadados, ou seja, dados que expressão algum controle de estrutura, de conteúdo e também a semântica de um documento XML.

O documento XSD serve para que se possa ter uma abrangência maior no que se diz respeito ao controle de restrições, pois nele é possível se definir novos tipos de dados, fazendo assim com que o documento associado a este esquema seja mais consistente ainda.

Todo o documento XSD começa pela marcação *schema*, nesse elemento pode ser definido um atributo chamado *version*, é através desse atributo que o autor do documento pode saber se está utilizando a última versão do XSD. (LIMA, 2008)

Este documento XSD utilizado como exemplo está usando *namespaces*, o que não é obrigatório, mas lembrando de que se a opção for utilizar *namespaces* ele deve ser utilizado em todas as definições do documento.

As duas principais definições utilizadas em documentos XSD são as definições dos elementos (*element*) e as definições dos atributos (*atribute*), tanto os elementos como os atributos sempre terão tipos e nomes. Quando definido um elemento este pode ser de dois tipos: tipo simple (*simpleType*) ou tipo complexo (*complexType*). (LAZZARETTI, 2005)

Quando se fala de tipo simples (*simpleType*), pode-se escolher em definir o elemento dentre diversos tipos, sendo que os principais já estão definidos pela norma, são eles:*string, integer, boolean, decimal, date.* Estes tipos de dados são exemplos de dados que já estão definidos pela W3C e estão prontos para serem usados. Na Figura 4 os elementos nome (linha 12) e estado (linha 13) são denominados elementos simples (*simpleTypeelement*).

Os elementos complexos (*complexType*) são elementos em que dentro deles estão contidos outros elementos, e também podem conter atributos para melhor defini-los. Na

Figura 4 encontra-se um elemento composto na linha 9, o elemento cidade, onde dentro dele contém subelementos (nome e estado) e também o mesmo possui um atributo chamada *id*.

Nos elementos do tipo complexo é possível fazer algumas restrições como tipo de dados que será aceito em determinada marcação, delimitar sequência pela marcação <sequence>, a condição OU pode ser imposta através da marcação <choice>.

Conforme Lazzaretti (2005), nos documentos XSD também é possível controlar a quantidade de ocorrências dos elementos, através dos atributos *minOccurs*e *maxOccurs*, sendo através destes atributos que se pode definir quantas vezes o elementos pode ser repetido ou não. Além desses elementos e atributos, existem outros que podem ser usados para que o documento XSD se torne cada vez mais rígido na sua validação.

Normalmente os documentos XSD são divididos em algumas partes, para que algumas restrições possam ser reutilizadas por outros documentos. Uma maneira de se dividir os documentos é criando um documento XSD geral, um documento para a definição dos tipos simples (*simpleType*) e outro para a definição dos tipos complexos (*complexType*).

Quando se necessitar usar um tipo definido em um documento externo, apenas se faz a referência ao documento em questão, através da marcação: <xs:includeschemaLocation="local onde esta o arquivo"/>

Figura 4 - Exemplo documento XSD

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="dados">
4          <xs:complexType>
5              <xs:sequence>
6                  <xs:element name="cidades" minOccurs="1" maxOccurs="1">
7                      <xs:complexType>
8                          <xs:sequence>
9                              <xs:element name="cidade">
10                                 <xs:complexType>
11                                     <xs:sequence>
12                                         <xs:element name="nome" type="xs:string"/>
13                                         <xs:element name="estado" type="xs:string"/>
14                                     </xs:sequence>
15                                     <xs:attribute name="id" type="xs:integer"/>
16                                 </xs:complexType>
17                             </xs:element>
18                         </xs:sequence>
19                     </xs:complexType>
20                 </xs:element>
21             </xs:sequence>
22         </xs:complexType>
23     </xs:element>
24 </xs:schema>
```

Fonte: Do Autor

2.2 BANCO DE DADOS

"Uma restrição de integridade é uma expressão *booleana* que está associada a algum banco de dados e precisar ser avaliada o tempo todo como *true*" (DATE, 2003).

Quando se fala em integridade, está se referindo à precisão ou à correção dos dados contidos em um banco de dados. Sendo assim, as regras que são definidas servem para garantir que quando o banco de dados sofra alguma alteração ele não saia de seu estado consistente, ou seja, os dados que ali estão devem estar protegidos contras eventuais danos. (DATE, 2003, SILBERCHATZ *et al*, 1999) Caso o banco de dados não garanta a consistência e a segurança dos dados, poderá em ações futuras ocorrer erros pelo fato de que os dados estão armazenados de forma inadequada na base de dados.

Para tratar dessa segurança e consistência existem algumas restrições que o banco de dados implementa. Segundo Heuser (1998), estas restrições de integridade são divididas em 4 categorias:

- Integridade de domínio: Referente aos valores contidos nos campos.
- Integridade de vazio: Especifica se o campo pode ou não ser vazio.
- Integridade de chave: Especifica que o valor da chave primária deve ser único.
- Integridade referencial: Especifica que o valor presente em uma chave estrangeira deve obrigatoriamente estar em uma chave primária.

A partir de agora será abordado um pouco mais sobre as restrições de domínio, às quais a ferramenta aqui proposta irá fazer as validações.

2.2.1 Restrições de Domínio

Conforme Lazzaretti (2005), as restrições de domínio são as restrições mais comuns encontradas em um banco de dados, sendo que estas são verificadas todas as vezes que algum elemento é inserido no banco de dados, sendo que as restrições de domínio se dividem em mais 5 categorias, são elas: restrições de atributo, restrições de tipo, restrições de tupla, restrições de banco de dados, restrições de transição de estado.

- Restrições de Atributo: nessa categoria são avaliadas as enumerações de valores. Como por exemplo, se tivermos um atributo chamado sexo, este somente atributo poderá receber os valores M (Masculino) ou F (Feminino).
- Restrições de Tipo: são as definições dos tipos dos elementos, para que cada atributo tenha um tipo específico. Por exemplo, o elemento sexo citado anteriormente será do tipo *string*, portanto aceitará *caracteres* alfanuméricos, mas

caso seja definido um elemento valor esse será do tipo *numeric*, portando só aceitará valores numéricos.

- Restrições de Tupla: é a restrição imposta sobre uma tupla individual, sendo que podem ser impostas várias restrições nos atributos contidos nesta tupla. Um exemplo seria que o campo voto_obrigatorio só teria o valor 'sim' caso a idade da pessoa seja superior a 18 anos.
- Restrições de Banco de Dados: são restrições que são compostas por duas ou mais tuplas dentro de uma tabela. Para que essas restrições sejam efetivadas as duas devem atender aos requisitos especificados. Um exemplo dessas restrições seria a soma do atributo chamado valor_unitarioda tabela de produtos deve ser igual ao atributo valor_total da tabela de vendas.
- Restrições de Transição de Estado: estas restrições estão ligadas as transições de estado de atributos do banco de dados, fazendo com que estas transições ocorram de maneira correta, impossibilitando que ocorram mudanças incoerentes. Por exemplo, um atributo chamado idade com o valor '18' só poderá ser alterado para idade superior a 18 e nunca para menor que esta.

A partir desse estudo foi possível constatar que o banco de dados é um gerenciador muito eficiente de restrições de integridade, podendo nele serem implementadas várias restrições que visam aperfeiçoar a segurança e integridade dos dados que nele estarão contidos.

2.3 XDC (XML DOMAIN CONSTRAINTS)

Após estudos realizados, contatou-se a necessidade de aplicar as restrições de integridade de domínio dos bancos de dados relacionais nos documento XML, a fim de melhorar a segurança e integridade dos dados contidos no documento XML.

Partindo deste princípio, Lazzaretti (2005) propôs um controle chamado XDC (*XML Domain Contraint*), o qual é composto pela linguagem XDCL (*XML Domain Contraint Language*) é por um mecanismo de validação chamado *parser*XDCL. Esta linguagem foi desenvolvida por Alexandre Tagliari Lazzaretti, portanto este capítulo será com base no trabalho por ele desenvolvido em sua dissertação de mestrado.

2.3.1 Arquitetura XDC

O controle XDC vem a ser um recurso adicional para a tecnologia XML existente, ele vem suprir algumas necessidades dos documentos XML, como a validação das restrições de

transição de dados, banco de dados e de tuplas, pois estas não são abordadas pelas DTDs e XSDs.

Para que o controle XDC seja usado, primeiramente ele deverá ser criado e salvo em um documento texto com a extensão XDC, para que quando os documentos XML ou os bancos de dados nativos XML forem usá-los, possam fazer a referência do arquivo.

A validação das restrições é feita pelo *parser* XDCL,ele valida as expressões XDCL com as linhas contidas no documento XML, retornando após a validação uma mensagem de validação correta ou a indicação de que alguma inconsistência foi encontrada, esta validação é feita através do uso da linguagem XDCL.

O documento XDC é um documento XML, portanto ele deve estar bem-formado e ser válido, para que seja possível o uso desse documento para realizar as validações. Pelo fato de o documento XDC seguir a sintaxe XML, dentro dele podem ser usados recursos já prontos para documento XML.

2.3.2 Linguagem XDCL

A seção aqui iniciada visa melhor explicar o funcionamento da linguagem XDCL, apresentando cada um dos elementos que compõem a linguagem. O detalhamento da linguagem é feito pela BNF (*BackusNaurForm*)[1].

A linguagem XDCL tem como uma das suas principais características o uso da sintaxe XML, portanto ela é uma linguagem composta por um conjunto de elementos definidos de forma hierárquica e todos esses elementos são armazenados em um documento textual com a extensão XDC. O documento XDC poderá ser usado em qualquer documento XML, para fazer uso dele é necessário no elemento raiz do documento incluir a informação xdcl="caminho_do_documento_xdc".

Por se tratar de um documento XML, o documento XDC segue um esquema XML que define sua estrutura e também valida os elementos e os atributos que compõem a linguagem XDCL. "O principal objetivo da linguagem XDCL é executar uma ação, a qual pode ou não ser executada a partir de uma condição estabelecida" (LAZZARETTI, 2005). A linguagem XDCL pode executar diversas ações como inserção, modificação, exclusão ou mensagem informativa, sendo que algumas dessas não precisam contar com condições pré-estabelecidas. A validação do documento XML baseado no XDC é feita através do *parser* XDCL. A Figura 5 demonstra a BNF da linguagem XDCL, e será feito o detalhamento de cada item abaixo.

[1]Metalinguagem usada para a definição da sintaxe de linguagens formais.

Figura 5 - BNF da linguagem XDCL

```
<xdcl_constraints>
   {<xdcl_constraint xdcl_name="{nome_constraint}">
      [<xdcl_on>{Expressão Xpath}</xdcl_on>]
      {<xdcl_statements>
         [{<xdcl_set_conditions>
            {<xdcl_condition xdcl_operator="{= | >= | <= | > | < | <>}">
               [<xdcl_old>
                  <xdcl_old_filexml>{caminho e nome do arquivo xml antigo}</xdcl_old_filexml>
                  <xdcl_old_identifier  type_id="{ElementDt|ElementNr|ElementSt}" [attr_identifier="{atributo}"]>{elemento}</xdcl_old_identifier>
               </xdcl_old>]
               {<xdcl_operand1 type_condition ="{ElementDt|ElementNr|ElementSt|ExpDt|ExpNr|ExpSt|ExpFuncNr}" [name_attr="{atributo}"] [old="{true}"]>
                  {expressão XPath | elemento}
               </xdcl_operand1>
               {<xdcl_operand2 type_condition ="{ElementDt|ElementNr|ElementSt|ExpDt|ExpNr|ExpSt|ExpFuncNr}" [name_attr="{atributo}"] [old="{true}"]>
                  {expressão XPath | elemento}
               </xdcl_operand2>
            </xdcl_condition>
            [{<AND/> | <OR/> | <NOT/>}][...n]}[...n]
         </xdcl_set_conditions>}]
         {<xdcl_actions>
            {[<xdcl_delete>
               <delete [name_attr= "{nome_atributo}"]>{nome delete}</delete>
             </xdcl_delete> |
            <xdcl_insert>
               {<insert type= "{Element | Attribute}" name_element="{nome_elemento_pai}" [type_place="{Append | Before}"]>{nome insere}</insert>
               [<insert-value>{valor insere}</insert-value>]}
            </xdcl_insert> |
            <xdcl_update>
               <update name_element= "{nome elemento}" [name_attr="{nome atributo}" ] type_value="{Constant | Function}">
                  {função XPath | constante }
               </update>
            </xdcl_update> |
            <xdcl_rename>
               <rename name_element= "{nome_elemento}" [name_attr="{nome atributo}"]>{novo nome}</rename>
            </xdcl_rename> |
            <xdcl_message> {mensagem informativa }</xdcl_message>]}[1...n]
         </xdcl_actions>}
      </xdcl_statements>}
   </xdcl_constraint>}[...n]
</xdcl_constraints>
```

Fonte: Lazzaretti (2005)

Todo o documento XDC começa pelo elemento <xdcl_constraints> e termina com </xdcl_constraints>, este é o elemento raiz do documento XDC, sendo que dentro desse elemento poderá ter vários elementos iniciados por <xdcl_contraint> e terminados por </xdcl_contraint>, sendo que estes definirão cada um uma restrição diferente.

Cada elemento xdcl_constraint contém o atributo xdcl_name e seu conteúdo indicará o nome da restrição o qual ele indica.Na Figura 6 existe um exemplo de documento XDC e nele pode-se perceber que no xdcl_name está definido o nome testa_datas.

O elemento xdcl_constraint terá somente dois filhos, que são o: xdcl_on e o xdcl_statements. O xdcl_on é onde se deve informar o caminho do elemento ou atributo que se deseja validar, seu conteúdo é uma expressão *XPath*, e só é necessário usar este elemento quando houver a necessidade de fazer teste com os elementos.

O elemento xdcl_statements também terá dois filhos, sendo eles, xdcl_set_conditions e xdcl_actions. O elemento xdcl_set_conditions é o responsável por armazenar dentro dele

as restrições que o usuário necessita. Dentro destes elementos poderão existir múltiplos elementos chamados xdcl_condition em cujo interior será especificado a condição, porém para usar mais de um xdcl_condition é necessário utilizar um elemento vazio que é um operador lógico. Para este uso existem 3 possibilidades, são elas: <AND/>, <OR/> ou <NOT/>, caso isso ocorra esses operadores devem ser usados após o elemento </xdcl_constraint>.

O elemento xdcl_contraint ainda deverá ter um atributo chamado xdcl_operator que é quem irá definir qual operação, deverá ser feita nos elementos <xdcl_operand1> e <xdcl_operand2>, os possíveis valores para o elemento xdcl_operator estão sendo mostrados na Tabela 1. Dentro de cada xdcl_operand, tanto o 1 como o 2, podem existir expressões *XPath* para a localização dos elementos, ou o nome dos elementos diretamente.

Tabela 1 - Valores do atributo xdcl_operator

Condição XDC	Condição	Significado
>	>	Maior
>=	>=	Maior ou igual
<	<	Menor
<=	<=	Menor ou igual
<>	<>	Diferente
=	=	Igual

O elemento xdcl_operand possuirá um atributo chamado type_condition, sendo que esse atributo irá definir o tipo do conteúdo do elemento. Na Tabela 2 estão sendo mostrados quais os tipos que se pode definir.

Tabela 2 - Valores do atributo type_condition

Valor	Siginificado
ElementDt	Elemento ou atribuo do tipo data.
ElementNr	Elemento ou atributo do tipo numérico.
ElementSt	Elemento ou atributo do tipo texto.
ExpDt	Elemento ou atributo relativo do tipo data.
ExpNr	Elemento ou atributo relativo do tipo numérico.
ExpSt	Elemento ou atributo relativo do tipo texto
ExpFuncNr	Função *XPath* numérica.

Caso seja usado o tipo de dados *ElementDt*, *ElementNr* ou *ElementSt*, o elemento que foi definido no elemento xdcl_operand, tanto o 1 como o 2, deve ser filho de algum elemento do caminho especificado no elemento xdcl_on, se o tipo de dados especificado for *ExpDt*, *ExpNr* ou *ExpSt*, o elemento a ser verificado pode estar em qualquer nível da hierarquia partindo do caminho definido no xdcl_on e ainda se o tipo da condição for *ExpFuncNr*, será obrigatório que nos elementos xdcl_operand1 e xdcl_operand2 exista uma expressão *XPath* numérica, a qual o resultado será comparado.

Também é definido no documento XDC qual a ação que será executada ao se encontrar alguma das inconsistências definidas. Essa definição ocorre dentro do elemento chamado <xdcl_action>. Dentro do xdcl_action poderão existir alguns outros elementos que definirão a ação a ser tomada. Os elementos que podem estar dentro do xdcl_action são: xdcl_delete (Apaga elementos e/ou atributos), xdcl_insert (Insere elementos e/ou atributos), xdcl_update (Atualiza elementos e/ou atributos), xdcl_rename (Renomeia elementos e/ou atributos) e xdcl_message (Mostra uma mensagem informativa).

No exemplo da Figura 6, foi feita a validação de duas datas, neste exemplo pode-se encontrar exemplos do que foi explicado até o momento. No elemento xdcl_on está especificado o caminho onde ele deve buscar os dois elementos, será buscado dentro da *tag* consulta os elementos especificados nos xdcl_operand1 e xdcl_operand2, que são, respectivamente, data_lancamento e data_realizacao.

Também pode se ver que o operador definido no xdcl_operator será o operador > (Maior), e que os dois elementos xdcl_operand tem o atributo type_condition com o valor *ElementDt*, isso significa que, dentro de cada um desses elementos serão encontrados dados

do tipo data, ainda neste exemplo pode-se encontrar uma operação lógica, no exemplo está sendo usada o operação AND, através do elemento <AND/>.

A ação que ser tomada, caso ocorra a inconsistência dos dados validados, será retornar uma mensagem informativa para o usuário, isso está definido dentro do elemento xdcl_action cujo conteúdo é o elemento chamado xdcl_message e seu conteúdo será a mensagem que o usuário irá receber.

Figura 6 - Exemplo documento XDC

```
<xdcl_constraints>
   <xdcl_constraint xdcl_name="testa_datas">
      <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
      <xdcl_statements>
         <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
               <xdcl_operand1 type_condition="ElementDt">data_lancamento</xdcl_operand1>
               <xdcl_operand2 type_condition="ElementDt">data_realizacao</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator="&lt;&gt;">
               <xdcl_operand1 type_condition="ElementDt" name_attr="meslan">
                  data_lancamento
               </xdcl_operand1>
               <xdcl_operand2 type_condition="ElementDt" name_attr="mesrea">
                  data_realizacao
               </xdcl_operand2>
            </xdcl_condition>
         </xdcl_set_conditions>
         <xdcl_actions>
            <xdcl_message>DATAS INCONSISTENTES</xdcl_message>
         </xdcl_actions>
      </xdcl_statements>
   </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Lazzaretti (2005)

3 FERRAMENTA PROPOSTA

Com base nos estudos realizados e considerando as necessidades encontradas, foi realizado o desenvolvimento de uma ferramenta que possibilita ao usuário definir através da linguagem XDCL as restrições de integridade e consiga fazer a verificação de seus documentos XML conforme as restrições definidas.

Neste capítulo são apresentadas a arquitetura da ferramenta, bem como seu funcionamento e as tecnologias utilizadas para o desenvolvimento da ferramenta.

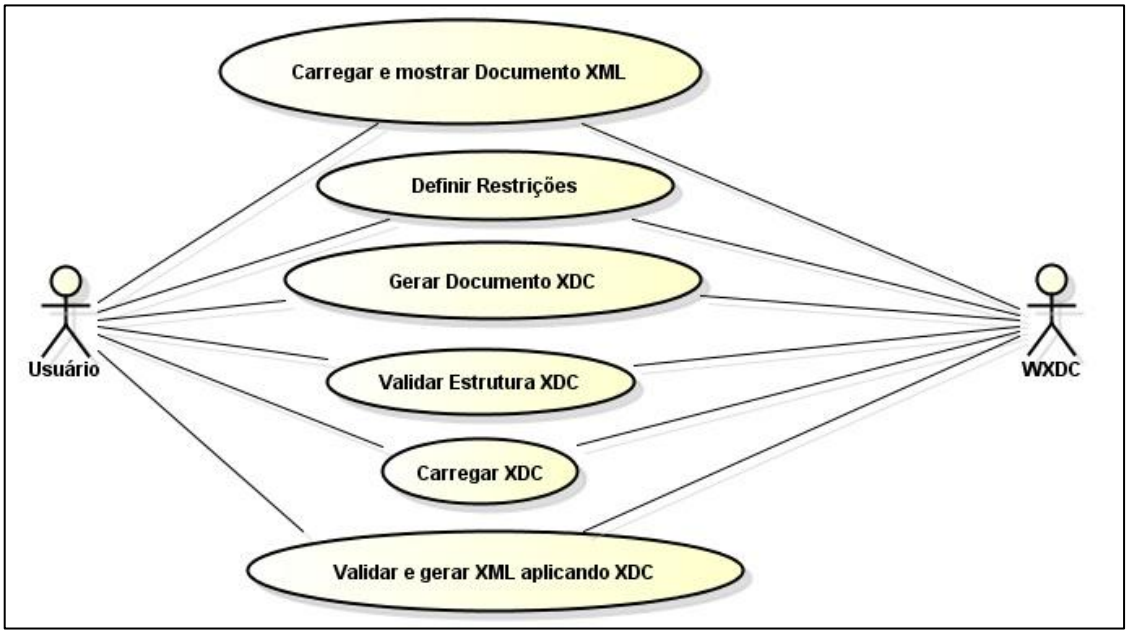
3.1 ARQUITETURA DA FERRAMENTA

A partir do estudo realizado foi possível definir os requisitos funcionais que a ferramenta deve atender:

- Carregar e mostrar documento XML existente;
- Definir restrições;
- Gerar documentos XDC a partir das restrições definidas;
- Carregar documento XDC.
- Validar estrutura de documento XDC;
- Validar e gerar XML aplicando XDC;

A Figura 7 apresenta o diagrama de casos de uso para os requisitos funcionais definidos.

Figura 7 - Diagrama de Caso de Uso

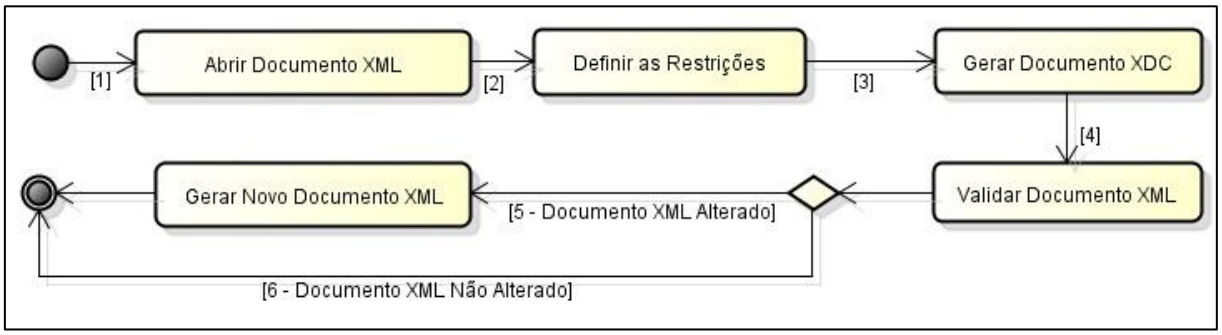


Fonte: Do Autor

3.1.1 Estrutura da Ferramenta

A ferramenta visa auxiliar os usuários na validação dos dados contidos em um documento XML de forma que menos erros ocorram quando armazenar os dados. No Diagrama de Atividades, encontrado na Figura 8, estão representadas as ações para que o usuário faça todo o processo pela ferramenta, desde a geração do documento XDC até o momento da validação.

Figura 8 - Diagrama de Atividades - processo completo



Fonte: Do Autor

Para que o usuário possa fazer a geração do documento XDC na ferramenta, terá que primeiramente carregar o documento XML, que será validado, conforme seta 1. Após ter o documento XML carregado na ferramenta, o usuário irá para a etapa de definição das restrições, seta 2, onde ele fará a inclusão de todas as restrições que devem ser validadas no documento XML.

Com todas as restrições incluídas, o documento poderá ser gerado (seta 3). Após a ferramenta ter gerado o documento XDC com as restrições incluídas pelo usuário, deverá fazer o *download* do documento XDC gerado.

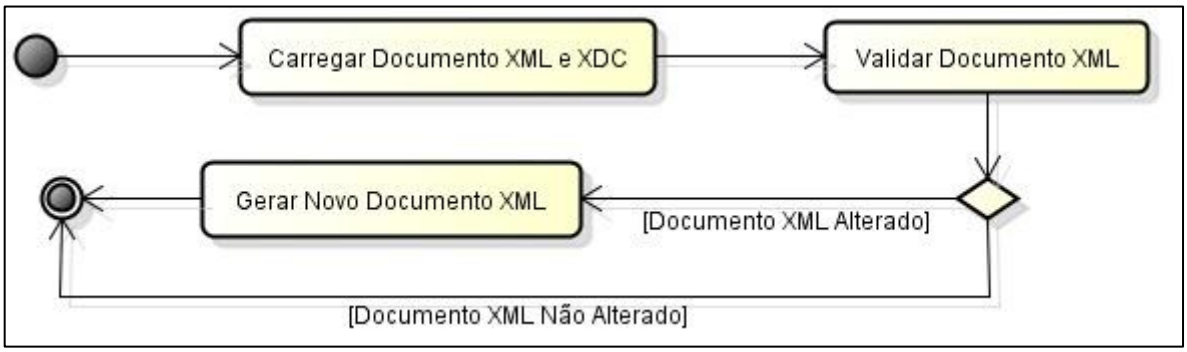
Ainda como recurso adicional, a ferramenta proporciona que o usuário faça a validação do seu documento XDC para garantir que é um documento bem-formado. Essa validação tem por base o *Schema XML* da linguagem XDCL, mostrado no Anexo 1. Caso o documento XDC não esteja bem formado é recomendado que sejam corrigidos os erros para que não ocorram problemas durante a validação do documento XML.

Após o usuário passar por todas estas etapas, segue para a etapa de validação do documento XML, com base no documento XDC, conforme mostrado na seta 4. Nesta etapa o usuário seleciona os dois documentos necessários, e clica no botão “Validar Documento”. A ferramenta faz a validação e retorna mensagens informando o que foi feito. Caso o documento

XML tenha sofrido alterações, será disponibilizado para *download*, este processo está sendo representado na seta 5. Caso o documento não tenha passado por alterações, serão apenas mostradas as mensagens definidas nas restrições (seta 6), terminando assim todo o processo de validação do documento.

Ainda é possível a utilização da ferramenta somente para a validação do documento XML, caso o usuário já tenha um documento XDC com as suas restrições, este processo parcial é mostrado na Figura 9.

Figura 9 - Diagrama de Atividades - processo parcial



Fonte: Do Autor

O processo de validação representado nesta Figura 9, mostra que é necessário somente carregar os dois documentos, XML e XDC, e fazer a validação, deste modo a ferramenta dará como resultado as mensagens informadas no documento XDC, e caso ocorra alguma alteração no documento XML será disponibilizado um novo documento XML em seu estado consistente.

3.1.2 Funcionamento da Ferramenta

Nesta seção será descrito o funcionamento da ferramenta, tomando por base que o usuário fará a geração do documento a partir da ferramenta, sendo assim, executará todas as etapas, inclusive a de validação do documento, para verificar se este foi bem formado.

Após acessar o sistema, o usuário deve como primeiro passo, carregar o documento XML que deseja impor a(s) restrição(ões), nesse momento estará na primeira etapa da geração do documento. A Figura 10 ilustra a tela da etapa descrita acima, já com o documento XML carregado. Pode-se notar na Figura 10 sinais de menos (-) e mais (+) ao lado dos elementos, estes sinais são usados, respectivamente, para comprimir e expandir os elementos do arquivo XML que está sendo visualizado, permitindo assim ao usuário uma melhor visualização de

seu documento, pois o documento após carregado é aberto com todos os seus elementos expandidos.

Figura 10 - Página inicial - carregar documento XML

Fonte: Do Autor

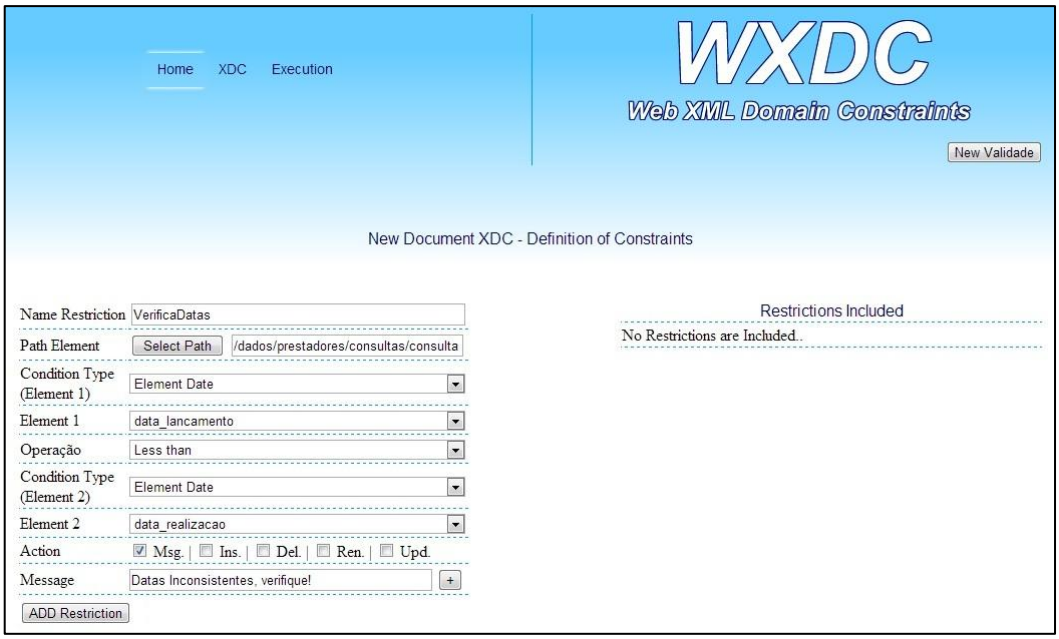
Após carregar o documento XML na ferramenta, o usuário pode passar para a etapa de definição das restrições, onde nesta etapa deverá informar o nome da restrição, o caminho *XPATH*, os dois elementos que deseja comparar, quais os tipos dos elementos, o modo de comparação (condição) e as ações que a ferramenta deve realizar no momento que o documento for verificado.

Na Figura 11 é apresentada a página onde o usuário deve preencher as informações para que o documento seja gerado com sucesso. Pode-se ver no exemplo abaixo os valores correspondentes a cada atributo que irá compor o documento XDC:

- Nome da Restrição: VerificaDatas
- Caminho *XPATH:* dados/prestadores/consultas/consulta/
- Tipo do Elemento 1: Elemento Data
- Elemento 1: data_lancamento
- Condição: Menor
- Tipo do Elemento 2: Elemento Data
- Elemento 2: data_realizacao

Neste exemplo foi definida apenas uma única ação que é a ação *Message*. O uso desta ação faz retornar uma mensagem ao usuário, sendo esta definida pelo próprio usuário. A mensagem usada no exemplo e que será retornada caso a condição seja verdadeira é “Datas inconsistentes, verifique!”. Após a inclusão de todos os dados no formulário o usuário deve clicar no botão “*Include Restriction*”, para que a restrição seja gerada.

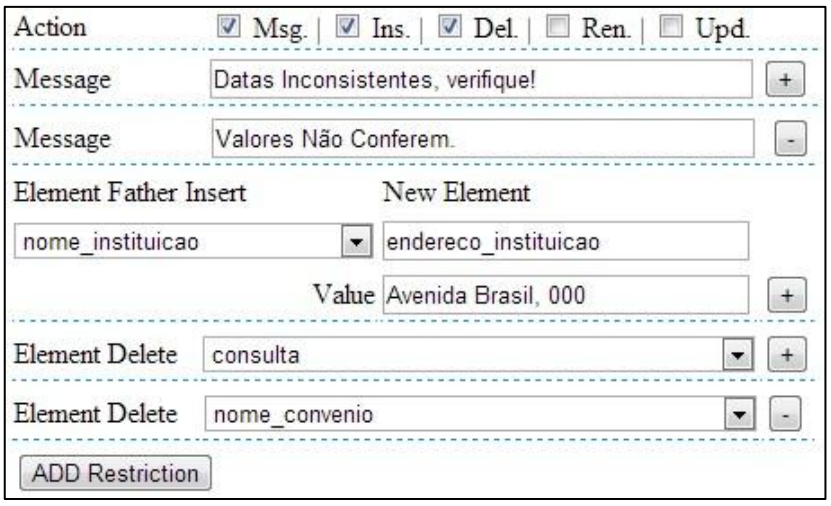
Figura 11 - Página para definição das restrições



Fonte: Do Autor

Existe a possibilidade também de inserir várias ações para a mesma restrição. O usuário pode optar por incluir mais de uma ação do mesmo tipo ou de tipos diferentes, como pode ser visto na Figura 12. Para fazer o uso desta funcionalidade basta clicar no botão de mais (+) ao lado de cada ação, para que um novo campo seja aberto.

Figura 12 - Inclusão de múltiplas ações



Fonte: Do Autor

Para selecionar o caminho *XPath* existe um botão chamado "*Select Path"*, ao clicar nele será aberta uma nova página com o documento XML carregado. Para selecionar o caminho clica-se no elemento no qual a validação deverá começar. Automaticamente a página será fechada e redirecionada para a tela de definição das restrição com o caminho *XPath* do elemento selecionado anteriormente. Esse processo é feito através de uma método que está incluso na biblioteca *XMLTree*, usada para a abertura do XML em forma de árvore, porém com alterações, para que o método retorne corretamente o caminho *XPath* (esse método pode ser visto na Figura 13).

Figura 13 - Método que retorna o caminho *XPath* do elemento selecionado

```
function returnXPathToNode(li){ // Retorna o XPath do elem. clicado.
    var caminho = ""; // XPath do Elemento
    var nome = []; // array que armazena os nomes dos atributos.
    var i = 0, k = 0;
    li.parents('li').andSelf().each(function() {
        // Pega o nome do elemento e adiciona no array
        nome[i] = $(this).children('.LIText').children('.tree_node').text();
        i++; // Incrementa a variavel que controla a posição do array
    });
    i--; // Decrementa o i para que ele fique no ultima elemento selecionado
    while(k <= i){
        caminho += "/" + nome[k];
        k++;
    }
    $('#caminho').val(caminho);
    returnValue();
}
```

Fonte: Do Autor

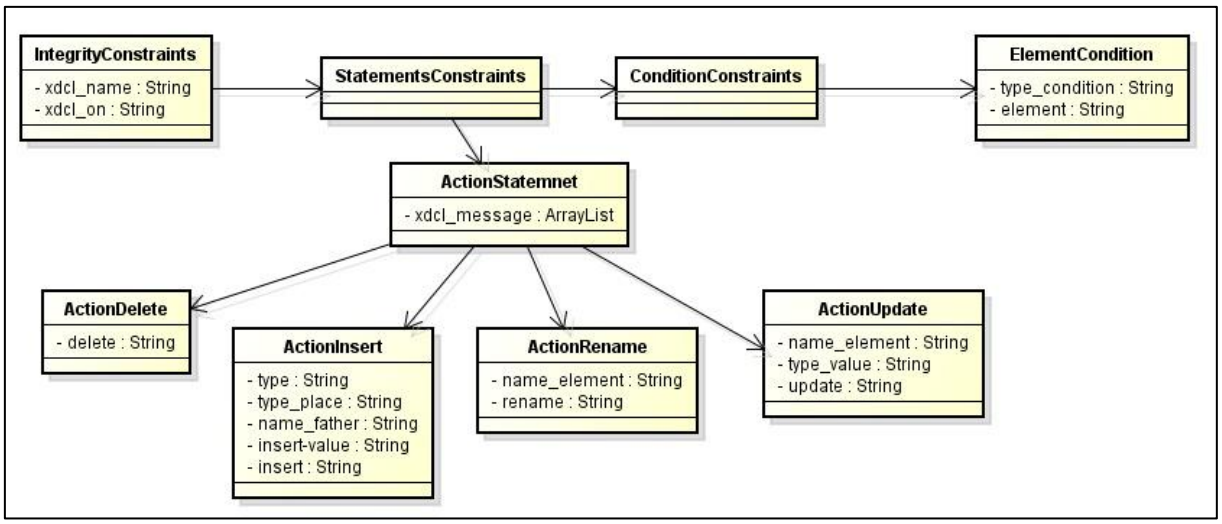
Também existem 5 tipos de ações que podem ser escolhidas, elas são detalhadas na Tabela 3.

Tabela 3 - Ações possíveis para a definição de restrições

Ações	*Descrição*	*Requisitos*
Message	*Mensagem Informativa*	*- Mensagem*
Insert	*Insere um novo elemento após o pai do mesmo.*	*- Nome do elemento pai.* *- Nome do novo elemento.* *- Valor do novo elemento.*
Delete	*Remove um elemento*	*- Nome do elemento.*
Rename	*Renomeia um elemento*	*- Nome do elemento.* *- Novo nome do elemento.*
Update	*Atualiza o valor de um elemento*	*- Nome do elemento.* *- Valor para o novo elemento.*

A Figura 14 apresenta o diagrama de classes do processo de definição das restrições. Após o usuário clicar no botão "*Include Restriction*", as classes são populadas com os dados vindos do formulário, e a classe *IntegrityConstraints* será colocada em uma lista de restrição que estará na sessão do usuário. Assim, é possível que o usuário inclua diversas restrições em um único documento, pois o documento será gerado com base nesta lista de restrições. Como pode-se notar na Figura 14, os dados serão empacotados nas classes e somente uma será adicionada na lista, pois tendo a referência para a primeira classe, ou seja, a classe *IntegrityConstraints*, tem-se acesso a todas as outras classes, promovendo assim um melhor controle sobre os dados que estão sendo enviados aos métodos no momento da geração do documento XDC.

Figura 14 - Diagrama de Classes do processo da definição das restrições

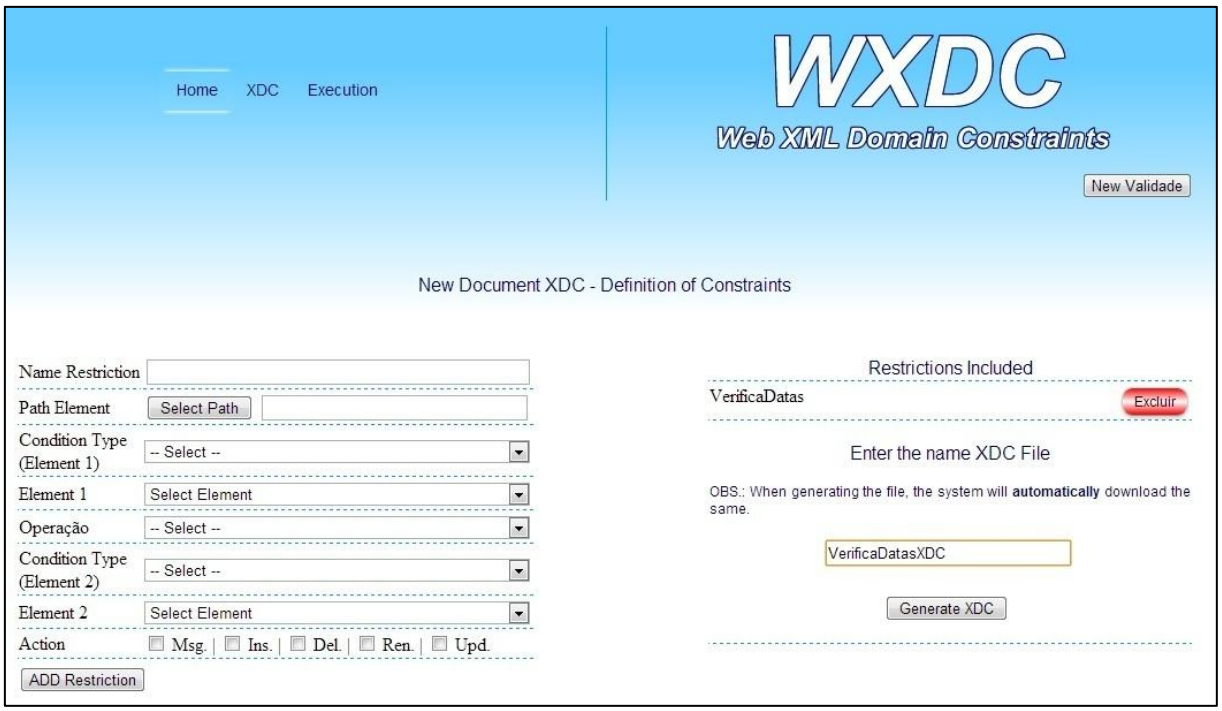


Fonte: Do Autor

Com a conclusão da geração das classes que definem uma restrição, será mostrada uma lista com as restrições já incluídas, estas restrições farão parte do documento XDC que será gerado. Há possibilidade de fazer a exclusão de restrições que foram incluídas, para fazer a exclusão de um registro, uma restrição, basta clicar no botão “Excluir” que se encontra ao lado da mesma.

Ao incluir a primeira restrição, será habilitado o espaço para a geração propriamente dita do documento XDC, mas para que o documento possa ser gerado, deve-se obrigatoriamente informar um nome para este documento. A Figura 15 representa esta etapa após a inclusão de pelo menos uma restrição. Em nosso exemplo o nome do documento é "VerificaDatasXDC", após isso o documento XDC poderá ser gerado. Ao término da geração do documento XDC, é solicitado ao usuário que faça o *download* do documento gerado, para o uso posterior.

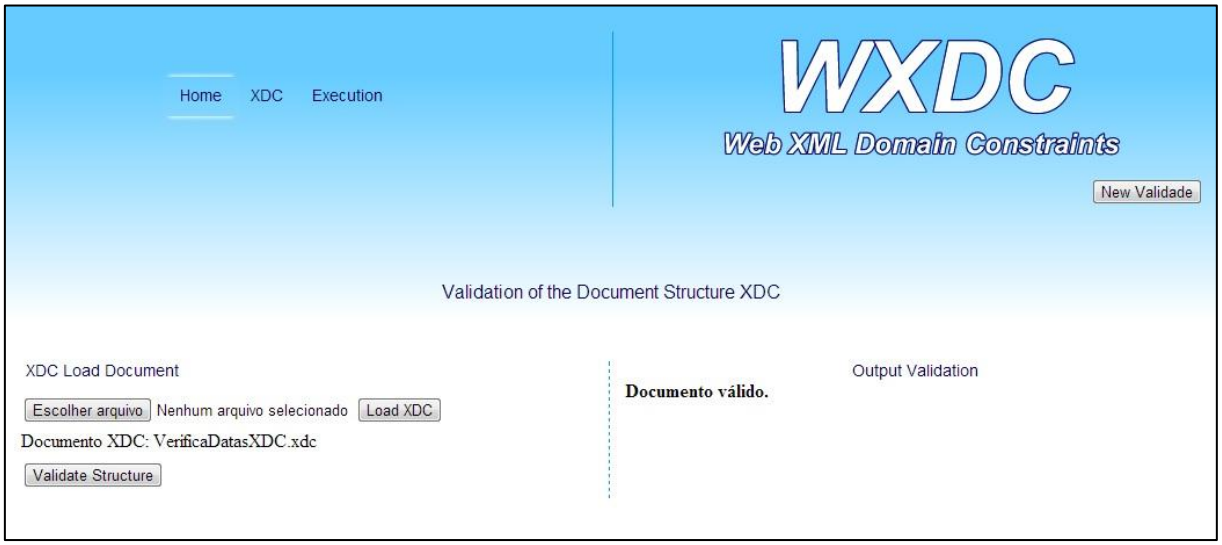
Figura 15 - Definição das Restrições (Restrição Incluída)



Fonte: Do Autor

Ao passo que o documento XDC foi gerado, pode-se passar para a etapa de validação do documento, para verificar se o documento está bem-formado. A página responsável por esta validação é a pagina "*Validate Structure"* ou "Validação da Estrutura", conforme Figura 16.

Figura 16 - Página para validação da estrutura do documento



Fonte: Do Autor

Para iniciar a etapa de validação do documento, primeiramente, deve-se informar o documento XDC correspondente, e após, clicar no botão “Validar Estrutura” ou “*Validate Structure*” para iniciar o processo de validação. Este processo é feito com base em um *schema XML* próprio da linguagem XDCL, arquivo que consta no Anexo 1.

A validação da estrutura do documento é feita por meio da tecnologia SAX. "SAX é uma API SAX baseada em eventos, seus métodos e estruturas de dados são mais simples do que os de DOM, podendo muitas vezes alcançar alta performance." (Veloso, 2003).

Para fazer a validação da estrutura foram utilizados dois métodos, um para carregar o *schema XML* na ferramenta, e outro para fazer a validação do documento. Para criar o *schema* é passado por parâmetro para o método *"loadSchema"*, o caminho do *schema XML* que se deseja carregar. O método pega o caminho e cria, a partir da classe *SchemaFactory*, um novo *schema XML*, o qual será usado como parâmetro para o outro método utilizado chamado "*validateXml"* (encontrado na Figura 17).

Figura 17 - Método de Validação da Estrutura

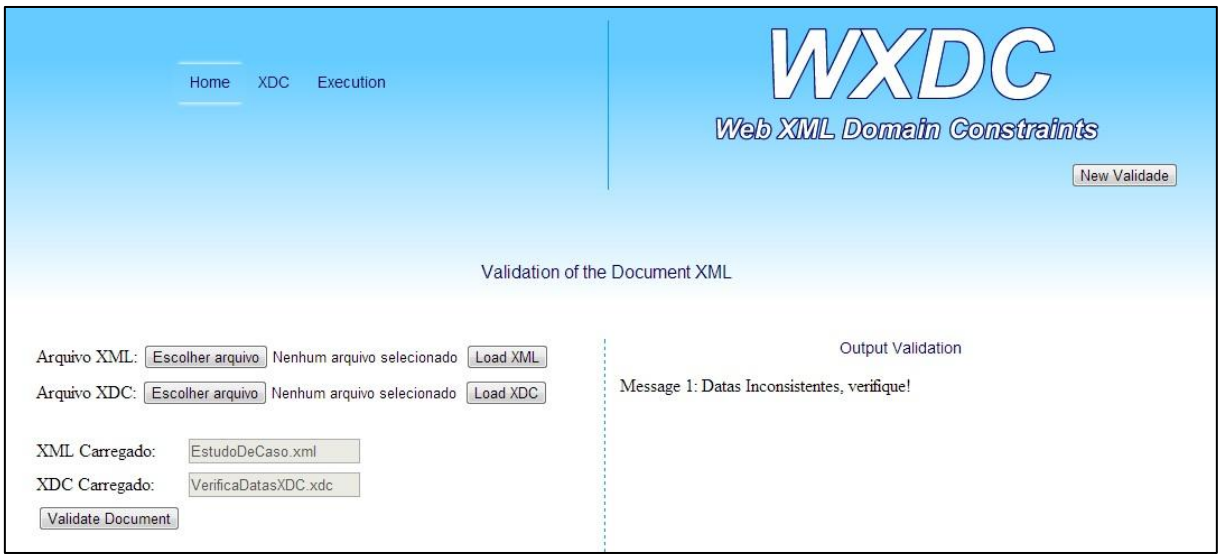
```
public String validateXml(Schema schema, String xmlName) {
    String retorno = "";
    try {
        // Criando uma instância do validador
        Validator validator = schema.newValidator();
        // Preparando o documento XML
        SAXSource source = new SAXSource(new InputSource(new java.io.FileInputStream(xmlName)));
        // Validando o documento XML
        validator.validate(source);

        retorno = "<strong>Valid Document.</strong>";
    } catch (SAXException | IOException e) {
        retorno = "<strong>Invalid Document</strong>, check please:";
        retorno += "<br>Line: " + ((SAXParseException) e).getLineNumber();
        retorno += "<br>Reason: " + ((SAXParseException) e).getMessage();
    } finally {
        return retorno;
    }
}
```

Fonte: Do Autor

Ao chamar o método *"validateXml"* é necessária a passagem de dois parâmetros, um é o *schema*, como mencionado anteriormente, e o outro é o nome ou caminho do documento XML para validação. Este método irá retornar uma *string* de documento válido ou documento inválido e mais alguns dados necessários para que o usuário possa verificar o erro com maior facilidade, e o retorna deste método será o resultado da validação, mostrado abaixo do título "*Output Validation*".

Após o documento XML ter sido carregado na ferramenta, o documento XDC gerado, e validada a estrutura do mesmo, chega-se na etapa de validação do documento XML com base no documento XDC. Está etapa irá necessitar que o documento XML e o documento XDC estejam carregados na ferramenta. Como o usuário carregou o documento XML na primeira etapa e validou a estrutura do documento XDC na terceira etapa do processo, nesta etapa não precisa selecionar novamente os documentos, pois eles já estarão carregados na sessão de usuário, somente é necessário clicar no botão "*Validate Document*" para iniciar o processo de validação, através do *parser XDCL*. A Figura 18 demonstra a página de validação do documento XML e na sequência será descrito o funcionamento do *parser*.

Figura 18 - Página de Validação do Documento



Fonte: Do Autor

A validação dar-se-á pelo *parser* XDCL, implementado por Lazzaretti (2005), que irá validar as condições e executar as ações definidas no documento XDC. O algoritmo do *parser* XDCL é definido conforme um conjunto de ações as quais se aplicam recursivamente sobre o documento XML. O *parser* segue uma lista de passos pré-definidos por Lazzaretti (2005), os quais são descritos em síntese a seguir.

Primeiramente o *parser* fará a leitura do documento XML e buscará no atributo do elemento raiz chamado xdclo seu conteúdo, o qual é o caminho para o arquivo XDC que contém as restrições. Caso exista um arquivo XDC referenciado, este será instanciado e será feita novamente a validação da estrutura, para garantir que não ocorram erros no momento da imposição das restrições. Com o documento XDC bem-formado, efetua-se a leitura do mesmo

e busca-se o elemento xdcl_constraints, para verificar se existe restrição ou não, caso exista, o *parser* irá buscar o elemento xdcl_condition e irá armazenar o valor do atributo xdcl_operator que indica qual a comparação que deverá ser feita ao se ler o conteúdo do xdcl_operand1 e o valor do atributo type_condition, o mesmo ocorre para o elemento xdcl_operand2. Após a leitura desses elementos é feita a leitura dos elementos correspondentes no arquivo XML.

Depois de feita a validação das condições e verificado se a condição é verdadeira ou falsa, é realizada a leitura das ações que se encontram no elemento xdcl_actions, caso encontre alguma ação, estas são lidas e executas pelo *parser* por meio da API DOM.

Após o término da validação do documento, a ferramenta retornará algumas mensagens de alerta para que o usuário saiba o que o ocorreu durante validação. As mensagens serão sempre as mesmas, exceto a mensagem que será retornada a partir da ação *Message.* Quando documento estiver consistente, também, será retornado uma mensagem. Abaixo segue a Tabela 4 que mostra quais as mensagens que a ferramenta poderá retornar.

Tabela 4 - Lista de Mensagens Retornadas pela Ferramenta

Ação	Mensagem Retornada
Message	Mensagem informada pelo usuário.
Insert	Insert efetuado com sucesso.
Delete	Delete efetuado com sucesso.
Rename	Rename efetuado com sucesso.
Update	Update efetuado com sucesso.
Nenhum Erro ou Documento Consistente	Documento Consistente.

Como no exemplo foi especificado apenas a ação *Message,* será retornado apenas a mensagem que o usuário definiu na hora da criação da restrição.

Após o término da validação será retornado ao usuário as mensagens, e caso tenha sido definida alguma ação, será disponibilizado um botão para o *download* do documento com as modificações feitas pelas ações que o usuário definiu.

Com isso a validação chega ao fim, e o documento agora estará consistente. Veremos mais exemplos de validações e de geração de documento XDC no Capítulo 4, onde serão mostrados alguns casos de uso.

3.2 TECNOLOGIAS UTILIZADAS

Diversas tecnologias foram utilizadas para o desenvolvimento da ferramenta. Esta seção apresenta as tecnologias divindido-as em *cliente* e *servidor*, sendo *cliente* aquelas interpretadas pelo navegador e *servidor* aquelas com execução no lado servidor.

3.2.1 Cliente

A seguir são apresentadas as tecnologias do lado cliente utilizadas no desenvolvimento da ferramenta.

a. *JavaScript*

O *JavaScript* foi desenvolvido pela Netscape em parceria com a Sun Microsystems, tem por finalidade proporcionar ao usuário mais interatividade com as páginas web. O *JavaScript* foi desenvolvido para rodar no lado cliente, ou seja, ela é interpretado pelo navegador que o usuário utiliza. Existem muitas funcionalidades que podem ser feitas a partir da linguagem *JavaScript*, como por exemplo: manipular o conteúdo de uma página, manipular o navegador, interagir com formulários, entre outras (SILVA, Maurício, 2010b).

A linguagem *JavaScript* também adota os conceitos de separação das camadas no momento do desenvolvimento, ou seja, a camada de apresentação e a camada de comportamento são separadas, isto significa que, todo e qualquer código *JavaScript* deve estar em um arquivo separado e onde for utilizar determinado código, deve-se referenciar o arquivo. Isso traz uma série de vantagens, tais como, eliminação de código redundante, reaproveitamento de código, busca e correção de eventuais *bugs*, facilita a manutenção e o entendimento dos códigos (SILVA, Maurício, 2010b).

Como citado anteriormente, o *JavaScript* permite manipular o conteúdo e a apresentação de uma página que está sendo visualizada no navegador. Neste trabalho adotou-se a biblioteca *jQuery* para prover esta funcionalidade.

b. Biblioteca *jQuery*

A biblioteca *jQuery* é uma biblioteca criada por John Resig, de software livre, que pode ser usada tanto para projetos pessoais, como para projetos comerciais. (SILVA, Maurício, 2010a). Esta biblioteca facilita e agiliza a programação e a criação de efeitos em *JavaScript*, o mais interessante é que para fazer a criação destes, não é preciso ser um profundo conhecedor da linguagem *JavaScript*, pois seu uso da biblioteca torna o

desenvolvimento mais simples e rápido, por isso uma das principais características do *jQuery* é a simplicidade.

Silva, Maurício (2010a) cita algumas outras características importantes da biblioteca *jQuery*, como: indiferença às inconsistências dos navegadores, interação implícita, extensível, entre outras. Silva, em seu livro, cita um pensamento de John Resig que define muito bem o que é a biblioteca *jQuery:* "O foco principal da biblioteca jQuery é a simplicidade. Por que submeter os desenvolvedores ao martírio de escrever longos e complexos códigos para criar um simples efeito?" (John Resig *apud* SILVA, Maurício, 2010a).

A biblioteca *jQuery* é usada para muitas recursos no desenvolvimento web, principalmente, para trazer dinamismo e interatividade as páginas, para trazer uma maior usabilidade, aumentar a acessibilidade e melhorar o *design* das páginas. É possível, por exemplo, adicionar efeitos visuais e animações, buscar informações no servidor via AJAX, fazer a alteração de conteúdo, simplificar as tarefas específicas de *JavaScript*. Um exemplo de uso do *jQuery*, que foi usado no desenvolvimento da ferramenta, pode ser encontrado na Figura 19, onde este é usado para fazer a manipulação de elementos HTML.

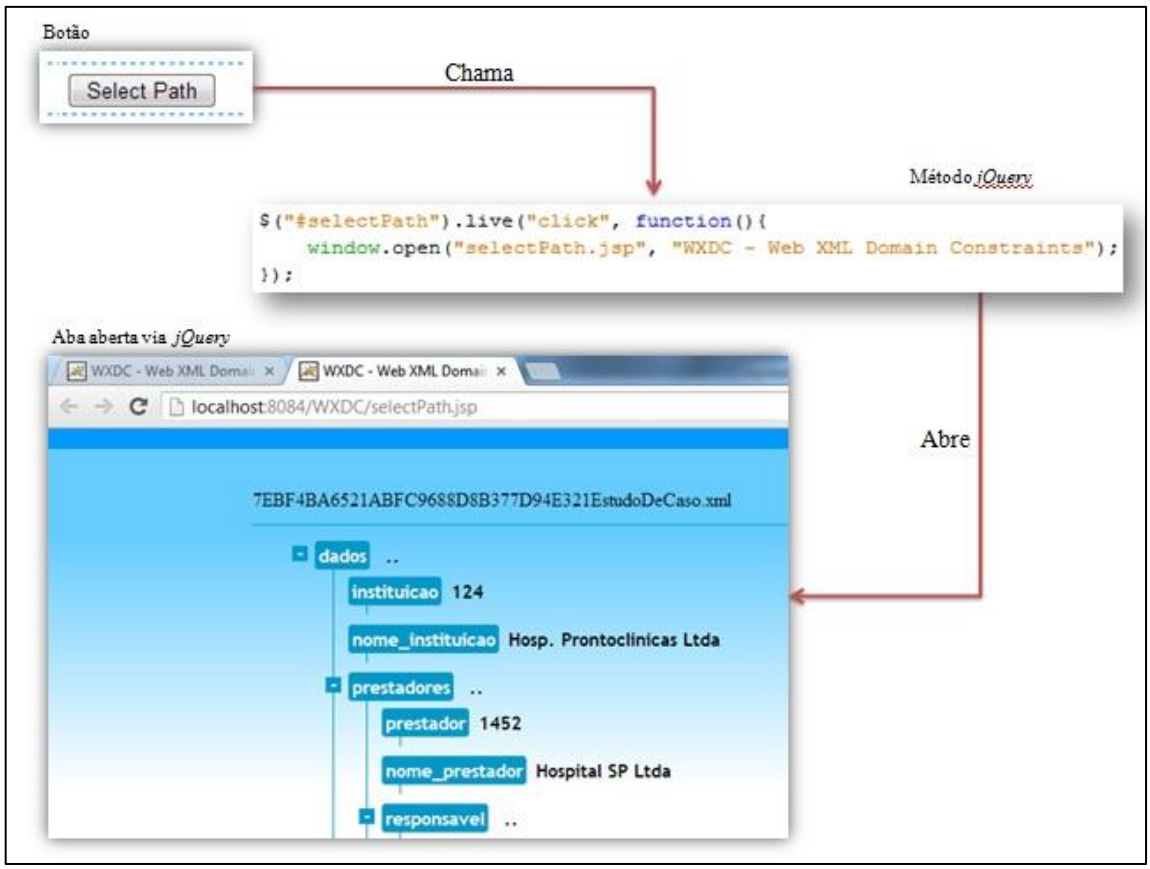
Figura 19- Exemplo do uso da biblioteca *jQuery*

```
function moreTrRename(){
    countRename++;

    var tr = $("#actionRenameTR").html();
    $("#moreActionRename").append("<table id='moreRenameTable'>" + tr + "</table>");
    $("#moreRenameTable #moreRename").hide();
    $("#moreRenameTable #lessRename").show();

    $("#countRename").val(countRename);
}
```

Fonte: Do Autor

Também foi usado *jQuery* para capturar o clique de um botão e abrir uma nova aba do navegador, este processo é mostrado na Figura 20.

Figura 20 - Abertura de uma nova aba via *jQuery*



Fonte: Do Autor

c. AJAX

AJAX ou *Asynchronos JavaScript And XML* não é uma tecnologia, mas sim um conjunto de tecnologias, como XML, *JavaScript*, DHTML, CSS, DOM, e *XMLHttpRequest*, usadas de forma integrada. O AJAX é responsável por fazer requisições assíncronas ao servidor, e quando o servidor responder, fazer modificações na página, sem que toda ela seja recarregada, fazendo assim que menos tráfego de internet seja usado e tornando a página mais interativa para o usuário.

Conforme Aramburu (2009), as requisições AJAX quando necessitam buscar muitas informações, como por exemplo, buscar dados no banco de dados, é utilizada a tecnologia XML para melhor estruturar os dados. Para fazer essa comunicação são utilizados dois tipos de requisição:

- GET: é o método mais comum de se encontrar, solicita um recurso por meio da URL.
- POST: neste método os dados são enviados junto com o comando, para serem processados.

Na ferramenta proposta neste trabalho o AJAX foi utilizado para carregar as opções da *combobox* com os nome dos elementos encontrados a partir do caminho *XPath* selecionado pelo usuário. O exemplo desta utilização é mostrado na Figura 21.

Figura 21 - Exemplo Requisição AJAX

```
$().ready(function(){
    $("#pathElement").blur(function(){
        // Recuperando os valores dos elementos do formulário.
        var xpath = $("#pathElement").val();
        var docXML = $("#pathDocument").val();
        // Chamando o método que retorna os elementos.
        $.post('CarregaCombo',
        {
            elementXPath:xpath,
            pathDocument:docXML
        },
        function(data){
            // setando para as combobox os dados retornados.
            $("#element1").html(data),
            $("#element2").html(data);
            $("#elementDelete").html(data);
            $("#elementInsert").html(data);
            $("#elementRename").html(data);
            $("#elementUpdate").html(data);
        },
        'html');
        return false;
    })
})
```

Fonte: Do Autor

d. Biblioteca XMLTree

XMLTree é uma biblioteca open-source desenvolvida por Croxall (2011). O desenvolvedor da biblioteca menciona que XMLTree é um utilitário para a visualização de documento XML. O script cria uma árvore a partir da passagem de um documento XML.

A biblioteca foi utilizada na ferramenta para a visualização do documento XML carregado pelo usuário na página inicial e também no momento que é escolhido o caminho *XPath* do elemento selecionado. A função que retorna o caminho *XPath* já vinha previamente incluída na biblioteca, porém para que o método retornasse corretamente o caminho *XPath* foram necessárias várias alterações no método. O método que traz esse retorno já foi ilustrado neste trabalho na Figura 13.

Para que a correta visualização da árvore é necessário a passagem de dois parâmetros obrigatórios, o primeiro parâmetro será o documento XML que será mostrado, para isso, existem 3 maneiras de passagem de parâmetro (conforme Tabela 5).

Tabela 5 - Parâmetros para informar o documento XML a ser aberto pelo XMLTree

Nome do Parâmetro	**Valor do Parâmetro**
fpath	Caminho do documento XML.
xml	Documento XML em forma de *string.*
jsonp	Documento XML no formato JSON[2].

Além destes parâmetros obrigatórios podem ser passados outros parâmetros opcionais, na Figura 22 pode-se observar que são passados mais dois parâmetros, um chamado *"startExpanded"*, que é para quando o documento for aberto todos os seus elementos apareçam expandidos, e o segundo é o *"clickCallback"*, que é usado para chamar o método que retorna o caminho *XPath* ao clicar-se no elemento.

Figura 22 - Chamada para abertura e visualização da árvore usando XMLTree

```
$(function() {
    new XMLTree({
        fpath: '<%= documento%>',
        container: '#tree',
        startExpanded: true,
        clickCallback: returnXPathToNode
    });
```

Fonte: Do Autor

3.2.2 Servidor

A seguir são apresentadas as tecnologias do lado servidor utilizadas no desenvolvimento da ferramenta.

a. Java

A tecnologia Java começou a ser criada em 1991 com o nome de "*Green Project*". Primeiramente a linguagem seria usada para pequenos dispositivos, mas com o advento da web, a *Sun* decidiu lançar o Java para rodar pequenas aplicações nos navegadores, mas com o passar dos anos a linguagem Java amadureceu e hoje é uma das linguagem mais usadas, por

[2] Padrão de transferência de dados na internet entre aplicações de linguagens diferentes, semelhante ao XML, porém mais fácil para usuários comuns compreenderem.

ser simples, robusto, seguro, portável, além de ter um alto desempenho e dinamismo (CORNELL e HORSTMANN, 2010).

A tecnologia Java não é somente uma linguagem mais sim, um conjunto de linguagem mais um programa de execução chamado máquina virtual ou *virtual machine*, esse conjunto é chamado tecnologia Java, onde a máquina virtual ou *virtual machine* é uma camada entre sistema operacional e a aplicação, sendo responsável por "traduzir" o código dependendo da plataforma utilizada, fazendo assim com que a tecnologia Java seja multiplataforma (SERSON, 2007).

b. Java Server Pages

"JSP é um acrônimo para *Java Server Pages* e consiste numa linguagem de *script* baseada em Java para criação de sites com conteúdos dinâmicos." (BRAZ, *on-line*).

O JSP é uma tecnologia para o desenvolvimento de aplicações web, similar ao ASP e ao PHP, só que baseada na linguagem Java. O JSP por ser baseado na tecnologia Java, herda as mesma características, como por exemplo, a portabilidade, podendo assim o desenvolvedor criar várias aplicações com acesso ao banco de dados, entre outros, sem se preocupar com a compatibilidade entre os diversos sistemas operacionais existentes.

Seguindo o mesmo molde de outras linguagens como o PHP, o JSP pode ser incorporado junto com os elementos HTML, fazendo assim que os comandos dinâmicos sejam processados e somente a resposta seja mostrada, no formato HTML.

A Figura 23, demonstra um trecho como descrito acima.

Figura 23 - Exemplo do uso de JSP

```
<input type="text" name="pathElement" id="pathElement"
       value="<% out.print(caminhoXPath);%>" size="33"/>
```

Fonte: Do Autor

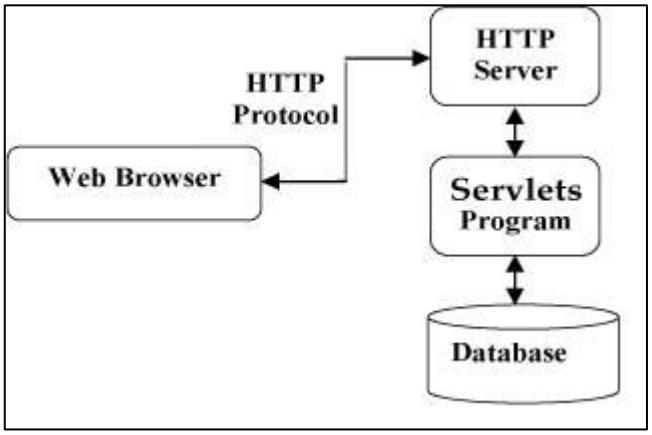
c. Servlets

Braz (*on-line*) explica o que são *servlets:* "*Servlets* são classes Java, que fundamentalmente, fazem a manipulação das requisições e respostas. Os *servlets* são os responsáveis por lidar tipicamente com características do HTTP, como por exemplo, os métodos GET e POST".

Os *servlets* servem basicamente para fazer a criação de páginas dinâmicas, podendo ser criadas páginas HTML, assim como outros dados, como por exemplo gerar um documento XML.

A principal diferença entre os *Servlets* e as páginas JSP, é que nos *servlets* o código HTML é incorporado dentro do código Java e já no JSP o código Java é incorporado ao HTML. A estrutura de uma requisição é mostra na Figura 24.

Figura 24 - Estrutura requisição *servlet*



Fonte: Do Autor

Nessa ferramenta os *servlets* serviram para fazer o controle de algumas funcionalidades, onde um *servlet* recebe uma requisição, faz o processamento dos dados utilizando as classes e retorna um resultado ao cliente, ao navegador. Na Figura 25, está um exemplo de *servlet* que faz o controle da inclusão das restrições.

Figura 25 - *Servlet* que controla a inclusão das restrições

```
@WebServlet(name = "includeRestriction", urlPatterns = {"/includeRestriction"})
public class IncludeRestriction extends HttpServlet {
    ArrayList<String> listNameRestriction = new ArrayList<String>();
    Utilidades util = new Utilidades();
    protected void processRequest(HttpServletRequest request,
                                  HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.print("Servlet de Restrições");
        this.util.generateClass(request);
        response.sendRedirect("setRestriction.jsp");
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

Fonte: Do Autor

d. DOM

DOM ou Document Object Model é um conjunto de classes que fazem a leitura do documento XML de forma hierárquica de nós e objetos, ou seja, usando DOM o documento XML será lido de nó em nó, fazendo assim com que uma árvore seja formada na memória. Dentre as vantagens de se utilizar DOM é que para consultadas via *XPath* se torna mais rápida a pesquisa pois já existe uma árvore. Esta tecnologia é a recomendação da W3C para a manipulação de documento XML (VELOSO, 2003).

Ao passar por cada nó do documento XML, este é transformado em um objeto *Node* e as ligações entre os objetos irão formando a árvore.

Nesta ferramenta a tecnologia DOM foi utilizada no momento da geração do documento XDC, onde será primeiramente criado um novo documento e depois sucessivamente serão criados os elementos filhos deste documento.

A Figura 26 mostra como é feita a criação de um novo documento XML, via DOM.

Figura 26 - Exemplo criando um documento via DOM

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
// Cria um novo documento.
this.docXDC = builder.newDocument();
```

Fonte: Do Autor

e. SAX

SAX ou *Simple API for XML* é o conjunto de classes que fazem a leitura de um documento XML por uma sucessão de eventos, ou chamadas de métodos. A tecnologia SAX não é uma recomendação da W3C, porém muitas vezes tem uma performance melhor do que a API DOM, além disso ocupa menos espaço em memória, uma vez que o documento não estará na memória (VELOSO, 2003).

Na ferramenta proposta a tecnologia SAX foi utilizada no *parser* desenvolvido por Lazzaretti (2005), que faz a validação e imposição das regras descritas no documento XDC sobre o documento XML.

Toda a validação do documento XML é feita através da API SAX que, segundo Veloso (2003), é processada na seguinte ordem de eventos: *startDocument(), startElement(), characters(), endElement() e endDocument().*

Na Figura 27 é demonstrado o método "*startElement*" que é disparado toda vez que um novo elemento é encontrado.

Figura 27 - Exemplo método SAX

```
public void startElement(String namespaceURI, String sName, String qName,
                         Attributes attrs) throws SAXException {

    //busca o valor do atributo xdcl_name do elemento xdcl_constraint
    if (qName.equals("xdcl_constraint")) {
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aqName = attrs.getLocalName(i);
                V_nomerestricao = aqName + attrs.getValue(i);
            }
        }
    }
```

Fonte: Do Autor

4 ESTUDO DE CASO

Esta seção tem por objetivo demonstrar a ferramenta em uso, portanto será apresentado um exemplo de documento XML, algumas restrições que podem ser impostas sobre o documento XML, o documento XDC gerado pela ferramenta, e também o documento XML que a ferramenta WXDC irá retornar como saída. O documento XML apresentado na Figura 28 será o documento de entrada do primeiro teste.

Figura 28 - Documento XML de entrada

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dados>
    <instituicao>124</instituicao>
    <nome_instituicao>Hosp. Prontoclinicas Ltda</nome_instituicao>
    <prestadores>
        <prestador>1452</prestador>
        <nome_prestador>Hospital SP Ltda</nome_prestador>
        <responsavel>
            <nome>Stevan Danielli Costa</nome>
            <endereco>Rua Tuiuti, 177</endereco>
            <telefone>99645188</telefone>
        </responsavel>
        <consultas>
            <nome_responsavel>Paulo Freitas</nome_responsavel>
            <consulta>
                <autorizacao>813321</autorizacao>
                <paciente>14578</paciente>
                <nome_paciente>Adalgisa Severo</nome_paciente>
                <convenio>78</convenio>
                <nome_convenio>Saude Brasil Individual</nome_convenio>
                <medico>200</medico>
                <nome_medico>Adao Soares</nome_medico>
                <data_lancamento>12/09/2004</data_lancamento>
                <data_realizacao>12/10/2004</data_realizacao>
                <valor_base>27.00</valor_base>
                <medico_aut>Adao Soares</medico_aut>
                <valor_pago>23.00</valor_pago>
            </consulta>
        </consultas>
    </prestadores>
</dados>
```

Fonte: Do Autor

Com base nesse documento serão validadas algumas restrições que são importantes impor, a seguir segue o documento XDC de cada uma das validações, a explicação e o documento XML consistente.

Na Figura 29 está o documento XDC que irá validar se o elemento nome é diferente do elemento nome_responsável, caso seja diferente, irá acontecer a ação de *Update* no elemento nome_responsável, informando que o valor correto deste elemento é "Stevan Danielli Costa" e também adicionará um elemento após o elemento consultas chamado telefone_responsavel, com seu valor igual a "99645188".

Figura 29 - Documento XDC - testa_responsavel

```
<?xml version="1.0" encoding="UTF-8"?>
<xdcl_constraints>
    <xdcl_constraint xdcl_name="testa_responsavel">
        <xdcl_on>/dados/prestadores</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ElementSt">
                        nome
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementSt">
                        nome_responsavel
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_insert>
                    <insert name_element="consultas" type="Element"
                            type_place="Append">
                        telefone_responsavel
                    </insert>
                    <insert-value>99645188</insert-value>
                </xdcl_insert>
                <xdcl_update>
                    <update name_element="nome_responsavel"
                            type_value="Constant">
                        Stevan Danielli Costa
                    </update>
                </xdcl_update>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Do Autor

Com base nesse documento XDC será feita a validação e o resultado é mostrado na Figura 30.

Figura 30 - XML resultante da validação com base no documento XDC testa_responsavel

```
<?xml version="1.0" encoding="UTF-8"?>
<dados xdcl="C:\Users\Stevan\Documents\NetBeansProjects\WXDC
             \build\web\ArquivoXDC
             \353AF499C3B321D4FE1398F8686C948Fvalida_resp.xdc">
    <instituicao>124</instituicao>
    <nome_instituicao>Hosp. Prontoclinicas Ltda</nome_instituicao>
    <prestadores>
        <prestador>1452</prestador>
        <nome_prestador>Hospital SP Ltda</nome_prestador>
        <responsavel>
            <nome>Stevan Danielli Costa</nome>
            <endereco>Rua Tuiuti, 177</endereco>
            <telefone>99645188</telefone>
        </responsavel>
        <consultas>
            <nome_responsavel>
                Stevan Danielli Costa
            </nome_responsavel>
            <consulta>
                <autorizacao>813321</autorizacao>
                <paciente>14578</paciente>
                <nome_paciente>Adalgisa Severo</nome_paciente>
                <convenio>78</convenio>
                <nome_convenio>Saude Brasil Individual</nome_convenio>
                <medico>200</medico>
                <nome_medico>Adao Soares</nome_medico>
                <data_lancamento>12/09/2004</data_lancamento>
                <data_realizacao>12/10/2004</data_realizacao>
                <valor base>27.00</valor base>
                <medico_aut>Adao Soares</medico_aut>
                <valor_pago>23.00</valor_pago>
            </consulta>
        </consultas>
    </prestadores>
</dados>
```

Fonte: Do Autor

O documento que será validado agora é o que resultou da validação anterior. Neste momento serão validados elementos valor_base e valor_pago. Caso o primeiro elemento

seja maior que o segundo elemento, o elemento valor_base deve ser renomeado para valor_base_pago e o elemento valor_pago deve ser excluído. A Figura 31 mostra o documento XDC que fará está validação.

Figura 31 - Documento XDC - testa_valores

```
<?xml version="1.0" encoding="UTF-8"?>
<xdcl_constraints>
    <xdcl_constraint xdcl_name="testa_valores">
        <xdcl_on>/dados/prestadores/consultas/consulta</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&gt;">
                    <xdcl_operand1 type_condition="ElementNr">
                        valor_base
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementNr">
                        valor_pago
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_delete>
                    <delete>valor_pago</delete>
                </xdcl_delete>
                <xdcl_rename>
                    <rename name_element="valor_base">
                        valor_base_pago
                    </rename>
                </xdcl_rename>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Do Autor

Após a imposição e validação do documento XML conforme o documento XDC que consta na Figura 31, temos o XML resultante, mostrado na Figura 32.

Figura 32 - XML resultante da validação com base no documento XDC testa_valores

```
<?xml version="1.0" encoding="UTF-8"?>
<dados xdcl="C:\Users\Stevan\Documents\NetBeansProjects\WXDC\build
             \web\ArquivoXDC
             \353AF499C3B321D4FE1398F8686C948Fvalida_valores.xdc">
    <instituicao>124</instituicao>
    <nome_instituicao>Hosp. Prontoclinicas Ltda</nome_instituicao>
    <prestadores>
        <prestador>1452</prestador>
        <nome_prestador>Hospital SP Ltda</nome_prestador>
        <responsavel>
            <nome>Stevan Danielli Costa</nome>
            <endereco>Rua Tuiuti, 177</endereco>
            <telefone>99645188</telefone>
        </responsavel>
        <consultas>
            <nome_responsavel>
                Stevan Danielli Costa
            </nome_responsavel>
            <consulta>
                <autorizacao>813321</autorizacao>
                <paciente>14578</paciente>
                <nome_paciente>Adalgisa Severo</nome_paciente>
                <convenio>78</convenio>
                <nome_convenio>Saude Brasil Individual</nome_convenio>
                <medico>200</medico>
                <nome_medico>Adao Soares</nome_medico>
                <data_lancamento>12/09/2004</data_lancamento>
                <data_realizacao>12/10/2004</data_realizacao>
                <valor_base_pago>27.00</valor_base_pago>
                <medico_aut>Adao Soares</medico_aut>
            </consulta>
        </consultas>
    </prestadores>
</dados>
```

Fonte: Do Autor

Como as duas validações anteriores tinham ações a serem executadas elas retornaram um novo documento XML, aplicando as imposições descritas no documento XDC, agora após estas validações, ainda é necessário verificar os elementos data_lancamento e data_realizacao, portanto o documento de entrada para esta validação é o documento que resultou da validação anterior, documento exposto na Figura 32. A Figura 33 mostra o documento XDC que fará a validação das datas.

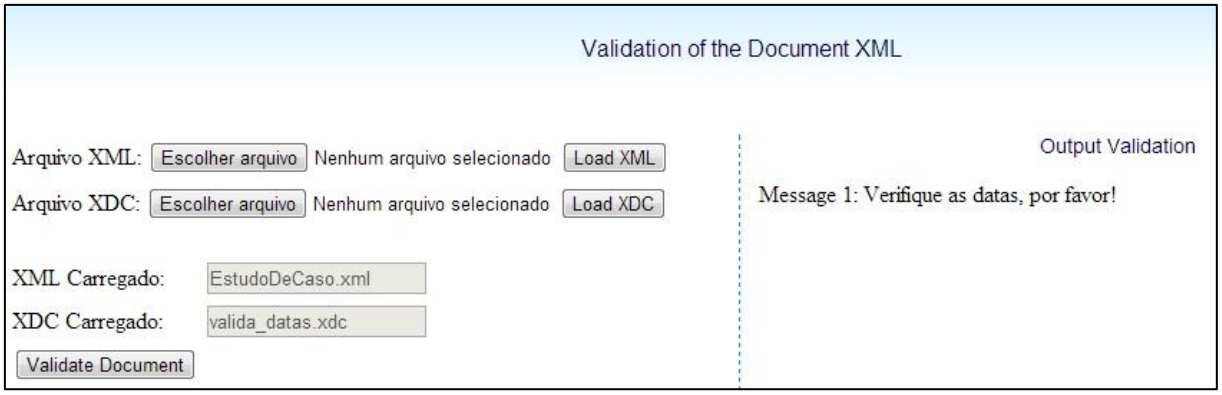
Figura 33 - Documento XDC - testa_datas

```
<?xml version="1.0" encoding="UTF-8"?>
<xdcl_constraints>
    <xdcl_constraint xdcl_name="testa_datas">
        <xdcl_on>/dados/prestadores/consultas/consulta</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;">
                    <xdcl_operand1 type_condition="ElementDt">
                        data_lancamento
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementDt">
                        data_realizacao
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>
                    Verifique as datas, por favor!
                </xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Do Autor

Esta validação não terá nenhum novo documento XML resultante, pois nenhuma ação de modificação do documento foi realizada, apenas foi informado que uma mensagem de alerta deve ser retornada para o usuário caso ocorra a inconsistência. Como foi encontrada a inconsistência, será mostrada na página do usuário a mensagem de alerta que ele mesmo definiu, no caso a mensagem é : "Verifique as datas, por favor!". Na Figura 34 encontramos o resultado da validação.

Figura 34 - Resultado da validação com base no documento XDC testa_datas



Fonte: Do Autor

5 CONSIDERAÇÕES FINAIS

O trabalho apresentou o desenvolvimento da ferramenta WXDC, uma ferramenta que faz a validação de documento XML. A validação do documento XML se dá através de um documento XDC e da linguagem XDCL.

Este trabalho se propôs a fazer a geração dos documentos XDC, abstraindo do usuário conhecimentos específicos sobre a linguagem XDCL, e também fazer a validação das restrições impostas e retornar um documento XML consistente para o usuário. Para que a ferramenta fosse desenvolvida foi necessário o estudo de diversas tecnologias, como por exemplo, a tecnologia Java, JSP e também um estudo específico da linguagem XDCL. Com base nestes estudos e no resultado demonstrado pela ferramenta pronta, os objetivos deste trabalho foram alcançados com sucesso.

Ao longo do desenvolvimento da ferramenta foram encontradas inúmeras dificuldades, como apresentar a visualização do documento XML para o usuário em forma de árvore, também ocorreram algumas dificuldades no momento de adequação do formulário de inclusão de restrições, para que este contemplasse e abstraísse o máximo possível da linguagem XDCL, e no momento de adequação do *parser* para a realidade apresentada neste trabalho.

Partindo destas dificuldades, o trabalho aqui proposto deixa algumas lacunas, como o aperfeiçoamento da ferramenta para que contemple ainda mais a linguagem XDCL, por exemplo com a incorporação de restrições referentes aos atributos, bem como o aprimoramento do *parser*, além da inclusão da opção de comparação com base em dois documentos XML.

REFERÊNCIAS

ANDERSON, Richard; BIRBECK, Mark; KAY, Michael; LIVINGSTONE, Steven; LOESGEN, Brian; MARTIN, Didier; MOHR, Stephen; OZU, Nikola; PEAT, Bruce; PINNOCK, Jonathan; STARK, Peter; WILLIAMS, Kevin. *Professional XML*. Ed. Wrox Press Ltda, 2000. 1a Edição**.**

ARAMBURU, Rodrigo. *Introdução a AJAX com jQuery*. Disponível em: <http://www.botecodigital.info/web/introducao-ao-ajax-com-jqurey/> Acesso em: 19 Nov 2012.

BAX, Macello Peixoto. *Introdução às linguagens de marcação*. In: *Ci. Inf*., Brasília, v. 30, n. 1, p. 32-38, jan./abr. 2001. Disponível em: <http://www.scielo.br/pdf/ci/v30n1/a05v30n1.pdf>. Acesso em: 10 Mai. 2012

BRADLEY, Neil. *The XML Schema companion*. Cidade: Ed.Addilson Wesley, 2004.

BRAZ, Christian Cleber Masdeval. *Introdução a Java Server Pages*. Disponível em: < http://www.webapostilas.com.br/imagens/apostilas/255/java20070312_1740_09_37_12.pdf>. Acesso em 20 Nov 2012.

CORNELL, Gary; HORSTMANN, Cay S. *Core Java, volume I: fundamentos.* São Paulo: Pearson Prentice Hall, 2010.

CROXALL, Andrew. *XML Tree - visualise and traverse your XML*. Disponível em: <http://www.mitya.co.uk/scripts/XML-Tree---visualise-and-traverse-your-XML-186> Acesso em: 10 Out 2012.

DATE, C.J. *Introdução a Sistemas de Banco de Dados*. 9. reimpressão. Tradução de Daniel Vieira. Rio De Janeiro: Elsevier, 2003.

FEDERIZZI, Gustavo Link. *APIs Java para XML*. Disponível em: < http://www.inf.ufrgs.br/gppd/disc/inf01008/trabalhos/sem01-1/t2/apis_xml_java/#IntroDOM>. Acesso em: 20 Nov 2012.

GONÇALVES, Edson. *Desenvolvendo aplicações web com JSP, Servlets, JavaServer Faces, Hibernate, EJB 3 Persistence e Ajax.* Rio de Janeiro: 2007.

HEUSER, Carlos Alberto. *Projeto de Banco de Dados*. 4. ed. Cidade: Ed.SagraLuzzato, 1998. Disponível em: <http://www.julianoribeiro.com.br/troca/banco_de_dados/material_der.pdf> Acesso em: 15 Mai. 2012.

LAZZARETTI, Alexandre Tagliari. *XDC: Uma Proposta de Controle de Restrições de Integridade de Domínio em Documento XML* . Dissertação. Universidade Federal de Santa Catarina Programa de Pós-Graduação em Ciências da Computação, Florianópolis, Fevereiro de 2005. Disponível em: <http://usuarios.upf.br/~lazzaretti/downloads/DI_LAZZA.pdf> Acesso em: 12 Mai. 2012

LIMA, Claudio de. *Uma Ferramenta para Conversão de Esquemas Conceituais EER para Esquemas Lógicos XML*. Disponível em: <http://www.projetos.inf.ufsc.br/arquivos_projetos/projeto_737/MONOGRAFIA_CLAUDIO_FINAL.pdf> Acesso em: 28 Mai 2012

NARDON, Fabiane Bizinella. *Utilizando XML para representação de informações de Saúde.* Disponível em: <http://www.tridedalo.com.br/fabiane/publications/XML-SBISNews.pdf> Acesso em: 18 Mai 2012. (Publicado Originalmente em: Newsletter da Sociedade Brasileira de Informática em Saúde, São Paulo/SP, v. 5, p. 9 - 13, 04 jul. 2000)

SERSON, Roberto Rubistein. *Programação orientada a objetos Java*. Rio de Janeiro: Brasport, 2007

SILBERCHATZ, Abraham; KORTH, Henry. F; SUDARSHAN, S. *Sistemas de Banco de Dados.* Ed. Makron Books, 1999. 3a Edição

SILVA, Maurício Samy. *jQuery: a biblioteca do programador JavaScript.* São Paulo: Novatec Editora, 2010a.

SILVA, Maurício Samy. *JavaScript: guia do programador.* São Paulo: Novatec Editora, 2010b. Disponível em: < http://novatec.com.br/livros/javascriptguia/capitulo9788575222485.pdf> Acesso em: 19 Nov 2012.

SILVA, Marinaldo Nunes Da. *Um Sistema de Controle de Integridade para Modelo de Dados Aberto.* Disponível em: <http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2000/Dissertacao_MarinaldoNunesdaSilva.pdf> Acesso em: 29 Nov 2012.

VELOSO, Renê Rodrigues. *Java e XML, Processamento de documentos XML com Java.* São Paulo: Novatec Editora, 2003.

XML (Extensible Markup Language). Disponível em: <http://apostilas.fok.com.br/attachments/042_apostila-de-xml.pdf>. Acesso em: 10 Mai. 2012

ANEXOS E APÊNDICES

ANEXO 1 – *SCHEMA XML* DA LINGUAGEM XDCL

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schemaelementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="xdcl_constraints">
<xs:complexType>
<xs:sequence>
<xs:element ref="xdcl_constraint" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_constraint">
<xs:complexType>
<xs:sequence>
<xs:element ref="xdcl_on"/>
<xs:element ref="xdcl_statements"/>
</xs:sequence>
<xs:attribute         name="xdcl_name"         type="xs:string"
use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_statements">
<xs:complexType>
<xs:sequence>
<xs:element ref="xdcl_set_conditions" minOccurs="0"/>
<xs:element ref="xdcl_actions"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_set_conditions">
<xs:complexType>
```

```
<xs:choicemaxOccurs="unbounded">
<xs:element ref="xdcl_condition" maxOccurs="unbounded"/>
<xs:element name="AND" minOccurs="0" maxOccurs="unbounded">
<xs:complexType/>
</xs:element>
<xs:element name="OR" minOccurs="0" maxOccurs="unbounded">
<xs:complexType/>
</xs:element>
<xs:element name="NOT" minOccurs="0" maxOccurs="unbounded">
<xs:complexType/>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_condition">
<xs:complexType>
<xs:sequence>
<xs:element ref="xdcl_old" minOccurs="0"/>
<xs:element name="xdcl_operand1">
<xs:complexType mixed="true">
<xs:attribute ref="type_condition" use="required"/>
<xs:attribute ref="name_attr" use="optional"/>
<xs:attribute ref="old" use="optional"/>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_operand2">
<xs:complexType mixed="true">
<xs:attribute ref="type_condition" use="required"/>
<xs:attribute ref="name_attr" use="optional"/>
<xs:attribute ref="old" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
```

```
<xs:attribute        name="xdcl_operator"        type="ope_log"
use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_old">
<xs:complexType>
<xs:sequence>
<xs:element name="xdcl_old_filexml" type="xs:string"/>
<xs:element name="xdcl_old_identifier">
<xs:complexType mixed="true">
<xs:attribute name="type_id" type="tip_old" use="optional"/>
<xs:attribute      name="attr_identifier"      type="xs:string"
use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_actions">
<xs:complexType>
<xs:sequence>
<xs:element ref="xdcl_delete" minOccurs="0"/>
<xs:element ref="xdcl_insert" minOccurs="0"/>
<xs:element ref="xdcl_rename" minOccurs="0"/>
<xs:element ref="xdcl_update" minOccurs="0"/>
<xs:element ref="xdcl_message" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- COMEÇA A DEFINIÇÃO DAS CLÁUSULAS  DO ACTIONS-->
<!-- DELETE -->
<xs:element name="xdcl_delete">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="delete">
<xs:complexType mixed="true">
<xs:attribute ref="name_attr" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- INSERT -->
<xs:element name="xdcl_insert">
<xs:complexType>
<xs:sequence>
<xs:element name="insert">
<xs:complexType mixed="true">
<xs:attribute name="type" type="tip_ins" use="required"/>
<xs:attribute       name="name_element"       type="xs:string"
use="required"/>
<xs:attribute         name="type_place"         type="tip_pla"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:element        name="insert-value"        type="xs:string"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- UPDATE -->
<xs:element name="xdcl_update">
<xs:complexType>
<xs:sequence>
<xs:element name="update">
<xs:complexType mixed="true">
<xs:attribute       name="name_element"       type="xs:string"
use="required"/>
```

```
<xs:attribute ref="name_attr" use="optional"/>
<xs:attribute name="type_value" type="tip_up" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- RENAME -->
<xs:element name="xdcl_rename">
<xs:complexType>
<xs:sequence>
<xs:element name="rename">
<xs:complexType mixed="true">
<xs:attribute ref="name_element" use="required"/>
<xs:attribute ref="name_attr" use="optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- MENSAGEM -->
<xs:element name="xdcl_message" type="xs:string"/>
<!-- DEFINIÇÃO DO ELEMENT -->
<xs:element name="xupdate_element">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element ref="xupdate_attribute" minOccurs="0"/>
</xs:sequence>
<xs:attribute ref="name" use="required"/>
</xs:complexType>
</xs:element>
<!-- DEFINIÇÃO DO ATTRIBUTE -->
<xs:element name="xupdate_attribute">
<xs:complexType mixed="true">
```

```
<xs:attribute ref="name" use="required"/>
</xs:complexType>
</xs:element>
<!-- DEFINIÇÕES -->
<xs:attribute name="select" type="xs:string"/>
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="old" type="xs:string" fixed="TRUE"/>
<xs:attribute name="name_attr" type="xs:string"/>
<xs:element name="xdcl_on" type="xs:string"/>
<xs:attribute name="type_condition" type="tip_con"/>
<xs:attribute name="name_element" type="xs:string"/>
<!-- Definicao dos operadores lógicos -->
<xs:simpleType name="ope_log">
<xs:restriction base="xs:string">
<xs:enumeration value="="/>
<xs:enumeration value=">"/>
<xs:enumeration value="&lt;"/>
<xs:enumeration value=">="/>
<xs:enumeration value="&lt;="/>
<xs:enumeration value="&lt;>"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="tip_con">
<xs:restriction base="xs:string">
<xs:enumeration value="ExpDt"/>
<xs:enumeration value="ExpNr"/>
<xs:enumeration value="ExpSt"/>
<xs:enumeration value="ExpFuncNr"/>
<xs:enumeration value="ElementDt"/>
<xs:enumeration value="ElementNr"/>
<xs:enumeration value="ElementSt"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="tip_old">
```

```
<xs:restriction base="xs:string">
<xs:enumeration value="ElementDt"/>
<xs:enumeration value="ElementNr"/>
<xs:enumeration value="ElementSt"/>
</xs:restriction>
</xs:simpleType>
<!-- Definicao dos tipos dos inserts -->
<xs:simpleType name="tip_ins">
<xs:restriction base="xs:string">
<xs:enumeration value="Element"/>
<xs:enumeration value="Attribute"/>
</xs:restriction>
</xs:simpleType>
<!-- definição do conteúdo do atributo type_place -->
<xs:simpleType name="tip_pla">
<xs:restriction base="xs:string">
<xs:enumeration value="Append"/>
<xs:enumeration value="Before"/>
</xs:restriction>
</xs:simpleType>
<!-- Definicao dos tipos dos updates -->
<xs:simpleType name="tip_up">
<xs:restriction base="xs:string">
<xs:enumeration value="Constant"/>
<xs:enumeration value="Function"/>
</xs:restriction>
</xs:simpleType>
</xs:schema>
```