# XDocValidator: Uma Interface Gráfica Para Atualização e Validação Incremental de Documentos XML

**Fabio Pasquali**[1*], **Denio Duarte**[2]

[1]Universidade Comunitária Regional de Chapecó / Unochapecó – CETEC

[2]Universidade do Estado de Santa Catarina (UDESC) – CCT

fabio_p@unochapeco.edu.br, denio@joinville.udesc.br

***Resumo.*** *Este artigo apresenta uma ferramenta que implementa uma interface gráfica para atualização de documentos XML baseada no banco de dados $eXist$. Esta interface também garantirá que documentos anteriormente válidos em relação a um esquema mantenham a validade após as atualizações. A verificação da validação é feita apoiada na abordagem incremental.*

## 1. Introdução

A linguagem de marcação XML tornou-se um padrão *de facto* na descrição de dados semi-estruturados que são compartilhados via *Internet*. Sua simplicidade permite que usuários possam criar suas próprias marcações, dando assim significado aos dados. Esta flexibilidade torna útil o uso de esquemas que definem a estrutura que um documento XML deve possuir para que ele seja manipulado correta e eficientemente por uma aplicação. Logo, é necessário verificar se um documento XML é válido em relação a seu esquema quando for atualizado.

Existem duas abordagens de validação de documentos XML. A validação *from scratch* é a mais simples pois analisa um documento na integra a cada atualização, fato que a torna pouco atrativa, pois acarreta em perda de performance, principalmente ao lidar com documentos grandes. Outra abordagem é a validação incremental, que valida apenas as partes modificadas de um documento XML após sua atualização, sendo, portanto, mais eficiente. Esta forma de validação, entretanto, não tem sido satisfatoriamente explorada [Balmin et al. 2004].

Além disso, há a necessidade das aplicações em manipular uma grande quantidade de documentos XML, tornando necessária uma forma eficiente de gerenciamento destes documentos em bancos de dados. É neste escopo que aparecem os bancos de dados XML nativos (BDXML) – SGBDs que armazenam dados apenas no formato XML, e são otimizados para esta finalidade. Após um estudo comparativo realizado entre alguns BDXML [Pasquali and Duarte 2009], o BDXML $eXist$ [eXist 2008] foi escolhido para servir como camada de armazenamento de dados para o projeto proposto. O $eXist$ [eXist 2008] possui código aberto implementado em Java. É um banco de dados facilmente integrável com aplicações que lidam com XML, além de possuir vários recursos e APIs para desenvolvedores. Implementações das linguagens XPath, XQuery e XUpdate são disponibilizadas para a realização de consultas e atualizações. O banco também possui uma interface gráfica, para executar operações administrativas

*Bolsista FAPESC - Prémio Mérito Universitário Catarinense 2009

como gerenciar usuários, realizar $backup$ de dados e configurações de segurança. Entretanto, o $eXist$, na versão 1.2.4, apresenta duas características negativas: ($i$) não possui uma interface gráfica amigável que permita com que usuários não-especialistas em programação possam manipular documentos XML, necessidade cada vez mais crescente [Costello and Schneider 2000, Roddick et al. 2000], e ($ii$) não garante a integridade de um documento em relação ao seu esquema após uma atualização.

O sistema $XDocValidator$ é uma proposta de ambiente onde um usuário pode carregar um documento XML armazenado no banco $eXist$ e visualizá-lo em forma de árvore. A partir dessa representação gráfica, o usuário poderá expressar alterações que serão refletidas no documento em questão. Estas alterações podem ser aplicadas a elemento e atributos, e englobam: criação, renomeação, exclusão, modificação de conteúdo textual. O sistema também fará verificações da validade do documento afetado. Esta última será realizada através da abordagem de validação incremental. Uma outra contribuição deste projeto é uma proposta para transformação de um esquema escrito em $XML$-$Schema$ em um modelo para a validação de documentos. Esse modelo é baseado em autômatos de árvore e é armazenado como um documento XML.

## 2. Arquitetura do Sistema

A arquitetura do sistema $XDocValidator$ foi desenvolvida com a linguagem de programação $Java$, e a IDE utilizada durante a implementação foi o $Eclipse\ Europa$. O sistema apresenta um documento XML armazenado no $eXist$ sob a forma de árvore gráfica, sendo que os nós-folha correspondem às informações enquanto os demais nós representam a estrutura (elemento e atributos). Uma implementação do $parser\ DOM$ é utilizada para ler o documento do banco e mantê-lo em memória. Durante esta leitura a representação de árvore gráfica é criada, sendo que a partir dela é possível expressar modificações a serem refletidas no documento carregado. As modificações englobam criação, renomeação, exclusão e modificação do conteúdo textual dos nós.

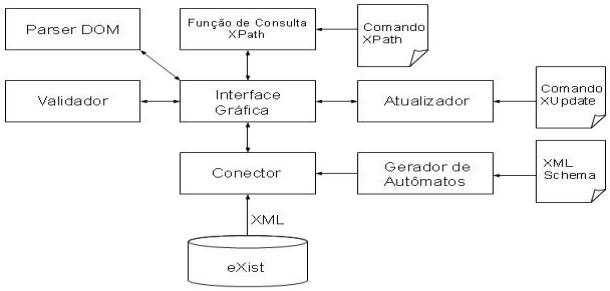


**Figura 1. Arquitetura do sistema XDocValidator.**

Para confirmar as atualizações realizadas no disco (no banco de dados), é necessário que o documento carregado em memória continue sendo válido caso estiver associado a um esquema. Este processo de validação é incremental, pois consiste em analisar as partes atingidas pelas últimas atualizações (efetuadas desde quando o documento foi carregado ou salvo com sucesso) através de um autômato previamente criado a partir do esquema. Se este processo resultar em sucesso (ou se não existir esquema associado), o documento do banco poderá ser substituído pela sua cópia atualizada. A Figura 1 ilustra a arquitetura do $XDocValidator$ que é explicada a seguir.

**Conector**: é o componente responsável pela comunicação entre a aplicação e o banco

de dados. Possui funções para estabelecer conexão, recuperar e sobrescrever documentos XML. O componente ainda encapsula a gerência de um arquivo auxiliar chamado $associacoes.xml$, que identifica os relacionamentos entre documentos e esquemas. Utiliza uma implementação da API $XML{:}DB$ para a interação com eXist.

**Atualizador**: Componente que aplica as atualizações solicitadas no documento XML. O Atualizador também se encarrega de marcar os elementos comprometidos pela modificação para que possam ser analisados pelo componente Validador durante o processo de validação incremental do documento.

**Validador**: Este componente é responsável pela validação incremental de um documento XML e funciona da seguinte forma: quando um documento XML $d$ é carregado, o sistema recupera, do banco $eXist$, o arquivo XML $A$ que contém o autômato construído a partir do esquema associado a $d$. Esta recuperação envolve uma consulta em um arquivo auxiliar que contém a associação de cada autômato já criado com os documentos XML reconhecidos por ele. O Validador utiliza $A$ para criar estruturas de dados que serão utilizadas na validação incremental do documento carregado. Estas estruturas incluem um conjunto de regras de transição, sendo que cada uma destas regras está associada a uma declaração de elemento. Uma regra de transição possui a estrutura $q \rightarrow A_{ob}\ A_{op}\ n\ R$, sendo que $q$ é um estado, $A_{ob}$ é um conjunto de atributos obrigatórios, $A_{op}$ é um conjunto de atributos opcionais, $n$ é uma etiqueta e $R$ é uma expressão regular de estados. Um elemento $e$, que possui um conjunto de atributos $\Lambda$, é associado a uma regra $q \rightarrow A_{ob}\ A_{op}\ n\ R$ se e somente se: ($i$) a etiqueta de $e$ é igual a $n$, ($ii$) $A_{ob} \subset \Lambda$, ($iii$) $(\Lambda - A_{ob}) \subset A_{op}$, e ($iv$) a concatenação dos estados associados aos filhos de $e$ pertence à linguagem definida por $R$. Um documento XML é válido se cada um dos seus elementos pode ser reconhecido por uma regra de transição. A validação incremental otimiza este processo, reconhecendo apenas os elemento comprometidos pela atualização.

Pelo fato do autômato ser gerado a partir de uma gramática $single\text{–}type$ (veja [Murata et al. 2005]), é possível que mais de uma regra de transição tenha a mesma etiqueta $n$. Para evitar ambigüidade de reconhecimento, o processo consulta um escopo para identificar a regra de transição correta. Mais especificamente, este escopo é o conjunto de regras cujos estados que ocorrem na expressão regular $R$ da regra associada ao pai do elemento em questão.

O processo de validação se divide em duas fases: ($i$) verificação de estrutura, e ($ii$) verificação das chaves primárias e estrangeiras (especificadas pelo $XML\text{-}Schema$). Na primeira etapa, o componente analisa apenas as partes do documento XML comprometidas pela modificação, ou seja, os elementos pais das subárvores atualizadas. O seguinte exemplo ilustra esta verificação:

```
<A> <B atrib1="valor1"/> <C> <D atrib2="valor2"><E>texto</E></D></C> </A>
```

Supondo que o usuário, ao fazer uma atualização no documento acima, excluiu o elemento <E>. Logo, este documento precisa ser validado em relação ao seu esquema antes de poder ser armazenado no banco. A validação incremental analisará somente a integridade do elemento <D> (pai de <E>), e não de todo o documento (elemento <A>). Esta verificação é feita combinando o nó D com a sua respectiva regra de transição do autômato. Se resultar em sucesso, então a alteração no documento é mantida.

Após a validação estrutural, o componente realizará a verificação da unicidade

do valor das chaves primárias, além da integridade referencial das chaves estrangeiras contidas na parte alterada do documento. A análise das chaves é o único tipo de validação de dados realizado. Isto significa que nenhuma verificação será feita no conteúdo de elementos textuais e atributos, pois ambos serão interpretados apenas como cadeias de caracteres, independente dos tipos de dados e restrições (exemplo: o valor do atributo $X$ deve ser um inteiro entre 2 e 9).

**Gerador de Autômatos**: Componente responsável pela transformação de um esquema - escrito em $XML\text{-}Schema$ - em um autômato de árvore descendente que reconhece os documentos XML associados. Esta transformação é realizada com base nas declarações de elementos, tipos e atributos. Estas regras são armazenadas no banco de dados em forma de documento XML, para serem recuperadas e repassadas para o componente Validador, que as usará na construção de um modelo computacional para a validação de documentos.

O analisador sintático de esquemas foi implementado para reconhecer apenas um subconjunto da linguagem $XML\text{-}Schema$ que corresponde a uma gramática $single\text{-}type$. Este subconjunto engloba as seguintes declarações: <xs:schema>, <xs:element>, <xs:attribute>, <xs:attributeGroup>, <xs:group>, <xs:sequence>, <xs:choice>, <xs:all>, <xs:complexType>, <xs:complexContent>, <xs:restriction>, <xs:extension>, <xs:key>, <xs:keyref>, <xs:field>, <xs:selector> e <xs:unique>.

Os exemplos a seguir apresentam como um autômato é construído a partir de instâncias de $XML\text{-}Schema$. São abordadas as declarações dos aspectos considerados mais importantes da linguagem $XML\text{-}Schema$ para construção de esquemas.

```
 1 <xs:complexType name="publicacao">
 2  <xs:sequence>
 3   <xs:element name="titulo" type="xs:string"/>
 4   <xs:element name="autor" minOccurs="2" maxOccurs="unbounded">
 5    <xs:complexType>
 6     <xs:attribute name="nome" type="xs:string" use="required"/>
 7     <xs:attribute name="email" type="xs:string" use="required"/>
 8    </xs:complexType>
 9   </xs:element>
10  </xs:sequence>
11 </xs:complexType>
12 <xs:element name="livro" type="publicacao"/>
```

A definição acima indica que o elemento $<livro>$ (linha 12) terá a estrutura definida pela cláusula *<xs:complexType>* com o atributo $name$ igual a $publicacao$, e pode ser convertida nas seguintes regras de produção: $Livro \rightarrow$livro$(Titulo, Autor^{+})$, $Titulo \rightarrow titulo(PCDATA)$, $Autor \rightarrow${nome,email}{} $autor(\epsilon)$. Perceba que a declaração de elemento pode possuir alguns atributos além de $name$. O atributo $type$, por exemplo, faz referência a uma estrutura declarada exteriormente ($simpleType$ ou $complexType$), ou a um tipo de dado textual (exemplo: $xs$:$string$ na linha 3). $minOccurs$ e $maxOccurs$ definem, respectivamente, o número mínimo e máximo de ocorrências de um elemento (linha 4). No caso da declaração do elemento $autor$, a transformação generalizou para o fechamento positivo de $Kleene$ ($^{+}$), pois o valor de $minOccurs$ é maior ou igual a 1 (caso contrário seria $^{*}$). $sequence$ estabelece que os elementos declarados em seu interior devem aparecer na mesma seqüência ($<titulo>$ seguido de $<autor>$, linhas 2 a 10). Caso $<xs$:$sequence>$ fosse substituído por $<xs$:$choice>$ (apenas um dos elementos declarados deve constar no elemento do tipo $publicacao$, então a regra $Livro$ seria: $Livro \rightarrow (Titulo \mid Autor^{+})$. A declaração de $<autor>$ indica que o elemento é vazio, além de possuir duas declarações de atributos

que, por serem obrigatórios, compõe o conjunto $A_{ob}$ da regra resultante ($A_{op}$ é um conjunto vazio). Este formalismo para tratamento de atributos é inspirado na abordagem proposta por [Bouchou et al. 2003].

Um outro tipo de declaração é a *Derivação*, utilizada para definir tipos complexos baseando-se em outros tipos complexos. Esta derivação pode ser feita por extensão ou restrição. O exemplo a seguir ilustra uma extensão do tipo complexo $Publicacao$ que acrescenta, em sua estrutura, um elemento textual universidade.

```
<xs:complexType name="artigo">
  <xs:complexContent>
    <xs:extension base="publicacao">
      <xs:sequence>
        <xs:element name="universidade" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Esta derivação pode ser transformada na seguinte regra: $Artigo \rightarrow artigo(Publicacao, Universidade)$, $Universidade \rightarrow universidade(PCDATA)$. O $XML\text{-}Schema$ ainda possui muitas outras funcionalidades que podem ser transformadas de forma semelhante às apresentadas, como: Grupo de Substituição, Tipo Abstrato, Grupo-Modelo, entre outras. Por razão de espaço, as mesmas não serão descritas aqui.

**Interface gráfica:** A partir desta interface (desenvolvida com a API $Swing$), o usuário pode navegar na hierarquia de coleções de documentos contidos no banco eXist. Ao escolher um documento, sua estrutura é mostrada graficamente (Figura 2). É fornecido, ainda, um meio de realizar consultas XPath no documento carregado. Os resultados das consultas também são mostrados em forma de árvores.

Ao selecionar um nó da árvore, opções de atualização são exibidas, variando para atributos, elementos textuais e estruturais, permitindo que o documento seja manipulado de forma bastante intuitiva, sem a necessidade do usuário possuir conhecimentos em programação. Atualizações no documento também podem ser feitas através de comandos escritos na linguagem XUpdate. Dessa forma é possível expressar atualizações mais complexas (como, por exemplo, modificações condicionais ou que atinjam vários elementos e/ou atributos). O menu da interface ainda disponibiliza opções para validar o documento, salvar alterações no banco e associar um esquema ao documento carregado. Quando esta última operação é executada, o sistema solicita ao componente Gerador de Autômatos a criação de um autômato de árvore a partir de um $XML\text{-}Schema$ escolhido. Se o documento for reconhecido pelo autômato, então este é armazenado no banco eXist, e a associação entre o documento e o esquema é registrada.

## 3. Conclusão

Este artigo apresentou $XDocValidator$, uma interface que auxilia usuários a manipularem documentos XML de um banco de dados $eXist$. Nesta interface, um documento (ou o resultado de uma consulta $XPath$) é representado graficamente em forma de árvore, sendo que seus nós equivalem aos elementos e atributos. Para cada um destes nós, é disponibilizado um conjunto de opções para a sua atualização, englobando ações de criação, remoção, renomeação e alteração de conteúdo. A interface $XDocValidator$

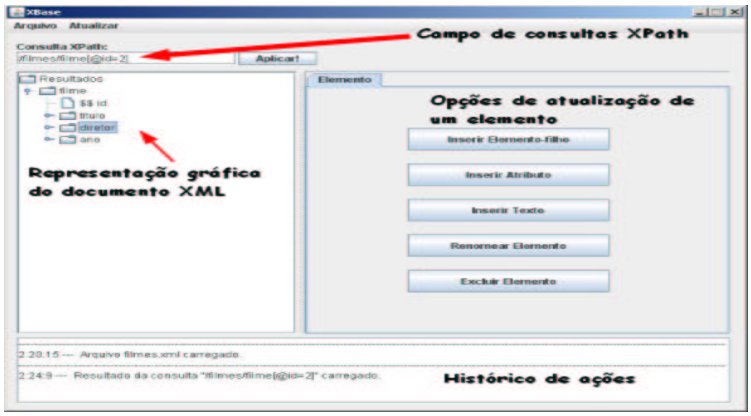


**Figura 2. Interface gráfica.**

também garante que os documentos XML, que são associados a um esquema (escrito em $XML\text{-}Schema$) continuem válidos após uma tentativa persistir as atualizações no documento XML do banco. Esta verificação é feita através da abordagem de validação incremental.

O sistema $XDocValidator$ permite que a manipulação de um documento XML seja mais intuitiva, especialmente para usuários não-especialistas em computação, que não possuem experiência com programação, pois, ao clicar nos nós da árvore, uma expressão $XPath$ equivalente é construída automaticamente. A partir dessa expressão, o usuário pode realizar as atualizações necessárias. O processo de validação XML, por ser incremental, analisa apenas as partes do documento modificadas pelas atualizações, o que poder melhorar a eficiência desse processo, principalmente ao lidar com documentos XML volumosos e com estrutura complexa. Segundo o conhecimento dos autores, não existe nenhuma ferramenta livre que permita a validação incremental de documentos XML similar ao $XDocValidator$.

## Referências

Balmin, A., Parapakonstantinou, Y., and Vianu, V. (2004). Incremental validation of XML documents. In *International Conference on Database Theory*.

Bouchou, B., Duarte, D., Halfeld Ferrari, M., and Laurent, D. (2003). Extending tree automata to model XML validation under element and attribute constraints. In *ICEIS*.

Costello, R. and Schneider, J. C. (2000). Challenge of XML schemas - schema evolution. In *Proceedings of XML Europe*.

eXist (2008). Open source native XML database. In *http://exist.sourceforge.net*.

Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of XML schema language using formal language theory. *ACM TOIT*, 5(4):660–704.

Pasquali, F. and Duarte, D. (2009). Estudo comparativo dos BDXML eXist e Xindice. In *V Escola Regional de Banco de Dados (ERBD)*.

Roddick et al. (2000). Evolution and change in data management - issues and directions. *SIGMOD Record*, 29(1):21–25.