Um Estudo das Abordagens para Extração de Esquemas XML

Geomar A. Schreiner[1], **Denio Duarte**[1]

[1]Universidade Federal da Fronteira Sul - UFFS
Campus Chapecó

geomarschreiner@gmail.com, duarte@uffs.edu.br

Abstract. *XML (Extensible Markup Language) is a mark-up language that models semistructured data. XML lets the user define his own customized markup language for different document classes. Users can create XML documents freely. This flexibility has a drawback: processing XML documents can be computational onerous. In this context, schemas play an important role. This work presents a survey of three XML schema extraction approaches. We use a running example to show how the approaches work.*

Resumo. *Documentos XML são caracterizados por possuírem a estrutura armazenada junto com os dados tornando a sua aplicação bastante flexível. Porém, o efeito colateral dessa flexibilidade é que o processamento dos documentos se torna custoso computacionalmente. Isso pode ser resolvido associando um esquema ao documento permitindo que a aplicação conheça 'a priori' a estrutura do documento a ser processado. Este trabalho tem como objetivo estudar algumas destas ferramentas apresentando as abordagens utilizadas por elas e uma breve comparação entre as mesmas.*

1. Introdução

Documentos XML são caracterizados por não possuirem tipos associados as suas estruturas, ou seja, as aplicações e/ou usuários podem criar e alterar a estrutura dos dados e os dados livremente. Essa característica é propria da proposta da XML: modelar dados semi-estruturados. Tais dados não possuem tipos associados, são auto-descritivos e irregulares, sem distinção entre suas estruturas e os dados propriamente ditos.

Ao *tipar* instâncias XML (documentos XML), as aplicações e/ou usuários não podem mais livremente criar e modificar tais instâncias. A atualização dessas instâncias deve respeitar as restrições impostas pelo tipo associado (esquema). Documentos XML associados a esquemas são chamados válidos quando respeitam as regras impostas pelos esquemas. A validade não é característica obrigatória para os documentos XML. Assim, um documento XML associado a um esquema e que não respeita o mesmo continua sendo um documento bem formado sendo tratado pelos processadores de documentos XML sem problemas.

Se um esquema restringe a liberdade de criação e atualização de documentos XML por que, então, utilizá-los? Apesar da flexibilidade dos dados semi-estruturados, esquemas são úteis pois: (i) descrevem os dados e auxiliam a consulta sobre os mesmos, (ii) permitem, também, otimizar tais consultas, (iii) são base para abordagens eficientes para armazenamento dos dados, (iv) permitem que projetistas das aplicações descrevam a estrutura dos documentos, criando classes de documentos, e (v) facilitam o processo de transformação dos documentos para diferentes formatos (*i.e.,*, dados relacionais).

A maioria dos documentos encontrados na WEB não possuem um esquema associado a eles ou o documento não respeita seu esquema [Barbosa et al. 2005]. O que torna muito pertinente que através de uma determinada coleção de documentos XML (mesmo que unitária) seja possível a extração de um esquema que descreva a estrutura da coleção. Segundo [Garofalakis et al. 2000], seres humanos fazem a melhor inferência possível de um esquema para determinada coleção de documentos XML. Evidentemente esta abordagem torna-se impraticável para documentos extensos ou coleções com muitos documentos. Sendo assim, é necessária uma ferramenta que realize a extração de forma automática ou semi-automática.

Diversas abordagens foram propostas para solucionar este problema: XTRACT [Garofalakis et al. 2000], Inferência Gramatical [Chidlovskii 2001], Min&Chung [Min et al. 2003], XStruct [Hegewald et al. 2006] e XTLSM [Amiel et al. 2008]. Este trabalho apresenta três das abordagens citadas anteriormente: XTRACT e Inferêncial Gramatical por terem o maior número de citações segundo ScholarGoogle© e XTLSM por ser o mais recente.

A metodologia de apresentação segue estes passos: o método é brevemente descrito com o seu pseudo-código e, em seguida, um exemplo do funcionamento é apresentado. Todos os métodos estudados utilizam uma coleção unitária de documentos XML como entrada para fins de simplificação.

2. Abordagens Estudadas para Extração de Esquemas

O problema de extração de esquema pode ser definido como: dada um coleção de documentos XML (possivelmente unitária) com alguma semelhança estrutural, inferir um esquema que descreva a coleção de entrada. Esta seção apresentará algumas abordagens para extração de esquemas XML a partir de documentos XML e utiliza como exemplo de estudo de caso o documento XML X apresentado na Figura 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<motos> <moto cilindradas = "250">
          <marca>Yamaha</marca> <modelo>Fazer</modelo>
          <opcional>Freio Disco</opcional>
          <revisao>  <km>15000</km><valor>350,00</valor>
                     <km>35000</km><valor>600,00</valor>  </revisao>
        </moto>
        <moto>
         <marca>Honda</marca>  <modelo>CBX</modelo>
         <opcional>Abs</opcional> <opcional>Alerta</opcional>
         <revisao>
            <km>1000</km><valor>17,00</valor>
            <km>5000</km><valor>35,00</valor>
            <km>10000</km><valor>320,00</valor>  </revisao>
       </moto>
       <moto cilindradas="125">
        <modelo>Biz</modelo> <ano_modelo>2013</ano_modelo>
       </moto>
</motos>
```

Figura 1. Exemplo de documento XML

2.1. Inferência Gramatical

O método descrito em [Chidlovskii 2001] é baseado em inferência gramatical, ou seja, dado um conjunto de entrada é inferida uma gramática que deve descrever todo o conjunto de dados. Este método consiste na execução de 3 passos: *(i)* os documentos são

representados em forma de uma árvore, *(ii)* é feita a indução de uma gramática com base na árvore gerada no primeiro passo e *(iii)* a gramática é transformada em um esquema na linguagem XML-Schema (XSD). O documento X (Figura 1) é utilizado para ilustrar o funcionamento do Algoritmo 1 que apresenta o funcionamento do método.

Algoritmo 1: Inferência Gramatical

```
1  Entrada : documento XML X;
2  arvoreDerivacao ← criarArvore(X);
3  eCFG ← infereGramaticaInicial(arvoreDerivacao);
4  juncaoNaoTerminaisPorContexto(eCFG);
5  juncaoNaoTerminaisPorConteudo(eCFG);
6  foreach regra in eCFG do
7  |   foreach elemento in regra.producao do
8  |   |   if elemento.valor == "Any" then
9  |   |   |   extraiTipoPrimitivo(elemento, Entrada);
10 |   |   end
11 |   end
12 end
13 esquema = converteECFGParaXMLSchema(eCFG);
```

No primeiro passo é feita a conversão de X para uma representação em árvore (linha 2). A árvore gerada será uma árvore de derivação de uma gramática livre de contexto sem os seus símbolos não terminais. A árvore t_1 da Figura 2 representa a árvore gerada a partir de X. Para maior clareza algumas subárvores de t_1 foram omitidas. Após a conversão, o segundo passo do método é dividido em quatro etapas:

(i) Na primeira etapa é gerada uma gramática livre de contexto extendida ($eCFG$ - *extended Context Free Grammar* [Brüggemann-Klein and Wood 1998]) (linha 3). Nesta etapa, t_1 é utilizada para criar as regras da gramática. Inicialmente, uma regra denominada *Start* que representa o elemento raiz da árvore é criada. Para cada um dos elementos presentes em t_1 são criadas regras. O nome da regra será da forma A_n, sendo *n* o identificador do número da regra. As produções contidas na regra representam os filhos que o elemento pode possuir na árvore. Assim, elementos complexos (possuem filhos) tem seus valores representados por rótulos de uma nova regra e elementos simples por *Any*. Após executado o método, a variável *eCFG* (linha 3) receberá o conjunto das regras contidas na Figura 3(a). Por exemplo, o elemento *revisao* de t_1 possui filhos então é gerada a regra A_5.

(ii) Nesta etapa são feitas as junções das regras que possuem o mesmo contexto, afim de eliminar ambiguidades (linha 4). O método irá comparar as regras e a forma que estas estão sendo utilizadas e fará a junção. Levando em consideração a gramática contida na variável *eCFG* (Figura 3(a)), as regras A_1, A_2 e A_3 estão sendo usadas no mesmo contexto (regra *Start*), sendo assim as regras A_2 e A_3 serão fusionadas à regra A_1 formando uma única regra.

(iii) Em seguida, são realizadas as junções das regras verificando se possuem produções semelhantes para tentar uní-las em uma única regra. As regras selecionadas são as regras A_4 e A_5. A produção da regra A_5 é incorporada à produção A_4 concatenando-as utilizando o operador *"ou"*. Ao final desta etapa a variável *eCFG* possui apenas as regras $Start$ (regra inicial), A_1 (fundida com as regras A_1, A_2 e A_3) e A_4 (adicionada com a regra A_5).

(iv) Na quarta etapa é realizada a extração de tipos básicos dos elementos simples (linhas 6 a 12). Para realizar esta etapa as produções de todas as regras são varridas, sendo selecionados os elementos que possuem como valor o terminal *Any* (elementos simples) e passados para a função *extraiTipoPrimitivo* (linha 9) que analisa o documento de entrada e infere através de tentativa e erro um tipo para o elemento. Ao fim desta parte a variável *eCFG* possui o conteúdo apresentado na Figura 3(b)

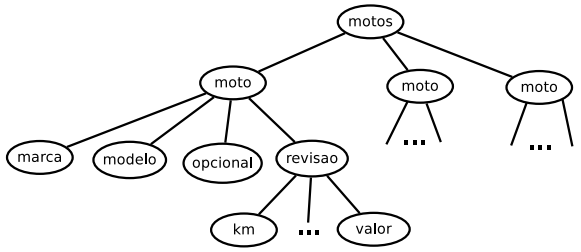

Figura 2. Árvore de derivação gerada a partir de X

O último passo do método faz, através de uma série de regras, a conversão da $eCFG$ em uma XSD. A Figura 4 apresenta o esquema gerado a partir da $eCFG$ apresentada na Figura 3(b).

(a) Inferência inicial das regras
$Start \leftarrow moto : A_1\ moto : A_2\ moto : A_3$
$A_1 \leftarrow marca : Any\ modelo : Any\ opcional : Any\ revisao : A_4$
$A_2 \leftarrow marca : Any\ modelo : Any\ opcional : Any\ opcional : Any\ revisao : A_5$
$A_3 \leftarrow modelo : Any\ ano\_modelo : Any$
$A_4 \leftarrow km : Any\ valor : Any\ km : Any\ valor : Any$
$A_5 \leftarrow km : Any\ valor : Any\ km : Any\ valor : Any\ km : Any\ valor : Any$
(b) Após a execução do segundo passo do algoritmo
$Start \leftarrow moto : A_1\ moto : A_1\ moto : A_1$
$A_1 \leftarrow marca : String\ modelo : String\ opcional : String\ revisao : A_4 \mid$
$marca : String\ modelo : String\ opcional : String\ opcional : String\ revisao : A_4 \mid modelo : String\ ano\_modelo : Int$
$A_4 \leftarrow km : Int\ valor : Float \mid km : Int\ valor : Float\ km : Int\ valor : Float$

Figura 3. Estruturas intermediárias para a construção do esquema.

2.2. XTRACT

O XTRACT [Garofalakis et al. 2000] é um método de extração que constrói uma DTD que representa a estrutura do documento XML dado como entrada. O Algoritmo 2 apresenta o seu pseudo-código e é utilizado para explicar o funcionamento do método tendo X com entrada.

O método pode ser resumido em três etapas principais: (i) o documento XML é transformado em um conjunto de sequências de símbolos: para cada elemento são geradas sequências que representam a ocorrência de seus sub-elementos, a partir dessas sequências, são construídas expressões regulares candidatas (ERc), (ii) ERc são otimizadas e simplificadas através do processo de fatoração, e (iii) do conjunto gerado é escolhida uma das candidatas para compor a DTD final, essa escolha é feita baseada no princípio do *Minimum Description Lenght* (MDL) que seleciona a candidata que melhor descreve estruturalmente a sequência no menor tamanho possível (em $bits$) [Vitanyi and Li 2000].

```
1 <?xml version="1.0"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3  <xs:element name="motos">
4    <xs:complexType>
5      <xs:element name="moto" minOccurs="1" maxOccurs="3" >
6       <xs:complexType>
7          <xs:element name="marca" type="xs:string"/>
8          <xs:element name="modelo" type="xs:string"/>
9          <xs:element name="opcional" type="xs:string" maxOccurs="2"/>
10         <xs:element name="revisao">
11            <xs:complexType>
12               <xs:sequence>
13                 <xs:element name="km" type="xs:int"/>
14                 <xs:element name="valor" type="xs:float"/>
15               </xs:sequence>
16             </xs:complexType>
17          </xs:element>
18         <xs:element name="ano_modelo" type="xs:int"/>
19     </xs:complexType>
20   </xs:element>
21  </xs:complexType>
22 </xs:element>
23 </xs:schema>
```

Figura 4. XSD criada utilizando o método de Inferência Gramatical

Inicialmente, é feita a busca dos elementos de X (linha 2). Cada *tag* de X é considerada um elemento. Sendo assim, a variável *elementos* receberá todos os elementos de X (*i.e. motos, moto, marca, modelo, opcional, revisao, km, valor, ano_modelo*). Para cada um dos elementos de *elementos* (linha 3) é executada a extração de sequências (linha 4). A extração das sequências, basicamente, é o processo de retirada dos elementos contidos em um determinado elemento. Por exemplo para o elemento *motos* a sequência correspondente é {*moto moto moto* }, pois a *tag motos* contém 3 elementos *moto*, e aparece uma vez. Assim, a variável *seqElementos* recebe {*moto moto moto*} (linha 4).

Para cada sequência de elementos de *seqElementos* serão construídas ERc a elementos da DTD (linhas 5 a 11). Essa etapa do método é chamada de generalização. Inicialmente, a sequência é adicionada ao vetor de candidatas (linha 6), então o método *geraCandidataE* é invocado 3 vezes variando o parâmetro *r* em 2, 3 e 4 que representa o número de vezes que um símbolo pode se repetir. *geraCandidataE* varre a sequência com o intuito de criar expressões regulares que correspondam a estrutura do elemento candidato. Assim, a primeira candidata c_1 a ser gerada ($r = 2$) será *moto**. c_1 é passada para o método *geraCandidatasOu* que tentará gerar outras candidatas baseadas em c_1, utilizando o operador *ou*. Todas as candidatas geradas são adicionadas ao vetor $candidatas$. Passando c_1 como parâmetro para o método, será obtida como resposta a candidata a própria c_1 pois não é possível obter mais candidatas utilizando o operador *ou*. Outra candidata gerada pelo método *geraCandidataE* (com $r = 3$) é *moto moto**, que ao passar pelo método *geraCandidataOu* não sofreria alterações. $r = 4$ não gera nenhuma nova candidata. Sendo assim, ao final da etapa de generalização para a sequência { *moto moto moto*} seriam geradas as candidatas: {*moto moto*, moto**}. Para a maior parte das candidatas geradas a partir dos elementos de X, não serão feitas alterações ao passar pelo método *geraCandidataOu* exceto a candidata *(km valor)**, onde o método retornará as candidatas *(km valor)** e *(km | valor)**.

A próxima etapa a ser executada é a fatoração onde as candidatas geradas são submetidas a processos de simplificação (linha 12). Basicamente, o sistema de fatoração

Algoritmo 2: XTRACT

```
1  Entrada : documento XML X;
2  elementos ← extraiElementos(X);
3  foreach elemento in elementos do
4      seqElementos ← extraiSeqElementos(elemento);
5      foreach sequencia in seqElementos do
6          adiciona sequencia em candidatas;
7          for r in (2,3,4) do
8              geraCandidataE(sequencia, r, candidata);
9              geraCandidatasOu(candidata,candidatas);
10         end
11     end
12     fatora(candidatas);
13     menor ← MDL(candidatas[0]);
14     DTD ← candidatas[0]; i ← 1;
15     while i++ < tamanho(candidatas) do
16         if MDL(candidatas[i]) < menor then
17             menor ← MDL(candidatas[i]); DTD ← candidatas[i];
18         end
19     end
20     adiciona DTD em DTDFinal;
21 end
22 return DTDFinal
```

varre todas as candidatas e simplificá-as, podendo até criar, unindo ou construindo, novas candidatas. Passando para o módulo de fatoração as candidatas obtidas anteriormente, *moto moto** será eliminada pois após fatorada ficará igual a candidata *moto**. Ao fim desta etapa, a variável *candidatas* terá somente a candidata *moto**. Para a maior parte dos conjuntos de candidatas que serão geradas ao longo da execução do exemplo, a etapa de fatoração não fará grandes mudanças, exceto quando esta receber as candidatas { *marca modelo, modelo ano_modelo* }, que resultará a candidata *(marca | modelo | ano_modelo)*.

A última etapa a ser executada é a eleição da candidata que participará dos elementos que constituem a DTD final. Esta eleição é baseada no princípio do MDL. Como a variável *candidatas* contém apenas uma candidata então está será a eleita para fazer parte da DTD final. Sendo assim, será calculado o MDL para a primeira candidata que estiver no vetor de candidatas (linha 13 - *i.e. moto**, que é 32 (pois moto* ocupa 5 bytes para ser armazenada mais 27 bytes dele convertido em elemento da DTD). Como não existem mais candidatas, a candidata é descrita em termos de DTD e é adicionada a DTD final (linhas 14 e 20 respectivamente). O elemento resultante deste passo é apresentado na Figura 5 (linha 2). Para a maior parte dos elementos de X apenas uma candidata será passada para o módulo do MDL. Exceto para as candidatas *(km valor)*, (km | valor)** para o elemento *revisao*. A eleita entre essas será *(km valor)** pois possui um custo de MDL menor que sua concorrente.

O processo é repetido para todos os elementos que foram extraídos do documento de entrada. Após a execução de todas as etapas para todos os elementos, a DTD obtida será a apresentada pela Figura 5. Percebe-se que o método faz a extração de um esquema que descreve a estrutura do documento XML utilizado como entrada, porém não se preocupa com a extração de atributos e de tipos básicos dos elementos.

```
<!DOCTYPE motos[
<!ELEMENT motos(moto*)>
<!ELEMENT moto((marca|modelo|opcional|revisao|ano_modelo)*)>
<!ELEMENT marca(#PCDATA)>
<!ELEMENT modelo(#PCDATA)>
<!ELEMENT opcional(#PCDATA)>
<!ELEMENT revisao(km, valor)>
<!ELEMENT km(#PCDATA)>
<!ELEMENT valor(#PCDATA)>
<!ELEMENT ano_modelo(#PCDATA)> ]>
```

Figura 5. DTD criada utilizando o Xtract

2.3. XTLSM

O método apresentado em [Amiel et al. 2008] é baseada em um novo tipo de máquina de estado a TLSM (*two layer state machine*) usada para validar gramáticas de árvore. Cada estado da TLSM contém uma máquina de estados regular que descreve os filhos de um elemento da coleção de documentos XML. O método propõe 4 etapas para a extração do esquema: (i) é feita a criação de um reconhecedor de árvores de prefixos (PTA - *prefix tree acceptor*) a partir de um documento XML, (ii) a PTA é convertida em uma máquina de estados combinando árvores semelhantes, (iii) através da máquina de estados criada na etapa anterior é criada uma TLSM, e (iv) a TLSM é convertida para uma XSD. O Algoritmo 3 apresenta um pseudocódigo do método e é utilizado na aplicação do estudo de caso utilizando X como entrada.

Algoritmo 3: XTLSM

```
1  Entrada : documento XML X;
2  PTA ← criaPTA(X, X.root);
3  fsm ← generaliza(PTA, X);
4  foreach estado in fsm do
5      estadoInterno ← novo estado inicial;
6      foreach nodo in PTA do
7          filho ← filhoDireita(nodo);
8          while filho do
9              if (estadoInterno, filho) não mapeado then
10                 if ∃ filho transição na TLSM then
11                     addTLSMtransição (estadoInterno, filho);
12                 else
13                     criaEstadoInterno (e, filho);
14                     estadoInterno ← e;
15                 end
16                 adicionaMapeado (estadoInterno, filho);
17             end
18             filho ← irmaoEsquerda(filho);
19         end
20     end
21 end
22 xsd ← converteTLSMParaXSD(tlsm);
```

Inicialmente, o algoritmo transforma X em uma PTA (Figura 6). Para tal, X é visto como uma árvore (linha 2). O método *criaPTA* recebe dois parâmetros: X é a raiz de X ($i.e.$, $motos$). Ao fim desse passo a variável *PTA* armazena a PTA gerada (Figura 6.

Onde as transições são rotuladas pelo nome do elemento, os círculos representam estados e os círculos duplos representam estados de reconhecimento.

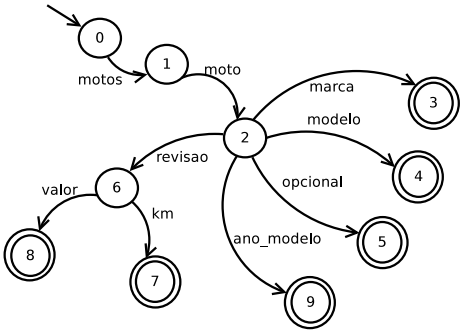

Figura 6. PTA gerada a partir do documento X

O algoritmo então, transforma a PTA em uma máquina de estados (linha 3). Essa transformação é feita realizando a junção de subárvores contidas na PTA levando em consideração suas semelhanças e contexto. Como a PTA gerada pelo exemplo não possui subárvores semelhantes, ela não sofrerá mudanças sendo somente transportada da variável *PTA* para *fsm* (linha 3). Sendo assim, ao fim do processo a máquina gerada será a mesma apresentada pela Figura 6.

O próximo passo a ser executado cria a TLSM (linhas 4 a 21). Durante este passo, a máquina de estados criada será transformada em uma TLSM. A TLSM se diferencia de uma máquina de estados normal por possuir estados com máquinas de estados internas. O XTLSM faz uso das máquinas de estado internas para realizar a validação dos filhos de um elemento presente no documento.

Para criar a TLSM, é feita uma análise de todos os estados presentes na máquina de estados existente (*fsm* - linha 4) em conjunto com os nodos presentes no documento de entrada ($i.e.$, X). Sendo assim, em cada estado da máquina é verificado se o nodo da transição possui filhos, caso existam, é criada uma nova camada, ou seja, uma máquina de estados interna e são adicionadas as transições que validam a ordem em que os filhos deverão aparecer. Seguindo a execução do exemplo, o estado selecionado será o estado inicial com o primeiro nodo (*motos*). Na linha 7, a variável *filho* recebe o filho mais a direita do elemento *motos* ($i.e.$, $moto$). Após verificado que o *estadoInterno* (agora um estado inicial) com o *filho* (*moto*) ainda não foi mapeado - linha 9 - e que não existe um estado para o filho na TLSM (linha 10), é criado um novo estado que é alvo de uma transição do estado *estadoInterno* pelo elemento *filho*, e o novo valor do *estadoInterno* é o estado criado (linhas 13, 14 e 16). Então, é selecionado o irmão a esquerda do *filho*, que é novamente *moto*. É criada uma transição para o estado *estadoInterno* por moto (linha 11). As camadas da TLSM geradas por esses passos estão representadas na Figura 7. As três máquinas geradas (Figura 7 (1), (2) e (3)) são utilizadas para criar as regras do esquema que representa X.

A última etapa do método consiste em transformar a TLSM em uma XSD (linha 22). Para isso, são propostos alguns algoritmos de redução da TLSM que simplificam e reduzem a TLSM gerada, então através de força bruta a TLSM é transformada em uma XSD. A XSD gerada é muito semelhante a XSD gerada pelo método de Inferência Gramatical (Figura 4). A única diferença entre as XSDs será nas linhas 5 e 9 (Figura 4)

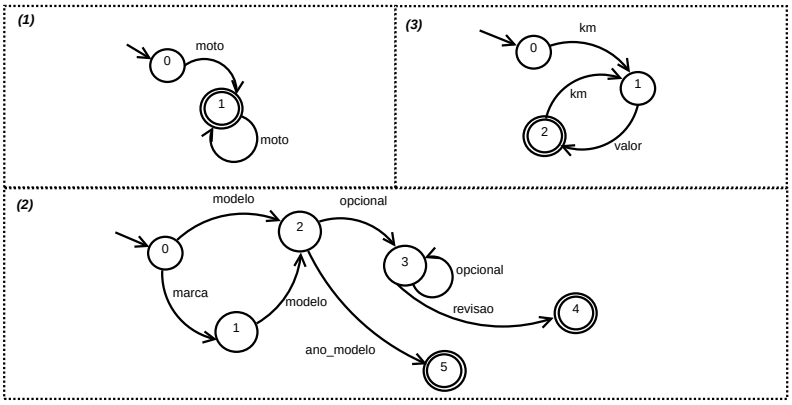

Figura 7. Camadas internas criadas a partir da TLSM criada no estudo de caso.

onde o atributo *maxOccurs* que apresenta os valores 3 e 2 respectivamente, serão alterados para *unbounded*, já que o XTLSM não limita o número de repetições máximas de um elemento.

2.4. Considerações Finais

As abordagens apresentadas realizam a extração da estrutura de um documento XML, ou seja, um esquema que descreve a estrutura de uma coleção de documentos XML. A Tabela 1 apresenta algumas das características dos métodos apresentados. Duas das abordagens (*Inferência Gramatical* e *XTRACT*) criam uma gramática que representa a estrutura do documento e a converte para uma XSD ou para DTD (coluna *Rep. Intermediária*). Já o XTLSM cria uma máquina de estados multinível que contém a estrutura do documento, então, através de força bruta converte essa máquina em uma XSD.

Apesar das abordagens de *Inferência Gramatical* e *XTRACT* possuírem uma semelhança (ambas extraem um gramática que armazena a estrutura do documento) a forma que realizam a extração é distinta. O XTRACT extrai para cada elemento presente no documento de entrada uma série de sequências, para cada sequência é criada uma regra. O conjunto de regras é submetido a uma série de processos que irão simplificá-las e então uma destas é escolhida para fazer parte de gramática que representa todo o documento. Em contrapartida, o método de *Inferência Gramatical* interpreta o documento em forma de árvore e infere para a árvore uma eCFG. A eCFG gerada é submetia a um processo de simplificação que leva em consideração o contexto e o conteúdo das regras para gerar uma gramática mínima.

O XTLSM propõe uma abordagem muito diferente das demais apresentadas. Apesar das demais representarem a estrutura do documento de entrada através de algum tipo de gramática, o XLTSM usa uma máquina de estados multinível para esta tarefa. Assim, o XTLSM inicialmente realiza a extração de uma PTA que é transformada em uma máquina de estados. A máquina de estados gerada apresenta apenas a hierarquia do documento. Então ela é transformada em uma TLSM onde são adicionados estados internos que validam a ordem em que os elementos devem aparecer no documento.

Pela coluna *Esquemas*, percebe-se que todas as abordagens transformam as estruturas intermediárias em linguagens de esquemas XML (duas delas em XSD). Percebe-se também que nenhuma das abordagens trata atributos e apenas o XTRACT não extrai os tipos, transformando todos os elementos simples em textuais ($string$).

Método	Rep. Intermediária	Esquema	Atributos	Tipos
XTRACT	ER	DTD	-	-
Inferência Gramatical	eCFG	XSD	-	X
XTLSM	Máquina de Estados	XSD	-	X

Tabela 1. Características métodos de extração de esquemas.

3. Conclusão

Este artigo apresentou um estudo de algumas abordagens para extração de esquemas XML a partir de um coleção de documentos. A extração de esquemas é importante pois conhecendo a estrutura de uma classe de documentos XML é possível otimizar consultas e armazenamento dos dados, criar índices, entre outros. Atualmente, na Web, existem várias coleções de documentos XML que não estão associadas a esquemas, por isso a importância de abordagens eficientes e corretas para a extração. Percebe-se que a maioria dos métodos apresentados utiliza conceitos de linguagens formais para a extração de esquema. Isso é esperado já que um documento XML pode ser representado por uma árvore e uma árvore pode ser gerada por uma gramática de árvore. Os leitores encontrarão em [Schreiner 2014] uma discussão mais detalhada sobre os métodos de extração de esquemas XML. Não é do conhecimento dos autores outros trabalhos que façam comparações entre métodos de extração de esquemas de documentos XML.

Espera-se que com este estudo seja possível propor novos métodos de extração de esquemas bem como aplicar os métodos aqui vistos em outros domínios similares: documentos JSON e RDF.

Referências

Amiel, S., Harrusi, S., and Averbuch, A. (2008). XML Schema Inference from Positive Example: The TLSM Approach. Technical report, Tel Aviv University.

Barbosa, D., Mignet, L., and Veltri, P. (2005). Studying the XML Web: gathering statistics from an XML sample. *World Wide Web*, pages 1–32.

Brüggemann-Klein, A. and Wood, D. (1998). Regular tree languages over non-ranked alphabets.

Chidlovskii, B. (2001). Schema Extration from XML Data: A Grammatical Inference Approach. *KRDB'01 Workshop (Knowledge Representation and Databases)*.

Garofalakis, M., Gionis, A., and Rastogi, R. (2000). XTRACT: a system for extracting document type descriptors from XML documents. *ACM SIGMOD*, pages 165–176.

Hegewald, J., Naumann, F., and Weis, M. (2006). Xstruct: Efficient schema extraction from multiple and large xml documents. In *Proceedings of the ICDEW '06*. IEEE Computer Society.

Min, J.-K., Ahn, J.-Y., and Chung, C.-W. (2003). Efficient extraction of schemas for XML documents. *Information Processing Letters*, 85(1):7–12.

Schreiner, G. A. (2014). Extração de esquemas de documentos XML: Uma abordagem probabilística. TCC, Universidade Federal da Fronteira Sul.

Vitanyi, P. and Li, M. (2000). Minimum description length induction, bayesianism, and kolmogorov complexity. *Information Theory, IEEE Transactions on*, 46(2):446–464.