TX — validação de XML baseada em tipos dinâmicos

José João Almeida and Alberto Manuel Simões

Departamento de Informática, Universidade do Minho
{jj|ambs}@di.uminho.pt

Resumo Desde o advento do SGML e posteriormente do XML, que a validação de documentos tem sido focada.
Esta validação surgiu para analisar a estrutura dos documentos SGML e XML usando DTDs. Além dessa, e devido às restrições do XML em relação ao SGML, a validação de XML bem formado também tem sido usada. Mais recentemente, os Schema e Schematron vieram permitir a validação a um nível superior: não só a estrutura do documento mas também alguma validação de conteúdo.
Neste artigo apresentamos a ferramenta TX que visa outro nível de validação, em que os tipos possam ser mais ricos e/ou calculados dinamicamente, e onde se possa definir funções de anotação e/ou correcção das porções do documento que não sigam as especificações.

1 Introdução

A definição de estrutura de um documento XML e a sua validação estrutural em relação a um DTD é, já desde os seus primórdios, uma tarefa que tem sido tratada com relativo sucesso.

A validação semântica tem vindo a ser uma preocupação crescente mas de difícil solução. A tipagem de elementos XML tem sido ignorada ou, recentemente com Schemas[3], Schematrons[5] e XCSL[7], demasiado rígidas em relação aos tipos usados.

Se considerarmos um XML que esteja a funcionar como base de dados (contendo informação fortemente estruturada), poderemos usar validadores de conteúdo baseadas em tipos como *strings* ou *inteiros* ou mesmo *inteiros menores do que 5 e maiores do que -5*. No entanto, a validação de conteúdo impõe frequentemente questões mais complexas.

Por exemplo, numa memória de tradução em formato TMX[8] (baseado em XML), uma tarefa da validação deverá, sem dúvida, verificar se o conteúdo das etiquetas de cada uma das memórias de tradução está na língua correcta, e sem erros ortográficos.

Esta foi a motivação para a construção da ferramenta a que chamamos TX (Typed XML) — uma *framework* de construção de validadores (usável como biblioteca Perl), incluindo um comando Unix para validação de documentos XML usando especificações TX. O use do TX é, por vezes, pouco genérica, ganhando-se em expressividade, nomeadamente com a possibilidade de validação com semântica operacional, tipos dinâmicos e anotação/correcção dos documentos.

1.1 Validação de Semântica Operacional

Consideremos os endereços de Internet: embora exista uma definição rígida para URIs, e embora seja possível definir uma expressão regular que valide essa sintaxe, isso não nos garante que se trate de um URI activo — todos sabemos que existem URIs totalmente válidos sintacticamente mas que não pertencem ao conjunto dos URIs vivos.

Frequentemente queremos que o nosso critério de validação verifique se os URLs estão activos, e não só se estão correctos. Um exemplo é a validação de um ficheiro de *bookmarks*.

Consideremos a correcção ortográfica de que já falamos: esta pode ser realizada por um programa externo como o ispell[4], aspell ou jspell, que suportam XML (e portanto, ignorará as etiquetas). No entanto, esta abordagem não é a mais correcta para a correcção de qualquer documento — já que analisa todo o documento independentemente do contexto.

Veja-se, por exemplo, o caso das memórias de tradução[6], em que o inicio do documento contém meta-informação e o corpo é constituído por frases em várias línguas.

Obrigaria ao utilizador duas passagens do ispell sobre o texto, uma em cada uma das línguas, tendo o cuidado de ignorar em cada passagem metade do texto.

Não sendo esta uma solução viável, torna-se importante existir a possibilidade de dar tipos a porções de documento com línguas diferentes, em que a correcção seleccione automaticamente a língua a usar.

1.2 Validação com Tipos Dinâmicos

A validação com tipos dinâmicos tem o objectivo de calcular o tipo do elemento e portanto o seu critério de correcção, com base no próprio elemento.

Por exemplo, é impossível tipar genericamente qualquer memória de tradução, já que as línguas usadas não são necessariamente as mesmas em cada uma[1]. Desta forma, é necessário que o validador consiga discernir em que língua está o elemento antes de o avaliar. Este tipo de informação pode ser obtida usando, por exemplo, em função do atributo `xml:lang` existente nas memórias de tradução.

1.3 Anotação e Correcção

Supondo que já temos um sistema para validar este tipo de informação, o passo seguinte será discutir a forma de reacção aos erros encontrados. A forma habitual é indicar que na linha x, o elemento y tem texto inválido. Embora útil, esta informação não é suficientemente precisa. Mesmo que a mensagem de erro inclua, por exemplo, a palavra com erro ortográfico, quando o utilizador tentar editar

[1] Uma memória de tradução tem frases em duas ou mais línguas, pelo que nada obsta a que se tenha uma memória de tradução que use apenas Português e Inglês, e uma outra que use apenas Francês e Dinamarquês

o XML para o corrigir irá, com certeza, ter-se esquecido da palavra em causa. Para colmatar este tipo de problema, propomos a marcação do texto analisado para facilitar a sua posterior correcção (eventualmente usando uma ferramenta específica para esse efeito).

Parece-nos igualmente importante que um validador seja capaz de não só validar mas também corrigir (ou propor correcções). Sem dúvida que grande parte dos erros que um validador encontra não podem ser corrigidos sem intervenção humana mas, isso não implica que o sistema não seja capaz de assistir, interactivamente, no processo de correcção dos erros encontrados (como é habitual, por exemplo, nas ferramentas de correcção ortográfica.

2 Abordagem Proposta

Na abordagem que aqui se propõe, pretendemos enriquecer o processo de validação de documentos XML através de:

- *associar* (estática ou dinamicamente) *tipos* a alguns dos elementos;
- *criar* (externamente) *validadores de tipos* que sejam capazes de marcar e alterar (automaticamente ou de modo interactivo) os elementos incorrectos desse tipo.
- mecanismos de validar tipos baseados em *semântica operacional*: ou seja que os tipos possam ficar associados a uma função (no caso presente imperativa, Perl) que tire partido da funcionalidade existente nas bibliotecas e no sistema operativo.
- criar mecanismos (uma API, um módulo Perl) de *definir novos tipos* e até validadores.

Na definição dos tipos pode haver necessidade de lhes associar:

- uma *função de marcação* de erros encontrados — esta função pode ser tão simples como: para a correcção ortográfica, preceder cada palavra desconhecida por um símbolo; pela inclusão de um comentário; pela adição de etiquetas à volta do elemento ou conteúdo errado;
- uma *função de correcção* automática ou interactiva das anomalias encontradas;
- uma *política de inclusão ou não dos filhos* (recursividade ou não pelas subárvores dos elementos filhos).

A ferramenta utiliza valores por omissão no caso destas propriedades não estarem definidas.

3 TX *by example*

Considere-se um dicionário definido em XML com cabeçalho e entradas, de acordo com a seguinte gramática (DTD):

```
<!ELEMENT dic       (cabecalho, entrada*)      >
<!ELEMENT cabecalho (...)                      >
<!ELEMENT entrada   (termo*, definicao, figura)>

<!ELEMENT termo     (#PCDATA)                  >
    <!ATTLIST termo xml:lang CDATA #REQUIRED   >

<!ELEMENT definicao (#PCDATA)                  >

<!ELEMENT figura    (#EMPTY)                   >
    <!ATTLIST figura url CDATA #REQUIRED       >
```

e, a titulo exemplificativo, considere-se o seguinte documento:

```
<dic>
  <cabecalho>...</cabecalho>
  <entrada>
    <termo xml:lang="pt">violino</termo>
    <termo xml:lang="en">violin</termo>
    <definicao>Instrumento de cordas...</definicao>
    <figura url="http://imagem.de/violino.png"/>
  </entrada>
  <entrada>
  ...
  </entrada>
</dic>
```

A validação da estrutura deste documento pode ser feita usando um validador comum sensível a DTDs (por exemplo, `xmllint`[9]). No entanto, o uso deste tipo de ferramenta nos garante que:

- cada *termo* está na língua definida, e sem erros ortográficos;
- a *definição* está em Português (para manter coerência em todo o dicionário);
- a *imagem* existe, e está acessível publicamente;

De acordo com a abordagem TX, o que pretendemos é associar a cada elemento um tipo. Para especificar o elemento, a forma mais expedita é o uso de XPath[2]. A cada um destes elementos queremos associar um tipo estático ou um tipo dinâmico:

$$TXdef \equiv XPath \rightharpoonup (Tipo + TipoDinamico(elemento))$$

em que `TipoDinamico` é uma função que, dado o elemento ou atributo XML a analisar nos devolve o tipo desse elemento.

$$TipoDinamico : elemento \longrightarrow Tipo$$

Um exemplo de associação de tipos ao DTD apresentado anteriormente, usando a sintaxe TX, seria:

```
1  //definicao      Text("PT")
2  //termo          Text(@xml:lang)
3  //figura@url     ActiveURL
```

Esta sintaxe é definida por duas colunas; um selector em XPath do elemento ou atributo que queremos validar, e a definição do tipo estático ou dinâmico a ser usado:

1. a primeira linha apresenta um tipo estático, que valida a língua do conteúdo do elemento definição, como sendo Português;
2. a linha seguinte, é semelhante à anterior, mas de ordem superior: a língua a ser validada é determinada a partir do atributo `xml:lang` desse elemento (usando uma directiva XPath);
3. o último exemplo é uma regra directa sobre um atributo. O tipo estático ActiveURL valida se determinado URL ainda está vivo.

Embora o tipo `ActiveURL` seja *built-in*, na secção 3.1 usá-mo-lo para demonstrar como se pode adicionar a esta lista de associações a definição de novos tipos.

O modo de utilização da ferramenta TX sobre a especificação completa seria:

```
1  tx file.txd file.xml
```

em que `file.txd` é a nossa definição de tipos e `file.xml` o ficheiro a validar. Por omissão o `tx` marca os erros encontrados. Se usada a opção `-correct` a ferramenta invoca a função de correcção associada a cada tipo (caso não exista usará a de marcação).

3.1 Definição de validadores para novos tipos

No exemplo que seguidamente se apresenta é criado um um novo tipo (ActiveURL), usado no exemplo anterior.

```
1  //url ActiveURL
2  %%
3  use LWP::Simple;
4  XML::TX::addType(
5     ActiveURL =>
6         { markit => sub{ $c = markAsErr($c) unless (LWP::Simple::head($c));
7                          toxml()}, } );
```

Esta especificação é composta por duas partes:

1. uma zona de associação Xpath → tipo (linha 1)
2. uma zona de criação de novos tipos: através de código Perl, invocando funcionalidade do módulo XML::TX para adicionar o tipo definido.

A função **addType** usada para definir o novo tipo urlActive, está a receber uma função (`markit`) para marcar as situações consideradas erradas. Essa função de marcação dos erros está a usar o módulo Perl `LWP::Simple` [1] para descarregar o cabeçalho (função `head`) ligado ao URL a validar.

Saliente-se que este tipo de validação é difícil de obter usando apenas especificações declarativas.

3.2 Validadores usando extrai-processa-reconstroi

Por uma questão de eficiência, não é aceitável arrancar um processo externo de validação para cada ocorrência de um determinado elemento[2].

Por exemplo um corrector ortográfico normalmente carrega para memória o dicionário associado, ou seja, carrega para memória alguns MBytes de informação. Se esse carregamento for feito para cada parágrafo encontrado, a validação vai ser extremamente lenta.

Este facto pode ser ultrapassado com a seguinte estratégia:

1. Criar um texto com os elementos de cada tipo (extrai)
2. executar interactivamente o validador sobre cada ficheiro de cada tipo criado (processa)
3. substituir no texto XML original os elementos que tiverem sido alteradas no processo de validação (reconstrói)

Esta estratégia de validação é genérica e o módulo XML::TX dispõe de função de ordem superior (`ext_proc_rec`) que pode ser usada na definição de novos tipos.

```
1 | Correcção=ext_proc_rec(CorrectorInteractivo,....)
```

3.3 Validadores usando apenas o módulo XML::TX

O módulo XML::TX, pode ser usado para criar validadores completos totalmente escritos em Perl.

No exemplo que seguidamente se apresenta é criado um validador.

```
1 | use XML::TX;
2 | my $types={ sentencePt => text("pt"),
3 |             sentenceEn => text("en"),
4 |             definition => sub{text($v{'xml:lang'} || "pt")},
5 |             url        => "urlActive",
6 |           };
  |
7 | XML::TX::addType(
8 |    urlActive =>
9 |        { markit => sub{ $c = markAsErr($c) unless (LWP::Simple::head($c));
```

[2] O esforço de arranque de uma aplicação externa pode ser grande.

```
10                          toxml()}, } );

11  XML::TX::markit($filename,$types);
```

O validador criado usa os tipos predefinidos:

- os elementos *sentencePt* são do tipo predefinido *text*("*pt*")
- os elementos *sentencePt* são do tipo predefinido *text*("*en*"),
- os elementos *definition* são de um tipo dinâmico calculado em função do atributo *xml:lang*,
- os elementos *url* são de um tipo *urlActive* que é definido através da invocação da função *addType*.

4 Tipos predefinidos

Na ferramenta TX existem alguns tipos predefinidos que resultaram de necessidades encontradas em problemas reais.

Tipo ActiveURL

O tipo ActiveURL foi já apresentado em secções anteriores.

Este tipo de validade é por vezes mais importante do que a simples validação sintáctica já que permite a detecção de gralhas que tipicamente mantêm o URL sintacticamente correcto.

Tipo Email

Para os e-mails não podemos, como para os URLs, validá-los de forma sistemática, podendo apenas definir os sintacticamente válidos ou inválidos.

Tipo Data

A validação de datas também é simples, sendo no entanto complicado validar todas as variantes possíveis para a escrita de uma data. Noutros casos, existe a noção de formato válido, e todos os outros são considerados errados.

Neste ponto ganha-se bastante em ter a possibilidade de correcção. Sabendo-se o conjunto de formas diferentes usadas para escrever datas, será possível, sem grande esforço, fazer a conversão para um formato de data *standard* no documento.

Tipo Texto-$\mathcal{L}_\alpha$

No processamento de textos de uma língua $\mathcal{L}_\alpha$ há necessidade de fazer a respectiva análise/correcção ortográfica.

A marcação de erros está a ser feita do seguinte modo:

1. enviar cada palavra a um processo Ispell[4] ou Aspell;
2. prefixar as palavras desconhecidas com uma marca (`##`)

A função de correcção está a usar a estratégia extrai-processa-reconstrói, atrás referida:

1. Criar um texto com as frases de cada língua (extrai)
2. executar interactivamente um corrector ortográfico sobre cada ficheiro de língua criado (processa)
3. substituir no texto XML original as frases que tiverem sido alteradas no processo de correcção ortográfica (reconstrói)

5 Conclusões

A tipagem dos métodos tradicionais é insuficiente para manter documentos com semântica forte, pelo que a possibilidade de validação com base em tipos operacionais e dinâmicos é importante.

No entanto, esta ferramenta não pode ser vista como uma substituição aos métodos tradicionais, mas antes mas um passo na validação de um documento XML:

- correcção sintáctica;
- correcção estrutural;
- correcção de tipos básicos;
- correcção de tipos dinâmicos e operacionais;

A possibilidade de programação da ferramenta obriga a maiores conhecimentos, nomeadamente da linguagem Perl, mas facilita a definição de novos tipos e validadores associados.

Referências

1. Sean M. Burke. *Perl & LWP*. O'Reilly, 2002.
2. James Clark and Steve DeRose. XML Path Language (XPath), 1999. `http://www.w3.org/TR/xpath`.
3. David C. Fallside. XML Schema — W3C Recommendation, 2001. `http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/`.
4. R. Gorin, P. Willisson, W. Buehring, and G. Kuenning. Ispell, a free software package for spell checking files, 1971.
5. Rick Jelliffe. Resource Directory (RDDL) for Schematron 1.5, 2003.
6. OSCAR. Open Standards for Container/Content Allowing Re-use — TMX home page, 2003. `http://www.lisa.org/tmx/`.
7. José Carlos Ramalho. Constrain content: Specification and processing. 2001. XML Europe.
8. Yves Savourel. TMX 1.4a Specification. Technical report, Localisation Industry Standards Association, 1997.
9. Libxml. `http://www.libxml.org`.