XML Templates for Constraints (XTC), *Um nível de abstracção para Linguagens de Especificação de Restrições*

Marta H. Jacinto* José C. Ramalho
Pedro R. Henriques

Departamento de Informática
Universidade do Minho
4710-057 Braga
marta.jacinto@itij.mj.pt,{jcr, prh}@di.uminho.pt

Resumo

Os DTDs permitem etiquetar um texto e validar a sua estrutura contra uma gramática.
As linguagens de especificação de restrições (XML Constraint Specification Languages), nomeadamente o XCSL, o Schematron e os XML-Schemas, *num nível mais elevado*, já permitem validar aspectos não estruturais dos documentos XML, tais como: relações entre elementos, ou atributos, pertencentes a diferentes contextos; invariantes sobre modelos de dados; e restrições ao valor dos elementos, ou atributos.
O sistema XCSL (XML Constraint Specification Language) nasceu no seio do nosso grupo de investigação [7]. No entanto esta linguagem foi testada em pé de igualdade com Schematron e XML-Schema. Usou-se um conjunto considerável de casos de estudo para testar e comparar estas três linguagens em termos: dos tipos de restrições especificáveis; da facilidade de aprendizagem/utilização; da informação devolvida ao utilizador. Os resultados mais significativos foram descritos em [3].
Fazendo esta comparação, apercebemo-nos que em cada linguagem e para cada tipo de restrição há um texto fixo e um conjunto de partes variáveis, sendo este último comum às várias linguagens. Tendo em conta estas partes variáveis, criámos *templates* para cada tipo de restrição, em cada uma das três linguagens. Com estes *templates* é possível gerar a especificação de restrições, em qualquer uma daquelas linguagens, a partir de um conjunto finito de parâmetros.
Neste artigo mostramos os **templates** para cada par tipo-de-restrição

*Presentemente a trabalhar no ITIJ - Instituto das Tecnologias de Informação na Justiça, Ministério da Justiça

/ linguagem. A partir das partes comuns desses *templates* construímos um conjunto de **templates genéricos**, designado XTC—XML Templates for Constraints—, um para cada tipo de restrição independentemente da linguagem escolhida. Com um documento XTC pode gerar-se todos os ficheiros de especificação de restrições, ou seja, um ficheiro de especificação para cada linguagem. Apresentamos, então, vários exemplos escritos em XTC.
A implementação final usa aquilo a que chamamos *sistema de folhas de estilo XSL de terceira geração*, três níveis de folhas de estilos. Com a primeira folha de estilos (a do XTC) e o documento XTC geramos o documento de especificação na linguagem pretendida; com este último e a segunda folha de estilos (específica da linguagem pretendida) geramos a terceira folha de estilos (documento com o qual já se vai poder validar a semântica das várias instâncias); por fim, aplicamos esta última folha de estilos aos vários documentos da família em estudo.
Terminamos o artigo mostrando como construímos esta arquitectura baseada apenas em XML e XSL.

Palavras-chave:
XML, DTD, XML-Schema, Schematron, XCSL, validação semântica, XSL

1 Introdução

O SGML e o XML alteraram profundadamente a forma como as pessoas pensam e lidam com documentos. Estes standards permitiram que nascesse uma nova metodologia e modelo de processamento de documentos.

Tornaram ainda possível a independência entre contexto e estrutura, validação estrutural, longevidade dos documentos, independência tanto de hardware como de software, e muito mais. Contudo, esta validação estrutural nem sempre e suficiente para obter a qualidade final desejada para os documentos.

Recentemente, propostas como DCD[1], XDuce[2], XML-Schema[3], TREX[4] e RelaxNG[5], foram submetidas ao W3C numa tentativa de resolver o problema. Daquelas, XML-Schema e RelaxNG têm sido as mais utilizadas pela comunidade XML.

No entanto, nenhuma destas duas linguagens é suficiente para atingir o tipo de validação que tratamos neste artigo. Para isso é necessária uma extensão, uma linguagem (e respectiva ferramenta) que permita especificar (e validar) restrições sobre documentos XML.

Por forma a formatar um documento, é necessário seleccionar fragmentos e aplicar-lhes um estilo ou, se quisermos questionar o conteúdo do documento, é necessário seleccionar os nodos desejados da árvore documental. Para isto temos o XPath — a linguagem de padrões inserida no XSLT, poderosa o suficiente para

[1] www.w3.org/TR/NOTE-dcd
[2] http://xduce.sourceforge.net/
[3] http://www.w3.org/XML/Schema
[4] http://thaiopensource.com/trex/
[5] http://www.oasis-open.org/committees/relax-ng/

seleccionar qualquer contexto dentro da árvore documental.

Começamos por descrever uma arquitectura operacional simples que permite aos utilizadores especificar restrições e testá-las dentro de um sistema XML-XSL (secção 2).

Depois, na secção 3, apresentamos as três linguagens de especificação de restrições que analisámos: XCSL, Schematron e XML-Schema, numa breve abordagem.

Na secção 4 mostramos os **templates** para cada par tipo-de-restrição / linguagem. A partir das partes comuns desses *templates* construímos um conjunto de **templates genéricos** (secção 5), que designámos por XTC—XML Templates for Constraints—, um para cada tipo de restrição independentemente da linguagem escolhida. Na secção 6 apresentamos a linguagem XTC e exemplos escritos nessa linguagem.

A implementação final usa aquilo a que chamamos *sistema de folhas de estilo XSL de terceira geração*: três níveis de folhas de estilos. Com a primeira folha de estilos (a do XTC) e o documento XTC, geramos o documento de especificação na linguagem pretendida; com este último e a segunda folha de estilos (específica da linguagem pretendida) geramos a terceira folha de estilos (documento com o qual já se vai poder validar a semântica das várias instâncias); por fim, aplicamos esta última folha de estilos aos vários documentos da família em estudo. Este processo encontra-se descrito na secção 7.

A secção 7 mostra ainda como construímos esta arquitectura baseada apenas em XML e XSL.

Por fim, generalizamos a ideia, especificando um método para implementar o modelo de processamento de restrições com qualquer ferramenta que suporte XSLT.

A conclusão pode ser encontrada na secção 8.

2 Restringindo o conteúdo

Em [6], foi estabelecido um paralelismo entre o Processamento de Linguagens Formais e o Processamento de Documentos.

Esta hipótese ajuda a perceber que o processamento de documentos actual é muito semelhante ao dos primeiros tempos da construção de compiladores.

Até há poucos anos, tinhamos apenas análise léxica e sintáctica. Os DTDs permitiram especificar o léxico e a gramática (quase abstracta) de uma determinada linguagem de anotação. Depois, em 1996, o DSSSL apareceu permitindo a especificação do que denominamos por semântica dinâmica, i.e., com DSSSL podemos especificar transformações sobre a estrutura produzida pelo *parser* quando este analisa o documento.

No entanto, este avanço não colmatou uma outra falha: a semântica estática.

Por Semântica Estática entendemos um conjunto de regras que impõem restrições sobre o conteúdo específico de cada documento, limitando o valor ab-

soluto ou relativo que certos elementos (ou componentes) podem ter. Para descrever essas regras sobre os valores admissíveis, é necessária uma linguagem que permita escrever condições sobre o conteúdo exacto das várias componentes estruturais, condições essas que são tipicamente dependentes do contexto preciso onde o elemento aparece. Para conceber uma dessas linguagens, a XCSL, Ramalho sentiu necessidade de classificar os tipos de restrições existentes [7]. Entretanto esta classificação foi refinada ([3, 4]). Relativamente à restrição de conteúdo, podemos assim classificar as condições contextuais a impor de acordo com as categorias seguintes:

Restrição de domínio Impõe que um determinado conteúdo esteja entre dois valores, isto é, num determinado domínio. Este é o tipo mais comum de restrição e, normalmente, aplica-se a dados numéricos ou do tipo data. Exemplo: a idade e a altura têm que ser valores positivos.

Dependência entre dois elementos ou atributos Permite relacionar o valor de um atributo ou elemento com o de outro atributo ou elemento localizado num ramo diferente da árvore documental (restrições dependentes do contexto) do qual ele depende. Exemplo: nome e verbo devem concordar em número.

Concordância com padrões descritos por Expressões Regulares Permite garantir que o conteúdo segue um determinado formato que pode ser padronizado por uma ER. Exemplo: Os números de telefone em Portugal têm nove dígitos e começam por 2, 7, 8, 91, 93 ou 96.

Restrições complexas Este grupo reúne todas as restrições que não podem pertencer a nenhum dos três grupos anteriores. Podem dividir-se em três casos (os dois primeiros são os casos particulares mais vulgares):

- **Restrições de conteúdo misto** Quando, por exemplo num elemento de conteúdo misto, se quer garantir determinada cardinalidade e/ou ordem dos elementos que podem aparecer no meio do texto. Exemplo: O fecho de um pedido de Certidão Fiscal é composto por texto livre, no qual têm que aparecer os sub-elementos *local* e *data*, uma vez cada e exactamente por esta ordem.
- **Restrições de unicidade** Quando se quer garantir que cada ocorrência de algum elemento, em determinado contexto, tem um valor diferente do de todas as outras ocorrências desse elemento nesse mesmo contexto. Exemplo: Não pode haver valores repetidos nos elementos chave de uma determinada tabela.
- **Restrições mistas** Sempre que uma restrição não pertence a nenhuma das categorias já enumeradas, ou resulta da mistura de algumas dessas categorias. Exemplo: Um poema deve ser constituído por duas quadras e dois tercetos (exactamente por esta ordem), mas apenas se for do tipo "soneto".

3 Linguagens de Especificação de Restrições

Nesta secção apresentam-se as três linguagens de especificação de restrições estudadas.

3.1 XCSL

O XCSL (XML Constraint Specification Language) é uma linguagem que foi desenhada e implementada por José Carlos Ramalho na Universidade do Minho, Portugal. Combina capacidades poderosas de validação com uma sintaxe e implementação simples. O XCSL foi definido formalmente em [7] e [5] e noutros artigos publicados. O DTD e XML-Schema que o definem podem ser encontrados em http://www.di.uminho.pt/~jcr/PROJS/xcsl-www/.

Para validar instâncias XML utilizando XCSL, é necessário executar dois passos. O primeiro consiste em compilar o documento de restrições com um gerador que vai produzir um validador específico em XSL. O segundo consiste em validar cada instância contra esse validador específico da família de documentos. O utilizador tem que efectuar os dois passos explicitamente invocando uma ferramenta como o Saxon[6] na linha de comandos.

3.2 Schematron

O Schematron é uma linguagem XML desenhada e implementada por Rick Jelliffe na Academia Sinica Computing Centre, Taiwan. Combina capacidades poderosas de validação com uma sintaxe e implementação simples. O Schematron está descrito em [1] e muitos outros artigos e foi recentemente submetido para standard ISO como ISO Schematron. O DTD e XML-Schema que o definem podem ser encontrados em http://www.ascc.net/xml/schematron/.

Toda esta abordagem (linguagem e processamento) é muito semelhante à anterior e por isso, para validar instâncias XML utilizando Schematron, também é necessário executar dois passos. O primeiro consiste em compilar o documento de restrições de modo a gerar um validador apropriado. O segundo consiste em validar cada instância contra esse validador específico da família de documentos. Para a execução destes dois passos, pode escolher-se uma de duas opções: Topologi Schematron Validator é a primeira — o processo de dois passos é transparente para o utilizador e tudo o que este tem que fazer é fornecer o documento com as restrições e as instâncias; e o Schematron-report — o utilizador tem que efectuar os dois passos explicitamente. Enquanto que este último, Schematron-report, é processado invocando uma ferramenta como o Saxon na linha de comandos; o Topologi Schematron Validator é um ambiente interactivo baseado em janelas.

[6] http://users.iclway.co.uk/mhkay/saxon

3.3 XML-Schema

O XML-Schema [2] do W3C foi criado uma vez que a sintaxe dos DTDs não satisfazia as necessidades dos utilizadores de XML. Quando se usa um DTD e se pretende especificar restrições semânticas mesmo que muito simples, é necessário usar uma linguagem de especificação de restrições (como o XCSL ou o Schematron) e, consequentemente, são necessários dois instrumentos para validar completamente um documento XML. Por outro lado, quando se usa antes um XML-Schema, será necessário, para um conjunto específico de restrições, apenas um documento para conseguir validar completamente as instâncias XML em vez de dois.

Há vários *parsers* disponíveis para validar as instâncias XML contra um XML-Schema. Esses *parsers* são semelhantes aos validadores de XML tradicionais mas agora, em vez do DTD, utilizam o XML-Schema.

4 Templates

Nesta secção mostramos os **templates** para cada par tipo-de-restrição / linguagem [4], organizado por linguagem.

4.1 XCSL

4.1.1 Restrição de domínio

```
<constraint>
 <selector selexp="path to the element"/>
 <cc>. | @attname relop value</cc>
 <action>
   <message>Message...
     <value selexp="path to any
       element/attribute | any expression
       applied to any element/attribute"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. Em *cc* especifica-se a restrição propriamente dita: o elemento (.) ou um atributo (@attname), o operador lógico e o valor (*value*) ou valores que limitam o domínio. A mensagem a ser mostrada sempre que a restrição não se verifica é descrita usando o elemento *message* onde se pode escrever texto livre (*Message...*) e/ou o valor de qualquer elemento/atributo no documento (*path to any element/attribute*) ou qualquer expressão a ele aplicada (*any expression applied to any element/attribute*).

4.1.2 Dependência entre dois elementos ou atributos

```
<constraint>
 <selector selexp="path to the 1st element"/>
 <cc>. | @attname relop path to the 2nd element
    [/@attname]
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição propriamente dita (*cc*) é agora: o elemento (.) ou um atributo (@attname), o operador lógico e o caminho para o segundo elemento/atributo (*path to the 2nd element*). A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.3 Concordância com padrões descritos por Expressões Regulares

```
<constraint>
 <selector selexp="path to the element"/>
 <cc>
   substring(.| @attname,i,n1)
   =literal_value and
   (string-length(number(substring(.| @attname,j,n2)))
   = value
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

Novamente, o caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição (*cc*) tem um determinado número de expressões (que dizem respeito a um elemento ou atributo) ligadas pelo operador lógico "e" (*and*); estas expressões podem ser:

- para uma parte fixa — *substring(. | @attname,i,n1) = literal_value*
  (por exemplo se os primeiros quatro caracteres do elemento seleccionado tiverem que ser '253-', escreve-se *substring(.,1,4) = '253-'*);

- ou para um conjunto contíguo de dígitos — *string-length(number(substring(. | @attname,j,n2))) = value*
  (por exemplo se os 6 caracteres do atributo x do elemento seleccionado que começam na posição 5 tiverem que ser dígitos, escreve-se *string-length(number(substring(@x,5,6))) = 6*).

A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.4 Restrições complexas - conteúdo misto

```
<constraint>
 <selector selexp="path to the parent element"/>
 <cc>
   (count(elt1)=c_elt1) and (count(elt2)=c_elt2)
   and ... (count(eltn)=c_eltn) and
   name(elo1[1]/following::*)='elo2' and
   name(elo2[1|2]/following::*)='elo3' and
   ... and
   name(elo(m-1)[1|2|...|m-1]/following::*)='elom'
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O elemento *selector* serve agora para seleccionar o caminho para o elemento pai (*path to the parent element*). A restrição (*cc*) tem duas partes ligadas pelo operador lógico "e" (*and*):

- a cardinalidade dos sub-elementos, constituída por várias expressões *(count(elti)= c_elti)* ligadas por "e" (*and*)
  (por exemplo se o elemento *lugar* tiver que aparecer três vezes no elemento pai, escreve-se *(count(lugar)=3)*);

- a ordem dos sub-elementos, constituída por várias expressões *name(elo(j-1)[1|2|...|j-1]/ following::*)='eloj'*[7] (deve indicar-se a ordem exacta pela qual os sub-elementos devem ocorrer — elo1, elo2, ...) também ligadas

[7] [1|2|...|j-1], significa que a ocorrência do sub-elemento que se está a testar pode ser a 1^a, 2^a, ...,(j-1)-ésima ocorrência desse sub-elemento no elemento pai.

pelo operador "e" (*and*)
(por exemplo se a terceira ocorrência do elemento *lugar* tiver que ser seguida de uma ocorrência do elemento *estado*, escreve-se *name(lugar[3]/ following::*)='estado'*).

A mensagem tem as mesmas características que a descrita em 4.1.1.

4.1.5 Restrições complexas - unicidade

```
<constraint>
 <selector selexp="path to X branch"/>
 <let name="nameKey1"
      value="elementX | @attributeX"/>
 <cc>(count(path to Y branch[elementY
     | @attributeY = $nameKey1]) = 1)
 </cc>
 <action>
   <message>
     Message...
     element | @attribute:
     <value selexp="$nameKey1"/>.
   </message>
 </action>
</constraint>
```

Para este tipo de restrição, suponha-se que se quer forçar todos os valores de determinado elemento/atributo (element|@attribute) que ocorram no ramo X a existir também no ramo Y (uma vez). Primeiro, selecciona-se o caminho para o ramo X (*path to X branch*) no elemento *selector*. Depois, usa-se o elemento *let* para guardar numa lista todos os valores que esse elemento/atributo assume naquele ramo X (*elementX | @attributeX*). O elemento *cc* serve agora para contar o número de vezes que cada ocorrência do elemento/atributo no ramo Y ocorre na lista previamente definida e comparar esse valor com 1. O elemento *message* pode ter texto livre (Message... element|@attribute:) e/ou o valor que não aparece no ramo Y ($nameKey1).

4.1.6 Outras Restrições complexas

Não é possível escrever *templates* para este tipo de restrições uma vez que, virtualmente, podem ser qualquer coisa, o que é impossível de normalizar.

Para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, tem-se:

```
<constraint>
 <selector selexp="path to the element"/>
```

```
 <cc>
   not (path/@attname | . | @attname = value) or
   (constraint)
 </cc>
 <action>
   <message>
     Message...
     <value selexp="path to any elt/att |
     any expression applied to any elt/att"/>
   </message>
 </action>
</constraint>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no elemento *selector*. A restrição (*cc*) é agora: o operador lógico "não" (*not*) seguido pela especificação do valor que certo elemento/atributo, naquele ou noutro ramo, toma (o valor único para o qual a restrição vai ser tida em linha de conta), seguida pelo operador "ou" (*or*) e, finalmente, a restrição. A mensagem tem as mesmas características que a referida em 4.1.1.

4.2 Schematron

Nesta sub-secção mostram-se os *templates* para a linguagem Schematron.

4.2.1 Restrição de domínio

```
<pattern name="pattern title">
 <rule context="path to the element">
 <assert test=" . | @attname relop value" diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any
     element/attribute | any expression applied
     to any element/attribute"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. No atributo *test* do elemento *assert* especifica-se a restrição propriamente dita: o elemento (.) ou um atributo (@attname), o operador lógico e o valor (*value*) ou valores que limitam o domínio. A mensagem, a ser mostrada sempre que a restrição não se verifica, é descrita usando o elemento *diagnostic* onde se pode escrever texto livre (Message...) e/ou o valor de qualquer

elemento/atributo no documento (*path to any element/attribute*) ou qualquer expressão a ele aplicada (*any expression applied to any element/attribute*).

4.2.2 Dependência entre dois elementos ou atributos

```
<pattern name="pattern title">
 <rule context="path to the 1st element">
   <assert test=" . | @attname relop path to the 2nd element
               [/@attname] " diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição propriamente dita (*test*) é agora: o elemento (.) ou um atributo (@attname), o operador lógico e o caminho para o segundo elemento/atributo (*path to the 2nd element*). A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.3 Concordância com padrões descritos por Expressões Regulares

```
<pattern name="pattern title">
 <rule context="path to the element">
   <assert test="
     substring(.| @attname,i,n1)=literal_value and
     (string-length(number(
     substring(. | @attname,j,n2))) = value"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

Novamente, o caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição (*test*) tem um determinado número de

expressões (que dizem respeito a um elemento ou atributo) ligadas pelo operador lógico "e" (*and*); estas expressões podem ser:

- para uma parte fixa — *substring(. | @attname,i,n1) = literal_value*
  (por exemplo se os primeiros quatro caracteres do elemento seleccionado tiverem que ser '253-', escreve-se *substring(.,1,4) = '253-'*);
- ou para um conjunto contíguo de dígitos — *string-length(number(substring(. | @attname,j,n2))) = value*
  (por exemplo se os 6 caracteres do atributo x do elemento seleccionado que começam na posição 5 tiverem que ser dígitos, escreve-se *string-length(number(substring(@x,5,6))) = 6*).

A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.4 Restrições complexas - conteúdo misto

```
<pattern name="pattern title">
 <rule context="path to the parent element">
   <assert test="(count(elt1)=c_elt1) and
     (count(elt2)=c_elt2) and ...
     (count(eltn)=c_eltn) and
     name(elo1[1]/following::*)='elo2' and
     name(elo2[1|2]/following::*)='elo3' and
     ... and
     name(elo(m-1)[1|2|...|m-1]/following::*)='elom'"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O atributo *context* serve agora para seleccionar o caminho para o elemento pai (*path to the parent element*). A restrição (*test*) tem duas partes ligadas pelo operador lógico "e" (*and*):

- a cardinalidade dos sub-elementos, constituída por várias expressões *(count(elti)= c_elti)* ligadas por "e" (*and*)
  (por exemplo se o elemento *lugar* tiver que aparecer três vezes no elemento pai, escreve-se *(count(lugar)=3)*);

– a ordem dos sub-elementos, constituída por várias expressões *name(elo(j-1)[1|2|...|j-1]/ following::*)='eloj'*[8] (deve indicar-se a ordem exacta pela qual os sub-elementos devem ocorrer — elo1, elo2, ...) também ligadas pelo operador "e" (*and*)
(por exemplo se a terceira ocorrência do elemento *lugar* tiver que ser seguida de uma ocorrência do elemento *estado*, escreve-se *name(lugar[3]/ following::*)='estado'*).

A mensagem tem as mesmas características que a descrita em 4.2.1.

4.2.5 Restrições complexas - unicidade

```
<pattern abstract="true" id="uID">
 <rule context="path to Y branch">
   <key name="nameKey1"
        path="elementY | @attributeY"/>
 </rule>
</pattern>
<pattern name="pattern title">
 <rule context="path to X branch">
   <assert test="count(key('nameKey1',elementX
                 | @attributeX ) = 1)"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   element | @attribute:
   <value-of select="elementX | @attributeX"/>
 </diagnostic>
</diagnostics>
```

Para este tipo de restrição, suponha-se que se quer forçar todos os valores de determinado elemento/atributo (element|@attribute) que ocorram no ramo X a existir também no ramo Y (uma vez). Primeiro selecciona-se o caminho para o ramo Y (*path to Y branch*) no atributo *context* de *rule* descendente de um elemento *pattern*. Depois usa-se o elemento *key* (descendente do referido elemento *rule*) para guardar numa lista todos os valores que esse elemento/atributo assume naquele ramo Y (*elementY | @attributeY*). O atributo *context* usual (atributo de um elemento *rule* descendente de um novo elemento *pattern*) serve para seleccionar o caminho para o ramo X (*path to X branch*) e o elemento *assert* serve agora para contar quantas vezes cada ocorrência da lista previamente

[8] [1|2|...|j-1], significa que a ocorrência do sub-elemento que se está a testar pode ser a 1^a, 2^a, ...,(j-1)-ésima ocorrência desse sub-elemento no elemento pai.

definida ocorre no ramo X e comparar esse valor com 1. O elemento *diagnostic* pode ter texto livre (Message... element|@attribute:) e/ou o valor que não aparece no ramo Y (elementX | @attributeX).

4.2.6 Outras Restrições complexas

Não é possível escrever *templates* para este tipo de restrições uma vez que, virtualmente, podem ser qualquer coisa, o que é impossível de normalizar.

Para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, tem-se:

```
<pattern name="pattern title">
 <rule context="path to the element">
   <assert test="
       not (path/@attname | . | @attname = value)
       or (constraint)"
   diagnostics="d01"/>
 </rule>
</pattern>
<diagnostics>
 <diagnostic id="d01">
   Message...
   <value-of select="path to any elt/att |
   any expression applied to any elt/att"/>
 </diagnostic>
</diagnostics>
```

O caminho para o elemento (*path to the element*) a restringir é especificado no atributo *context*. A restrição (*test*) é agora: o operador lógico "não" (*not*), seguido pela especificação do valor que certo elemento/atributo, naquele ou noutro ramo, toma (o valor único para o qual a restrição vai ser tida em linha de conta), seguida pelo operador "ou" (*or*) e, finalmente, a restrição. A mensagem tem as mesmas características que a descrita em 4.2.1.

4.3 XML-Schema

Nesta secção mostram-se os *templates* para a linguagem XML-Schema.

4.3.1 Restrição de domínio

```
<xs:simpleType name="name of the type of the
                          element/attribute">
 <xs:restriction base="xs:integer">
   <xs:minInclusive value="value"/>
 </xs:restriction>
</xs:simpleType>
```

Cria-se um tipo simples para o elemento ou atributo em causa, por restrição, a partir de um conjunto conhecido.

4.3.2 Dependência entre dois elementos ou atributos

```
Não é possível especificar...
```

4.3.3 Concordância com padrões descritos por Expressões Regulares

```
<xs:simpleType name="telement">
  <xs:restriction base="xs:string">
    <xs:pattern value="literal_value\d{value}"/>
  </xs:restriction>
</xs:simpleType>
```

Cria-se, por restrição, um tipo simples para o elemento ou atributo em causa a partir do conjunto *xs:string*. Depois, especifica-se, no atributo *value*:

- para cada parte fixa — literal_value (por exemplo '253-');
- para cada conjunto contíguo de dígitos — \d{value} (por exemplo \d{6}).

4.3.4 Restrições complexas - conteúdo misto

```
<xs:complexType name="tparent element"
                                  mixed="true">
 <xs:sequence>
   <xs:element name="elo1" type="telo1"
       minOccurs="elo1Min" maxOccurs="elo1Max"/>
   <xs:element name="elo2" type="telo2"
       minOccurs="elo2Min" maxOccurs="elo2Max"/>
   ...
   <xs:element name="elon" type="telon"
       minOccurs="elonMin" maxOccurs="elonMax"/>
 </xs:sequence>
</xs:complexType>
```

Cria-se um tipo complexo para o elemento pai. Depois, especifica-se a sequência de sub-elementos referindo, para cada um, o nome (*name*), e o número mínimo (*minOccurs*) e máximo (*maxOccurs*) de ocorrências contínuas.

4.3.5 Restrições complexas - unicidade

```
Não é possível especificar...
```

4.3.6 Outras Restrições complexas

Aparentemente, nenhuma das restantes restrições tratadas nas outras duas abordagens pode ser especificada com XML-Schema. Por exemplo, para o caso particular em que se quer que uma restrição (de um dos tipos especificados acima) seja aplicada apenas se um determinado elemento/atributo tiver um valor específico, não se pode produzir uma restrição.

5 Templates Gerais

Olhando para os templates da secção anterior, é fácil aperceber-nos das fortes semelhanças entre as linguagens de especificação de restrições. Do ponto de vista do construtor de compiladores, podemos dizer que partilham a sintaxe abstracta com algumas diferenças de pouco relevo.

Estando interessados em criar templates para os vários tipos de restrições já enumerados, extraímos a gramática abstracta de cada template. Até ao momento considerámos apenas as linguagens XCSL e Schematron. As linguagens como o XML-Schema exigem que seja produzido o documento completo para efectuar a validação das instâncias, não são escaláveis para um conjunto de restrições semânticas.

Na subsecção que se segue especifica-se a gramática abstracta para as restrições de domínio e de dependência entre dois elementos ou atributos numa notação formal de gramáticas.

5.1 Restrição de domínio

A sintaxe abstracta das restrições de domínio é a seguinte:

```
Constraint -> address1 address2 Test

Test -> min | max | minin |maxin
```

em que:

address1 caminho para o nodo relevante na árvore documental abstracta;

address2 complementa o caminho escrito em *address1* — pode ser o nodo actual (".") ou um seu atributo;

TestList lista na qual se escrevem os limites do domínio (*minin* e *maxin* são usados para incluir os limites).

5.2 Dependência entre dois elementos ou atributos

Para as restrições de dependência entre dois elementos ou atributos, tem-se a sintaxe abstracta seguinte:

```
Constraint -> address1.1 adress1.2 address3 relop
```

em que:

address1.1 caminho para o nodo relevante (o ponto principal para a comparação) na árvore documental;

address1.2 complementa o caminho escrito em *address1* — pode ser o nodo actual (".") ou um seu atributo;

address3 caminho para o segundo argumento da comparação.

relop operação relacional a avaliar.

Para os outros tipos de restrições a sintaxe abstracta é obtida de modo análogo, pelo que nos abstemos de a colocar aqui.

6 A linguagem XTC

Para descrever os templates genéricos em XML, desenvolveu-se o XTC — uma linguagem com a qual se descrevem os parâmetros das restrições pretendidas e se indica a linguagem de restrições escolhida, sem ter que conhecer a sintaxe específica dessa linguagem.

Com um documento XTC pode gerar-se todos os ficheiros de especificação de restrições, ou seja, um ficheiro de especificação de restrições XCSL, Schematron ou ambos.

O DTD que se segue representa a gramática desta linguagem (extraímos as linhas dos elementos terminais, que são *#PCDATA*):

```
<!ELEMENT xtc (rest)+>
<!ATTLIST xtc
    lang (XCSL | Schematron) #REQUIRED
    date CDATA #IMPLIED
    version CDATA #IMPLIED>
<!ELEMENT rest (path, comp?, relop?, fixed*, digits*, subel*, seq?,
    message?)>
<!ATTLIST rest
    kind (domain | dependency | regexp | mixedcont | unicity)
        #REQUIRED>
<!ELEMENT path (path1, node1?, path2?, node2?)>
<!ELEMENT comp (min?, max?)>
<!ATTLIST comp
    inc (min | max | both) #IMPLIED>
<!ELEMENT fixed (first, number, value, relop?)>
<!ELEMENT digits (first, numberc, numberd, relop?)>
<!ELEMENT subel (name, card)>
<!ELEMENT seq (elt)+>
```

```
<!ELEMENT message (submesg)+>
<!ATTLIST submesg
    type (1 | 2 | 3) #REQUIRED>
```

Não podendo apresentar um exemplo completo, com todos os tipos de restrições, devido à sua extensão, escolhemos adaptar o exemplo da Certidão Fiscal por forma a mostrar como é que se processa a escrita das restrições em XTC. Nas sub-secções que se seguem apresenta-se uma restrição de domínio e uma restrição de dependência entre dois elementos ou atributos, escritas em XTC.

Como se verá mais à frente, cada um dos documentos XML que se apresentam nas sub-secções seguintes, ao passar pela arquitectura de três níveis, é transformado num validador para aquela restrição na linguagem escolhida.

Um requerimento de Certidão Fiscal é feito aquando do falecimento de uma pessoa que deixou bens, depois da relação de bens, para comprovar que determinados bens constam de uma herança e que foram cumpridos os deveres fiscais devidos.

6.1 Restrição de domínio

Vamos supor que toda e qualquer data do documento (neste caso a data do falecimento e a data do pedido) deve estar entre os anos 1900 e 2990. Esta é uma restrição de domínio, pelo que deveremos usar os parâmetros (a que correspondem elementos XTC) *path*, *comp* e *message*, como se mostra no exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE xtc SYSTEM "xtc.dtd">
<xtc lang="XCSL">
  <rest kind="domain">
    <path>
      <path1>//data</path1>
      <node1>@valor</node1>
    </path>
    <comp inc="min">
      <min>19000000</min>
      <max>29900000</max>
    </comp>
    <message>
      <submesg type="1">A data do pedido: </submesg>
      <submesg type="2">/cert_fis/corpo/pedido/data</submesg>
      <submesg type="1">não está entre 1900 e 2990!</submesg>
    </message>
  </rest>
</xtc>
```

Este documento XML especifica a restrição de domínio enunciada: *path* tem os sub-elementos *path1* e *node1* instanciados; em *comp* especifica-se que o ano de-

verá ser posterior ou igual (o atributo *inc* vale 'min') a 1900 e anterior a 2990; em *message* (mensagem a devolver ao utilizador) especifica-se, nos sub-elementos *submesg* com atributo *type* igual a '1' o texto corrido, e naquele com atributo *type* igual a '2', o caminho para o elemento cujo valor se pretende apresentar. Neste caso atribuímos o valor 'XCSL' ao atributo *lang*, pedindo assim a geração de um ficheiro de restrições em XCSL.

Recebendo este documento, o processador XTC gera um documento escrito em XCSL, como pretendido:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
    generated by: XTC language processor
    Copyright (C) 2003
    Marta Jacinto, José C. Ramalho, Pedro Henriques
    GePL (DI, Uminho, Portugal)
    Version: 1.1  2003.02.09
-->
<!DOCTYPE cs SYSTEM "xcsl.dtd">
<cs>
  <constraint>
    <selector selexp="//data"/>
    <cc>
      ( @valor &gt;= 19000000 )
      and
      ( @valor &lt; 29900000 )
    </cc>
    <action>
      <message>A data do pedido:
        <value selexp="/cert_fis/corpo/pedido/data"/>
        não está entre 1900 e 2990!
      </message>
    </action>
  </constraint>
</cs>
```

6.2 Dependência entre dois elementos ou atributos

Num requerimento de Certidão Fiscal, exige-se que a data do pedido seja posterior à data de falecimento da pessoa a quem se refere o pedido. Para garantir que esta restrição se verifica, torna-se necessário comparar as duas datas referidas acima. Esta é uma restrição de dependência entre dois elementos ou atributos, pelo que deveremos usar os parâmetros (a que correspondem elementos XTC) *path*, *relop* e *message*, como se mostra no exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

```
<!DOCTYPE xtc SYSTEM "xtc.dtd">
<xtc lang="Schematron">
  <rest kind="dependency">
      <path>
          <path1>//pedido/data</path1>
          <node1>@valor</node1>
          <path2>/cert_fis/fecho/data/@valor</path2>
      </path>
      <relop>&lt;</relop>
      <message>
          <submesg type="1">A data de falecimento indicada: </submesg>
          <submesg type="2">/cert_fis/corpo/pedido/data</submesg>
          <submesg type="1">é posterior à data do pedido: </submesg>
          <submesg type="2">/cert_fis/fecho/data</submesg>
      </message>
  </rest>
</xtc>
```

Este documento XML especifica a restrição de domínio enunciada: *path* tem os sub-elementos *path1*, *node1* e *path2* instanciados (os dois primeiros para o primeiro ponto da comparação e o terceiro para o segundo ponto da comparação); *relop* especifica o operador a utilizar para a comparação (neste caso menor, que se representa pela entidade <); em *message* (mensagem a devolver ao utilizador) especifica-se, nos sub-elementos *submesg* com atributo *type* igual a '1' o texto corrido, e naqueles com atributo *type* igual a '2', caminhos para os elementos cujo valor se pretende apresentar. Neste caso atribuímos o valor 'Schematron' ao atributo *lang*, pedindo assim a geração de um ficheiro de restrições em Schematron.

Recebendo este documento, o processador XTC gera um documento escrito em Schematron, como pretendido:

```
<?xml version="1.0" encoding="iso-8859-1"?>
...
<schema>
  <pattern name="p2">
    <rule context="//pedido/data">
    <assert test="@valor&lt;/cert_fis/fecho/data/@valor"
                                          diagnostics="d2"/>
  </rule>
  </pattern>
  <diagnostics>
    <diagnostic id="d2">A data de falecimento indicada:
    <value-of select="/cert_fis/corpo/pedido/data"/> é posterior à data
    do pedido: <value-of select="/cert_fis/fecho/data"/>
    </diagnostic>
  </diagnostics>
</schema>
```

7 Implementação

As linguagens de especificação de restrições baseadas em regras como o XCSL e o Schematron estão implementadas sobre o XSLT naquilo a que se pode chamar uma arquitectura de dois níveis.

7.1 Arquitecturas de dois níveis

Por arquitectura de dois níveis, queremos dizer que alguém (o utilizador ou algum programa como por exemplo uma página web que ofereça uma interface ao utilizador) está consciente de que existem (e tem que seguir) dois passos por forma a obter o resultado final.

Como se disse na secção 3, o XCSL e o Schematron partilham a metodologia de implementação. Ambos os autores se aperceberam que se usassem XSLT para especificar as restrições, os processadores funcionariam, sem qualquer alteração, em qualquer plataforma.

Contudo, o utilizador ter que especificar uma folha de estilos XSL cada vez que pretende verificar algumas restrições, pode ser uma tarefa complexa e morosa. Assim, uma linguagem XML que "esconda" é tremendamente útil já que os detalhes "chatos" e a complexidade ficam do lado do processador XSL (único para aquela linguagem XML).

Em suma, se pretendermos usar o XCSL ou o Schematron, basta-nos seguir estes passos:

1. Especificar as restrições num documento XML (usando XCSL ou Schematron).
2. Usar um processador XSL para processar o documento XML usando a folha de estilos especial para a linguagem que estamos a utilizar. Este processador vai produzir uma folha de estilos que implementa completamente as restrições especificadas no passo anterior.
3. Validar as instâncias XML aplicando a folha de estilos gerada no passo anterior às várias instâncias XML da família de documentos.

Neste artigo propomos um terceiro nível de abstracção — abstracção da linguagem de especificação de restrições para "esconder" os detalhes do utilizador.

7.2 O processador XTC, uma arquitectura de três níveis

A arquitectura de três níveis que propomos pode ser descrita pelos passos que descrevemos de seguida e que será necessário seguir para obter a validação dos documentos XML:

1. Especificar as restrições num documento XML (usando o XTC). A escolha da linguagem final é feita através de um parâmetro.

2. Usar um processador XSL para processar o documento XML do passo anterior usando a folha de estilos especial: uma folha de estilos geradora de restrições de nível 2. Este processador vai produzir um documento XML em XCSL, Schematron ou XML-Schema (por enquanto ainda só numa das duas primeiras) dependendo do valor do parâmetro.

3. Usar um processador XSL para processar o documento XML gerado no passo anterior usando a folha de estilos especial para a linguagem que estamos a utilizar. Este processador vai produzir uma folha de estilos que implementa completamente as restrições especificadas no primeiro passo.

4. Validar as instâncias XML aplicando a folha de estilos gerada no passo anterior às várias instâncias XML da família de documentos.

Na secção 6 mostraram-se os dois primeiros passos desta arquitectura.

Poder-se-ia dizer que o preço que pagamos por ter mais níveis de abstracção é o aumento de complexidade na altura da implementação, mas esta abordagem possibilitará prosseguir com o estudo das transacções em tempo real.

8 Conclusão

Presentemente estão a ser discutidos nos comités W3C e ISO uma nova linguagem de esquema XML e talvez até mesmo um novo modelo de processamento XML. Tendo avaliado o que havia de diferente em várias linguagens e criado uma abstracção com as características melhores de cada uma delas, podemos contribuir activamente para esta nova linguagem.

Outra ideia que surgiu deste trabalho tem a ver a especificação de uma estrutura XML Pipe. As arquitecturas em vários níveis estão a ser desenvolvidas cada vez mais e há a necessidade de tornar os vários níveis transparentes para o utilizador. Há uma proposta no W3C para uma linguagem chamada "XML Pipe Definition Language", mas ainda não há qualquer implementação. Não obstante, a implementação de tal sistema levanta questões interessantes como: o que acontece se um componente falhar? (a resposta mais óbvia é abortar todo o processo, mas há respostas mais interessantes como ignorar ou tentar compensar); podemos ter sequências de componentes paralelas? (nesta situação, o contexto de falha é mais complexo...). Trabalhar nesta linha, criar uma álgebra para transacções em tempo real e encontrar uma implementação adequada para essa álgebra é bastante interessante e é o que se seguirá no futuro próximo.

No que diz respeito ao XTC, continuaremos a incorporar os tipos de restrições.

Referências

[1] L. Dodds. Schematron: Validating xml using xslt. In *XSLT UK Conference*, Keble College, Oxford, England, 2001.

[2] Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Ian Stokes-Rees, Kevin Williams, Kurt Cagle, Nikola Ozu, and Jeni Tennison. *Professional XML-Schemas*. Wrox Press, 2001.

[3] M. Jacinto, G. Librelotto, J. C. Ramalho, and P. Henriques. Constraint specification languages: comparing XCSL, Schematron and XML-Schemas. In *XML Europe'2002*, Barcelona, Spain, May 2002.

[4] M. H. Jacinto. Validação semântica em documentos xml. Master's thesis, Dep. Informática, Escola de Engenharia, Universidade do Minho, Outubro 2002.

[5] M. H. Jacinto, G. R. Librelotto, J. C. L. Ramalho, and P. R. Henriques. XCSL: XML Constraint Specification Language. In *XXVIII Conferencia Latino Americana de Informática (CLEI02)*, Uruguai, 2002.

[6] J. C. Ramalho. *Anotação Estrutural de Documentos e sua Semântica*. PhD thesis, Dep. Informática, Escola de Engenharia, Universidade do Minho, July 2000.

[7] J. C. Ramalho and P. Henriques. Constraining content: Specification and processing. In *XML Europe'2001*, Internationales Congress Centrum (ICC), Berlin, Germany, Apr. 2001.