Document Composer: uma aplicação XML para extracção de informação de repositórios XML

José Carlos L. Ramalho

Departamento de Informática, Universidade do Minho
{jcr}@di.uminho.pt

Resumo *Document Composer* é uma ferramenta desenvolvida como um exercício de reflexão em XSLT e que nasce da tentativa de criar um nível de abstracção para alunos dum curso de XML. Com este nível de abstracção pretendia-se que o aluno conseguisse realizar tarefas para as quais já havia adquirido os conceitos mas não a capacidade técnica.
A ferramenta permite que um utilizador interrogue um repositório de documentos XML (os documentos poderão pertencer a várias classes, i.e., estar de acordo com DTDs ou Schemas diferentes ou, simplesmente serem bem formados). O resultado de cada operação de interrogação é um novo documento XML. A ferramenta foi concebida de maneira a que os resultados individuais de cada interrogação possam ser combinados da forma que o utilizador entender para produzir o documento resultado (a composição é feita declarativamente na definição da estrutura do novo documento).
O processo de interrogação/extracção de informação é especificado numa linguagem XML desenvolvida para o efeito. Com esta linguagem o utilizador especifica o esqueleto de um novo documento e indica, através de interrogações (*queries*) escritas em XPath, como é que vai povoar cada componente do novo documento com a informação extraída dos vários documentos no repositório.
Neste artigo, serão descritos os passos seguidos para a criação desta ferramenta e alguns casos de estudo onde ela já foi utilizada.

1 Introdução

A ideia original para a construção desta ferramenta surgiu quando, numa aula de um curso sobre tecnologia XML, foi necessário colocar os alunos a trabalhar com XPath antes destes saberem construir *stylesheets* XSL ou manipular sistemas baseados em XQuery. O requisito principal era dar aos alunos a capacidade de seleccionar partes de documentos XML tendo apenas conhecimento do XPath.

Existem algumas ferramentas que permitem visualizar o resultado de uma expressão de selecção especificada em XPath. Por exemplo, o XMLSpy [Alt] incorpora um utilitário que permite o cálculo de expressões XPath fornecendo ao utilizador uma lista dos nodos da árvore documental abstracta que podem ser visualizados e manipulados graficamente por este. No nosso caso, pretendia-se um pouco mais. Desejava-se que o resultado da selecção fosse retornado como um novo documento XML para depois se poder processá-lo e transformá-lo noutro documento ou simplesmente poder navegar sobre ele.

Para resolver este problema resolveu-se utilizar o XSLT como motor de processamento. O seu modelo de programação segue o paradigma de programação por regras, ou seja, padrão-acção. E em XSLT, o XPath é usado para os padrões de selecção, i.e., as partes do documento às quais irão ser aplicadas acções são seleccionadas por expressões XPath. A ideia de usar o XSLT como motor de cálculo baseado em regras surgiu em 1999 com o XCSL [Ram01], e o Schematron [Jel04]. Estas aplicações adicionam um nível semântico aos DTDs e Schemas que pode ser especificado em regras escritas em XPath. Nos anos que se seguiram, surgiram mais algumas aplicações deste tipo em áreas diferentes mas onde era necessário criar um nível de abstracção, por exemplo: a arquitectura MTRANS [PBG01] desenvolvida para a conversão de modelos MOF e o GRAPHOTRON [Nic00] para a definição de vistas sobre grafos.

Logo à partida, surgiu um problema. Os alunos desconheciam o XSL e portanto não poderiam especificar as *stylesheets*. Consequentemente, estas teriam de ser geradas automaticamente.

O processo de geração automática de *stylesheets* é um processo de geração a dois níveis, envolvendo aquilo que normalmente se designa por *stylesheet genérica de segunda geração* (nalguns artigos são também referidas como *stylesheets* com reflexão [KK03,Kos03]). Uma transformação XSL de segunda geração é uma transformação (um pouco abstracta) que vai transformar um documento XML, que contem a especificação de um determinado processo, e que vai gerar como resultado uma nova transformação XSL (mais específica) que implementa o processo especificado. Esta nova *stylesheet* quando aplicado a um determinado documento XML irá então produzir o resultado final esperado. Há vários exemplos de aplicações que utilizam esta metodologia para criar níveis de abstracção encapsulando, desta forma, alguma da complexidade dos processos que implementam.

Ao longo deste artigo, começa-se por descrever a evolução da ferramenta até ao seu estado actual. Continua-se com a discussão da estrutura actual da ferramenta e de cada um dos seus componentes. A seguir apresenta-se um caso de estudo que exemplifica a utilização da ferramenta. Por fim, apresentam-se algumas conclusões e apontam-se algumas linhas para trabalho futuro.

2 Do *XPath Wrapper* ao *Document Composer*

Para, colocar alunos, que não tinham conhecimentos de XSL, a trabalhar com XPath como se esta fosse uma linguagem de query houve que criar um nível de abstracção que tornasse a complexidade de criação duma *stylesheet XSL* invisível para o utilizador. O nível de abstracção consistiu numa especificação XML composta por uma sequência de expressões XPath devidamente colocadas no seio de alguns elementos especificamente criados para o efeito, ou seja, as expressões XPath são colocadas como conteúdo de elementos XML e cada uma corresponde a uma query sobre o documento alvo da pesquisa.

2.1 Especificação de queries

Para a linguagem XML de especificação foi desenvolvido um pequeno DTD que se apresenta a seguir:

```
<!ELEMENT query-set (query+)>
<!ELEMENT query (#PCDATA)>
```

O conteúdo do elemento *query* deverá ser uma das expressões XPath a serem calculadas.

Considere o seguinte exemplo.

Exemplo 1: Queries sobre o Arquivo Sonoro de Ernesto Veiga de Oliveira

---

Suponha que possuía uma versão do Arquivo Sonoro de Ernesto Veiga de Oliveira em XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <arq>
  <doc>
    <prov>Alentejo</prov>
    <local>Santa Vitória, Beja</local>
    <tit>Disse a laranja ao limão</tit>
    <musico>Jorge Montes Caranova</musico>
    <file t="MP3">d1/evo001.mp3</file>
    <duracao>1:02</duracao>
  </doc>
- <doc>
    ...
  </doc>
  ...
</arq>
```

e que sobre ele queria calcular as seguintes queries:

1. Seleccionar o título das músicas que contêm a palavra "Jesus"no título.
2. Seleccionar o nome de todos os músicos referidos.
3. Seleccionar o título de todas as músicas de Castelo Branco.

Estas queries seriam então especificadas na linguagem descrita acima:

```
<?xml version="1.0"?> <query-set>
  <query>/arq/doc/tit[contains(.,'Jesus')]</query>
  <query>//musico</query>
  <query>//doc/tit[contains(../local),'Castelo Branco')]</query>
</query-set>
```

Como se pode ver do exemplo, o utilizador desta ferramenta apenas precisa de conhecer o XPath. Na secção seguinte, apresenta-se o método seguido para o cálculo das queries.

2.2 Cálculo das queries

A maneira mais fácil de processar uma expressão XPath é imbuti-la numa stylesheet XSL e processar o documento XML sobre o qual se querem calcular as queries com esta stylesheet.

Para isso, desenvolveu-se uma stylesheet XSL especial (vamos designá-la por XPP - "XPath Processor") que implementa este nível de abstracção. Esta stylesheet quando aplicada a um documento XML com a especificação de queries gera uma stylesheet XSL mais específica que quando aplicada ao documento produz o resultado do cálculo das queries.

A seguir mostra-se o resultado da aplicação de XPP ao documento com as queries apresentado no exemplo 1.

Exemplo 2: Stylesheet XSL que implementa o cálculo das queries

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<axsl:stylesheet
  xmlns:axsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <axsl:output method="xml" omit-xml-declaration="no"
    encoding="iso-8859-1" standalone="yes" indent="yes"/>

  <axsl:template match="/">
    <doc-result>
      <result-set id="q0" qexp="/arq/doc/tit[contains(.,'Jesus')]">
        <axsl:apply-templates mode="Q0"/>
      </result-set>
      <result-set id="q1" qexp="//musico">
        <axsl:apply-templates mode="Q1"/>
      </result-set>
      <result-set id="q2" qexp="//doc/tit[../prov='Castelo Branco']">
        <axsl:apply-templates mode="Q2"/>
      </result-set>
    </doc-result>
  </axsl:template>

   <axsl:template mode="Q0" match="/arq/doc/tit[contains(.,'Jesus')]">
     <result>
       <axsl:copy-of select="."/>
     </result>
   </axsl:template>

   <axsl:template match="text()" priority="-1" mode="Q0"/>

   <axsl:template match="text()" priority="-1"/>
</axsl:stylesheet>
```

Esta stylesheet é um pouco diferente das stylesheets que especificam no dia-a-dia:

linha 1-33 – o prefixo utilizado não é o `xsl` e sim `axsl`; isto deve-se ao facto da stylesheet ter sido gerada e por isso estes elementos tiveram de coexistir na stylesheet XPP com elementos do XSL que estariam activos; a maneira de se conseguir distinguir na mesma stylesheet elementos XSL activos de elementos XSL que se estão a gerar como resultado é associando estes ao mesmo `Namespace` mas com um prefixo diferente.

linha 10-22 – esta é a template principal que coordena todo o processo de cálculo; como se pode ver o documento resultado segue também uma estrutura que se pode especificar no seguinte pseudo DTD:

```
<!ELEMENT doc-result (result-set+)>
<!ELEMENT result-set (result+)>
<!ATTLIST result-set
              id ID #REQUIRED
              qexp CDATA #REQUIRED>
<!ELEMENT result (subárvore do documento original)>
```

Referimo-nos a este DTD como pseudo DTD pois o conteúdo de `result` é uma subárvore de elementos que depende da instância que está a ser processada e da expressão de query e por isso sempre variável, o que inviabiliza a sua formalização.

linha 13,16,19 – estas linhas três linhas contêm a invocação de três travessias à instância documental (atributo `mode`), uma por cada query especificada; não é possível calcular as queries em simultâneo pois nada nos garante que estas sejam disjuntas e todos os processadores de XSL exigem que as templates sejam disjuntas na sua selecção de nodos.

linha 24-28 – esta `template` corresponde à implementação da primeira query; vai copiar cada uma das subárvores, cujo nodo raiz obedeça à expressão XPath especificada na query, para dentro dum elemento `result` no documento resultante.

linha 30 – como as travessias iniciadas na `template` principal são globais, vão visitar todos os nodos, é necessário filtrar aqueles que não interessam; é o que faz esta `template` para a travessia `Q0`.

linha 32 – esta `template` à semelhança da anterior filtra nodos residuais na travessia principal, que nesta aplicação apenas é utilizada para desencadear as outras travessias.

No exemplo seguinte, apresenta-se o resultado da aplicação desta stylesheet mais específica ao arquivo sonoro de Ernesto Veiga de Oliveira.

Exemplo 3: Resultados finais

Depois da aplicação da stylesheet específica o documento XML resultante tem a seguinte forma:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-result>
   <result-set id="q0" qexp="/arq/doc/tit[contains(.,'Jesus')]">
      <result>
         <tit>Versos ao Menino Jesus</tit>
      </result>
      ...
   </result-set>
   <result-set id="q1" qexp="//musico">
      <result>
         <musico>Jorge Montes Caranova</musico>
      </result>
      <result>
         <musico>Catarina Chitas</musico>
      </result>
      ...
   </result-set>
   <result-set id="q2"
        qexp="//doc/tit[contains(../local,'Castelo Branco')]">
      <result>
         <tit>Dança dos Homens;
            dedicada a Nª Senhora dos Altos Céus</tit>
      </result>
      ...
   </result-set>
</doc-result>
```

Como se pode ver, cada `result-set` tem a ele associados um identificador e a expressão XPath que o originou. Por acaso, neste exemplo, todos os nodos seleccionados eram nodos folha na árvore documental da instância, se fossem nodos intermédios cada `result` conteria a respectiva subárvore completa.

2.3 *Pretty-print* dos resultados

Uma vez que os resultados são gerados estruturadamente num documento XML, ficam à distância duma stylesheet XSL de qualquer outra representação. Para já, desenvolveu-se uma stylesheet XSL que gera uma página HTML que permite uma fácil navegação nos resultados das queries.

2.4 Pipeline de execução

Nos exemplos apresentados, da especificação das queries até aos resultados finais passamos por três processos de transformação encadeados em série. Actualmente esta cadeia de processos está implementada num ficheiro `bat` mas está prevista uma solução mais genérica baseada em XPipe [DuC02,PS03]. Com efeito, já se realizou um projecto

[RTFR04] cujo objectivo era encapsular estas pipelines de execução por detrás de um Web Service.

A figura 1 ilustra as duas primeiras etapas desta cadeia de transformações.

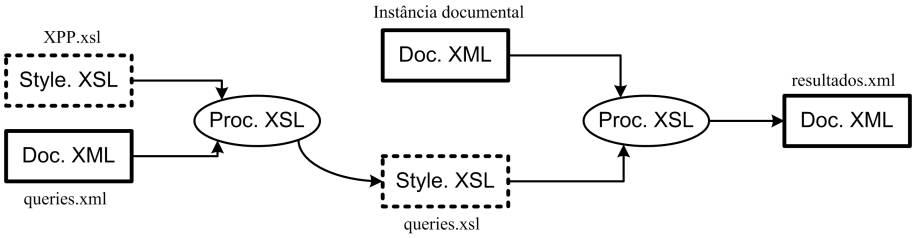

Figura 1. Pipeline de execução

Na próxima secção apresenta-se o `Document Composer` que resultou da generalização do `XPath Wrapper` permitindo ao utilizador combinar resultados de queries colocadas a vários documentos compondo deste forma novos documentos.

3 *Document Composer*

Depois de alguma experiência com o `XPath Wrapper` em contextos lectivos e aplicacinais, surgiu a ideia de tentar adaptá-lo para permitir colocar queries de uma só vez a todo um repositório documental em lugar de a um documento apenas.

Para isso, os vários componentes do sistema tiveram que sofrer alterações e alguns compromissos tiveram que ser estabelecidos.

Nas secções seguintes, descrevem-se as alterações, mais profundas nuns casos do que noutros, introduzidas nos vários componentes da ferramenta.

3.1 Especificação de queries

As diferenças tiveram que começar a ser introduzidas logo no módulo inicial. Assim, a linguagem para especificação de queries sofreu algumas alterações no sentido de generalizar o sistema de interrogação e de permitir uma maior facilidade na composição dos resultados finais.

Na figura 2, apresenta-se a estrutura do XML Schema desenvolvido para a nova linguagem.

Em que:

– `new-doc` é o elemento raiz da especificação e é composto por um subelemento `target`, que identifica o elemento raiz do documento resultado, e por um mais elementos `part`, cada um correspondendo a um conjunto de queries.

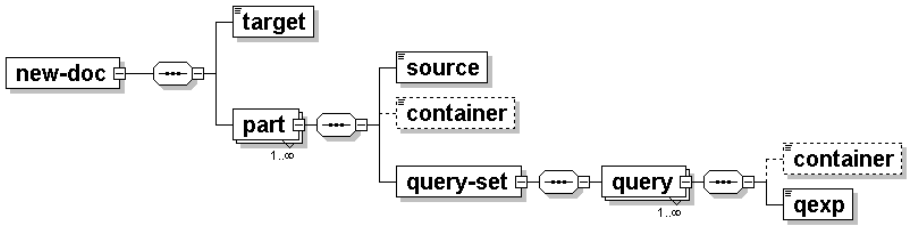

Figura 2. Estrutura da *Query Language* do `Document Composer`

– Cada elemento `part` é constituído por um triplo: um elemento `source` que indica o documento XML sobre o qual irão ser calculadas as queries, um elemento opcional `container` que identifica o elemento no documento resultado que irá conter os resultados deste conjunto de queries e, um elemento `query-set` que contem um conjunto de queries.
– O elemento `query-set` manteve a sua definição anterior e é composto por um ou mais elementos `query`.
– O elemento `query` é composto por um par: um elemento `container` que é opcional e que indica o elemento no documento resultado que irá conter o resultado da query em causa e, um elemento `qexp` que irá conter a expressão XPath que especifica a query.

O exemplo seguinte mostra uma aplicação concreta do sistema.

Exemplo 4: Lista de publicações e autores

Imagine que guardava num documento XML o registo das suas publicações. Para optimizar, um autor só é preenchido na primeira vez que aparece, nas vezes subsequentes é utilizado por referência. Acabou a de receber uma solicitação para o relatório anual: a sua lista de publicações no ano de 2003. Aparentemente bastaria uma query mas devido às referências aos autores é preciso criar um novo documento com duas partes: a lista de publicações e a lista de autores. A especificação em `Document Composer` seria a seguinte:

```
<?xml version="1.0" encoding="UTF-8"?> <new-doc>
  <target>minhaspubs2003</target>
  <part>
    <source>minhaspubs.xml</source>
    <container>autores</container>
    <query-set>
      <query>
        <qexp>//author</qexp>
      </query>
    </query-set>
  </part>
```

```
12   <part>
13     <source>minhaspubs.xml</source>
14     <container>pubs</container>
15     <query-set>
16       <query>
17         <qexp>//*[year='2003']</qexp>
18       </query>
19     </query-set>
20   </part>
21 </new-doc>
```

Está-se a criar um documento com a seguinte estrutura:

```
<?xml version="1.0" encoding="UTF-8"?> <minhaspubs2003>
  <autores>
    Todos os nodos referentes a autor
  </autores>
  <pubs>
    Todos os nodos com um filho year
    com conteúdo igual a 2003
  </pubs>
</minhaspubs2003>
```

Como se viu a principal modificação para além da especificação do documento fonte sobre o qual se querem calcular as queries, foi a introdução opcional da identificação de um elemento que irá conter os resultados de um grupo de queries.

3.2 Cálculo das queries

Para calcular as queries vai-se, de novo, gerar uma stylesheet XSL específica e proceder da mesma maneira. Como podem existir vários documentos de entrada, a pipeline de execução irá usar um documento XML produzido para este efeito e que nem sequer é analisado: `comp-driver.xml`.

Como na discussão do `Xpath Wrapper` não se apresentou a meta-stylesheet vai-se agora apresentá-la em pequenos excertos.

A template principal cria o elemento raiz (linha 15) e gera as várias travessias (linhas 16) para processar cada conjunto de queries.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:axsl="http://www.w3.org/1999/XSL/TransformAlias">
  <xsl:namespace-alias stylesheet-prefix="axsl" result-prefix="xsl"/>

  <xsl:output method="xml" omit-xml-declaration="no"
              standalone="yes" indent="yes"/>

```

```
  <xsl:template match="/">
    <axsl:stylesheet version="1.0">
      <axsl:output method="xml" omit-xml-declaration="no"
        encoding="iso-8859-1" standalone="yes" indent="yes"/>
      <axsl:template match="/">
        <xsl:element name="{new-doc/target}">
          <xsl:apply-templates select="new-doc/part"  mode="do-all-parts"/>
        </xsl:element>
      </axsl:template>
      <xsl:apply-templates/>
      <axsl:template match="text()" priority="-1">
        <!-- strip characters -->
      </axsl:template>
    </axsl:stylesheet>
  </xsl:template>
```

A template que trata os conjuntos de queries, elemento `part`, apenas coloca os resultados daquelas como conteúdo do elemento identificado pelo utilizador em `container`.

```
<xsl:template match="part" mode="do-all-parts">
    <xsl:choose>
      <xsl:when test="container">
          <axsl:element name="{container}">
              <xsl:apply-templates mode="do-all-parts" select="query-set"/>
          </axsl:element>
      </xsl:when>
      <xsl:otherwise>
           <xsl:apply-templates mode="do-all-parts" select="query-set"/>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Cada query é calculada sobre o documento identificado em `source` (linha 40) e numa travessia distinta (linha 39).

```
<xsl:template match="query" mode="do-all-parts">
  <axsl:apply-templates
    mode="P{count(../../preceding-sibling::*)}Q{count(preceding-sibling::*)}"
    select="document('{../../source}')"/>
</xsl:template>
```

Por fim, cada query é calculada na sua travessia específica, e o respectivo resultado é colocado no elemento `container` se este foi definido pelo utilizador (linhas 49-51), caso contrário, o resultado é simplesmente concatenado ao documento resultado (linha 54).

```
<xsl:template match="query">
  <axsl:template
    mode="P{count(../../preceding-sibling::*)}Q{count(preceding-sibling::*)}"
    match="{qexp}">
```

```
    <xsl:choose>
      <xsl:when test="container">
        <xsl:element name="{container}">
          <axsl:copy-of select="."/>
        </xsl:element>
      </xsl:when>
      <xsl:otherwise>
        <axsl:copy-of select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </axsl:template>
</xsl:template>
```

Nesta breve apresentação, foram omitidos alguns pormenores como a geração de comentários que documentam o processo e a filtragem dos nodos textuais que estão fora do contexto das expressões de query.

3.3 Produção de resultados

Uma vez que o utilizador pode "interferir"na estrutura do documento final de resultados não há possibilidade de definir uma stylesheet normalizada para os resultados. No entanto, pode sempre utilizar-se a stylesheet desenvolvida para o `XPath Wrapper` ou desenvolver outra. Para se poder utilizar a stylesheet do `XPath Wrapper` a especificação `Document Composer` teria de ser:

```
<?xml version="1.0" encoding="UTF-8"?> <new-doc>
  <target>doc-result</target>
  <part>
    <source>fonte.xml</source>
    <container>result-set</container>
    <query-set>
      <query>
        <container>result</container>
        <qexp>XPath exp</qexp>
      </query>
    </query-set>
  </part>
</new-doc>
```

3.4 Pipeline de execução

O Pipeline de execução é muito semelhante ao do `XPath Wrapper` e está, neste momento, implementado através dum ficheiro `bat`.

4 Conclusão e Trabalho Futuro

Quer o `XPath Wrapper` quer o `Document Composer` têm sido bastante úteis em contextos lectivos no entanto, acreditamos que também podem sê-lo em contextos

aplicacionais. Ainda não houve oportunidade de fazer-lhes testes sérios neste último domínio. Até agora, foram usados em exemplos de pequena dimensão mas está prevista a sua aplicação e mesmo inclusão em sistemas de maior porte como por exemplo um sistema para interrogação de um reposiório de documentos XML.

Quanto a trabalho futuro, existem muitas linhas de trabalho que se poderão desenvolver:

- Desenvolver duma interface gráfica integrada com o `Document Composer` que transforme este numa ferramenta WYSIWYG de composição documental.
- Desenvolver o suporte a Namespaces no `Document Composer`. Tornaria possível a optimização do número de travessias que se fazem para cálculo das queries.
- Permitir a associação de novos atributos aos elementos `container`.
- Normalizar os nomes dos elementos na linguagem de especificação, criar um Namespace próprio para a aplicação e criar uma lista de casos de uso.

Brevemente, estas ferramentas ficaão disponíveis na Web.

Referências

[Alt] Xmlspy website. http://www.altova.com.

[DuC02] Bob DuCharme. Maintaining schemas for pipelined stages. In *Proceedings of the XML Conference 2002*, Baltimore, USA, 2002.

[Jel04] Rick Jelliffe. Document schema definition language (dsdl) - part 3: Rule-base validation (schematron). `http://www.jtc1sc34.org/repository/0524c.htm`, Jun 2004.

[KK03] Oleg Kiselyov and Shriram Krishnamurthi. Sxslt: Manipulation language for xml. *Lecture Notes in Computer Science*, 2562:256 – 272, Jan 2003.

[Kos03] Jirka Kosek. Xslt reflection. `http://www.xml.com/pub/a/2003/11/05/xslt.html`, Nov 2003.

[Nic00] Miloslav Nic. Graphotron. `http://www.zvon.org/ZvonSW/ZvonGraphotron/`, 2000.

[PBG01] Mikael Peltier, Jean Bézivin, and Gabriel Guillaume. Mtrans: A general framework, based on xslt,for model transformations. In *WTUML01, Proceedings of the Workshop on Transformations in UML*, Genova, Italy, 2001.

[PS03] Michael Pediaditakis and David Shrimpton. Device neutral pipelined processing of xml documents. In *Poster Proceedings of the Twelfth International World Wide Web Conference, WWW2003*, 2003.

[Ram01] José Carlos Ramalho. Constraining content: Specification and processing. In *Proceedings of XML Europe 2001*, Berlin, Germany, May 2001.

[RTFR04] José Carlos Ramalho, Pedro Taveira, Ricardo Ferreira, and Vasco Rocha. Gerador de web services para cadeias de transformações de documentos xml. In *XATA2004, XML Aplicações e Tecnologias Associadas, 2ª Conferência Nacional*, Faculdade de Engenharia da Universidade do Porto, Fevereiro 2004.