XSLT 2.0 vs. XSLT 1.0

Jean-Michel Hufflen

HAL Id: hal-00661896
https://hal.science/hal-00661896

Submitted on 20 Jan 2012

XSLT 2.0 vs XSLT 1.0[*]

Jean-Michel HUFFLEN
LIFC (EA CNRS 4157)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

This article focuses on the new features introduced by Version 2.0 of XSLT, the language of transformations used for XML texts. We show why these new features — groups of XML subtrees, functions, interface with schemas — ease the development of some applications. Some examples, related to bibliography management, will be demonstrated.

Keywords   XPath 2.0, XSLT 2.0, Muenchian method, sequences, multiple outputs, character mapping, datatype binding.

Streszczenie

Artykuł dotyczy nowych własności wprowadzonych przy wersji 2.0 XSLT, języka transformacji dla tekstów XML-owych. Pokażemy dlaczego te nowe własności — grupy poddrzew XML-owych, funkcje, interfejs ze schematami — ułatwiają tworzenie pewnych aplikacji. Pokażemy przykłady z obszaru zarządzania bibliografiami.

Słowa kluczowe   XPath 2.0, XSLT 2.0, metoda Muencha, ciągi, wyjście wielostrumieniowe, przekształcanie znaków, wiązanie typów danych.

0   Introduction

This article follows [5, 6], which are introductions to XSLT[1], proposed to the attenders of the 2005 and 2006 BachoTeX conferences. As for these last two demonstrations, reading this article only requires basic knowledge about XML[2].

The first version (1.0) of XSLT [26], the language of transformations used for XML texts, has succeeded and is now widely used to perform some computations, to convert an XML text into another XML-like format, or to generate HTML[3] pages. However, some operations are difficult to perform with the basic constructs of Version 1.0 and this led to the design of new versions. The first attempt was Version 1.1 [29]. The main problem addressed by this proposal is the portability of XSLT stylesheets [27, § 3]: some functionalities tedious or impossible to express with XSLT's basic constructs can be implemented in XSLT 1.0 by means of *extensions* written using a 'more classical' programming language. But the available programming languages depend on the XSLT processor used: if you develop with xsltproc[4] [23], part of the GNOME[5] project, you can extend your stylesheets with C functions; if you develop with Xalan [2], part of the Apache project, your extensions may be written using Java [10] or C++ [21]; if you develop with Microsoft's MSXML3, you can use the ECMAScript[6] language; ... Moreover, even if several XSLT processors support the same programming language for writing extensions, they may do that in incompatible ways. Let us consider the XML text given in Figure 1, specifying the contents of some omnibus volumes. For each story included into such a book, we make precise its title and the year of its first publication. The XSLT program given in Figure 3 includes an example of

---

[*] Title in Polish: XSLT *2.0* versus XSLT *1.0*.

[1] e**X**tensible **S**tylesheet **L**anguage **T**ransformations.

[2] e**X**tensible **M**arkup **L**anguage. Readers interested in an introductory book to this formalism can refer to [19].

[3] **H**yper**T**ext **M**arkup **L**anguage. [17] is a good introduction to this language.

[4] We used this XSLT processor for the examples demonstrated in [5, 6].

[5] GNU **N**etwork **O**bject **M**odel **E**nvironment.

[6] This script language comes from JavaScript [3] and has been standardised by the ECMA (**E**uropean **C**omputer **M**anufacturers **A**ssociation).

Jean-Michel HUFFLEN

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

<books>
  <omnibus series="Doc Savage">
    <author>        <!-- The organisation of author elements is the same than in [9].  -->
      <name><personname><first>Kenneth</first><last>Robeson</last></personname></name>
    </author>
    <booktitle>Doc Savage Omnibus #9</booktitle>
    <year>1989</year>
    <story><title>The Invisible-Box Murders</title><year>1941</year></story>
    <story><title>Birds of Death</title><year>1941</year></story>
    <story><title>The Wee Ones</title><year>1945</year></story>
    <story><title>Terror Takes 7</title><year>1945</year></story>
  </omnibus>
  <omnibus>
    <author>
      <name><personname><first>Kenneth</first><last>Robeson</last></personname></name>
    </author>
    <booktitle>Doc Savage Omnibus #10</booktitle>
    <year>1989</year>
    <story><title>The Devil's Black Rock</title><year>1942</year></story>
    <story><title>Waves of Death</title><year>1943</year></story>
    <story><title>The Too-Wise Owl</title><year>1942</year></story>
    <story><title>Terror and the Lonely Widow</title><year>1945</year></story>
  </omnibus>
</books>
```

Figure 1: Specification of some stories collected in omnibus volumes.

such an extension: in addition to the standard output, another output file — named 'Doc Savage-years' when this stylesheet is applied to Figure 1's text — is created and contains all the years associated with stories, these years being sorted.

The main addition provided by XSLT 1.1 is an `xsl:script` element [29, § 14.4], allowing some additional functions to be directly included in XSLT texts. There is a wide choice among available programming languages, an `xsl:script` element — including a `language` attribute[7] — is ignored by an XSLT processor if it is unable to deal with a particular programming language.

This `xsl:script` element has been viewed only as a partial solution to the portability problem. In addition, the need for a deeper revision of XSLT appeared at this time. As a consequence, XSLT 1.1 did not go past the working draft stage, and the new 'official' version of XSLT is 2.0; here are the main requirements for it [28]:

- authoring extension functions should be allowed [28, § 2.6];
- grouping must be simplified and made more efficient [28, § 14];

- XML Schema [30] must be supported[8] [28, § 3]: in particular, it must be possible to construct XML Schema-typed elements and attributes.

In addition, XSLT 2.0 provides:

- *multiple output documents* for one XSLT program;
- *datatype binding*, allowing processing data according to their datatypes;
- *character mapping*, improving the error-prone character escaping of XSLT 1.0;
- *temporary trees*, replacing the *result tree fragments* of Version 1.0 [26, § 11.1], but more operations are permitted on temporary trees, that is, users can address parts by means of XPath expressions[9].

---

[7] Several implementations using different programming languages are allowed for a function.

[8] *Schemas* allow users to define types precisely, which makes more precise the validation of a XML text. A short comparative study of some schema languages, including XML Schema, is [24].

[9] In fact, this operation is permitted by most of XSLT 1.0 processors, after a conversion of the result tree fragments into a *node set* — which is the type used by XPath to handle parts of an XML document — but this functionality does not belong to standard XSLT 1.0. If you use Xalan, this can by done by the `xalan:nodeset`, the `xalan` namespace prefix being `http://xml.apache.org/xalan`. Another way is to

XPath, the language used to address parts of an XML text, has been revised, too [32, 33]. More precisely, XSLT 1.0 (resp. 2.0) uses expressions of XPath 1.0 [25] (resp. 2.0). In a first section, we briefly show what is new in XPath 2.0. Then Sections 2 to 7 give some illustrations of most features of XSLT 2.0. Of course, these following sections do not aim to replace the reference manuals [33, 35], they just give some representative idea of the improvements provided by XPath 2.0 and XSLT 2.0.

1 XPath 2.0's new features

In the XPath 2.0's data model [11, Ch. 2], every value is a *sequence*. An atomic value is a special case of a sequence: a one-element sequence. Some syntactic constructs allows all the elements of a sequence to be processed. As an example, let s be a sequence whose elements are the numbers 30, 4, 2008 — in XSLT 2.0, such a sequence may be introduced by:

```
<xsl:sequence select="30,4,2008"/>
```

anonymously or by:

```
<xsl:variable name="s" select="30,4,2008"
              as="xsd:integer+"/>
```

as a variable's value — :

- `for $x in $s return $x + 1`
  yields the sequence 31,5,2009;
- `every $x in $s satisfies $x gt 0`
  returns `true` because every number belonging to s is positive;
- `some $x in $s satisfies $x eq 0`
  returns `false`: zero does not belong to s.

An 'if' expression avoids using an `xsl:choose` element for simple conditional expressions:

```
if (empty($s)) then 0 else $s[1]
```

yields the first element of the s sequence if it is not empty, zero otherwise. Notice that:

- both branches — 'then' and 'else' — must be present within a conditional expression;
- given X and Y two XPath expressions, the two conditional expressions:

```
if (empty($s)) then X else Y
if ($s)        then Y else X
```

have two different meanings: in the first case, the test yields `true` for an empty sequence — '()' — `false` otherwise [11, Ch. 10]; in the second case, the test yields `false` for an empty string, an empty node set, and zero, it returns `true` for all the other values [20, pp. 77–80];

---

use the `node-set` function provided by the `common` module of EXSLT (**E**xtensions to XSLT) [20, App. A].

```
<items>
  <by-year year="1941">
    <title>The Invisible-Box Murders</title>
    <title>Birds of Death</title>
  </by-year>
  <by-year year="1942">
    <title>The Devil's Black Rock</title>
    <title>The Too-Wise Owl</title>
  </by-year>
  <by-year year="1943">
    <title>Waves of Death</title>
  </by-year>
  <by-year year="1945">
    <title>The Wee Ones</title>
    <title>Terror Takes 7</title>
    <title>Terror and the Lonely Widow</title>
  </by-year>
</items>
```

Figure 2: Grouping elements of Fig. 1's text.

- atomic values should be compared using the operators eq, ne, lt, le, gt, ge[10] — as we do in previous examples — whereas the operators =, !=, <, <=, >, >= are also allowed for sequences [11, Ch. 6].

Parentheses can be used throughout XPath expressions [11, Ch. 7]:

```
(if (author) then author else editor)/name
(:  If there is an author element, select its name
    child, otherwise, select the name child of the
    editor element.
:)
```

Notice that XPath 2.0's expressions can embed *comments*, surrounded by '(:' and ':)'.

XPath 2.0 offers more numerical data types than XPath 1.0. In particular, some numerical types defined in XML Schema — e.g., `xsd:integer` for the relative integers — are allowed [31]. Likewise, other data types for dates, times, and durations are provided.

XPath 2.0 also provides functions that allows the contents of strings to be processed w.r.t. regular expressions: `matches`, `replace`, `tokenize` [11, Ch. 10].

Last, let us mention that XQuery, a query language comparable to the languages used in database management, is an extension of XPath 2.0. A didactic introduction to XQuery is [36], the official document is [34].

---

[10] Resp.: '**EQ**uals', '**N**ot **E**quals', '**L**ess **T**han', '**L**ess than or **E**qual to', '**G**reater **T**han', '**G**reater than or **E**qual to'.

Jean-Michel HUFFLEN

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

<xsl:stylesheet version="1.0" id="grouping" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
                extension-element-prefixes="redirect">
  <!-- The extension-element-prefixes attribute gives the prefixes of extension functions or elements. If
       need be, these external elements (resp. functions) are invoked (resp. called) when the stylesheet is applied
       [26, § 14]. Xalan includes a Java class, org.apache.xalan.xslt.extensions.Redirect, providing three
       methods: open(), close(), and write().
    -->

  <xsl:output method="xml" encoding="ISO-8859-1" indent="yes"/>

  <xsl:key name="by-year" match="story" use="year"/>

  <xsl:template match="books">
    <xsl:variable name="the-stories" select="omnibus/story"/>
    <items>
      <xsl:for-each select="$the-stories[generate-id() = generate-id(key('by-year',year)[1])]">
        <!-- That is, each subtree corresponding to the first occurrence of a year, given by the first position of
             the node set returned by the key function.
          -->
        <xsl:sort select="year" data-type="number"/>
        <!-- Sort them w.r.t. year information. The data-type attribute defaults to text.  -->
        <by-year year="{year}"> <!-- Some attributes — e.g., select — are interpreted, but not all. In these
                                     last cases, the '{...}' notation forces the value to be evaluated as an
                                     XPath expression [26, § 7.6.2].
                                  -->
          <!-- Copy the complete fragments [26, § 11.3] of all the title elements of the stories coming out in
               the same year:
            -->
          <xsl:for-each select="key('by-year',year)"><xsl:copy-of select="title"/></xsl:for-each>
        </by-year>
      </xsl:for-each>
    </items>
    <xsl:if test="element-available('redirect:write')">
      <!-- This test allows us to invoke the beginning of this template even if the redirect:write extension
           element is unavailable. In other words, we can run this stylesheet with another XSLT processor than
           Xalan. If Saxon [13] is used as an XSLT 1.0 processor, the equivalent element is saxon:output, the
           saxon prefix being bound to the namespace http://icl.com/saxon.
        -->
      <redirect:write file="{@series}-years">
        <xsl:for-each select="$the-stories/year">
          <xsl:sort select="." data-type="number"/>
          <xsl:value-of select="concat(.,' ')"/>
        </xsl:for-each>
      </redirect:write>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Figure 3: Muenchian method of grouping elements in XSLT 1.0.

2   Grouping

Given the XML text given at Figure 1, let us try to group the titles of included stories by year. More precisely, we are seeking for the XML text given in Figure 2.

In XSLT 1.0, the only way to group elements is to define a key partitioning these elements[11]. Applying it to such an example results in a complex and memory-intensive method, called the **Muenchian**

---

[11] We already showed how to use keys in [6].

TUGboat, Volume 0 (2060), No. 0 — Proceedings of the 2060 Annual Meeting

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

<xsl:stylesheet version="2.0" id="grouping-plus" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="xml" encoding="ISO-8859-1" indent="yes"/>
  <xsl:output method="text" encoding="ISO-8859-1" name="additional-text"/>

  <xsl:template match="books">
    <xsl:variable name="the-stories" select="omnibus/story" as="element(story)*"/>
    <items>
      <xsl:for-each-group select="$the-stories" group-by="year">
        <xsl:sort select="xsd:integer(year)"/>   <!-- Shorthand for 'year cast as xsd:integer'. -->
        <!-- The contents of the year elements are coerced into integers, so they are sorted as numbers. -->
        <by-year year="{year}"><xsl:copy-of select="current-group()/title"/></by-year>
      </xsl:for-each-group>
    </items>
    <xsl:result-document href="{@series}-years" format="additional-text">
      <!-- The outputs performed when this element is invoked are stored into the file whose name is given by
           the href attribute.
      -->
    <xsl:perform-sort select="distinct-values($the-stories/year)">
      <!-- This 'new' element is used to receive the result of a sort, as a sequence [12, Ch. 5]. The
           distinct-values function eliminates the duplicates values of a sequence [11, Ch. 10]. Since the
           order of the values in the result sequence is undefined, we have to use it before sorting.
        -->
      <xsl:sort select="." data-type="number"/>
      <!-- Of course, using the data-type attribute still remains possible. -->
    </xsl:perform-sort>
    </xsl:result-document>
  </xsl:template>

</xsl:stylesheet>
```

Figure 4: Grouping elements by values in XSLT 2.0.

---

method, after Steve Muench of Oracle [16, § 6.2]. Let us look at Figure 3: the `by-year` key allows us to group all the `story` elements sharing the same year. We retain the `story` elements corresponding to the first occurrences of each year, given inside a `year` element. To do that, we label each `story` element with a unique identifier by means of the `generate-id` function [26, § 12.4]. Then these `story` elements are sorted according to the information about years [26, § 10]. Approaching our goal, for each `story` element corresponding to the first occurrence of a year, we consider all the `story` elements whose the information about the year is the same, given by the `by-year` key. At last (!), the `title` elements of each of these `story` elements are copied into the result, that is, they are embedded into a `by-year` element. In analogous applications, several keys may be needed, corresponding to group levels.

The implementation of the same functionality in XSLT 2.0 using *groups* — see Figure 4 — is indisputably easier to understand. . . and much more ef-ficient. The `story` elements sharing the same year information are grouped. Then the `current-group` function allows us to obtain the successive items — `story` elements in this example — that are members of the group we are processing [12, Ch. 7]. So when XPath 2.0's expression `current-group()/title` is applied, we get all the successive `title` elements of each member of this group. Let us remark that we need neither keys, nor generated identifiers associated with nodes of the source text.

The main element processing groups in XSLT 2.0 is `xsl:for-each-group`. More precisely, nodes of the XML text that are selected by an XPath expression given by the `select` attribute may be grouped since they share the same value, specified by the `group-by` attribute, as we do in Figure 4. This `xsl:for-each-group` element may be used with another attribute, `group-adjacent`, to group only adjacent elements. That is, the first item of adjacent elements starts a new group, and the subsequent item belongs to the same group if it shares the same

Jean-Michel HUFFLEN

value w.r.t. the `group-adjacent` attribute, otherwise, a new group is started. This rule is iterated until the end of adjacent elements. Two other possible attributes of this element — `group-starting-with` and `group-ending-with` — are also used to group adjacent elements. In the first (resp. second) case, just specify a pattern — as an XPath expression [12, Ch. 6] — matching the first (resp. last) element of each group. More details and many examples are given in [12, Ch. 5].

3 Multiple outputs

As mentioned in the introduction, we can implement this functionality in XSLT 1.0 by using some extension functions. Figure 3 shows how to proceed if you use Xalan. If another XSLT processor is used, this feature is protected by an `xsl:if` element, so no error occurs. However, a complete implementation should take each XSLT processor into account, and let us recall that XSLT processors do not have to provide it. Finally, let us notice that in this implementation provided by Xalan, the output mode is supposed to be the same than the main stream's — here, `xml` — which may be unsuitable in some applications.

On the contrary, the program given in Figure 4 runs under any processor of XSLT 2.0. There is a main output stream, and additional ones can be managed by means of the `xsl:result-document` element [12, Ch. 5]. Additional output methods can be specified, in which case they must have been given a `name` attribute. If the `format` attribute of the `xsl:result-document` is given, it must be such a name, otherwise the corresponding output stream refers to the default `xsl:output` element. About possible formats, let us mention that you may set the `method` attribute to `xhtml` for generating XHTML[12] outputs, in addition to the methods already known in XSLT 1.0: `html`, `text`, `xml`.

4 Functions

XSLT 2.0 allows the definitions of functions that may be used inside XPath expressions. These functions use XSLT's constructs, so they are more portable than XSLT 1.1's `xsl:script` elements. They must belong to a namespace. As shown by Figure 5, the `as` attribute allows us to make precise the type of the function's result, as well as the type of a parameter or a variable. Let us recall that such values are sequences, the '?' marker used inside `as` attributes denotes a zero-or-one-length sequence.

---

[12] e**X**tensible **H**yper**T**ext **M**arkup **L**anguage. This a reformulation of HTML using XML conventions. See [17] for more details.

```
<xsl:function name="add:month-position"
              as="xsd:integer">
  <!-- The add namespace prefix is supposed to be
       defined [19, pp. 41–45]. xsd is the prefix
       used for XML Schema's types.
  -->
  <xsl:param name="the-month"
             as="element(month)?"/>
  <xsl:variable
    name="the-index"
    select="index-of((('jan','feb',...,'dec'),
                      name($the-month/*[1]))"
    as="xsd:integer?"/>
  <!-- If the item does not belong to the sequence,
       the index-of function returns an empty
       sequence [11, Ch. 10].
  -->
  <xsl:value-of
    select="if (empty($the-index)) then 13 else
               $the-index"/>
</xsl:function>
```

Figure 5: XSLT function returning a month's number.

If we assume that the XPath expression `month` gives access to an element defined as follows[13]:

```
<!ELEMENT month (jan | feb | ... | dec)>
<!ELEMENT jan    EMPTY>
<!ELEMENT feb    EMPTY>
...
<!ELEMENT dec    EMPTY>
```

by using a DTD[14], the function given in Figure 5 returns the number of a month and can be used throughout an XPath expression, as follows:

```
<xsl:sort
  select="add:month-position(month)"/>
```

Since this function returns `13` when the optional element `month` is not found, items without month information are ranked at the end. When such a function is called, formal parameters — given by successive `xsl:param` elements — are bound to actual values regarding positions: the first parameter is bound to the first actual value, and so on. An actual value is required for each parameter of a function, so default values are not allowed for such parameters[15]. The

---

[13] XML trees generated by MlBIBTEX — our reimplementation of the BIBTEX bibliography processor [18] — are conformant to this definition [9].

[14] **D**ocument **T**ype **D**efinition. A DTD defines a document markup model, see [19, pp. 148–155] for more details.

[15] Concerning a parameter of a `xsl:template` element, a `required` attribute set to `yes` forces it to be passed explicitly when the template is invoked. In this case, default values are not allowed, either. This `required` attribute did not exist in XSLT 1.1 and defaults to `no`.

---

override attribute controls what happens if a user-written function and a vendor-supplied one have the same name. If it is set to **yes** (resp. **no**), the former (resp. latter) wins. It defaults to **no**.

5 Datatype binding

A complete description of this feature would be too long and outside the scope of this article, because it requires good knwoledge of the way used in XML Schema to define types. However, we can mention some simple cases showing that the type information is widely used in XSLT 2.0.

When a node set is to be sorted in XSLT 1.0, the **data-type** attribute defaults to **text**, that is, for a lexicographical sort [26, § 14]. So you have to set it to **number** for a numerical sort. When the key sort has been recognised as a number in XSLT 2.0, the sort is numerical by default [35, § 13.1]: an example is given in Figure 4. Another example is provided by the function given at Figure 5. If this function is used as a sort key, as shown in Section 4, we do not have to make precise the **data-type** attribute of the **xsl:sort** element, either, since our function includes the declaration **as="xsd:integer"**. The **as** attribute provides type information and can be associated with the elements **xsl:function**, **xsl:param**, **xsl:variable**, and **xsl:with-param**. Even if **as** attributes are not always needed, we recommend to use them as far as possible, as we do in our examples. Besides, this choice allows us to show the expressive power of the language of types: e.g., the use of '?', '*', '+', related to regular expressions[16]. If there is no **as** attribute, any value of any type is acceptable, that is equivalent to specifying ' **as="item()*"** '. Of course, type-checking may result in significant loss of efficiency, but it can help develop a stylesheet; it allows you to control what happens when data are passed by functions, variables or parameters. In addition, the specification of some operations may be made easier, as abovementioned about sorting.

Let *X* be an XPath expression and *T* be a type expression, the following operators, described in [11, Ch. 9], are usable inside XPath expressions:

X **cast as** *T* coerces *X* into an expression being type T (see Figure 4), if this operation fails, an error is signalled;

X **castable as** *T* returns **true** if the corresponding coercion operation would succeed, **false** otherwise;

X **instance of** *T* returns **true** if *X* is of type *T*, **false** otherwise;

---

[16] A complete specification of this language is given in [12, pp. 74–79].

X **treat as** *T* returns *X* if it is of type *T*, otherwise an error is signalled.

6 Character maps

Another new feature introduced by XSLT 2.0 is given by *character maps*, especially interesting if you derive source texts for the LaTeX word processor [15] from XML texts.

In XSLT 1.0, all the characters belonging to a constant string or the contents of an **xsl:text** element are copied *verbatim* into the result, except if the **disable-output-escaping** attribute — which defaults to **no** — is set to **yes**, in which case the entities specifying special characters used throughout XML texts [19, pp. 48–49] are replaced by the characters themselves [26, § 16.4]. For example:

```
<xsl:text disable-output-escaping="yes">
  ... Mickey &amp; Mallory...
</xsl:text>
```

will generate '... Mickey & Mallory...'. If we are interested in deriving texts suitable for LaTeX, we have to pay attention to LaTeX's special characters, that is, we have to produce '... Mickey \& Mallory...' for this example. The **translate** function [25, § 4.2] allows us to replace a character by another, or to remove a character[17]. But it cannot be used to replace a single character by a sequence of several characters. So processing LaTeX's special characters by means of XPath 1.0's functions is tedious[18].

As shown by Figure 6, the character maps of XSLT 2.0 allows the replacement of a single character by a string, by means of **xsl:output-character** elements. You can use several character maps for an output stream, given by the **use-character-maps** attribute[19]. We think that the best method consists of putting characters down as they must appear within the result. Control characters — e.g., '{' and '}' in LaTeX — are represented by positions belonging to a private use area of Unicode encoding [22]. Then the character map is used for characters that must be escaped or belong to the private use area.

---

[17] To remove a character, call the **translate** function with a source character with no corresponding position within the third argument. For example, **translate(*s0*,"$&","£")** returns a string in which all the occurrences of the '$' character (resp. the '&' character) in the *s0* string are replaced by the '£' character (resp. removed).

[18] This problem is solved in MlBIBTEX, which uses the **nbst** language, close to XSLT 1.0, for specifying bibliography styles. We have added the value **LaTeX** to the possible values for the **mode** attribute of the **nbst:output** element [4, App. A], analogous to the **xsl:output** element.

[19] What to do if several **xsl:output-character** declarations conflict is unspecified: an XSLT processor may report an error, or use the one that occurs last in the stylesheet.

Jean-Michel HUFFLEN

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

<!DOCTYPE stylesheet [<!ENTITY start-command "&#xE000;">
                       <!ENTITY start-group "&#xE001;">
                       <!ENTITY end-group "&#xE002;">
                       <!ENTITY start-math-mode "&#xE003;">
                       <!ENTITY end-math-mode "&#xE003;">]>

<xsl:stylesheet version="2.0" id="using-tex-map" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsl:output method="text" encoding="ISO-8859-1" use-character-maps="TeX-map"/>

  <xsl:strip-space elements="*"/>
  <!--  Rules blank nodes out from the source text: cf. [6] & [35, § 4.4].  -->

  <xsl:character-map name="TeX-map">
    <xsl:output-character character="#" string="\#"/>
    <xsl:output-character character="%" string="\%"/>
    <xsl:output-character character="$" string="\$"/>
    <xsl:output-character character="&amp;" string="\&amp;"/>
    <xsl:output-character character="\" string="$\backslash$"/>
    <xsl:output-character character="^" string="{\char&quot;5E}"/>   <!--  Using hexadecimal  -->
    <xsl:output-character character="_" string="\_"/>                <!--  code. In Plain TeX,  -->
    <xsl:output-character character="{" string="$\{$"/>   <!--  the commands '\{' and '\{' are  -->
    <xsl:output-character character="|" string="$|$"/>    <!--  only usable in mathematical mode  -->
    <xsl:output-character character="}" string="$\}$"/>   <!--  (cf. [14, Exercise 16.12]).  -->
    <xsl:output-character character="~" string="{\char&quot;7E}"/>
    <xsl:output-character character="£" string="{\it\char&quot;24}"/>
    <xsl:output-character character="&start-command;" string="\"/>
    <xsl:output-character character="&start-group;" string="{"/>
    <xsl:output-character character="&end-group;" string="}"/>
    <xsl:output-character character="&start-math-mode;" string="$"/>
    <xsl:output-character character="&end-math-mode;" string="$"/>
  </xsl:character-map>

  <xsl:variable name="eol" select="'&#xA;'" as="xsd:string"/>
  <!--  Convenient way to write the end-of-line character: cf. [6].  -->

  <xsl:template match="books">
    <xsl:apply-templates/>
    <xsl:value-of select="concat($eol,'&start-command;end',$eol)"/>   <!--  Yields '¶\end¶'.  -->
  </xsl:template>

  <xsl:template match="omnibus">
    <xsl:apply-templates select="booktitle,story/title"/>       <!--  Process them in turn.  -->
  </xsl:template>

  <xsl:template match="booktitle">      <!--  First-level enumeration: omnibuses' titles.  -->
    <xsl:text>&start-command;item&start-group;&start-math-mode;&start-command;bullet</xsl:text>
    <xsl:text>&end-math-mode;&end-group; </xsl:text>
    <xsl:value-of select="concat(.,$eol)"/>
  </xsl:template>

  <xsl:template match="title">      <!--  Second level: titles of each story.  -->
    <xsl:text> &start-command;itemitem&start-group;&start-math-mode;&start-command;star</xsl:text>
    <xsl:text>&end-math-mode;&end-group; </xsl:text>
    <xsl:value-of select="concat(.,$eol)"/>
  </xsl:template>

</xsl:stylesheet>
```

Figure 6: Using a character map when a source text for Plain TeX is generated.

```
\item{$\bullet$} Doc Savage Omnibus \#9
 \itemitem{$\star$} The Invisible-Box Murders
 \itemitem{$\star$} Birds of Death
 \itemitem{$\star$} The Wee Ones
 \itemitem{$\star$} Terror Takes 7
\item{$\bullet$} Doc Savage Omnibus \#10
 \itemitem{$\star$} The Devil's Black Rock
 \itemitem{$\star$} Waves of Death
 \itemitem{$\star$} The Too-Wise Owl
 \itemitem{$\star$} Terror and the Lonely Widow

\end
```

Figure 7: Applying Fig. 6's stylesheet to Fig. 1's text.

The XSLT stylesheet given in Figure 6 generates an output suitable for Plain TeX [14]: applying it to the XML text given in Figure 1 yields Figure 7's text. We use the first positions of the range U+E000–U+F8FF — which is a private use area of Unicode's basic multilingual plane — for the character opening a command name, for the beginning and end of an argument of a TeX command, and for on-off switch to the mathematical mode. We define some character entities by using a 'trick' already described in [7]. Such technique is used in the example given in [12, pp. 234–235], although additional characters belonging to the private area could also be defined as global variables, i.e., by means of `xsl:variable` elements that are children of the `xsl:stylesheet` root element. The character codes given in the `TeX-map` character map have been established according to the tables of [14, App. F].

As another example, Figure 8 shows how the Polish letters that do not belong to the Latin 1 encoding[20] can be replaced by the LaTeX commands producing them[21].

7 Other features

Let us consider the `month` element defined in Section 4, Figure 9 shows a function returning the English name corresponding to such an element. When this element is absent, an empty string is returned. As shown in this figure, a temporary tree is created by using an `xsl:variable` element with no `as` attribute. Parts of such a tree can be accessed by means of XPath expressions.

---

[20] The encoding encompassing all the Polish letters is Latin 2.

[21] As we explain in [8], MlBibTeX is presently based on the Latin 1 encoding, and letters belonging to other encodings are replaced by LaTeX commands. In fact, tables analogous to this character map are used internally.

In addition to the XPath functions mentioned in Section 1 about analysing a string w.r.t. regular expressions, let us notice the elements:

```
xsl:analyze-string   xsl:matching-substring
        xsl:non-matching-substring
```

and the `regex-group` function [12, Ch. 5 & 7], which also serve this purpose.

8 Going further

A more didatic introduction to the differences between Versions 1.0 and 2.0 is [16]. More details about the `xsl:sort` element are given in [9]. Stylesheets using XSLT 1.0 are roughly compatible if they are run by an XSLT 2.0 processor, except for the points signalled in [11, App. C] and [12, pp. 123–128]. Here is a short overview.

- The main incompatibility between XPath 1.0 and 2.0 concerns numbers: as an example, let us consider the test `"10" > "2"`. In XPath 1.0, the two strings are dynamically converted into numbers and this test yields `false`. In Version 2.0, if the two operands have not been recognised as numbers by means of a type annotation, they are considered as strings, so the test uses the lexicographical order and yields `true`. Likewise, comparisons with a boolean value may yield different values.

- In XSLT 1.0, if a single item is expected and the supplied value contains more than one item, the first item is returned and the rest is ignored. This rule concerns the parameters of a function, and the values of the `select` attribute of the elements `xsl:sort`, `xsl:value-of`. In XSLT 2.0, the `xsl:sort` element signals an error; in the other cases, all the items are processed: the results are separated by space characters, except if the `separator` attribute is set when the `xsl:value-of` element is used.

- If an `xsl:call-template` element supplies a parameter undefined in the called template, it is ignored in XSLT 1.0. In XSLT 2.0, an error is signalled.

- Most of other incompatibilities are related to datatype management, more dynamic in Version 1.0: see above about XPath's two versions.

At the present time, there are only a few processors for XSLT 2.0. The most known, Saxon, exists in two versions [13]: an open-source version implementing the basic conformance to XSLT 2.0, and a full commercial product, schema-aware. Another choice is AltovaXML[TM] 2008 [1]. All the examples given throughout this article have been tested with Saxon's open-source version.

Jean-Michel HUFFLEN

```
<xsl:character-map name="polish-letters-nlatin1">
  <xsl:output-character character="&#x104;" string="{\Aob}"/>      <!-- 'Ą' -->
  <xsl:output-character character="&#x105;" string="{\aob}"/>      <!-- 'ą' -->
  <xsl:output-character character="&#x106;" string="\'{C}"/>       <!-- 'Ć' -->
  <xsl:output-character character="&#x107;" string="\'{c}"/>       <!-- 'ć' -->
  <xsl:output-character character="&#x118;" string="{\Eob}"/>      <!-- 'Ę' -->
  <xsl:output-character character="&#x119;" string="{\eob}"/>      <!-- 'ę' -->
  <xsl:output-character character="&#x141;" string="{\L}"/>        <!-- 'Ł' -->
  <xsl:output-character character="&#x142;" string="{\l}"/>        <!-- 'ł' -->
  <xsl:output-character character="&#x143;" string="\'{N}"/>       <!-- 'Ń' -->
  <xsl:output-character character="&#x144;" string="\'{n}"/>       <!-- 'ń' -->
  <xsl:output-character character="&#x15A;" string="\'{S}"/>       <!-- 'Ś' -->
  <xsl:output-character character="&#x15B;" string="\'{s}"/>       <!-- 'ś' -->
  <xsl:output-character character="&#x17B;" string="\.{Z}"/>       <!-- 'Ż' -->
  <xsl:output-character character="&#x17C;" string="\.{z}"/>       <!-- 'ż' -->
</xsl:character-map>
```

Figure 8: Character map for the Polish letters not included in the Latin 1 encoding.

```
<xsl:variable name="months">
  <!-- This declaration is allowed in XSLT 1.0, but the contents of this variable, containing XML elements that do
       not come from the source text, would be viewed as a whole, that is, an XPath expression such as
       $months/month-abbr[. = "feb"] would result in an error.
    -->
  <month-abbr english-name="January">jan</month-abbr>
  <month-abbr english-name="February">feb</month-abbr>
  ...
  <month-abbr english-name="December">dec</month-abbr>
</xsl:variable>
<xsl:function name="add:english-month-name" as="xsd:string">
  <xsl:param name="the-month" as="element(month)?"/>
  <xsl:variable
    name="the-position"
    select="index-of(for $node in $months return $node/month-abbr,name($the-month/*[1]))"
    as="xsd:integer?"/>
  <xsl:value-of
    select="if (empty($the-position)) then '' else $months/*[$the-position]/@english-name"/>
</xsl:function>
```

Figure 9: Using a temporary tree.

9   Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has written the Polish translation of the abstract. Tomasz Przechlewski has helped translate keywords, thanks to him, too.

References

[1] *AltovaXML*. 2008. http://www.altova.com/altovaxml.html.

[2] *The Apache Xalan project*. 2005. http://xalan.apache.org.

[3] David FLANAGAN: *JavaScript. The Definitive Guide*. 5th edition. O'Reilly. August 2006.

[4] Jean-Michel HUFFLEN: "MlBiBTEX's Version 1.3". TUG*boat*, Vol. 24, no. 2, pp. 249–262. July 2003.

[5] Jean-Michel HUFFLEN: "Introduction to XSLT". *Biuletyn* GUST, Vol. 22, pp. 64. In *BachoTEX 2005 conference*. April 2005.

[6] Jean-Michel HUFFLEN: "Advanced Techniques in XSLT". *Biuletyn* GUST, Vol. 23, pp. 69–75. In *BachoTEX 2006 conference*. April 2006.

[7] Jean-Michel HUFFLEN: "Introducing LaTeX users to XSL-FO". TUG*boat*, Vol. 29, no. 1, pp. 118–124. EuroBachoTEX 2007 proceedings. 2007.

[8] Jean-Michel HUFFLEN: "Managing Order Relations in MlBibTEX". TUGboat, Vol. 29, no. 1, pp. 101–108. EuroBachoTEX 2007 proceedings. 2007.

[9] Jean-Michel HUFFLEN: "Revisiting Lexicographic Order Relations on Person Names". In: this volume. BachoTEX. April 2008.

[10] Java Technology. March 2008. http://java.sun.com.

[11] Michael H. KAY: XPath^{TM} 2.0 Programmer's Reference. Wiley Publishing, Inc. 2004.

[12] Michael H. KAY: XSLT 2.0 Programmer's Reference. 3rd edition. Wiley Publishing, Inc. 2004.

[13] Michael H. KAY: Saxon. The XSLT and XQuery Processor. March 2008. http://saxon.sourceforge.net.

[14] Donald Ervin KNUTH: Computers & Typesetting. Vol. A: The TEXbook. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[15] Leslie LAMPORT: LATEX: A Document Preparation System. User's Guide and Reference Manual. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[16] Sal MANGANO: XSLT Cookbook. 2nd edition. O'Reilly. December 2005.

[17] Chuck MUSCIANO and Bill KENNEDY: HTML & XHTML: The Definitive Guide. 5th edition. O'Reilly & Associates, Inc. August 2002.

[18] Oren PATASHNIK: BIBTEXing. February 1988. Part of the BIBTEX distribution.

[19] Erik T. RAY: Learning XML. O'Reilly & Associates, Inc. January 2001.

[20] John E. SIMPSON: XPath and XPointer. O'Reilly & Associates, Inc. August 2002.

[21] Bjarne STROUSTRUP: The C++ Programming Language. 2nd edition. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts. 1991.

[22] THE UNICODE CONSORTIUM: The Unicode Standard Version 5.0. Addison-Wesley. November 2006.

[23] Daniel VEILLARD: The XSLT C Library for Gnome. http://xmlsoft.org/XSLT. March 2003.

[24] Eric VAN DER VLIST: Comparing XML Schema Languages. http://www.xml.com/pub/a/2001/12/12/schemacompare.html. December 2001.

[25] W3C: XML Path Language (XPath). Version 1.0. W3C Recommendation. Edited by James Clark and Steve DeRose. November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[26] W3C: XSL Transformations (XSLT). Version 1.0. W3C Recommendation. Edited by James Clark. November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[27] W3C: XSL Transformation Requirements. Version 1.1. Working draft. Edited by Steve Muench. August 2000. http://www.w3.org/TR/2000/WD-xslt11req-20000825.

[28] W3C: XSLT Requirements Version 2.0. W3C Working Draft. Edited by Steve Muench and Mark Scandina. February 2001. http://www.w3.org/TR/WD-xslt20req-20010214.

[29] W3C: XSL Transformations (XSLT). Version 1.1. W3C Working Draft. Edited by James Clark. August 2001. http://www.w3.org/TR/2001/WD-xslt11-20010824.

[30] W3C: XML Schema. November 2003. http://www.w3.org/XML/Schema.

[31] W3C: XML Schema Part 2: Datatypes. W3C Recommendation. Edited by Paul V. Biron, Ashok Malhotra. October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[32] W3C: XPath Requirements Version 2.0. W3C Working Draft. Edited by Mark Scandina and Mary F. Fernández. June 2005. http://www.w3.org/TR/2005/WD-xpath20req-20050603.

[33] W3C: XML Path Language (XPath) 2.0. W3C Working Draft. Edited by Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael H. Kay, Jonathan Robie and Jérôme Siméon. January 2007. http://www.w3.org/TR/2007/WD-xpath20-20070123.

[34] W3C: XQuery 1.0: an XML Query Language. W3C Working Draft. Edited by Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon. January 2007. http://www.w3.org/TR/xquery.

[35] W3C: XSL Transformations (XSLT). Version 2.0. W3C Recommendation. Edited by Michael H. Kay. January 2007. http://www.w3.org/TR/2007/WD-xslt20-20070123.

[36] Priscilla WALMSLEY: XQuery. O'Reilly. April 2007.