A NEW GROUPING METHOD FOR XSL 1.0

Isidro Vila Verde

Feup, Faculdade de Engenharia da Universitdade do Porto
Rua Dr. Roberto Frias, 4200-465 Porto Portugal.
jvv@fe.up.pt

Abstract. One of the classic problems in XSL 1.0 is the nodes grouping. The known solutions are not trivial and have limited range of application. Additionally, the problem is not easy to state and until now all that has been done so far, is presenting it as use cases. In the this article we present a first approach to stating the problem in a formal way and introduce an algorithm based on a method known as "method of Muenchian", we identify XML document structures where this does not work well and we propose a new method for overcoming those limitations. In the end we identify the limits of our solution but we show that it generalizes the Muenchian method.

Keywords: XSL, XSLT, nodes grouping problem

1. Introduction

In XSL 1.0 there are several grouping techniques [1], [2], [3] and in XSL 2.0 there are support in proper elements of the language [4]. However, the support for XSL 2.0 is still not widely implemented in the current XSLT processors and does not solve all situations. As such it makes sense to analyze these techniques in XSL 1.0. This article appears as a consequence of the necessity to create a XSLT to transform into HTML list elements, the items of XML document that appear in mixing elements:

```
<!ELEMENT PARA (#PCDATA|item)*>
```

The search for a solution to this problem led to the research of grouping algorithms. In elapsing of this research some methods were found, but a more detailed analysis of these allow to conclude that these are limited to some very specific XML structures. This article intends to make a revision of one of these methods and with some alterations present a new algorithm.

We start by stating the problem in a formal way but it is not completely formalized. This is due to the fact the XSL specification was done in natural language and until now, so far we know there are not any formal specification[1]. We based our problem's formalization with ideas from some related works [7,8,9,10,11].

1 Very recently and after finishing this paper, we found a formal specification for XSL on W3C Draft [12] but it is a document with 191 pages and we have not had the time to study it yet.

A grouping algorithm in XSL 1.0 is presented using the Muenchian method [1] and we show where it fails. To overcome those limitations we propose another method and show how it works even when the Muenchian method fails.
However, the new proposed method does not solve all grouping problems. Nevertheless, it is more generic than the previous one. The situations where it will not work are also identified.
In the following section the partial formalization of the problem is presented, using ideas from others [7,8,9,10,11]. In section 3 two algorithms are presented, one based on Muenchian method and our proposal for a new method. We discuss both on section 4 and finish with conclusions on section 5.

2. Stating the Problem

A XML document can be represented in a tree structure. Each DTD specifies the possible structures of a set of trees. These structures can be seen as being a type t of trees. For a given type t of trees zero or more XML documents d of type[2] t can exist and, theoretically, it can have an infinite number of types t .

Therefore, we can define the following sets:

$$D_t = \{d : d \text{ is a XML tree of type } t \text{ and } t \in T\}$$
$$T = \{t : t \text{ is some type of XML tree}\}$$

D_t is the set of all XML documents of type t and T is the set of all possible types[3]

A XML document is composed by a content c and represented in a tree of type t . However, the same content[4] can be represented by trees of different types without modifying its meaning.
Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
      <person><age>20</age><name>Ana</name></person>
      <person><age>25</age><name>Joana</name></person>
      <person><age>20</age><name>Pedro</name></person>
      <person><age>25</age><name>Sofia</name></person>
</persons>
```

Fig. 1: A XML Document $d1$ representing a given content c

2 A XML document is said valid, according to one given DTD, if its structure is conform to type t defined by that DTD.
3 A tree structure can be explicitly defined or implicitly assumed from a XML document when it is not associated with any DTD or XSD or any other language.
4 Here, by content , we mean the set of real world facts that the XML text represent

The type $t1$ for the XML document above is defined in Fig. 2

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT persons (person+)>
<!ELEMENT person (age,name)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

Fig. 2: Type Definition $t1$

We can say document $d1$ belongs to set D_{t1}, that is, $d1$ is one instance of type $t1$. D_{t1} is the set of all possible instances of type $t1$.
That is:

$$d1 \in D_{t1}$$

However the same content c can be represented as in Fig. 3, which is a tree of type $t2$ (defined in Fig. 4)

```
<?xml version="1.0" encoding="UTF-16"?>
<persons>
      <age years="20">
            <person><name>Ana</name></person>
            <person><name>Pedro</name></person>
      </age>
      <age years="25">
            <person><name>Joana</name></person>
            <person><name>Sofia</name></person>
      </age>
</persons>
```

Fig. 3: Another XML Document $d2$ representing the same content c as $d1$

To define this kind of relation between $d1$ and $d2$ we use the similar operator modified as follows:

$$d1 \underset{c}{\simeq} d2$$

This means $d1$ is similar to $d2$ in content.
Here, on this example, we transform the document of type $t1$ to a document of type $t2$ by grouping the elements person, which share the same value in its leaf age, under a new element age with an attribute years assigned to value of the old leaf age.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT persons (age+)>
<!ELEMENT age (person+)>
<!ELEMENT person (name+)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST age years CDATA #REQUIRED>
```

Fig. 4: Type Definition $t2$

On a generic transformation we need to identify the leaves for grouping and to choose a name for the new grouping element in a such way that:

$$d1 \underset{c}{\simeq} d2$$
$$where \quad d1 \in D_{t1} \text{ and } d2 \in D_{t2}$$
$$\text{and } t1 \neq t2 \text{ and } (t1, t2) \in (T \times T)$$

Assuming we are able to identify the leaves, the problem is finding a function F that transform any XML document of type t1 into another document of type t2 such as:

$$F : D_{t1} \rightarrow D_{t2}$$
$$F(d1) = d2$$
$$where \; d1 \underset{c}{\simeq} d2$$
$$\text{and } t1 \neq t2 \text{ and } (t1, t2) \in (T, T)$$
$$\text{and } d2 \; groups \; some \; elements \; of \; d1$$
$$based \; on \; leaves \; with \; the \; same \; value$$

Obviously this is not a completely formal problem statement but it is a start and we can see what kind of problems we are dealing with.

3. XSL Grouping Methods

As the XSL[5, 6] is the main programing language to transform XML documents, the function F must be implemented in XSL.

We do not investigate how easy it is to implement function F in an imperative Language like Java or C++, but we suppose it will not be difficult. However, in XSL 1.0 the solution is not immediate because XSL has some particularities.

A XSL Transformer (XSLT) is set of template rules [8] where each rule consists of a pattern matching and a template. A XSLT processor receives as input a XML tree and starting from the root node it will transform each node according to a template defined in the XSLT. The selection of the template to use is made by the pattern matching of the rules with the processed node.

The XSL algorithm to implement the function F is not trivial due to not being enough to declare a set of templates which are applied to the nodes. It is also necessary to identify the groups of the target elements (person) that have the same value in the grouping node (age) and to guarantee that a template exists that will be executed for each one of those groups and not for each node of that group.

In favor of the clearness of the text the following conventions are assumed for the remaining portion of this article:

- The element that we intend to group will be called "target element"
- The sub-node (or leaf) used to group ascendant nodes will be called "grouping node"
- The parent element of the intended elements will be called "context".

Occasionally, to remember and to reinforce the idea, the name of the referred node will be placed between parentheses.

3.1 The Muenchian Method

A solution for the function F in XSL 1.0 was proposed by Steve Muenchian and published by Jeni Tennison on the book *"XSLT and XPath On The Edge"*[1].

A possible implementation of F to convert any XML of type $t1$ (defined in Fig. 2) to a XML tree of type $t2$ (defined in Fig. 4) based on the method of Muenchian, is presented in Fig. 5.

The target element here is the element person, the context is the element persons and the grouping node is the element age.

In Fig. 5, a key k is created (in line 3) for the target elements (person) and indexed by the value of the grouping node (age). Each entry of this key identifies a set of the target elements that have the same value in the grouping node. Thus this key will contain the sets of the target elements to be grouped together. This solves the first part of the problem: group identification.

From line 4 to 8 a trivial template, which applies to any node, is defined. This template simply copies the node and applies the correspondent rules to its children. Eventually, depending on the objectives, these 4 lines can be omitted or replaced.

In line 9 a template is defined which applies only to one of the elements of each set indexed in key k (in this implementation we chose the first target element in each set to match the template rule, but any element of these sets can be selected). The xpath expression on the match attribute guarantees that the template will be executed only once (and only once) for each set. It is in this template that the grouping will be performed. This solves the second part of problem: a template applied only once to each group.

Note that in the predicate the cardinal is only one ([count (...) = 1]), as it is known, when the union (|) of the current element (.) with the first element indexed by the key (key(' k ', age)[1]), results in a set of only one node, i. e., when the current element is the first element indexed by the key k. In all other cases the union result will contain two elements (the current and the first indexed element) therefore, the condition will not be verified. An alternative method to the use of the count function can be seen in reference [1].

From line 10 to 17 the new grouping element (age) is generated with the attribute years assuming the corresponding value, which is nothing more nothing less than the value of the grouping node.

From line 12 to 16 all target elements sharing the same value on the grouping node are copied. As those are joint together in the same set and indexed by key k it is enough to perform the copy operation over each element in that key.

```
1     <?xml version="1.0" encoding="UTF-8"?>
2     <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3       <xsl:key name="k" match="person" use="age"/>
4       <xsl:template match="*|@*">
5        <xsl:copy>
6         <xsl:apply-templates select="@*|node()"/>
7        </xsl:copy>
8       </xsl:template>
9     <xsl:template match="person[
                    count(. | key(' k', age)[1]) = 1]">
10       <xsl:element name="age">
11      <xsl:attribute name="years">
            <xsl:value-of select="age"/></xsl:attribute>
12        <xsl:for-each select="key('k', age)">
13         <xsl:copy>
14          <xsl:apply-templates select="@*|node()"/>
15         </xsl:copy>
16        </xsl:for-each>
17       </xsl:element>
18      </xsl:template>
19      <xsl:template match="person"/>
20      <xsl:template match="person/age"/>
```

Fig. 5: A grouping algorithm based on the Muenchian Method

In line 19 an empty template for those target elements that are not the first ones in the respective set indexed in the key k. It is guaranteed, thus, that those elements are not copied a second time for the result as they are already present.

Finally in line 20 omits the copy of grouping nodes as the information contained in those elements is already present in attribute years. One notices that this is a decision that can be not taken in other situations.

In this section we presented a possible algorithm and noted, however, that there are other XSL grouping algorithms using the Muenchian Method as well. In the reference [1] there is a slightly different algorithm from the algorithm described here. Nevertheless, this one is a little bit more generic.

3.2 Two Keys Method

The Muenchian Method works well when the target elements belong all to the same context (that is, they are all children of the same parent element) and the parent does not have other children than the target elements. That is:

count((xpnodes)/parent:: *) = 1
and count((xpnodes)/parent::*/nodes()) = count((xpnodes)) (1)
where xpnodes is the xpath expression for selecting target elements.

In a real case, the probability of this scenario occurring, limits the use of this method. It will be most likely to find diverse elements that we intend to group, distributed by several nodes of the tree, that is, belonging to different contexts.

Let us see the case shown in Fig. 6, where we have elements person, that we intend to group, distributed by more than one context.

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <group n="1">
    <person><age>20</age><name>Ana</name></person>
    <person><age>25</age><name>Joana</name></person>
    <person><age>20</age><name>Pedro</name></person>
  </group>
<group n="2">
    <person><age>20</age><name>Rita</name></person>
    <person><age>20</age><name>Tiago</name></person>
    <person><age>25</age><name>Sofia</name></person>
  </group>
</persons>
```

Fig. 6: Example of a XML structure where Muenchian Method does not work well

In this example we have the target elements (person) in two different contexts. We have some elements in the context of the first element group and others in the context of the second element group. If this XML document is transformed by the XSLT in Fig. 5, the result will not make any sense as shown in Fig. 7. As it can be verified, the elements person of group 2 appear in the result as being of group 1. This result will hardly be desired because it breaks the information relations.

The result that, certainly, is expected from a grouping algorithm for this case is represented in Fig. 8. The XSL code in Fig. 5 does not work with this structure for two reasons. First, the group identification occurs only with the first element on the set, independently of the context used. So the groups whose value of the grouping node has already occurred in a previous context are simply ignored by the template in line 19. Second, this set omits the context of the elements. Thus, these are selected indistinctly and appear in the first group whose grouping node shares the same value.

The elements in key k are grouped by the value of the grouping node independently of the context where they exist, when they should be grouped by the

value of the grouping node and by the context to which they belong. Thus it is necessary to modify the XSLT to take into consideration the context.

```
<persons>
  <group n="1">
    <age years="20">
     <person><name>Ana</name></person>
     <person><name>Pedro</name></person>
     <person><name>Rita</name></person> <!--wrong
-->
     <person><name>Tiago</name></person> <!--wrong
-->
    </age>
    <age years="25">
     <person><name>Joana</name></person>
     <person><name>Sofia</name></person> <!--wrong
-->
    </age>
  </group>
  <group n="2" /> <!--wrong -->
</persons>
```

Fig. 7: Result when use the XSLT in Fig. 5 to transform XML Document of Fig. 6

The XSL algorithm to transform the XML in Fig. 6 into the XML showed in Fig. 8, having in consideration the context where the target elements exist, is represented in Fig. 9.

This algorithm has the base structure of the algorithm on Fig. 5 but it contains some alterations derived from a different problem approach. The key of the Muenchian Method is divided in two, one to index the first occurrences on each context and other to index the remaining occurrences on that same context. These two keys identify the group (that is, the first element of that group) and the remaining elements of that same group. The way the keys are created allows the groups to be identified in context and not as it happens with the Muenchian Method.

The first key (k1) is created in line 3a for the first occurrence of a target element (person) which has a grouping node (age) with a distinct value of all others which have already occurred on the same context. The key is indexed by the ID generated for the element. This ID will be used for later verification of the existence or not of an element in this key. That is, it will be used to verify if an element identifies a group or not. It must be noticed that to each key it corresponds to one and only one element. Also, it must be noticed that there is an index for each distinct value of the grouping element in each context. So, if it has *m* contexts and each context has *n* distinct values the key will have *nxm* indexes which identify *nxm* distinct groups.

The second key (k2) contains all the target elements (person) sharing the same value in the grouping node (age) and being in the same context which are not the first occurrences. That is obtained by the use of the predicate [preceding-sibling::*/age = age] which guarantees the existence of, at least, a previous element in the same context and with the same value in the grouping node. In this key those elements are

indexed by the ID of the first occurred element. Thus both k1 and k2 keys share the same indexes. And for a given index key k1 returns the element that occurs in first place, while key k2 returns the set of the elements that occur after the first and all share the same value in the grouping node. It can be said that in certain way they are complementary.

```
<persons>
  <group n="1">
    <age years="20">
      <person><name>Ana</name></person>
      <person><name>Pedro</name></person>
    </age>
    <age years="25">
      <person><name>Joana</name></person>
    </age>
  </group>
  <group n="2">
    <age years="20">
      <person><name>Rita</name></person>
      <person><name>Tiago</name></person>
    </age>
    <age years="25">
      <person><name>Sofia</name></person>
    </age>
  </group>
```

Fig. 8: Expected result after grouping XML document in Fig. 6

To generate the ID of the first occurred element to also index key k2, we use the axis preceding-sibling, with the predicate [age = current()/age]. This returns the set of the preceding elements sharing the context and with the same value in the grouping node. The first one of these is the element that we are interested in but, as it is known, when this set is gotten by the axle preceding-sibling the first one to occur in the XML document appears in the last position, reason why we use a second predicate with function last().

Once these two keys defined, it is now easy to define a XSL template for each group and, inside of this, select the elements sharing the context and with the same value in the grouping node. In line 9 the predicate is used to impose the condition of the element being present in key k1, that is, to guarantee that it represents the group. We have, thus, a template executed for each group that we want to generate.

The selection of the elements to be copied to this new element is performed in line 12 using the second key k2. As this key is indexed by the ID of the first occurrence, that is, by the current element, to get all the elements for the group it is enough to index this key by the current element generated ID. Since the current node must also be copied and it is not present in the key k2, it is also included on the XSL match attribute using the union operator.

The remaining part of the algorithm, as it can be observed, remained relatively unalterable with respect to Fig. 5.

```
1     <?xml version="1.0" encoding="ISO-8859-1"?>
2     <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3a   <xsl:key name="k1" match="person[
          not(preceding-sibling::*/age = age)]"
      use="generate-id()"/>
3b     <xsl:key name="k2" match="person[
       preceding-sibling::*/age=age]"
      use="generate-id(preceding-sibling::*[
    age=current()/age][last()])" />
4      <xsl:template match="*|@*">
5       <xsl:copy>
6        <xsl:apply-templates select="@*|node()"/>
7       </xsl:copy>
8      </xsl:template>
9      <xsl:template match="person[key('k1',
                                    generate-id())]">
10      <xsl:element name="age">
11       <xsl:attribute name="years">
           <xsl:value-of select="age"/>
         </xsl:attribute>
12       <xsl:for-each select=".|key('k2',
                                     generate-id())">
13        <xsl:copy>
14         <xsl:apply-templates select="@*|node()"/>
15        </xsl:copy>
16       </xsl:for-each>
17      </xsl:element>
18     </xsl:template>
19     <xsl:template match="person"/>
20    </xsl:stylesheet>
```

Fig. 9: Two keys method

4 Discussion

There are other solutions for this kind of grouping problems but as they don't use keys they have complexity of order O(N^2) (see of Michael Kay's argument in thread initiated in the reference [2]). This solution, being based on keys as the Muenchian Method, has complexity order equivalent, that is, O(N log N). However, we have not made any performance tests over large datasets yet. This is essential but at the present stage of our work, we have not had time to perform those tests.

The presented solution removes the limitation of all target element sharing the same context. Thus, the first condition in (1) could be eliminated. That is, this solution works for any XML document structure where:

count((xpnodes)/parent::*/nodes()) = count((xpnodes)) (2)
where xpnodes is the xpath expression for selecting target elements.

Of course it still has the limitation of all children elements of contexts being target elements and nothing else. But this is a little more common situation than the restriction for Muenchian Method.

5 Conclusions

In this article we present an algorithm of grouping based on the method of Muenchian and show what conditions must be satisfied for it to work. We show why it is too limited for the real world and we present an alternative method to solve one of those limitations. We also show the limitations of our own solution although the range of amplitude of applicability is much more. Finally we must say our solution is not a generic solution nor even resolve our original problem (grouping continuous items under mixed elements) but it is a step forward and we think we can work on a solution for that problem based on the two keys methods with some changes at predicates level. This will be our next goal and we hope to present the solution on a future article.

The complexity issue is particularly relevant for large datasets and the performance tests have not been done yet. We need to work urgently on that component to consolidate our solution.

However, the big challenge for the future will be stating the problem in a formal way. For achieving that, we need to study and learn from the W3C draft XQuery 1.0 and XPath 2.0 Formal Semantics [12]

Acknowledgments

This work was based on a previous paper presented in "IADIS International Conference WWW/Internet", Oct 2006 and was partially support by Departamento de Engenharia Electrotécnica e de Computadores of FEUP and was done as a part of assignments of the course Methodologies for Scientific Research of Doctoral Program in Informatics Engineering

References

1. Jeni Tennison, "*XSLT and XPath On The Edge",* 2001, Unlimited Edition, USA.

2. Sergiu Ignat , “Recursive grouping - simple, XSLT 1.0, fast non-Muenchian grouping method”, In a mailing list, Dec, 2004, http://www.biglist.com/lists/xsl-list/archives/200412/msg00865.html
3. M. David Peterson, “New Alternative to Muenchian Method of Grouping”, In a mailing list, Dec, 2004, http://www.xsltblog.com/archives/2004/12/new_alternative.html
4. Bob DuCharme, “Grouping With XSLT 2.0”, In XML.com, Nov, 2003, http://www.xml.com/lpt/a/2003/11/05/tr.html
5. W3C, “XSL Transformations (XSLT) Version 1.0”, W3C Recommentation, Nov, 1999, http://www.w3.org/TR/xslt
6. W3C, “XML Path Language (XPath) Version 1.0”, W3C Recommentation, Nov, 1999, http://www.w3.org/TR/xpath
7. P. Wadler. “A Formal Semantics of Patterns in XSLT and Xpath” In Markup Tecnologies, Philadelphia, Dec, 1999. Revision version in Markup Languages, MIT Press, Jun, 2001.
8. G. J. Bex, S. Maneth, and F. Neven. “A formal model for an expressive fragment of XSLT”. 1st International Conference on Computational Logic, pp :1137-1151, Lecture Notes in Artificial Intelligence, volume 1861. Springer, 2000
9. S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. “Fast detection of XML structural similarity”, In Transactions on Knowledge and Data Engineering, Volume 17, Issue 2, pp:160-175, Feb, 2005
10. T. Milo, D. Suciu and V. Vianu, “Typechecking for XML transformers “ In Journal of Computer and System Sciences, Volume 66, Issue 1, pp:66 - 97, Feb, 2003
11. G. Gottlob, C. Koch, and R. Pichler. “*Efficient Algorithms for Processing XPath Queries”* In Proc. VLDB'02, 2002
12. W3C, “XQuery 1.0 and XPath 2.0 Formal Semantics”, W3C Draft, Jan, 2007, http://www.w3.org/TR/xquery-semantics/