XSLT - Optimierungen - Hinweise

1. Wenn man einen String aus mehreren zusammensetzen möchte ist die eingebaute concat() Funktion besser geeignet als mehrere <xsl:value-of> hintereinander zu setzen
2. Bevor man eine Funktion mehrmals in einem Template auswertet sollte versucht werden, dass Ergebniss in einer Variable zu speichern und diese zu verwenden. Das gilt eigentlich generell, bevor man etwas doppelt macht speichert man es lieber in einer Var und benutzt diese dann. Im Bezug darauf kann man mit node-set() auf eine Menge an gespeicherten Nodes zugreifen. Das ist besonder nützlich wenn man diese Nodes für ein Template in sortierter Form braucht. Dabei ist zu beachten das Variablen in XSL nicht im zwischengespeichert werden. D.H. sie werden dynamisch bei jedem Aufruf generiert. Um das effizienter zu machen sollta mam explizietes type-Casting mit bspw. string() oder boolean () benutzen.
3. Grunsätzlich sind Templates for-each-Loops vorzuziehen
4. Wenn man häufig eine bestimmte Menge an Nodes betrachtet, kann es sich lohnen diese mit einem <xsl:key> zu gruppieren und mit key() anzusprechen.
5. Mann sollte möglichst präzise Xpath-Anweisungen verwenden und // bzw. * vermeiden
6. Der Kontext eines Templates sollte so nahe wie möglich an den relevanten Nodes sein, so dass man sich möglichst wenig durch den Baum bewegen muss.
7. Wenn man xsl-choose benutzt sollte man darauf achten das der häufigste Fall duch das erste when abgefangen wird und man möglichst selten bis zum otherwise kommt.
8. Besser als tief verschachtelte choose Ausdrücke sind Templates. Statt:

```
<xsl:template match="something";>
    <xsl:choose>
     <xsl:when test="@id">
     ...case 1 </xsl:when>
     <xsl:otherwise>
     ...case 2
     </xsl:otherwise>
   </xsl:choose>
```

sollte man lieber:

```
<xsl:template match="something[@id]">
    ...case 1
 </xsl:template>
 <xsl:template match="something";>
    ...case 2
 </xsl:template>
```

verwenden.

Neben der Möglichkeit effizenteren XSLT-Code zu schreiben kann man auch versuchen das Stylesheet in kleinere Aufgabenteile zu zerlegen und diesee parallel bearbeiten zu lassen. Je nach XSL-Processor und Aufgabe ist das mehr oder weniger Möglich. Eine Einführung in Saxon die auch Multithreading behandelt findet man hier

- INFO: Techniques to Improve Performance of XSL Transformations
- StackOverflow - XSLT Performance

- SlideShare- Some Anti-Pattern in the later Slides