FH Aargau: Vorlesung XML Sommersemester 2005

Test XSLT

MUSTERLÖSUNG (Lösungen in grün & underline)

27.6.2005, 8^{30} - 9^{30}

Bitte schreiben Sie *sorgfältig* und *gut lesbar*! Danke!
Für diesen Test dürfen Sie *keinerlei Unterlagen* verwenden, insbesondere auch keine elektronischen Hilfsmittel wie Laptop oder Organizer!

1. XSLT 1.0 kann recht viele Probleme produzieren im Umgang mit XPaths, weil es bei diesen *keine Typüberprüfung* gibt, dafür aber sehr viele *implizite Typkonvertierungen*. Das Problem der impliziten Konvertierungen ist, dass sie u.U. Programmierfehler "beseitigen", bei denen dann unerwartetes Verhalten auftritt, anstatt eines (an sich hilfreicheren) Laufzeitfehlers. Ein klassisches Beispiel ist der Vergleich zweier Node Sets über den `=` Operator, der *nicht auf Identität der Nodes* testet, sondern *Gleichheit der String Values* (die genaue Definition ist im übrigen noch komplizierter). Erläutern Sie an einem Beispiel, weshalb der `=` Operator mit Node Sets ein oft überraschendes Verhalten an den Tag legt.
   Vergleicht man zwei Node Sets, so wird der String Value der enthaltenen Nodes verglichen. Also werden zwei unterschiedliche Nodes mit dem gleichen String Value (insbesondere Nodes ohne Text, die den leeren String ergeben, sind hier ein Risiko) als gleich betrachtet. Genaugenommen testet der `=` Operator, ob es in den beiden Node Sets Nodes gibt, deren String Values gleich sind, aus diesem Grunde sind `$a="x"` und `not($a!="x")` nicht identisch (ersterer XPath ist `true()` wenn es in `$a` mindestens einen Node mit dem String Value `"x"` gibt, letzterer dagegen, wenn alle Nodes in `$a` den String Value `"x"` haben), was zwar in XPath eindeutig so definiert, aber für Programmierer oftmals verwirrend ist.
2. XSLT erlaubt *Parameter für Templates und für Stylesheets*. In beiden Fällen werden Werte aus einer aufrufenden Umgebung an das aufgerufene Konstrukt übergeben. Beschreiben Sie *mindestens drei* wichtige Unzulänglichkeiten von Parametern und deren Übergabe in XSLT, die Vorsicht erfordern im Umgang mit Parametern.
   a. Es findet keine Typüberprüfung der Parameter statt, d.h. man muss sich im Programm selber darum kümmern, dass ein übergebener Wert auch als der erforderliche Typ interpretierbar ist.
   b. Man kann Parameter übergeben, die nicht deklariert wurden im aufgerufenen Konstrukt, diese werden ignoriert.
   c. Man kann auf die Übergabe von Parametern verzichten, die deklariert wurden im aufgerufenen Konstrukt, diese sind dann leer oder

bekommen den Default-Wert zugewiesen (falls der Parameter mit Default-Wert deklariert wurde).

   d. Die beiden vorherigen Punkte zusammengenommen bewirken, dass es kein Fehler ist, wenn es einfache Tippfehler im Namen gibt bei der Übergabe von Parametern, diese werden einfach nicht übergeben.
   e. Es kann schwierig sein, aus der Umgebung eines Stylesheets Node Sets an ein Stylesheet zu übergeben, hier muss die Umgebung genau bekannt sein und es sollte getestet werden, ob die Übergabe von Node Sets wirklich funktioniert wie erwartet.

3. Erklären Sie, wozu *Attribute Value Templates* gut sind (zur Erinnerung, das sind in `{}` eingeschlossene XPaths in Attributwerten), warum sie in XSLT notwendig sind, und spekulieren Sie darüber, wie man das Problem, für dessen Lösung sie eingeführt wurden, anders hätte lösen können.
   Bestimmte Attribute in XSLT Elementen (z.B. `select` oder `test`) werden prinzipiell als XPath ausgewertet, andere nicht. Für Attribute, die nicht prinzipiell als XPath ausgewertet werden, und vor allem auch für Attribute von Literal Result Elements, muss daher ein Mechanismus existieren, der diese Auswertung auslöst. Dies wird mit den Attribute Value Templates erreicht, die signalisieren, dass jetzt ein XPath benutzt wird, der ausgewertet werden muss. Folgende Alternativen wären denkbar:
   a. Alle Attributwerte werden als XPath ausgewertet, dann allerdings müssten alle Attributwerte, die nur Strings sein sollen, als XPath Strings in Anführungszeichen verpackt werden. Die Leserlichkeit würde leiden.
   b. Soll ein Attributwert eines Literal Result Elements sich durch einen XPath ergeben, so muss das Attribut durch `xs:attribute` erzeugt werden, in dem dann `xs:value-of` benutzt werden kann. Auch hier würde die Leserlichkeit leiden.
   c. Würde XML Elemente in Attributwerten erlauben (was es nicht tut), könnte man einfach `xs:value-of` in den Attributwerten benutzen.
4. Basierend auf einem XML Schema erhalten Sie ein Dokument, in dem zwei Attribute ein `xs:dateTime` enthalten (ein Beispielwert für diesen Typ ist `2002-12-07T12:20:46.275+01:00`). Sie sollen ein XSLT schreiben, dass sicherstellt, dass die beiden Attributwerte nicht mehr als `x` Stunden auseinanderliegen. Was für Unterstützung bietet ihnen XSLT?
   Nicht viel neben dem Zugriff auf die Attributwerte als Strings, da es die XML Schema Typen in XSLT nicht gibt. Mittels der String-Funktionen kann man zwar die Werte untersuchen und bestimmte Teile extrahieren, aber den eigentlichen Vergleich, und vor allem die aufwendigen Spezialfälle (unterschiedliche Zeitzonen, Intervalle über Datumsgrenzen hinweg) muss man von Hand programmieren. U.U. bietet EXSLT hier etwas an (es ist allerdings nur sehr wenig), dann könnte man auf etwas bessere Unterstützung durch den Prozessor hoffen.
5. Beschreiben Sie das Prinzip der `keys` in XSLT. Die beiden wichtigen Aspekte der `keys` sind das *Deklarieren* eines Keys (beschreiben Sie die Deklaration und was dies beim XSLT Prozessor bewirkt) und die *Anwendung* des `keys` in einem XPath. Beschreiben Sie anhand eines kleinen Beispiels, wozu `keys` typischerweise benutzt werden.
   Die Deklaration eines Keys geschieht am Beginn eines XSLT Stylesheets, diese Deklaration instruiert den XSLT Prozessor, dass später auf die im Key

selektierten Knoten zugegriffen werden soll. `<xsl:key name="zipkey" match="person" use="address/zip"/>` ist eine solche Deklaration, alle `person` Elemente des Dokuments werden mit dem Wert des `adresse/zip` Elements (von der `person` aus selektiert) in den Key eingetragen. Typische XSLT Prozessoren bauen aus diesem Grunde eine optimierte Zugriffsstruktur für diese Nodes auf, über die schnellere Zugriffe möglich sind. Mit der XSLT `key()` Funktion kann dann auf den Key zugegriffen werden, `key("zipkey","8092")` liefert also ein Node Set, dessen Inhalt alle `person` Elemente sind, deren `adresse/zip` Kind den Wert 8092 hat. Dieses Beispiel zeigt eine typische Benutzung, vor allem wenn man viele `person` Elemente hat, auf die man oftmals gefiltert nach der PLZ zugreifen will.

6. Angenommen, eine DTD läge in XML-Syntax vor (also Element- und Attributdeklarationen in XML-Konstrukten, nicht in der speziellen und zeichenbasierten DTD Syntax), dann wäre es möglich, einen XML Validator in XSLT zu implementieren. Beschreiben Sie in wenigen groben Schritten, wie eine solche Implementierung aussehen würde.
    a. Einlesen des zu validierenden Dokuments als Eingabedokument.
    b. Einlesen der XML DTD mit der `document()` Funktion, optional Anlegen von Keys für alle Element- und Attributdeklarationen.
    c. Es gibt nur zwei Templates, eines für Elemente, das andere für Attribute. Das Element-Template geht alle Kinderknoten durch und überprüft ihre Übereinstimmung mit der XML DTD, danach (oder davor) werden alle Attribute überprüft, abschliessend werden alle Kinderelemente mit `apply-templates` selektiert (rekursives Vorgehen). Das Attribut-Template schaut für jedes Attribut, ob es auf dem Eltern-Element erlaubt ist mit dem vorgefundenen Wert.
    d. Abschliessend (das könnte aber auch schon im Attribut-Template geschehen) werden noch die `ID/IDREF` Attribute überprüft.