Klausur "Semistrukturierte Daten und XML"
Sommersemester 2005
Prof. Dr. Wolfgang May
19. Juli 2005, 14-16 Uhr
Bearbeitungszeit: 90 Minuten

Vorname:

Nachname:

Matrikelnummer:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner etc.) erlaubt. Handies müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie nur die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller etc.; Bleistift ist nicht erlaubt. Beantwortung der Fragen auf Deutsch oder Englisch.
Auf dem letzten Blatt finden Sie ein XML-Dokument, das in den Aufgaben 3 und 4 verwendet wird. Trennen Sie es ggf. zur Bearbeitung der Aufgaben ab.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

☐ meine Note soll mit Matrikelnummer so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ meine Note soll nicht veröffentlicht werden; ich erfahre sie dann aus Munopag (bzw. für nicht im Munopag geführte Studierende: beim Abholen des Scheins).

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (Allgemeines)	17	8
Aufgabe 2 (Diverses)	30	12
Aufgabe 3 (XML, XQuery)	32	16
Aufgabe 4 (XSLT)	16	8
Summe	95	44

Note:

Aufgabe 1 (Allgemeines [17 Punkte])
(kurz und stichpunktartig beantworten)

1. Welche grundsätzliche Vorgehensweise (nicht Eigenschaften, sondern Vorgehensweise!) ist den "modernen" deklarativen, mengenorientierten Anfragesprachen SQL, OQL, XQuery, XML-QL gemeinsam (5 P)?

2. XPath folgt nicht diesem Konzept. Welche Eigenschaften hat es mit den oben genannten Anfragsprachen noch gemeinsam? Welche Funktionalität geht verloren (in der Vorlesung wurde hierauf insbesondere bei der Diskussion von XQL eingegangen)? Was kann man damit noch tun, bzw. wozu wird es verwendet? (6 P)

3. Bei dem am Anfang der Vorlesung besprochenen "Netzwerk-Datenmodell" gab es noch keine moderne Anfragesprache. Könnten Sie eine solche mit Ihrem Wissen jetzt entwerfen? Begründen Sie kurz, bzw. skizzieren Sie kurz Ihren Vorschlag. (6 P)

Lösung

1. Datensätze (Tupel, Knoten, Objekte) auswählen und an Variablen binden (FROM, for-let, Regelrumpf), Bedingungen darüber auswerten (WHERE, Regelrumpf), Ergebnis erzeugen (SELECT, return) (5 P)

   Dies ermöglicht Operationen wie Selektion, Join, Projektion und die Mengenoperationen.

   Hinweis: gesucht waren nicht *Eigenschaften* wie Abgeschlossenheit, Orthogonalität etc.

2. Gemeinsam: Es ist immer noch eine mengenorientierte, deklarative Sprache.
   Was fehlt: Joins (man hat nur noch Semijoins innerhalb der Filter), Projektionen/Weglassen von Teilen, (Re)strukturierung eines Ergebnisses.
   Was man damit noch tun kann: Adressieren von Knotenmengen; dazu wird es in XQuery auch verwendet (diese Mengen werden dann an Variablen gebunden).

   Weitere Verwendung u.a. in XSLT, XLink, XML Schema.

3. Ja, könnte man problemlos. Das Netzwerkmodell ist auch nicht viel anders als das DOM, oder insbesondere das objektorientierte oder das OEM-Modell (d.h., man wird Pfadausdrücke zur Navigation benutzen). Skizze: OQL, OEM, F-Logic, XQuery. Alles funktioniert wie das in (a) beschriebene Verfahren auf solchen graph-basierten Daten. Variablen bindet man eben an die Datensätze.

Aufgabe 2 (Diverses [30 Punkte])
Gegeben ist das folgende XML-Fragment:

```
<persons>
  <person>
    <firstname>Hans</firstname>
    <name>Wurst</name>
  </person>
  <person>
    <firstname>Karl</firstname>
    <firstname>Heinz</firstname>
    <name>Napf</name>
  </person>
  <person>
    <firstname>Lieschen</firstname>
    <name>Mueller</name>
  </person>
</persons>
```

1. Geben Sie eine DTD für Personenlisten dieser Form an. Interpretieren Sie Dinge, die durch das obige Fragment nicht eindeutig sind, "sinnvoll"; ggf. mit kurzem Kommentar.
(Hinweis: eine Personenliste darf auch leer sein) (5P)

2. Welche Ergebnisse liefern die vier folgenden Ausdrücke (Einrückung, Formatierung nicht berücksichtigen) (4 $\times$ 2P)?

   (a) XPath: `//name`

   (b) XQuery:
   ```
   for $x in //name/text()
   return <a>{$x}</a>
   ```

   (c) XQuery:
   ```
   let $x := //name
   return <a>{$x}</a>
   ```

   (d) XSLT:
   ```
   <xsl:stylesheet ... >
    <xsl:template match="*">
     <a><xsl:copy-of select=".//name"/></a>
    </xsl:template>
   </xsl:stylesheet>
   ```

3. Geben Sie einen endlichen Automaten (als Diagramm) zur DTD an, mit dem sich ein Dokument zu dieser DTD validieren läßt (8 P).

4. Welchen asymptotischen Aufwand hat die Validierung eines Dokuments mit n Knoten mit Ihrem Automaten (2 P)?

5. Wenn die Daten der Personen stattdessen wie folgt (um ein `id`-Attribut erweitert) abgelegt werden sollen,

```
<persons>
  <person id="01">
    <firstname>Hans</firstname>
    <name>Wurst</name>
  </person>
  :
</persons>
```

was müssen Sie dazu an Ihrer DTD ändern (2 P)?

6. Beschreiben Sie die Grundidee des SAX-API für XML anhand des Fragmentes aus (5.) . Welche Folge von Ereignissen tritt ein (4 P)?

7. Skizzieren Sie, wie Sie Ihren Automaten ergänzen müssen um entsprechend der erweiterten DTD zu validieren (1 P).

Lösung

1. ```
   <!ELEMENT persons (person*)>
   <!ELEMENT person (firstname+,name)>
   <!ELEMENT firstname (#PCDATA)>
   <!ELEMENT name (#PCDATA)>
   ```

2. Ergebnisse:

   (a) XPath: selektiert die Knoten und gibt sie aus:

   ```
   <name>Wurst</name>
   <name>Napf</name>
   <name>Mueller</name>
   ```

   (b) for: für jeden Knoten einzeln, aber nur den Textinhalt nehmen:

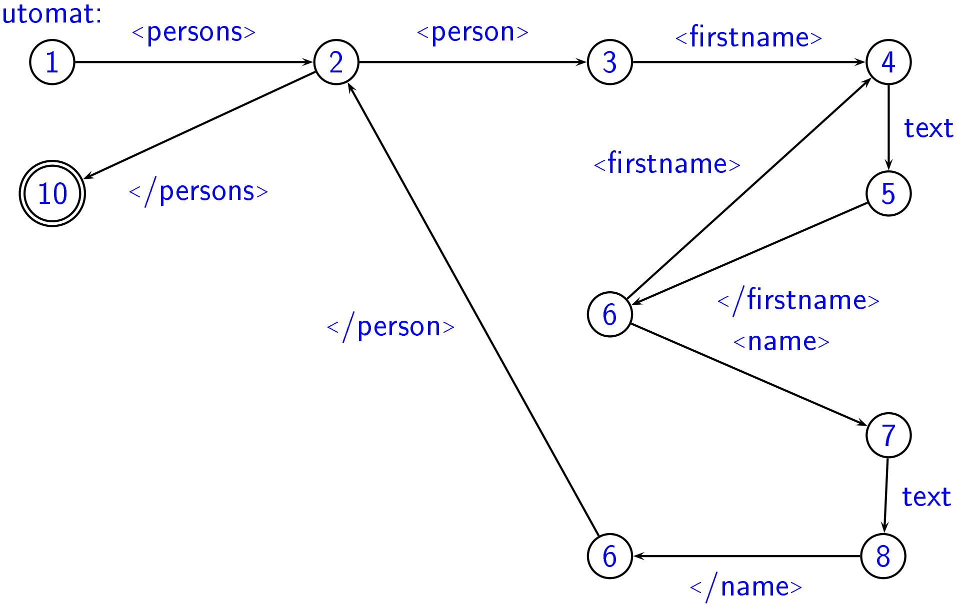
   ```
   <a>Wurst</a>
   <a>Napf</a>
   <a>Mueller</a>
   ```

   (c) let: alle zusammen

   ```
   <a><name>Wurst</name>
      <name>Napf</name>
      <name>Mueller</name></a>
   ```

   (d) XSL: alle zusammen; dasselbe wie bei “let”.
   Hinweis: Beim Aufruf eines XSL-Stylesheets wird automatisch quasi ein <xsl:apply-templates select="/" /> für das root-Element ausgeführt. Dieses ruft genau das angegebene Template einmal auf, das die oben genannte Ausgabe erzeugt.
   ```

3. Automat:



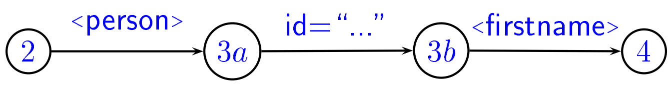
4. Aufwand: $O(n)$, bzw. linear.
   Das kann man sagen, ohne Ihren Automaten zu kennen; endliche Automaten haben *immer* linearen Aufwand! Jeder Knoten wird ein- bzw. zweimal (schließender Tag) angefasst.

5. Hinzufügen der Attributdeklaration:

```
<!ATTLIST person id ID #REQUIRED>
```

6. Das SAX-API ist ereignisbasiert, d.h., das XML-Beispiel wird als Stream von Ereignissen betrachtet: Beginn eines "persons"-Elementes; Beginn eines "person"-Elementes; ein id-Attribut mit Wert "01"; Beginn eines "firstname"-Elementes; Text: "Hans"; Ende des "firstname"-Elementes usw.

7. Zustand 3 wird wie folgt aufgespalten:



Aufgabe 3 (XML, XQuery [32 Punkte])
Diese Aufgabe verwendet die auf dem hintersten Blatt zu findende Lufthansa-Datenbasis.
Geben Sie für die Anfragen (2)-(7) je eine XPath- oder XQuery-Anfrage an, die das Ergebnis in dem jeweils angegebenen Format ausgibt (falls kein Format angegeben, beliebig).

1. Geben Sie an, welche Attribute welcher Elemente Sie in der entsprechenden DTD als ID auszeichnen würden (6 P). (Diese dürfen Sie im folgenden auch als IDs verwenden)

2. Geben Sie die Abkürzungen aller Flughäfen an, zu denen man direkt von Frankfurt (FRA) fliegen kann. (2 P)

3. Geben Sie alle Nummern von Flügen an, die Freitags von Frankfurt (FRA) nach Lissabon (LIS) gehen. (2 P)

4. Geben Sie die Menge der Namen aller Piloten an, die einen Einsatz mit einer B737 fliegen (4 P).

5. Geben Sie die Abkürzungen aller Flughäfen aus, die man von Frankfurt aus mit genau einer Zwischenlandung (die beliebig lange dauern darf) erreichen kann. Format: (5 P)
   <erreichbar>*ziel*</erreichbar> ,
   wobei *ziel* die Abkürzung des Zielflughafens ist.

6. Geben Sie die Nummern, Abflughafen-Kürzel und Abflugzeiten aller Flüge an, mit denen man am 26.12.2005 von Deutschland direkt nach Australien fliegen kann: (5 P)
   <result von= "*flughafen*" um= "*abflugzeit*" nr="*nr*" />

7. Wie würden Sie die Datenbank um Informationen ergänzen, welcher Pilot welche Flugzeugtypen (möglicherweise mehrere) fliegen darf (geben Sie ein Beispielfragment an, das diese Ergänzung zeigt)? (4 P)

   Geben Sie einen Ausdruck an, der mit dieser Ergänzung überprüft, ob Pilot "Schmidt" bei seinen Einsätzen nur Flugzeuge fliegt, die er auch wirklich fliegen darf. Der Ausdruck soll alle diejenigen "Einsatz"-Elemente zurückgeben, bei denen dies *nicht* der Fall ist. (4 P)

Lösung

1. IDs: flughafen/@abbrev, pilot/@name, flugzeug/@name, flug/@nr
   Hinweis: im Gegensatz zu Schlüsseln im relationalen Modell sind IDs immer einstellig – "flug"-Elemente haben also keine ID.

2. `//flug[@von='FRA']/@nach`

3. `//flug[freitag and @von='FRA' and @nach='LIS']/@nr`

4.
```
for $p in //pilot
where //einsatz[@pilot="{$p/@name}" and id(@flugzeug)/@typ='B737']
return $p/@name
```

   Hinweis:
   `//einsatz[id(@flugzeug)/@typ='B737']/@pilot`
   tut dasselbe, entfernt aber keine Duplikate!
   Wenn man das zu

```
id(//einsatz[id(@flugzeug)/@typ='B737']/@pilot)/@name
```

   erweitert, verschwinden die Duplikate (die id-Funktion wählt die Menge der Knoten aus).
   Man kann auch die Funktion distinct-values(...) verwenden:

```
distinct-values(//einsatz[id(@flugzeug)/@typ='B737']/@pilot)
```

5. Mehrere Alternativen:
   Mit einem klassischen symmetrischen Join

```
for $f1 in //flug[@von='FRA'], $f2 in //flug
where $f1/@nach = $f2/@von
return
  <erreichbar> {string($f2.@nach)} </erreichbar>
```

   Mit einem korrelierten Join (was es ja in SQL nicht gab, aber seit OQL möglich ist):

```
for $f1 in //flug[@von='FRA']
 for $f2 in //flug[@von = $f1/@nach]
return
  <erreichbar> {string($f2.@nach)} </erreichbar>
```

   Oder nur mit einem längerem XPath-Ausdruck:

```
for $ziel in //flug[@von=//flug[@von='FRA']/@nach]/@nach
return
  <erreichbar> {string($ziel.@nach)} </erreichbar>
```

   Hinweis: die interne Optimierung per XQuery-Algebra müßte alles auf dieselbe Auswertung abbilden.

6.
```
for $e in //einsatz
let $f:=id($e/@flug)
where $f/@datum="20051226"
  and id($f/@von)/@country='D'
  and id($f/@nach)/@country='AUS'
return
 <result von="{$f/@von}" um="{$f/@abflug}" nr="{$f/@nr}"/>
```

7. Möglichkeit 1: Pilot um Subelemente ergänzen:

```
<pilot name="Mueller">
  <darf-fliegen typ="B737"/>
  <darf-fliegen typ="A380"/>
</pilot>
//einsatz[@pilot="Schmidt"
   and not(id(@flugzeug)/@typ =
           //pilot[@name="Schmidt]/darf-fliegen/@typ)]
```

   Möglichkeit 2: Paare aufzählen (wie man es in einer relationalen DB machen würde)

```
<darf-fliegen pilot="Meier" typ="B737"/>
<darf-fliegen pilot="Meier" typ="A380"/>
//einsatz[@pilot="Schmidt"
   and not(id(@flugzeug)/@typ =
           //darf-fliegen[@pilot="Schmidt"]/@typ)]
```

   Möglichkeit 3: mehrstelliges Attribut zu Pilot als IDREF, wozu man aber alle Flugzeugtypen einzeln anlegen muss, um die IDs zur Verfügung zu haben:

```
<flugzeugtyp id="B737"/>
<flugzeugtyp id="A380"/>
<pilot name="Mueller" darf-fliegen="A380 B737"/>
//einsatz[@pilot="Schmidt"
   and not(id(id(@flugzeug)/@typ) =
           id(//pilot[@name="Schmidt"]/@darf-fliegen))]
```

   (Hinweis: wenn man das Attribut als NMTOKENS deklariert, kann man nicht splitten, sondern muss mit contains() arbeiten, was z.B. bei "B747" und "B747-400" problematisch ist.)

   Hinweis zu den Anfragen: Hierbei wird bei dem Vergleich jeweils geprüft ob nicht eines der Elemente beider Mengen übereinstimmt!

Aufgabe 4 (XSLT [16 Punkte])
Diese Aufgabe verwendet ebenfalls die auf dem hintersten Blatt zu findende Lufthansa-Datenbasis.

1. Geben Sie ein XSLT-Stylesheet an, das eine große Tabelle mit den Einsatzplänen aller Piloten nacheinander in XHTML wie folgt ausdruckt (13 P): [Nehmen Sie hierbei an, dass die Einsätze bereits im Eingabedokument nach Abflugsdatum- und Zeit geordnet sind].
   - 1. Zeile: Name des ersten Piloten
   - Je eine Zeile für jeden Einsatz dieses Piloten mit den Spalten "Datum", "Flugnummer", "Abflugszeit", "Abflughafen", "Flugzeugname", "Flugzeugtyp".
   - Name des zweiten Piloten
   - usw.
2. Was müssen Sie ändern, wenn die Einsätze nicht bereits im Eingabedokument geordnet sind, und die Tabelle sie aber dennoch nach Datum und Uhrzeit geordnet ausgeben soll? (3 P)

Lösung

1. Stylesheet:

```
<xsl:stylesheet>
  <xsl:template match="/">
   <html><body><table>
    <xsl:apply-templates select="//pilot"/>
   </table></body></html>
  </xsl:template>
  <xsl:template match="pilot">
    <tr><td>
      <xsl:value-of select="@name"/>
    </td></tr>
    <xsl:apply-templates select="//einsatz[@pilot=current()/@name]"/>
  </xsl:template>
  <xsl:template match="einsatz">
   <tr>
    <td><xsl:value-of select="@datum"/></td>
    <td><xsl:value-of select="@flug"/></td>
    <td><xsl:value-of select="id(@flug)/@abflug"/></td>
    <td><xsl:value-of select="id(@flug)/@von"/></td>
    <td><xsl:value-of select="@flugzeug"/></td>
    <td><xsl:value-of select="id(@flugzeug)/@typ"/></td>
   </tr>
  </xsl:template>
</xsl:stylesheet>
```

2. Nach Datum und Zeit ordnen: <xsl:sort> in das passende <xsl:apply-templates> einbauen:

```
<xsl:apply-templates select="//einsatz[@pilot=current()/@name]"/>
  <xsl:sort select="@datum"
            data-type="number" order="ascending"/>
  <xsl:sort select="id(@flug)/@abflug"
            data-type="number" order="ascending"/>
</xsl:apply-templates>
```

Für Aufgaben 3 und 4 (XQuery und XSLT) sei das folgende XML-Dokument, das die Datenbank einer Fluggesellschaft beschreibt, gegeben:

- Flughäfen liegen in einem Land (durch das Landeskennzeichen abgekürzt), gehören zu einer Stadt und haben eine Kurzbezeichnung.
- Piloten haben einen Namen.
- Flugzeuge haben ebenfalls einen Namen (sind nach Komponisten benannt) und sind von einem Flugzeugtyp.
- Flüge haben eine Flugnummer und führen von einem Flughafen zu einem anderen mit einer festen Abflugs- und Ankunftszeit, und werden an bestimmten Wochentagen geflogen. Zeiten sind hierbei durch Integers repräsentiert (also z.B. 1100 für 11:00h); dies ermöglicht, sie einfach numerisch zu vergleichen).
- Welcher Pilot zu einem bestimmten Datum den Flug fliegt, und mit welchen Flugzeug, ist im Einsatzplan festgehalten. Das Datum ist hierbei ebenfalls durch Integers repräsentiert (also z.B. 20050719 für 19.07.2005) um einfache Vergleiche zu erlauben.

```
<lufthansa>
  <flughafen country="D" abbrev="FRA">Frankfurt</flughafen>
  <flughafen country="D" abbrev="MUC">Muenchen</flughafen>
  <flughafen country="P" abbrev="LIS">Lisbon</flughafen>
  <flughafen country="AUS" abbrev="SYD">Sydney</flughafen>
  :
  <pilot name="Meier"/>
  <pilot name="Mueller"/>
  :
  <flugzeug name="Beethoven" typ="B737"/>
  <flugzeug name="Mozart" typ="A310"/>
  <flugzeug name="Strauss" typ="A310"/>
  <flugzeug name="Brahms" typ="A380"/>
  :
  <flug nr="LH306" von="FRA" abflug="730" nach="FCO" ankunft="920">
    <montag/><mittwoch/><freitag/>
  </flug>
  <flug nr="LH245" von="CDG" abflug="1100" nach="MUC" ankunft="1300">
    <montag/><dienstag/> ..
  </flug>
  :
  <einsatz datum="20050719" flug="LH245"
       pilot="Meier" flugzeug="Beethoven"/>
  <einsatz datum="20050720" flug="LH245"
       pilot="Schmidt" flugzeug="Liszt"/>
  :
</lufthansa>
```