# XML-Praktikum SS 2015

**Belinda Zahra**

**Thea Heim**

**Youlia Goldberg**

# XML-Praktikum SS 2015

Belinda Zahra
Thea Heim
Youlia Goldberg

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Einleitung

## Beschreibung und Motivation von CalendarX

CalendarX ist ein im Web-Browser dargestellter Kalender, der neben den normalen Kalenderfunktionen noch zusätzliche Kalenderfunktionen mit sich bringt. Diese können folgendermaßen beschrieben werden.

**Untersützung zusätzlicher Event-Typen (Superevents):**CalendarX macht es möglich, Beziehungen zwischen Events abzubilden, die keinem einzelnen Wiederholungsmuster enstprechen. So können nicht nur sich wöchentlich oder monatlich wiederholende Events abgebildet werden, sondern auch sich nach einem ganz anderen Muster wiederholende Events. Z.B. kann eine Vorlesung, die wöchentlich Montags von 10 bis 12 Uhr und Mittwochs von 13 bis 15 Uhr stattfindet, in einem einzelnen Event repräsentiert werden.

**Unterstützung verschiedener Wiederholungs-Muster (reccurence patterns):**CalendarX unterstützt die gewöhnlichen Wiederholungsmuster (täglich, wöchentlich, monatlich), aber auch die Kombination dieser Wiederholungsmuster untereinander, einschließlich Ausnahmen. Z.B. kann CalendarX den Fall repräsentieren, dass ein Event an jedem ersten Montag und jeden dritten Mittwoch im Monat stattfindet, mit Ausnahme der Semesterferien.

Zusammenfassend kann man die Motivation für dieses Praktikum darin beschreiben, dass wir einen Kalender entwickeln, der gegenüber den herkömmlichen Kalendersystemen über erweiterte Funktionalitäten verfügt, und somit auch von uns in der Realität gerne eingesetzt würde.

## Ziel und Organisation des XML-Praktikums

Ziel des Praktikums ist es, das System CalendarX zu implementieren und zu dokumentieren.

Client-seitig soll CalendarX im Web-Browser dabei die Möglichkeit bieten, Datumsangaben zu machen sowie eine Kalendersicht (Tages-, Wochen- oder Monatssicht) auszuwählen und die zugehörigen Kalenderdaten als SVG-Graphik berechnen zu lassen und anzuzeigen. Zudem soll jedes Event in der Kalendersicht mit einem Link versehen werden, welcher die Event-Daten in ein editierbares Formular lädt. Die im Eventformular durchgeführten Änderungen, sollen nach Bestätigung in die Datenbasis übertragen werden, wobei das Datum eines Events nicht änderbar ist.

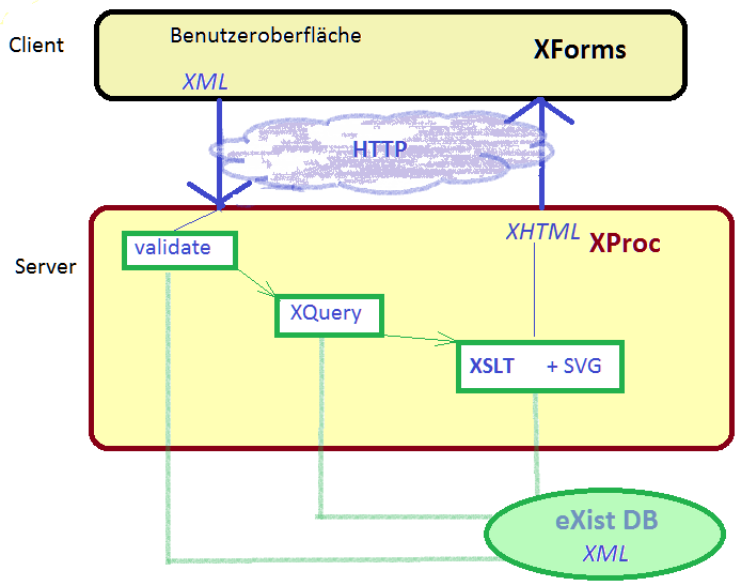
Die Dokumentation der Projektes erfolgt in DocBook und soll für die Abgabe in das PDF-Format umgewandelt werden.

Der Ablauf des Praktikums verlief in zwei Phasen. In der ersten Hälfte des Semesters wurden in 5 freiwilligen Terminen die Grundlagen von XML, XDM, XPath, XSLT, XQuery, eXist, XForms und Docbook vermittelt. Zu jedem Termin gab es jeweils ein Aufgabenblatt, das auf das Praktikum vorbereitete bzw. schon Teile der späteren Projektentwicklung enthielt. Neben den Vorträgen der Kursleitung waren auch die Präsentationen der einzelnen Gruppen über die Lösungen der Aufgabenblätter sehr hilfreich für die eigene CalendarX-Entwicklung. In der zweiten Hälfte des Semesters arbeitete jede Gruppe für sich am eigenen Projekt. Das Ergebnis wird am Ende des Semesters in einer Präsentation vorgestellt.

# Chapter 2. Das Kalender-System CalendarX

## Die XRX-Architektur

**Figure 2.1. CalendarX Component Layer Structure**




# XML Schema: Die Architektur von CalendarX

Das konzeptuelle Schema für Kalender-Daten

# Umsetzung der Tages-, Wochen und Monatssicht

## SVG zur Erstellung der Templates

Die Templates der Tages-, Wochen und Monatssicht wurden mit Hilfe von scalable Vector Graphics (SVG) erstellt und mit Hilfe der Tages-, Wochen- und Monatsstylesheets aufgerufen. Im ersten Schritt erhält das Template einen Namen, um es später eindeutig zuordnen zu können. Danach wird die SVG-Fläche mit den einleitenden Höhen- und Breitenangaben definiert. Innerhalb der SVG-Tags wird die entsprechende Graphik gezeichnet.

Nachfolgend wird beispielhaft das Template "tagessicht" beschrieben. Dieses enthält 25 horizontale Linien, innerhalb zweier Linien steht jeweils die volle Stunde von 0:00 Uhr bis 23:00 Uhr. Zudem enthält das Template eine vertikale Linie, um die Uhrzeiten von den Kalendereinträgen abzugrenzen.

```
<!-- Schreibt jede volle Stunde auf eine horizontale Linie -->
 <xsl:template name="tagessicht">
     <svg width="1300" height="1000" xmlns="http://www.w3.org/2000/svg"
         xmlns:xlink="http://www.w3.org/1999/xlink">
```

```
            <!-- Im defs-Container werden die Objekte definiert -->
            <defs>
                <!-- Linie horizontal-->
                <line x1="50" y1="50" x2="1150" y2="50"
                stroke-width="2" stroke="grey" id="li"/>
                <!-- Linie vertikal-->
                <line x1="100" y1="50" x2="100" y2="800"
                stroke-width="2" stroke="grey" id="li2"/>
            </defs>
            <!-- Zeichne die oben definierte Linie 25 mal mit Abstand 30 -->
            <use xlink:href="#li" y="30"/>
            <use xlink:href="#li" y="60"/>
            <use xlink:href="#li" y="90"/>
            <use xlink:href="#li" y="120"/>
            <use xlink:href="#li" y="150"/>
            ...
            <!-- Schreibe bestimmte Uhrzeit auf jede Linie -->
            <text x="55" y="100">00:00</text>
            <text x="55" y="130">01:00</text>
            <text x="55" y="160">02:00</text>
            <text x="55" y="190">03:00</text>
            ...
            <!-- Zeichne vertikale Linie neben die Uhrzeiten -->
            <use xlink:href="#li2"/>
        </svg>
```

# XSLT zur Generierung der Event-Elemente

Um die Event-Elemente eines Tages, einer Woche oder eines Monats in das entsprechende Template einzutragen wird die XSL Transformation (XSLT), eine Programmiersprache zu Transformation von XML Dokumenten, angewendet. Im ersten Schritt wird der Datei, welche die Beispielevents enthält (sampleEvents.xml) ein Stylesheet zugeordnet, das diese Events absteigend nach Datum und Startzeitpunkt sortiert (sampleEvents.xsl). Das Ergebnis ist eine XML-Datei mit den sortierten Event-Elementen (events_sortiert.xml). Aus dieser Zwischendatei heraus werden dann die entsprechenden Stylesheets, entweder für die Tages-, die Wochen-, oder die Monatssicht aufgerufen (tagessicht.xsl, wochensicht.xls, monatssicht.xsl). Im ersten Schritt haben wir diese Funktionalität nur für einfache Events (Einzelevents, die sich nicht wiederholen) umgesetzt. Später soll dieselbe Vorgehensweise auch für die Superevents genutzt werden, indem ein Superevent als eine Reihe von Einzelevents gesehen wird und ebenfalls nach Datum und Startzeitpunkt sortiert wird.

Nachfolgender Programmcode zeigt beispielhaft, wie die Events aus der sortierten Zwischendatei gefiltert und jeweils als Rechteck, dessen Höhe von der Dauer des Events abhängt, in das oben beschriebene SVG-Template der Tagessicht hineingezeichnet werden.

```
<xsl:template match="/">
        <svg  height="200%" width="100%">
            <!-- Template "tagessicht" wird aufgerufen -->
            <xsl:call-template name="tagessicht"/>

            <!-- Globale Variablen -->
            <xsl:variable name="aktuellesDatum" as="xs:date"
                select="document('aktuellesDatum.xml')/datum"/>

            <!-- Schreibe das Datum des Tages in die obere Mitte
            der Seite (festgelegt in aktuellesDatum.xml) -->
            <text x="600" y="25" text-anchor="middle"
            font-size="20" fill="red">
```

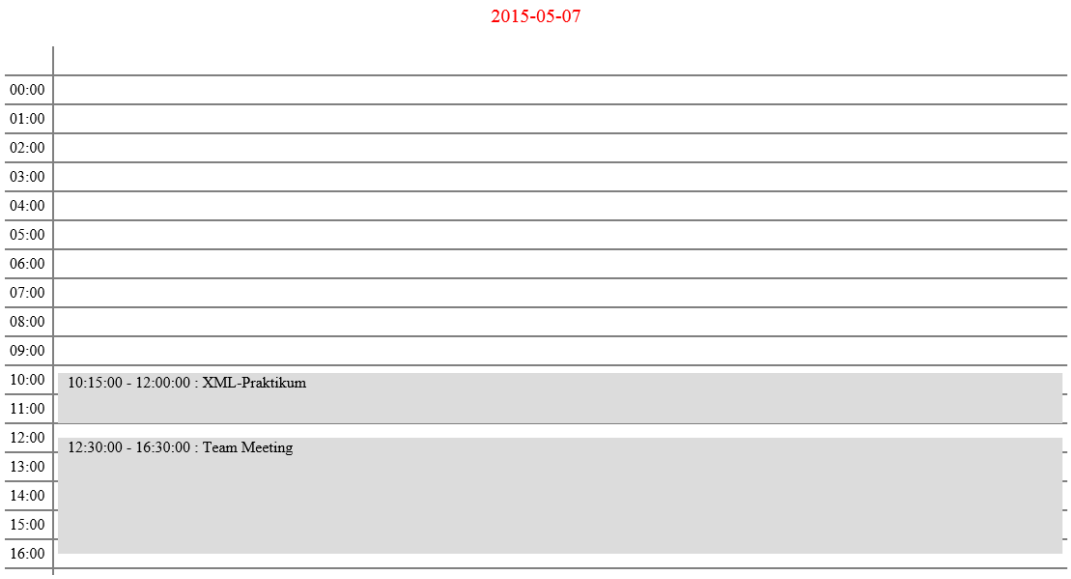
```
                <xsl:value-of select="$aktuellesDatum"/>
            </text>

            <!-- Falls vorhanden, trage die Termine für den
            Tag in das Template ein -->
            <xsl:for-each
                select="document('events_sortiert.xml')/events/
                event[datum = $aktuellesDatum]">
                <xsl:variable name="startRechteck" select="80 +
                (startZeitInMin div 2)"/>
                <xsl:variable name="endeRechteck" select="80 +
                (endZeitInMin div 2)"/>

                <!-- Zeichne ein Rechteck für die Zeitspanne,
                in der ein Termin stattfindet -->
                <rect x="105" y="{$startRechteck}" width="1040"
                    height="{($endeRechteck)-$startRechteck}"
                    fill="gainsboro"/>
                <!-- Schreibe Startzeit, Endzeit und die
                Beschreibung in das Rechteck -->
                <text x="115" y="{$startRechteck +15}">
                    <xsl:value-of select="startZeit"/> -
                    <xsl:value-of select="endZeit"/> :
                        <xsl:value-of select="beschreibung"/>
                </text>
            </xsl:for-each>
        </svg>
    </xsl:template>
```

Das Ergebnis enthält nun das Datum des aktuellen Tages, das Template der Tagessicht und die Rechtecke für die Events an dem betrachteten Tag. Folgendes Bild zeigt die Tagessicht nach der XSL Transformation:

### Figure 2.2. Template Tagessicht




# Wiederverwendung der Templates

Bei der Erstellung der Templates wurde darauf geachtet, dass sie nicht nur für eine Sicht verwendet werden können, sondern auch für die Erstellung der anderen Sichten brauchbar sind.

Mit Hilfe des Befehls <xsl:include href="tagessicht.xsl"/> wird das Stylesheet der Tagessicht, dessen Template in der Wochensicht benötigt wird, in diese mit eingebunden. Genauso wird in der Monatssicht das Stylesheet der Tagessicht eingebunden. Mit Hilfe des Befehls <xsl:call-template name="tagessicht"/> können einzelne Templates aufgerufen werden.

## Uhrzeit

Die Uhrzeit wird durch xs:time dargestellt. Die voreingestellte Darstellung eines xs:time ist Stunden:Minuten:Sekunden (hh:mm:ss). Dank der Formatierungsfunktion format-time() in XSLT 2.0 kann man xs:time Werte verwenden und dann das Format dieser ändern. Somit können wir die Sekunden in den Sichten weglassen und erhalten stattdessen die Darstellung Stunden:Minuten (hh:mm).

## Einbindung der FuncX Bibliothek

Für die Manipulation von Datums- und Zeitangaben wird die XSLT-Bibliothek FuncX (http://www.xsltfunctions.com/) verwendet. Die Bibliothek wird v.a. dazu benötigt, um die Wochen und Monate mit den richtigen Daten zu füllen. Mit Hilfe ihrer Funktionen können so beispielsweise der erste Tag im Monat, die Anzahl der Tage eines Monats, die Nummer eines Tages im Jahr, u.v.m. bestimmt werden.

# Benutzeroberfläche

Beschreibung der unterstützen Seiten, den darauf angebotenen Interaktionsmöglichkeiten und dem Layout, umgesetzt in XHTML, CSS und XForms

# Protokoll zwischen Client und Server

HTTP-Nachrichten mit URLs und XML-Daten

# Server-Komponente

Folgende Technologien werden benutzt: XQuery, XSLT und XProc.

## XProc

An der Server-Seite werden die Kunden-Anfragen einen XProc-Prozess auslösen, bei dem das angegebene Datum als Element im XML-Dokument aktuellesDatum.xml gespeichert und mit Schema aktuellesDatum.xsd validiert wird.

Als nächstes, wird das XML-Dokument sampleCalendarX.xml durch XQuery-XSLT-Pipe transformiert . Als Output erhalten wir entsprechenden Browser-Sicht. XProc erlaubt es, Workflow zu automatisieren und veranschaulichen.

Bemerkung: In der eXist-Umgebung gibt es die Möglichkeit, diesen Prozess mithilfe von Funktion transform:transform innerhalb von XQuery durchzuführen, wo man ein Dokument, ein Stylesheet und die Transformationsparameter als Argumente eingibt, und das Ergebnis bekommt. Wir haben uns dennoch letztendlich für XProc entschieden, und zwar aus folgenden Gründen.

Erstens, ist die Unterstützung von dieser Funktion in eXist noch nicht ganz ausgereift, was Fehlermeldungen bedeutet. (Dasselbe gilt übrigens auch für XProc-Module in eXide). Darüber hinaus, wird bei der Benutzung von XProc die Architektur (*welche* Operationen *wann* passieren) von den eigentlichen Code für diesen Operationen (*wie* sie ausgeführt werden) getrennt: Separation of

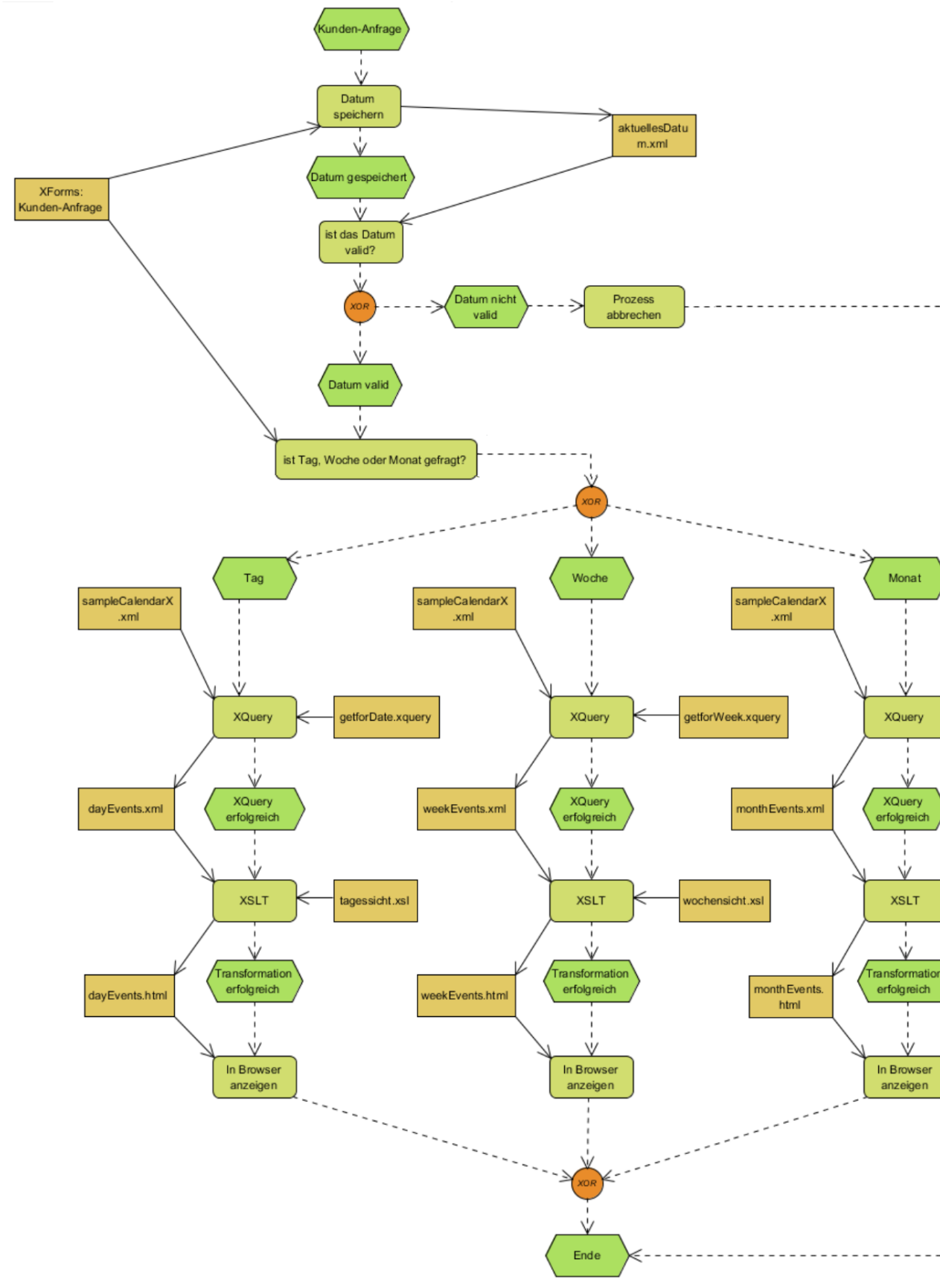
Concerns. Eine Integration von unterschiedlichen Technologien (XQuery, XSLT, etc.) innerhalb von XProc ist auch nicht zu unterschätzen.

Um XProc durchführen zu können, mussten wir alle XQueries, die wir als Version 3.0 geschrieben haben, als Version 1.0 umzuschreiben. Saxon akzeptiert Version 3.0 nicht, auch wenn sie explizit erlaubt wurde. Es gibt auch angeblich keine Pläne, 3.0 in der Zukunft als Default-Version zu setzen. Saxon im Calabash nehme sie als nicht ausgereift genug dafür an, so Oxygen-Webseite. Die Umstellung lief ganz problemlos: Außer einer filter-Funktion, waren alle von uns deklarierten Funktionen mit Version 1.0 völlig kompatibel.

Der Pipe-Prozess läuft nach der Datumsvalidierung folgendermaßen ab. Als Input kommt das Dokument sampleCalendarX.xml rein. Dann wird Kunden-Anfrage getestet: ob Tages-, Wochen- oder Monats-Events gefragt wurde. Dies erfolgt via p:choose step, wobei eine Verzweigung von drei Optionen entsteht. Jede Option löst eine eigene Subpipe aus, wobei in jeder Subpipe das XML-Dokument sampleCalendarX.xml als Input reingeht. Wenn die Option "Tag" gewählt wurde, erfolgt auf Input zunächst XQuery mit getforDate.xquery. Dieser Step erzeugt als Output eine XML-Zwischendatei dayEvents.xml, welche eine geordnete Sequenz von Events an diesem Tag darstellt. Sie geht als Input in den XSLT-Transformation-Step. Dort wird Kalendersicht im Browser mithilfe von tagessicht.xsl erzeugt. Dabei wird als sekundäres Output eine Zwischendatei dayEvents.html gespeichert. Für die Option "Monat" erfolgt XQuery mit getforMonth.xquery und danach die Transformation mit monatssicht.xsl. Die "Woche"-Subpipe funktioniert genau so mitgetforWeek.xquery und wochensicht.xsl

Output für alle drei Subprozesse wird der in Browser angezeigte Kalendersicht.

**Figure 2.3. EPK: XProc-Prozess**



Ein anderer Prozess wird bei der Events-Änderung ausgelöst. Die Änderung wird durchgeführt und in Datenbank gespeichert, am Ende wird sampleCalendarX.xml mit dem Schema CalendarX.xsd validiert.

# XQuery

Client hat die Möglichkeit, für ein von ihm auserwähltes Datum drei unterschiedliche Anfragen an DB zu stellen:

- Events für dieses Datum
- Events für eine Woche, inklusive dieses Datum
- Events für einen Kalendermonat, inklusive dieses Datum

Module:

Um Durchblick und Wiederverwendung von den Hilfsfunktionen zu gewährleisten, haben wir die Queries in die folgenden Module aufgeteilt:

- functX.xquery - Module

  Das ist eine Sammlung von den FunctX-funktionen, die andere Hilfsfunktionen benutzen. Hauptsächlich sind diese Zeit-verbundenen Funktionen, die in XSLT-Abschnitt bereits besprochen wurden.

- isinPat.xquery (Hilfsfunktionen)- Module

  Eine Sammlung von den aneinanderabhängigen Funktionen.

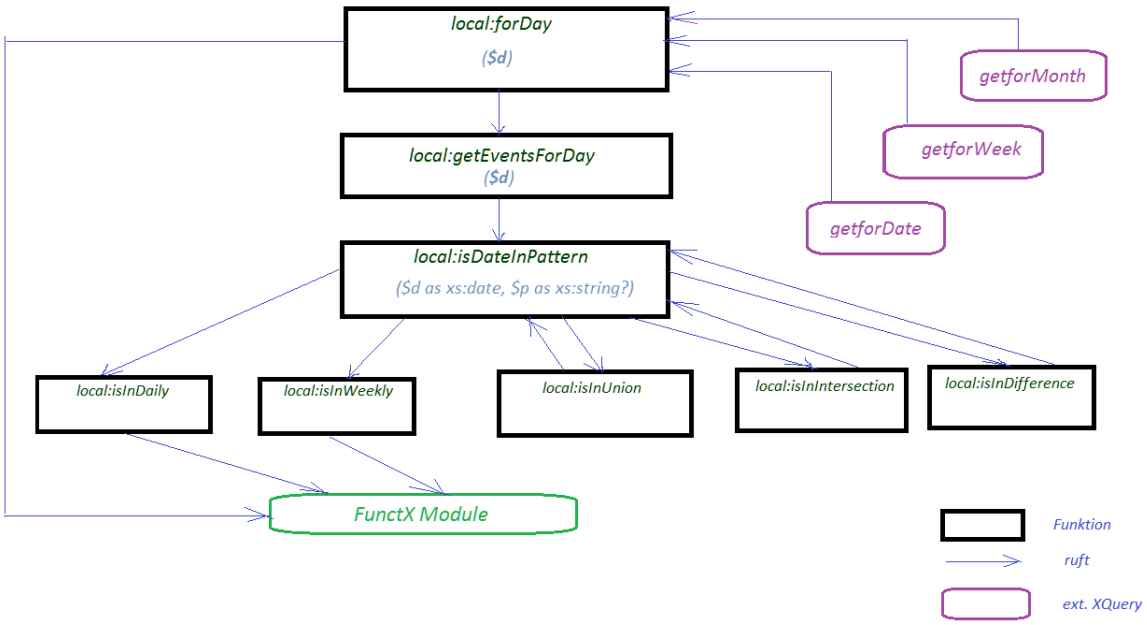
### Table 2.1. Event XQueries Summary

| ***Events*** | **Day Events** | **Week Events** | **Month Events** |
|---|---|---|---|
| *Main .xquery file* | getforDate.xquery | getforWeek.xquery | getforMonth.xquery |
| *additional modules used* | • isinPat.xquery<br>• functX.xquery | • isinPat.xquery<br>• functX.xquery | • isinPat.xquery<br>• functX.xquery |
| *specific declared auxiliary functions* | none | • w:getEventsForWeek($d, $i)<br>• w:getEvents ($d as xs:date?,<br>$i as xs:integer) | • m:getEventsForMonth ($d, $i,<br>• m:getEvents ($d as xs:date?,<br>$i as xs:date, $j as xs:date) |

**ModulisinPat.xqueryund Queries für Tag, Woche und Monat**

Die einzige Hilfsfunktion aus dem "isinPat"-Modul, die von den Day-, Week- und Month-XQueries direkt benutzt wird, ist *local:forDay ($d)* Funktion. .

Folgendes Diagramm zeigt den Zusammenspiel von unterschiedlichen Funktionen innerhalb des "isinPat"-Moduls, sowie mit den externen Module und Queries.




XQuery für die Events an dem Tag heißt ***"getforDate.xquery"***. XQuery *"getforDate"* setzt als Argument das von dem Client angegebene Datum und ruft die Funktion local:forDay($d) aus dem "isinPat"-Modul. Diese erzeugt als Ergebnis eine XML-Datei, welche Root-Element <events> mit einem Attribute "date" - das Datum, hat . Darin enthalten sind <event>Elemente - alle Events, die an diesem Datum stattfinden

```
<?xml version="1.0" encoding="UTF-8"?>
<events date="2015-06-16">
   <event>
      <datum>2015-06-16</datum>
      <datumWochenTag>2</datumWochenTag>
      <datumJahresTag>167</datumJahresTag>
      <startZeit>12:15:00</startZeit>
      <startZeitInMin>735</startZeitInMin>
      <endZeit>13:45:00</endZeit>
      <endZeitInMin>825</endZeitInMin>
      <beschreibung>test 2nd event in day</beschreibung>
      <tagZuvor>2015-06-15</tagZuvor>
      <tagDanach>2015-06-17</tagDanach>
      <location>00.09.13A</location>
      <attendees>
         <attendee>Anne Brüggemann-Klein</attendee>
      </attendees>
   </event>
   <event>
 ...
   </event>
</events>
Ergebnis von getforDate.xquery
```

Die XQuery ***"getforWeek.xquery"*** prüft, an welchem Tag der Woche das gesuchte Datum stattfindet und ruft die Hilfsfunktionw:getEventsForWeek($d, $i) auf, wo $i - Nummer des Wochentages ist. Falls das Datum am Montag vorkommt, ruft die Funktion für dieses Datum die Funktion local:forDay($d) aus dem "isinPat"-Modul. Für den nächsten Tag, ruft sie die zweite Hilfsfunktion "w:getEvents". Falls das Datum an einem anderen Wochentag vorkommt, ruft die Funktion sich selbst rekursiv für den vorigen Tag auf, usw. bis sie zum Montag dieser Woche kommt.

Die Funktion w:getEvents($d, $i) ruft local:forDay($d) aus dem "isinPat"-Modul auf und für den nächsten Tag ruft sie sich selbst auf. Usw. bis sie zum Ende der Woche kommt. Somit entsteht eine Sequenz von den Events für jeden Tag der Woche, die das gesuchte Datum enthält. Das ganze wird unter <events> Root-Element verpackt.

Ähnlich verhält sich auch ***"getforMonth.xquery"***, mit Monatsevents als Ergebnis.

Was passiert innerhalb von "isinPat"-Modul eher das Ergebnis geliefert wird?

Als Erstes, ruft die Funktion *local:forDay($d)* die Funktion *local:getEventsForDay ($d as xs:date?)*. Diese muss eine Sequenz ohne Root-Element von allen an diesem Tag stattfindenden Events , als "eventRule" mit allen dazugehörigen Attribute und Unterelemente, liefern- genauso, wie sie im sampleCalendarX.xml erscheinen.

Um diese Sequenz zu bekommen, geht die Funktion *local:getEventsForDay ($d as xs:date?)* durch alle im sampleCalendarX.xml gespeicherte Events (als "eventRule") und prüft deren "recurrencePattern"-Unterelement mithilfe von Funktion *local:isDateInPattern ($d as xs:date, $p as xs:string?)* :

Diese Funktion nimmt das Datum und den Patternnamen von "recurrencePattern" als Argumente und liefert eine boolische Antwort, ob an diesem Datum dieses Pattern stattfindet.

Weil "recurrencePattern" ausschließlich einen Namen hat, ohne Verweise auf Datum oder Pattern-type, muss die Funktion alle unter <patterns> gespeicherte Patterns durchgehen, bis sie den gesuchten Pattern findet und ihn dann bzgl. des Datums prüft.

```
declare function local:isDateInPattern ($d as xs:date, $p as xs:string?) as xs
{
((local:isInDaily($d, $p)) or (local:isInWeekly($d,$p))  or (local:isInUnion($
};
```

Es gibt 2 einfache und 3 komplexe Patterns, die in Frage kommen. Für jeden Pattern gibt es eine entsprechende Funktion: "isinDaily", "isinWeekly", "isinDifference", isinUnion" und "isinIntersection".

Die simple Patterns sind "daily" und "weekly" Patterns. Bei jedem "daily"-Pattern genügt es, Attribute "startDate" und "endDate" als Argumente der functx:between-inclusive ($d, $dp/@startDate ,$dp/@endDate) übergeben und prüfen, ob das Datum zwischen Anfang und Ende des Patterns liegt. Bei dem "weekly" Pattern kommt die (functx:day-of-week-name-en ($d) eq $wp/@dayOfWeek) Funktion im Spiel, die prüft, an welchem Tag der Woche das gesuchte Datum vorkommt und ob dieser Tag dem "dayofWeek"-Attribut des Patterns entspricht.

"isinDifference", "isinUnion" und "isinIntersection" alle rufen die Funktion *local:isDateInPattern* für "firstPattern" und für jede "furtherPattern" rekursiv auf. Die Ergebnisse liefern true(), wenn:

"isinDifference": das Datum befindet sich im "firstPattern" und in keinem der "furtherPatterns".
"isinUnion": das Datum befindet sich im "firstPattern" oder in einem der "furtherPatterns".
"isinIntersection":das Datum befindet sich im "firstPattern" und in jedem der "furtherPatterns".

Die Gedanke hinter der Aufbau der Funktion *local:isDateInPattern* war, dass weil XQuery eine funktionelle Sprache ist, wird die Funktion alle Pattern-Types nacheinander durchgehen ( als erstes alle daily, dann alle weekly Patterns, etc.), und, sobald der gesuchte Pattern zutrifft, Ergebnis liefern, ohne alle <pattern>-Elemente durchgehen zu müssen. Das spart bei der Suche die Zeit und lässt auch den rekursiven Aufrufe die Antwort schneller zu bekommen.

# Chapter 3. Reflexion

## Organisation der Arbeit im Team

## Thematik CalendarX

Text

## Organisation und Betreuung im Praktikum

Text