M. Bagić Babac, M. Kušek

Preddiplomski studij

Računarstvo

Telekomunikacije i informatika

Informacija, logika i jezici

Skripta: Jezici za označavanje sadržaja

Ak. g. 2011./2012.

Slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **remiksirati** — prerađivati djelo

pod sljedećim uvjetima:

- **imenovanje**. Morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno**. Ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima**. Ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela. Najbolji način da to učinite je linkom na ovu internetsku stranicu.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licencije preuzet je s http://creativecommons.org/.

Sadržaj

1 UVOD U XML 2

1.1 Osnovni elementi i struktura 2
1.2 Oblikovanost i valjanost dokumenata 4

2 KORIŠTENJE XML-a U JAVI 6

2.1 Knjižnica XStream 6

2.1.1 Serijalizacija objekata u XML 8
2.1.2 Deserializacija objekata iz XML-a 9
2.1.3 Podešavanje parametara serijalizacije i deserijalizacije 9

2.2 JAXB 11

2.2.1 Obilježavanje klasa 11
2.2.2 Preslikavanje objetka u XML 13
2.2.3 Preslikavanje XML-a u objekt 14

2.3 DOM 14

2.3.1 Stvaranje dokumenta u memoriji 14
2.3.2 Pretvaranje DOM stabla u XML 15
2.3.3 Stvaranja DOM stabla koji predstavlja objekt 16
2.3.4 Učitavanje XML dokumenta i stvaranje DOM stabla 16

2.4 JDOM 17

2.4.1 Stvaranje DOM stabla u memoriji 17
2.4.2 Pretvaranje DOM stabla u XML 18
2.4.3 Stvaranja DOM stabla koji predstavlja objekt 18
2.4.4 Učitavanje XML dokumenta i stvaranje DOM stabla 18

2.5 SAX 19

3 STRUKTURIRANJE XML DOKUMENATA 23

3.1 Prostor imena elemenata 23
3.2 DTD 24
3.3 XML Schema 27

4 ODABIR ELEMENATA POMOĆU XPath-a 31

4.1 Lokacijski putovi 32
4.2 Složeni lokacijski putovi 33
4.3 Neskraćeni lokacijski putovi 35

5 TRANSFORAMACIJA XML DOKUMENATA 36

5.1 Jezik XQuery 36

5.1.1 Adresiranje dokumenta 36
5.1.2 Stvaranje elemenata 36
5.1.3 Stvaranje atributa 38
5.1.4 Izrazi FLWOR 38
5.1.5 Izvođenje XQuerya u Javi 40

5.2 Jezik XSLT 41

5.2.1 Predlošci 44
5.2.2 Ugrađeni predlošci 45
5.2.3 Sadržaj elementa 45
5.2.4 Grananje 46
5.2.5 Korištenje XSLT-a u Javi 47

6 Literatura 48

1 UVOD U XML

Kratica **XML** označava ***eXtensible Markup Language***, dakle proširljivi jezik za označavanje. XML nastao je iz potrebe prijenosa i skladištenja podataka, a ne zbog njihova prikaza, za razliku od HTML-a kojemu je sličan po oznakama. Ali se ipak radi o potpuno drukčijim paradigmama. HTML ima definirani skup oznaka s univerzalnim značenjem pa nije moguće stvaranje vlastitih oznaka. HTML služi oblikovanju webskih sadržaja, ali ne omogućava strukturiranje podataka. Upravo XML omogućava strukturiranje podataka i fleksibilnost korisnika. Osnovna razlika između ova dva jezika je da XML zanima što je podatak, a HTML zanima kako podatak izgleda. XML nosi podatak neovisno o softverskoj i hardverskoj platformi.
XML omogućuje korisniku definiranje vlastitih oznaka odnosno definiranje jezika specifičnog za određeno područje primjene. Budući da se XML-om teži odvojenom prikazu podataka od samih podataka, da bi se jedan XML dokument prikazao u WWW pregledniku, potrebno je koristiti XSL (eng. *eXtensible Style Language*) ili CSS (eng. *Cascading Style Sheet*), tehnologije koje podržavaju XML. Ako treba prikazati dinamičke podatke u HTML dokumentu, svaki put kad se nešto promijeni trebat će dosta posla za urediti HTML dokument. S XML-om podaci mogu biti pohranjeni u posebnim XML datotekama. Na ovaj način možemo se potpuno skoncentrirati na HTML za prikaz podataka i osigurati se da promjene u samim podacima neće zahtijevati promjene u HTML-u.
Stvarni računalni sustavi i baze podataka sadržavaju podatke u nekompatibilnim formatima. XML pohranjuje podatke u običnom tekstualnom formatu čime se omogućava neovisnost pohrane podataka o softveru i hardveru, a olakšano je i dijeljenje podataka među različitim aplikacijama. Ovdje leži i najveći izazov istraživačima – kako preko interneta razmijeniti podatke između nekompatibilnih sustava, a XML pruža ohrabrujući odgovor na to.
Nadgradnje postojećih sustava, bilo sofverskih ili hardverskih platformi, obično zahtijeva jako puno vremena. Treba konvertirati velike količine podataka pa je čest gubitak nekompatibilnih podataka. Budući da XML pohranjuje podatke u običnom tekstualnom formatu, njime je olakšana ekspanzija ili ažuriranje novih operacijskih sustava, aplikacija, preglednika i slično, bez gubitaka podataka. Također, XML podatke mogu čitati razne vrste „čitajućih“ uređaja pa i oni koji pomažu osobama koje dobro ne vide ili imaju neke druge mane.

1.1 Osnovni elementi i struktura

XML dokument sastoji se od prologa, elemenata i eventualno epiloga.
Prolog se sastoji od XML deklaracije i eventualno reference na vanjski dokument za strukturiranje. Npr.

```
<?xml version="1.0" encoding="UTF-16"?>
```

Ova deklaracija specificira da se radi o XML dokumentu te definira vrstu kodiranja znakova prema odgovarajućem sustavu (ovdje UTF-16), što inače nije obvezno navoditi, ali se smatra dobrom praksom navesti je. Također, ponekad specificiramo odnosi li se dokument na neki vanjski dokument za strukturiranje.

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
```

Refereca na vanjski dokument za strukturiranje izgleda ovako:

```
<!DOCTYPE book SYSTEM "book.dtd">
```

Informacija o strukturi ovdje se pohranjuje u lokalnu datoteku `book.dtd`. Referenca može biti i internetski resurs (URL). U slučaju lokalnog imena ili URL-a, koristi se oznaka SYSTEM, a ako se koriste oba izvora, onda treba koristiti oznaku PUBLIC.
XML elementi govore o čemu se u XML dokumentu „radi" i oni čine njegov glavni sastojak. Svaki element ima svoju otvarajuću oznaku, sadržaj i zatvarajuću oznaku. Npr.

```
<student>Bero Berislavić</student>
```

Imena oznaka su potpuno proizvoljna, postoji tek nekoliko ograničenja. Najvažnije je da prvi znak mora biti slovo, donja crta ili dvotočka ne smije počinjati s „xml" slovima bez obzira na velika i mala slova. Sadržaj elementa može biti tekst ili drugi znakovi, ili ništa. Npr.

```
<zavod>
  <ime>Zavod za telekomunikacije</ime>
  <faks>0038516129832</faks>
</zavod>
```

Ako nema sadržaja, kažemo da je element prazan. Prazni element izgleda ovako:

```
<student></student>
```

i može se kraće zapisati ovako: `<student/>`.
Prazni element ne mora biti ni najmanje beznačajan jer on može sadržavati atribute. Atributi su parovi „ime - vrijednost" unutar otvarajuće oznake elementa.

```
<student ime="Bero" prezime="Berislavić"/>
```

Primjer praznog elementa koji nije prazan:

```
<zavod id="ZTEL">
  <predmet ime="Informacija logika i jezici" broj-studenata="135"/>
  <labosi broj="6" tema="Java"/>
</zavod>
```

Isti se element može zapisati i na sljedeći način ako zamijenimo atribute ugniježđenim elementima:

```
<zavod>
   <id>ZTEL</id>
      <predmet>
          <ime>Informacija, logika i jezici</ime>
          <broj-studenata>135</broj-studenata>
      </predmet>
      <labosi>
         <broj>6</broj>
         <tema>Java</tema>
      </labosi>
</zavod>
```

Nema univerzalnog recepta kad koristiti elemente, a kad atribute pa je to stvar ukusa, ali svakako treba primijetiti da se atributi ne mogu ugnježđivati.
Komentari su dijelovi teksta koje XML *parser* zanemaruje, a izgledaju ovako:

```
<!—Ovo je komentar-->
```

Naredbe za obradu (engl. *processing instructions*) omogućavaju informiranje aplikacije koja koristi naš dokument o tome kako ga obraditi. Opća je forma:

```
<?ciljana naredba?>
```

Npr. `<?stylesheet type="text/css" href="mystyle.css"?>` omogućava proceduralnu obradu iako se načelno radi o deklarativnom tekstu.

1.2 Oblikovanost i valjanost dokumenata

XML dokument je dobro oblikovan (engl. *well-formed*) ako je sintaktički korektan. Neka od osnovnih sintaktičkih pravila su sljedeća:

- samo je jedan korijen dokumenta, a tako se i zove, element korijen (root element),
- svaki element mora imati otvarajuću i zatvarajuću oznaku.
- oznake se ne mogu preklapati,
- (npr. `<autor><ime>Kafka</autor></ime>` nije dobro definirano).
- atributi unutar imena moraju imati jedinstvena imena,
- vrijednosti atributa moraju biti zatvorene u navodnike,
- elementi i imena oznaka moraju biti dopustivi,
- unutar oznaka ne smije biti komentara niti naredbi za obradu i
- u znakovnim podacima elemenata i atributa ne smije biti znakova „<“ ili „&“ koji nisu pretvoreni u izlazni slijed.

Dobro definiran XML dokument može se ilustrirati stablom pa zbog toga stablo i čini formalnu osnovu XML dokumenta.

```
<?xml version="1.0" encoding="UTF-16">
<!DOCTYPE email SYSTEM „email.dtd">
<email>
   <head>
      <from name="Perica" address="pero@fer.hr"/>
      <to name="Berislav" address=bero@fer.hr/>
      <subject>hej</subject>
   </head>
   <body>Bero... gdje ti je pero?
   </body>
</email>
```

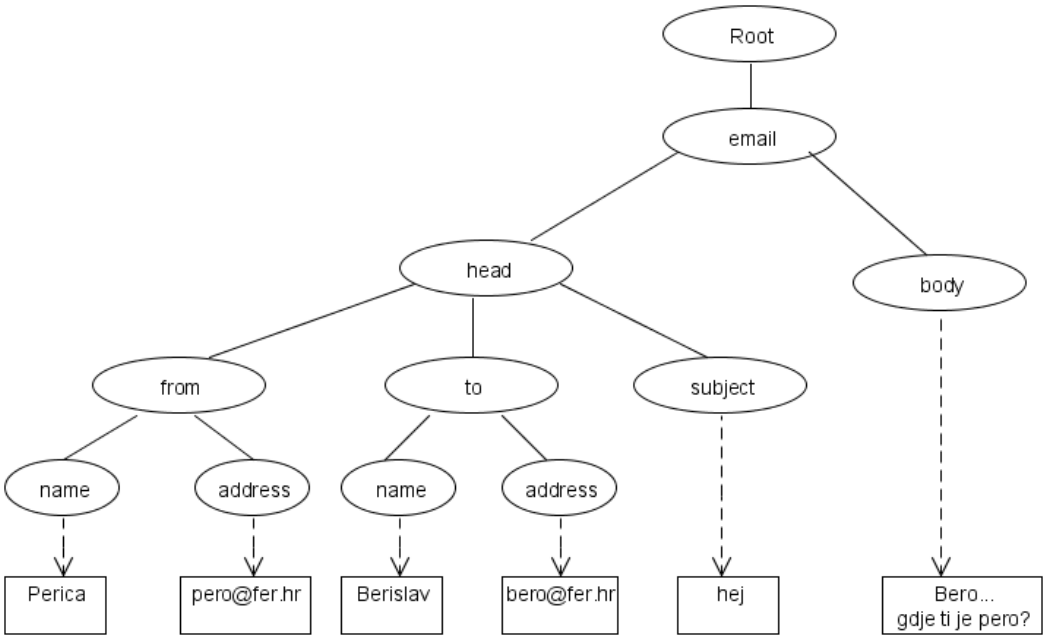
Slika 1.1. prikazuje stablastu strukturu XML dokumenta koju odlikuju sljedeća svojstva:

- samo je jedan korijen,
- nema zatvorenih petlji,
- svaki čvor, osim korijena, ima točno jedan roditeljski čvor,
- svaki čvor ima svoju oznaku i
- redoslijed elementa je bitan.

Iako je redoslijed elementa bitan, redoslijed atributa nije pa su sljedeća dva elementa ekvivalentna:

```
<dijagram čega="stanja" vrsta="statički">
<dijagram vrsta="statički" čega="stanja">
```

Slika predstavlja važnu razliku između korijena dokumenta (engl. *root*) i korijenskog elementa (u ovom primjeru element „email“). Ova je razlika važna kod adresiranja i kreiranja upita kod XML dokumenata.



Slika 1.1. Stablasti prikaz XML dokumenta

2 KORIŠTENJE XML-a U JAVI

Tipično korištenje bilo kojeg formata u nekom programskom jeziku je učitavanje podataka s diska ili s mreže i stavljanje u strukturu u memoriji te spremanje strukture iz memorije u format zapisa. XML nije iznimka pa se tako korištenje XML-a u Javi može svesti na učitavanje i spremanje XML-a. Ovisno o tome koje mogućnosti koristimo u XML-u i koliko je veliki dokument koji koristimo ovisi i knjižnica koju ćemo koristiti za učitavanje i spremanje XML-a.

2.1 Knjižnica XStream

Knjižnica XStream koristi se za serijalizaciju objekata u XML i deserijalizaciju iz XML-a u objekte. Ona je jednostavna za korištenje, nisu potrebna posebna podešavanja preslikavanja objekata u XML i obrnuto. Podešavanje kako će se neki objekt pretvoriti u XML su moguća, ali nisu nužna. Performanse su zadovoljavajuće jer koristi malo memorije i pretvorba je brza što je ključno za velike dokumente ili objekte. XML dokument koji predstavlja objekt je jednostavan i lagano ga je pročitati za razliku od standardne serijalizacije koju koristi Java. Objekte koji se serijaliziraju nije potrebno mijenjati.

Kao primjer objekata koje ćemo serijalizirati uzeti ćemo klasu `DifferentialCounter` sa sljedećim kodom:

```
package hr.fer.tel.ilj;

public class DifferentialCounter {
  public SimpleCounter cntr1 = new SimpleCounter();
  public SimpleCounter cntr2 = new SimpleCounter();

  public void increment1() {
    this.cntr1.increment();
  }

  public int getCntr1() {
    return cntr1.getState();
  }

  public void increment2() {
    this.cntr2.increment();
  }

  public int getCntr2() {
    return cntr2.getState();
  }

  public int getDifference() {
    return this.cntr2.getState() - this.cntr1.getState();
  }

  public void reset() {
    this.cntr1 = new SimpleCounter();
    this.cntr2 = new SimpleCounter();
  }

  public String toString() {
    return "DifferentialCounter: cntr1=" + cntr1.getState() +
      ", cntr2=" +  cntr2.getState();
  }
}
```

Kao što vidimo u kodu ova klasa ima atribute koji su vrste `SimpleCounter`. Pogledajmo kod klase `SimpleCounter`:

```
package hr.fer.tel.ilj;

public class SimpleCounter extends AbstractCounter {
  public SimpleCounter(int initialState) {
    super(initialState);
  }

  public SimpleCounter() {
  }

  public void increment() {
    this.counterState++;
  }
}
```

Klasa `SimpleCounter` nasljeđuje klasu `AbstractCounter` čije je implementacija sljedeća:

```
package hr.fer.tel.ilj;

public abstract class AbstractCounter implements Counter {
  int counterState;

  public AbstractCounter(int initialState) {
    this.counterState = initialState;
  }

  public AbstractCounter() {
  }

  public int getState() {
    return this.counterState;
  }

  public abstract void increment();
}
```

Da bismo jedan objekt klase `DifferentialCounter` serijalizirali u XML u postupku serijalizacije se mora serijalizirati svaki atribut tj. U ovom slučaju će se serijalizirati atributi `cntr1` i `cntr2` koji su objekti klase `SimpleCounter` koji nasljeđuje `AbstractCounter`. To zapravo znači da će se serijalizacijom objekta `DifferentialCounter` knjižnica XStream serijalizirati i objekte vrste `SimpleCounter` tj. `AbstractCounter`. Knjižnica XStream serijalizira samo atribute. Sljedeći kod pokazuje pripremu knjižnice XStream, serializaciju i deserijalizaciju objekata:

```
  public static void main(String[] args) {
    DifferentialCounter dc1 = createCounters();

    XStream xs = prepareXStream();

    String xml = serializeObjectToXMLString(dc1, xs);

    deserializeObjectFromXMLString(xs, xml);
  }
```

Kao što vidimo u kodu prvo se stvara objekt vrste `DifferentialCounter` u metodi `createCounters`. Nakon toga se priprema knjižnica XStream u metodi `prepareXStream`, te se iza toga vrši serijalizacija objekta vrste

`DifferentialCounter` u XML dokument koji je spremljen u niz znakova (metoda `serializeObjectToXMLString`), a nakon toga se iz niza znakova deserijalizacijom natrag stvara objekt (metoda `deserializeObjectFromXMLString`).
U metodi `createCounters` se stvori objekt dc1 i nad njim se pozivaju metode `increment1` i `increment2` tako da ga postave u stanje koje ćemo serijalizirati. Nakon toga se ispiše trenutno stanje objekta.

```
private static DifferentialCounter createCounters() {
  DifferentialCounter dc1 = new DifferentialCounter();
    dc1.increment1();
    dc1.increment1();
    dc1.increment2();
    System.out.println( "Pocetni objekt: " + dc1 + "\n\n" );
  return dc1;
}
```

U metodi `prepareXStream` se vrši priprema za serializaciju i deserijalizaciju objekata. U njoj se stvara objekt XStream koji se vraća natrag jer je on zadužen za serijalizaciju i deserijalizaciju. Minimalna konfiguracija je da se samo stvori objekt.

```
private static XStream prepareXStream() {
  XStream xs = new XStream();
  return xs;
}
```

2.1.1 Serijalizacija objekata u XML

U metodi `serializeObjectToXMLString` vrši se serializacija objekta u XML dokument. To se radi tako da se u objaktu `xs` pozove metoda `toXML` koja vraća `String` koji sadrži serijalizirani objekt. Nakon serializacije se XML ispisuje i vraća iz ove metode.

```
private static String serializeObjectToXMLString(
  DifferentialCounter dc1, XStream xs)
{
  String xml = xs.toXML( dc1 );
  System.out.println( xml );
  return xml;
}
```

Ispis ove metode je sljedeći:

```
<hr.fer.tel.ilj.DifferentialCounter>
  <cntr1>
    <counterState>2</counterState>
  </cntr1>
  <cntr2>
    <counterState>1</counterState>
  </cntr2>
</hr.fer.tel.ilj.DifferentialCounter>
```

Ovdje vidimo da je objekt `DifferentialCounter` stavljen u element s imenom `hr.fer.tel.ilj.DifferentialCounter`. Unutar njega imamo po jedan element za svaki atribut. Kako imamo dva atributa `cntr1` i `cntr2` onda u XML-u imamo dva elementa čija imena su jednaka imenima atributa. Ti atributi su vrste `SimpleCounter` tako da u njima postoje elementi koji predstavljaju atribute iz klase `SimpleCounter`. Pošto `SimpleCounter` nema deklarirane atribute to bi značilo da elementi `cntr1` i `cntr2` nemaju sadržaj, ali to nije tako. Klasa `SimpleCounter` ima nasljeđeni atribut `counterState` iz klase `AbstractCounter` i on se stavlja u sadržaj elementa. Atribut `counterState` je primitivne vrste int pa se njegov sadržaj stavlja kao tekstualni element unutar elementa `counterState`.

Kada bismo imali veliki objekt s puno atributa, a ti atributi su objekti koji imaju još atributa i tako duboko u hijerarhiji i kada bismo željeli takav objekt serijalizirati rezulata serijalizacije bi mogao biti `String` koji je toliko velik da zauzima puno memorije procesorski je zahtjevno s njim manipulirati ili bi se moglo dogoditi da takav `String` nije moguće napraviti jer je previše znakova u njemu (`String` ima ograničenje veličine na `Integer.MAX_VALUE` jer je interno `String` implementiran kao polje `char`-ova). Da nam se to ne dogodi objekte možemo serijalizirati direktno u tok podataka (npr. datoteku).

```
Writer w = new FileWriter( "diffCounter.xml" );
xs.toXML( dc1, w );
```

U prethodnom kodu smo napravili toko podataka u datoteku i njega proslijedili kao drugi parametar metode `toXML`. Postoji dvije varijante metode `toXML`, jedna prima kao drugi argument `Writer`, a druga `OutputStream`.
Iz ispisa serijalizirnog koda vidimo da je objekt koji je serijaliziran stavljen u element `hr.fer.tel.ilj.DifferentialCounter` što nije u duhu XML-a. Ako bismo željeli to promijeniti onda se treba u metodi za pripremu serijalizacije i deserializacije postaviti određene postavke. To će biti opisano malo kasnije.

2.1.2 Deserializacija objekata iz XML-a

Deserijalizacija objekata vrši se u metodi `deserializeObjectFromXMLString`:

```
private static void deserializeObjectFromXMLString(
  XStream xs, String xml)
{
  DifferentialCounter dc2 = (DifferentialCounter) xs.fromXML( xml );
  System.out.println( "\n\nObjekt stvoren iz XML-a: " + dc2 );
}
```

Kao što vidimo iz koda deserializacija se vrši pozivom metode `fromXML` koja za parametar ima XML dokument iz kojeg treba pročitati podatke i stvoriti objekt. Ova metoda vraća deserijalizirani objekt. Pošto u općem slučaju metoda `fromXML` može deserijalizirati bilo koji objekt onda je u deklaraciji metode stavljeno da vraća vrstu `Object`. Ako želimo pristupiti metodama objekta koji je vraćen potrebno je prvo *castati* objekt u neki konkretan objekt (u ovom slučaju `DifferentialCounter`) i onda ga dalje koristiti. Nakon deserializacije u ovoj metodi se ispiše objekt tako da se u ispisu vidi što je pročitano.
Kao i kod serijalizacije ako je XML dokument velik onda je neefikasno prvo pročitati čitav objekt iz nekog toka u `String` u memoriji pa ga onda deserijalizirati. Zbog toga postoji metoda `fromXML` koja kao parametar prima toko podataka (`Reader` ili `InputStream`). Sljedeći primjer to pokazuje:

```
Reader r = new FileReader( "diffCounter.xml" );
DifferentialCounter dc2 = (DifferentialCounter) xs.fromXML( r );
```

2.1.3 Podešavanje parametara serijalizacije i deserijalizacije

Ako želimo da se u serializiranom XML-u umjesto oznake `hr.fer.tel.ilj.DifferentialCounter` koristi drugo ime onda je potrebno podesiti parametre u objektu `XStream`. Dakle, za promjenu imena oznake koristimo sljedeći kod:

```
private static XStream prepareXStream() {
  XStream xs = new XStream();
  xs.alias( "diff-counter", DifferentialCounter.class );
  return xs;
}
```

Podebljani redak pozivom metode `alias` postavlja drugo ime za klasu `DifferentialCounter`. Novo ime je `diff-counter`. Rezultat serijalizacije je:

```
<diff-counter>
  <cntr1>
    <counterState>2</counterState>
  </cntr1>
  <cntr2>
    <counterState>1</counterState>
  </cntr2>
</diff-counter>
```

Ako želimo da se atributi `cntr1` i `cntr2` u XML-u nalaze u elementu s imenima `start` i `end` onda koristimo sljedeći kod inicijalizacije `XStream`-a:

```
private static XStream prepareXStream() {
  XStream xs = new XStream();
  xs.alias( "diff-counter", DifferentialCounter.class );
  xs.aliasField("start", DifferentialCounter.class, "cntr1");
  xs.aliasField("end", DifferentialCounter.class, "cntr2");
  return xs;
}
```

Poziv metode `aliasField` postavlja konfiguraciju tako da atribut `cntr1` u klasi `DifferentialCounter` u XML-u prikazuje elementom `start`. Na sličan način se postavlja konfiguracija za atribut `cntr2`. Rezultat serijalizacije je sljedeći:

```
<diff-counter>
  <start>
    <counterState>2</counterState>
  </start>
  <end>
    <counterState>1</counterState>
  </end>
</diff-counter>
```

Ako želimo da se element counterState prikaže kao atribut elementa start odnosno end gdje je ime atributa value onda moramo koristiti sljedeći kod za konfiguraciju XStream-a:

```
private static XStream prepareXStream() {
  XStream xs = new XStream();
  xs.alias( "diff-counter", DifferentialCounter.class );
  xs.aliasField("start", DifferentialCounter.class, "cntr1");
  xs.aliasField("end", DifferentialCounter.class, "cntr2");
  xs.useAttributeFor(AbstractCounter.class, "counterState");
  xs.aliasField("value", AbstractCounter.class, "counterState");
  return xs;
}
```

Prvi podebljani redak pozivom metode `useAttributeFor` postavlja u konfiguraciju da se atribut `counterState` u klasi `AbstractCounter` prikaže kao atribut elementa u kojem se nalazi. Drugi podebljani redak pomoću metode `aliasField` definira da se ime atributa `counterState` iz klase `AbstractCounter` u XML-u zove `value` tj. u ovom slučaju će to biti naziv atributa u elementu, a u prethodnom primjeru je to bio naziv elementa. Rezultat ovakve konfiguracije je sljedeći:

```
<diff-counter>
  <start value="2"/>
  <end value="1"/>
</diff-counter>
```

Osim ovih postavki moguće je još definirati i prilagođeno pretvaranje pojedinih klasa, ali to je napredno korištenje knjižnice XStream koji je izvan sadržaja ove skripte.

2.2 JAXB

JAXB je akronim za *Architecture for XML Binding*. JAXB služi za procesiranje XML dokumenata tako da dokumente preslikava u objekte i obrnuto. Preslikavanja se mogu definirati u posebnom specifikacijskom XML dokumentu ili pomoću bilješki (*annotations*) u kodu što je uvedeno i inačici 2.0.

2.2.1 Obilježavanje klasa

Za primjer ćemo obilježiti klasu `DifferentialCounter` i `SimpleCounter` i njih preslikati u XML i obrnuto.

```
package hr.fer.tel.ilj;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="diff")
public class DifferentialCounter {
  @XmlElement
  private SimpleCounter cntr1 = new SimpleCounter();

  @XmlElement
  private SimpleCounter cntr2 = new SimpleCounter();

  public void increment1() {
    this.cntr1.increment();
  }

  public int getCntr1() {
    return cntr1.getState();
  }

  public void increment2() {
    this.cntr2.increment();
  }

  public int getCntr2() {
    return cntr2.getState();
  }

  public int getDifference() {
    return this.cntr2.getState() - this.cntr1.getState();
  }

  public void reset() {
    this.cntr1 = new SimpleCounter();
    this.cntr2 = new SimpleCounter();
  }

  public String toString() {
    return "DifferentialCounter: cntr1=" + cntr1.getState() +
      ", cntr2=" +  cntr2.getState();
  }
}
```

Bilješke JAXB-a se nalaze u paketu `javax.xml.bind.annotation`. Klasa je obilježena `@XmlRootElement(name="diff")` što predstavlja da ta klasa može biti

korijenski element dokumenta i u tom slučaju će korijenski element imati naziv `diff`. Osim obilježavanja klase potrebno je obilježiti i atribute koji se mogu preslikati ili u atribute elementa `diff` ili u elemente unutar elementa `diff`. U ovom primjeru su atributi `cntr1` i `cntr2` obilježeni bilješkom `@XmlElement` što znači da će biti preslikani u elemente. Pošto su ti atributi objekti klase `SimpleCounter` onda je potrebno obilježiti i klasu `SimpleCounter`.

```
package hr.fer.tel.ilj;

import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="counter")
public class SimpleCounter extends AbstractCounter {
  public SimpleCounter(int initialState) {
    super(initialState);
  }

  public SimpleCounter() {
  }

  public void increment() {
    this.counterState++;
  }

  @XmlAttribute(name="value")
  @Override
  public int getState() {
    return super.getState();
  }

  public void setState(int state) {
    counterState = state;
  }
}
```

Iz ovog primjera se vidi da je klasa obilježena bilješkom `@XmlRootElement(name="counter")`, a za razliku od prošlog primjera ovdje je obilježena metoda `getState` bilješkom `@XmlAttribute(name="value")`. Dakle, u klasi možemo obilježiti ili atribut ili metodu `get` nekog atributa s istim efektom. U ovom slučaju je obilježena metoda. Metoda je obilježena tako da se atribut `state` preslikava u atribut elementa s imenom `value`. Kako bi bilo omogućeno preslikavanje u oba smjera onda je za atribute koji su obilježeni metodama potrebno imati i metode `set` i `get`. Zbog toga je dodana metoda `setState` u ovu klasu.
Najvažnije bilješke su:

- `XmlRootElement` – bilježi klasu
  - može biti dokument
  - dodatni elementi:
    - `name` – naziv elementa u XML-u
    - `namespace` – naziv prostora imena za taj element u XML-u
- `XmlElement` – bilježi atribut ili metodu `get`
  - atribut klase će u XML-u biti element
  - dodatni elementi: `name`, `required`, `namespace`
- `XMLAttibute` – bilježi atribut ili metodu `get`
  - atribut klase će u XML-u biti element
  - dodatni elementi: `name`, `required`, `namespace`

- `XmlTransient` – bilježi atribut ili metodu `get`
  - preskače preslikavanje atributa klase

2.2.2 Preslikavanje objekta u XML

Preslikavanje objekta u XML se vrši pomoću API-ja koji se prvo inicijalizira i nakon toga koriste objekti za preslikavanje. Sve klase koje se koriste za preslikavanje se nalaze u paketu `javax.xml.bind`. Prvo je potrebno stvoriti objekte koje ćemo preslikavati:

```
SimpleCounter sc = new SimpleCounter();
sc.increment();

DifferentialCounter dc1 = new DifferentialCounter();
dc1.increment1();
dc1.increment1();
dc1.increment2();
System.out.println( "Pocetni objekt: " + dc1 );
```

Kod preslikavanja se mogu baciti iznimke `JAXBException` pa je potrebno taj dio koda staviti u blok `try/catch`. Nakon stvaranja objekata slijedi inicijalizacija konteksta:

```
JAXBContext context = JAXBContext.newInstance(
  DifferentialCounter.class, SimpleCounter.class);
```

Kontekst se inicijalizira pozivom statičke metode `newInstance` koja za parametre prima klase koje mogu biti korijenski element XML-a. U ovom slučaju su to dvije klase koje smo prije obilježili.
Za preslikavanje objekta u XML dokument koristi se objekt `Marshaller`.

```
Marshaller marshaller = context.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
```

On se stvara tako da se na konteksu pozove metoda `createMarshaller`. Ako želimo da nam XML dokument bude formatiran (uvlačenje elemenata i prelasci u novi red) onda je to potrebno podesiti pomoću poziva metode `setProperty` što je prikazano u primjeru.
Nakon toga pomoću objekta Marshaller možemo preslikati objekt u XML:

```
marshaller.marshal(sc, System.out);
marshaller.marshal(dc1, System.out);
marshaller.marshal(dc1, new FileWriter("counter.xml"));
```

Prvi redak preslikava objekt `sc` u XML koji se ispisuje na standardni izlazni tok. Dokument izgleda ovako:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<counter value="1"/>
```

Drugi redak preslikava objekt `dc1` također na standardni izlazni tok:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<diff>
    <cntr1 value="2"/>
    <cntr2 value="1"/>
</diff>
```

Treći redak preslikava objekt `dc1` u datoteku `counter.xml`.

2.2.3 Preslikavanje XML-a u objekt

Preslikavanje XML dokumenta u objekt je vrlo slično preslikavanju objekta u XML. Razlika je u objektu koji se koristi. Ovdje se koristi objekt klase `Unmarshaller`:

```
Unmarshaller unmarshaller = context.createUnmarshaller();

DifferentialCounter dc2 = (DifferentialCounter)
  unmarshaller.unmarshal(new File("counter.xml"));
```

Prvo je potrebno napraviti objekt `unmarshaller` pomoću konteksta (prvi redak), a nakon toga se pozove metoda `unmarshal` koja kao parametar prima ulazni tok ili datoteku (objekt `File`) iz kojeg/e se čita XML dokument. Metoda `unmarshal` je deklarirana da vraća objekt klase `Object`. Programer mora znati kakav objekt je stvarno vraćen pa onda može castanjem pretvoriti objekt u onaj koji je stvarno preslikan.

2.3 DOM

World Wide Web Consortium (W3C) je definirao DOM (Document Object Model) i on je standard kao i XML. DOM nije napravljen specifično za Javu, već je napravljen da predstavlja sadržaj XML-a u objektnom obliku koji je prihvatljiv različitim programskim jezicima i alatima. On je definiran u specifikaciji CORBA koja ima preslikavanje u različite objektno orijentirane programske jezike.
DOM je model koji predstavlja sadržaj XML-a u obliku stabla. Kada se pomoću alata učita dokument u memoriju onda je rezultat DOM stablo u memoriji. Učitanim stablom u memoriji se može manipulirati ili dohvatiti bilo koji element. Nakon manipulacije stablom ono se može ponovno spremiti u datoteku u obliku XML-a.

2.3.1 Stvaranje dokumenta u memoriji

Osnovna sučelje koja se koriste u Javi nalaze se u paketu `org.w3c.dom`. Konkretne implementacije nalaze se u drugim paketima i nije ih moguće direktno koristiti. Za stvaranje DOM stabla potrebno je koristiti uzorak *builder*. Implementacija tog uzorka se nalazi u paketu `javax.xml.parsers`. Prvo je potrebno napraviti tvornicu objekata:

```
DocumentBuilderFactory builderFactory =
  DocumentBuilderFactory.newInstance();
```

Nakon toga je potrebno napraviti `DocumentBuilder`:

```
DocumentBuilder builder = builderFactory.newDocumentBuilder();
```

Tek kada imamo objekt `builder` onda pomoću njega možemo napraviti dokument:

```
Document document = builder.newDocument();
```

Za stvaranje elemenata koristimo metodu `createElement` u dokumentu. Parametar je ime elementa, a rezultat je objekt `Element`:

```
Element root = document.createElement("diff-counter");
```

Nakon stvaranja elementa potrebno ga je dodati u neki čvor u dokumentu. Kako je to korijenski čvor onda ga dodajemo u čvor `document`. To radimo pozivom metode `appendChild`:

```
document.appendChild(root);
```

Nakon toga stvaramo novi element. Za stvaranje atributa u nekom elementu koristi se metoda `setAttribute` koja kao prvi parametar ima ime atributa, a drugi je njegova vrijednost:

```
Element start = document.createElement("start");
start.setAttribute("value", "2");
root.appendChild(start);

Element end = document.createElement("end");
end.setAttribute("value", "1");
root.appendChild(end);
```

2.3.2 Pretvaranje DOM stabla u XML

Nakon što je dokument napravljen u memoriji možemo da pretvoriti u XML i snimiti u datoteku ili poslati preko mreže. Pošto je to transformacija DOM-a u XML onda je prvo potrebno napraviti tvornicu objekata `Transformer`:

```
TransformerFactory transformerFactory =
  TransformerFactory.newInstance();
```

Nakon toga stvaramo novi objekt `Transformer` koji može transformirati DOM:

```
Transformer transformer = transformerFactory.newTransformer();
```

Objekt `Transformer` ima metodu za transformaciju koja prima kao prvi parametar izvor transformacije. Izvor transformacije može biti DOM stablo, ali ga je potrebno omotati objektom `DOMSource`:

```
DOMSource domSource = new DOMSource(root);
```

Drugi parametar je rezultat transformacije. U ovom ćemo koristiti rezultat koji zna proslijediti transformirani oblik u tok podataka (objekt `StreamResult`), a on će to proslijediti u standardni izlazni tok:

```
StreamResult output = new StreamResult(System.out);
```

Metoda koja transformira izvor u rezultat je `transform`:

```
transformer.transform(domSource, output);
```

Ispis na standardni tok je sljedeći:

```
<?xml version="1.0" encoding="UTF-8"?><diff-counter><start
value="2"/><end value="1"/></diff-counter>
```

Kao što se vidi rezultat je XML dokument, ali on nije lijepo formatiran pa ga je čovjeku teško čitati. Za lijepi ispis potrebno je koristiti klasu `LSSerializer`. Sljedeći kod radi lijepi ispis DOM stabla:

```
DOMImplementation domImplementation = document.getImplementation();
DOMImplementationLS domImplementationLS = (DOMImplementationLS)
  domImplementation.getFeature("LS", "3.0");
LSSerializer lsSerializer =
  domImplementationLS.createLSSerializer();
lsSerializer.getDomConfig().setParameter(
  "format-pretty-print", Boolean.TRUE);
LSOutput lsOutput = domImplementationLS.createLSOutput();
lsOutput.setByteStream(System.out);
lsSerializer.write(document, lsOutput);
```

Iz prethodnog koda se vidi da je za lijepi ispis DOM stabla potrebno dosta koda. Ovaj kod radi samo na Javi 1.6.

2.3.3 Stvaranja DOM stabla koji predstavlja objekt

Stvaranje DOM stabla iz objekta `DifferentialCounter` nije automatski, već je potrebno napraviti metodu koja to radi. Sljedeći kod radi stvaranje DOM stabla iz objekta `DifferentialCounter`:

```
private static Document createDocument(DifferentialCounter counter)
    throws ParserConfigurationException {
  DocumentBuilderFactory builderFactory =
    DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = builderFactory.newDocumentBuilder();

  Document document = builder.newDocument();
  Element root = document.createElement("diff-counter");
  document.appendChild(root);

  Element start = document.createElement("start");
  start.setAttribute("value", Integer.toString(counter.getCntr1()));
  root.appendChild(start);

  Element end = document.createElement("end");
  end.setAttribute("value", Integer.toString(counter.getCntr2()) );
  root.appendChild(end);
  return document;
}
```

2.3.4 Učitavanje XML dokumenta i stvaranje DOM stabla

Za učitavanje dokumenta potrebno je prvo napraviti tvornicu objekata `DocumentBuilder`:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

Nakon toga trebamo napraviti novi objekt `DocumentBuilder`:

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

Objekt `DocumentBuilder` ima nekoliko varijanti metode `parse` koja iz parametra stvara DOM stablo. Ako je parametar ulazni tok onda podatke za XML čita iz njega, a ako je parametar `String` onda je vrijednost `String`-a ime datoteke iz koje će se pročitati XML dokument:

```
Document document = db.parse("counter.xml");
```

Rezultat metode `parse` je DOM stablo (objekt `Document`) i sada je potrebno iz tog stabla pročitati podatke i staviti ih u objekt `DifferentialCounter`. To radi sljedeća metoda:

```
private static DifferentialCounter loadObject(Document document) {
  DifferentialCounter counter = new DifferentialCounter();

  Element root = document.getDocumentElement();

  NodeList nodes = root.getChildNodes();
  if (nodes != null && nodes.getLength() > 0) {
    for (int i = 0; i < nodes.getLength(); i++) {
      Node node = nodes.item(i);
      if (node.getNodeType() == Node.TEXT_NODE)
        continue;
      System.out.println("Node: " + node.getNodeName());

      if (node.getNodeName().equals("start")) {
        int startValue = getValue(node);
```

```
        for (int j = 0; j < startValue; j++)
          counter.increment1();
      }

      if (node.getNodeName().equals("end")) {
        int startValue = getValue(node);
        for (int j = 0; j < startValue; j++)
          counter.increment2();
      }
    }
  }
  return counter;
}
```

Čitanje primitivne vrijednosti `int` iz atributa čvora radi se u metodi `getValue`:

```
private static int getValue(Node node) {
  NamedNodeMap nodeMap = node.getAttributes();
  for (int i = 0; i < nodeMap.getLength(); i++) {
    Node nodeAttribute = nodeMap.item(i);
    System.out.println(" Atribut: " + nodeAttribute.getNodeName()
        + " = " + nodeAttribute.getTextContent());
    if (nodeAttribute.getNodeName().equals("value")) {
      return Integer.parseInt(nodeAttribute.getTextContent());
    }
  }
  return 0;
}
```

2.4 JDOM

Iz koda iz prethodnog poglavlja se vidi da je učitavanje i spremanje XML dokumenata dosta složeno. To je zbog toga što je DOM definiran za različite programske jezike. DOM prilagođen Javi je implementiran u knjižnici JDOM koja koristi specifičnosti jezika Java.

2.4.1 Stvaranje DOM stabla u memoriji

Pro je potrebno napraviti dokument:

```
Document doc = new Document();
```

Nakon toga stvaramo element koji pomoću metode `setRootElement` dodajemo kao korijenski element u dokument:

```
Element root = new Element( "diff-counter" );
doc.setRootElement( root );
```

Onda možemo stvoriti i ostale elemente i dodati ih u korijenski element. Za stvaranje atributa koristimo metodu `setAttribute`:

```
Element start = new Element( "start" );
root.addContent( start );
start.setAttribute( "value",  "2" );

Element end = new Element( "end" );
root.addContent( end );
end.setAttribute( "value", "1" );
```

2.4.2 Pretvaranje DOM stabla u XML

Za pretvaranje DOM stabla u XML potrebno je napraviti `XMLOutputer`. On kao parametar može imati objekt `Format` koji određuje kako će biti stvoren XML. U ovom slučaju želi se lijepo ispisati dokument:

```
XMLOutputter xOut = new XMLOutputter( Format.getPrettyFormat() );
```

Nakon stvaranja objekt `XMLOutputter` u njemu se poziva metoda `output` koja pretvara dom stablo koje je poslano kao prvi parametar u XML i ispisuje ga u izlazni tok:

```
xOut.output( doc, System.out );
```

2.4.3 Stvaranja DOM stabla koji predstavlja objekt

Za stvaranje DOM stabla koje predstavlja objekt DifferentialCounter potrebno je također napraviti metodu koja to radi isto kao i kod DOM-a. Sljedeći kod radi pretvaranje objekta DifferentialCounter u DOM stablo u memoriji:

```
Document doc = new Document();

Element root = new Element( "diff-counter" );
doc.setRootElement( root );

Element start = new Element( "start" );
root.addContent( start );
start.setAttribute( "value", Integer.toString(counter.getCntr1()));

Element end = new Element( "end" );
root.addContent( end );
end.setAttribute( "value", Integer.toString(counter.getCntr2()) );

XMLOutputter xOut = new XMLOutputter( Format.getPrettyFormat() );
xOut.output( doc, System.out );
```

2.4.4 Učitavanje XML dokumenta i stvaranje DOM stabla

Za učitavanje XML dokumenta i stvaranje DOM stabla potrebno je napraviti objekt `DifferntialCounter` i onda u njemu postaviti podatke iz XML dokumenta:

```
DifferentialCounter counter = new DifferentialCounter();
```

Za učitavanje XML dokumenta i stvaranje DOM stabla prvo treba napraviti objekt koji stvara DOM stablo. U ovom slučaju je to SAXBuilder:

```
SAXBuilder sxb = new SAXBuilder();
```

Nakon toga pozivom metode `build` stvara se DOM stablo iz datoteke čije ime je poslano kao parametar metode `build`:

```
Document doc = sxb.build("counter.xml");
```

Iz stvorenog stabla koje je predstavljeno objektom Document treba metodom getRootElement dohvatiti korijenski element:

```
Element root = doc.getRootElement();
```

Metodom getChildren dohvaća se lista elemenata unutar korijenskog elementa:

```
List<Element> elements = root.getChildren();
```

U `for` petlji prolazi se kroz sve elemente. Ime elementa može se dohvatiti pozivom metode `getName`, a vrijednost elementa metodom `getValue` koja vraća tekst unutar elementa:

```
for (Element el : elements) {
  System.out.println("Element: " + el.getName());

  if(el.getName().equals("start")) {
    int startValue = getValue(el);
    for(int i = 0; i < startValue; i++)
      counter.increment1();
  }

  if(el.getName().equals("end")) {
    int startValue = getValue(el);
    for(int i = 0; i < startValue; i++)
      counter.increment2();
  }
}
```

I ovdje je napravljena posebna metoda getValue koja vraća vrijednost atributa:

```
private static int getValue(Element el) throws
  DataConversionException
{
  List<Attribute> attributes = el.getAttributes();
  for (Attribute att : attributes) {
    System.out.println(" Atribut: " + att.getName() + " = "
        + att.getValue());
    if(att.getName().equals("value")) {
      return att.getIntValue();
    }
  }

  return 0;
}
```

U ovom kodu je korištena metoda `getIntValue` koja vraća vrijednost atributa u obliku primitivne vrijednosti `int`. Umjesto čitave metode mogli smo koristiti i sljedeći kod:

```
el.getAttribute("value").getIntValue();
```

Ovdje bi u slučaju da element nema atributa `value` došlo do iznimke pa treba biti pažljiv prilikom takvog koda.

2.5 SAX

SAX (Simple API for XML) je programsko sučelje koje omogućuje učitavanje XML dokumenata. Osnovna ideja SAX-a je da se prilikom učitavanja XML dokumenta generiraju događaji i prosljeđuju se objektu koji je zadužen za obradu tih događaja. Na ovakav način moguće je iz velikih dokumenata izvući samo nužne podatke za rad aplikacije koja treba samo neke podatke iz XML dokumenta. Pomoću SAX-a nije potrebno čitav dokument pretvoriti u DOM stablo u memoriji. Na taj način se povećavaju performanse aplikacije koja se programira.
Prilikom učitavanja dokumenta potrebno je napraviti tvornicu `SAXParser`-a:

```
SAXParserFactory parserFactory = SAXParserFactory.newInstance();
```

Nakon toga napravimo objekt `SAXParser`:

```
SAXParser parser = parserFactory.newSAXParser();
```

Metodom `parse` pokreće se učitavanje XML dokumenta. Prvi parametar je ili ulazni tok podataka ili ime datoteke u kojoj je XML. Drugi parametar je objekt koji nasljeđuje `DefaultHandler`:

```
parser.parse("counter.xml", this);
```

Klasa `DefaultHandler` ima prazne metode koje se pozivaju kada `SAXParser` učita pojedini dio XML dokumenta. Najvažnije metode u obradi XML-a koje se pozivaju u `DefaultHandler`-u su:

- `startDocument` – poziva se kada se počinje učitavati dokument,
- `endDocument` – poziva se na kraju učitavanja dokumenta,
- `startElement` – poziva se kada je učitana oznaka otvaranja elementa,
- `endElement` – poziva se kada je učitana zatvarajuća oznaka elementa i
- `characters` – poziva se kada je učitan tekstualno element.

Primjer jednostavne klase koja ispisuje najvažnije elemente XML-a (elementi, atributi i tekst) je u sljedećem kodu:

```
public class PrintSAXEvents extends DefaultHandler {

  public static void main(String[] args) throws Exception {
    new PrintSAXEvents().parse();
  }

  private void parse() throws Exception {
    SAXParserFactory parserFactory = SAXParserFactory.newInstance();

    SAXParser parser = parserFactory.newSAXParser();

    parser.parse("counter.xml", this);
  }

  @Override
  public void startDocument() throws SAXException {
    System.out.println("početak dokumenta");
  }

  @Override
  public void endDocument() throws SAXException {
    System.out.println("kraj dokumenta");
  }

  @Override
  public void startElement(String uri, String localName,
    String qName, Attributes attributes) throws SAXException
  {
    System.out.println("početak elementa: " + qName);
    for (int i = 0; i < attributes.getLength(); i++) {
      System.out.println("atribut: ime=" + attributes.getQName(i) +
        " vrijednost=" + attributes.getValue(i));
    }
  }

  @Override
  public void endElement(String uri, String localName, String qName)
      throws SAXException {
    System.out.println("kraj elementa: " + qName);
  }

  @Override
  public void characters(char[] ch, int start, int length)
```

```
      throws SAXException {

    System.out.println("tekst element: /" +
      new String(ch,start,length) + "/");
  }
}
```

Učitavajući sljedeći XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff-counter>
  <start value="2" />
  <end value="1" />
</diff-counter>
```

kada pokrenemo program dobijemo sljedeći ispis:

```
početak dokumenta
početak elementa: diff-counter
tekst element: /
  /
početak elementa: start
atribut: ime=value vrijednost=2
kraj elementa: start
tekst element: /
  /
početak elementa: end
atribut: ime=value vrijednost=1
kraj elementa: end
tekst element: /
/
kraj elementa: diff-counter
kraj dokumenta
```

Slijedi primjer klase koja učitava `DifferentialCounter` pomoću SAX-a:

```
public class Load extends DefaultHandler {

  public static void main(String[] args) throws Exception {
    new Load().parse();
  }

  private String tempValue;
  private int intValue;
  private DifferentialCounter counter;

  private void parse() throws Exception {
    SAXParserFactory parserFactory = SAXParserFactory.newInstance();

    SAXParser parser = parserFactory.newSAXParser();

    parser.parse("counter.xml", this);

    System.out.println(counter.toString());
  }

  public void startElement(String uri, String localName,
    String qName, Attributes attributes) throws SAXException
  {
    if(qName.equalsIgnoreCase("diff-counter")) {
      counter = new DifferentialCounter();
    } else if (qName.equalsIgnoreCase("start")) {
      tempValue = attributes.getValue("value");
      if(tempValue != null)
```

```
        intValue = Integer.parseInt(tempValue);
      else
        intValue = 0;

      for (int j = 0; j < intValue; j++)
        counter.increment1();
    } else if (qName.equalsIgnoreCase("end")) {
      tempValue = attributes.getValue("value");
      if(tempValue != null)
        intValue = Integer.parseInt(tempValue);
      else
        intValue = 0;

      for (int j = 0; j < intValue; j++)
        counter.increment2();
    }
  }

  public void characters(char[] ch, int start, int length)
    throws SAXException
  {
    tempValue = new String(ch,start,length);
  }
}
```

3 STRUKTURIRANJE XML DOKUMENATA

Kažemo da je XML dokument dobro definiran ako poštuje svoja sintaktička pravila. No, ta pravila ne govore ništa niti o strukturi niti o semantici dokumenta. Uvođenjem strukture u dokument povećava se mogućnost provjere njegove ispravnosti. Kažemo da je dokument **valjan** (validan) ako je dobro i sintaktički i strukturno definiran. Dva su načina definiranja strukture XML dokumenata; stariji i stroži način je DTDs (*Document Type Definitions*) i XML Schema s većom mogućnosti definiranja podatkovnih vrsta.

3.1 Prostor imena elemenata

Kad netko želi stvoriti svoj jezik temeljen na XML-u, može to učiniti na dva načina: ili sâm kreirati sve oznake ili kreirati samo dio oznaka, a ostale uzeti iz nečijeg XML dokumenta. Korištenjem ovog drugog načina mogao bi se pojaviti problem istoimenih elemenata različitog konteksta ili značenja. Budući da računalni programi, za razliku od ljudi, ne razlikuju kontekste, potreban je mehanizam koji će omogućiti razlikovanje spomenutih elemenata i time ispravan rad programa. Takav mehanizam sastoji se u dodjeljivanju jedinstvenih imena strukturama dokumenta, a naziva se XML-ovim imenicima (*XML Namespaces*). Imenici pružaju uslugu pridjeljivanja konteksta elementima, što elementima omogućuje da budu jedinstveni.
Imenik općenito predstavlja neku grupu elemenata i atributa čijim imenima dodajemo imenički prefiks (prefiks imenika) po kojem parser prepoznaje o kojem se imeniku radi odnosno koja pravila oblikovanja dokumenta vrijede za tu grupu elemenata i atributa. Zamislimo da je hrvatski jezik podijeljen u imenike prema različitim temama. Uzmimo, recimo, dvije teme za dva imenika. Npr. imenik alata i imenik ljudskog tijela. Neka su u imeniku alata stvari poput lopate, čekića, grablji, itd., a u imeniku ljudskog tijela oči, nos, usta, ruke, itd. Oba ova imenika neka sadrže riječ kosa, ali u različitim značenjima – kosa kao alat za košenje trave u imeniku alata i kosa na našim glavama u imeniku ljudskog tijela (sigurno i sami znate dosta primjera homonima u hrvatskom jeziku). Ovo s kosama je primjer kolizije imena u XML-u gdje dva objekta u različitim imenicima imaju potpuno ista imena. Da bismo oba ova značenja mogli koristiti u istom dokumentu, dodajemo imenički prefiks svakom elementu i atributu kojeg koristimo na sljedeći način:

```
imenički prefiks : lokalno ime elementa ili atributa
```

Za naše imenike imamo elemente, npr. `<alati:kosa>` i `<ljudsko_tijelo:kosa>`.
Imenici nisu samo važni zbog prevencije preklapanja imena nego i zbog toga što pomažu XML procesoru složiti različite grupe elemenata za različitu obradu (npr. preglednicima weba naznačuju kako obraditi određenu grupu podataka).
Imenik se mora deklarirati u dokumentu prije negoli se koristi. Deklaracija je u obliku atributa elementa koji počinje riječju „xmlns“, a sintaksa je sljedeća:

```
xmlns : imenički prefiks = "URL"
```

Evo primjera:

```
<zavod xmlns:tel="http://www.tel.fer.hr/">
```

Kod izbora prefiksa treba paziti da se uzme neka od XML-ovih rezerviranih riječi (xml, xsl, itd.). Vrijednost xmlns:atribut-a je URL koji obično pripada organizaciji koja održava imenik. XML procesor inače ne mora ništa činiti s tim URL-om jer na toj lokaciji čak niti ne mora nužno biti dokument. URL je više formalnost kojom se pruža dodatna informacija o imeniku poput njegova vlasnika ili verzije koja se koristi. Bilo koji element u dokumentu može sadržavati deklaraciju imenika, no to najčešće sadrži korijenski element. Imenik se može ograničiti na samo jedan dio dokumenta

deklariranjem imenika u nekom dubljem elementu. U tom se slučaju imenik odnosi samo na taj element i njegove podelemente.

3.2 DTD

Elementi DTD-a mogu se definirati u posebnoj, tzv. vanjskoj datoteci ili unutar samog XML dokumenta (tzv. unutarnji DTD). Načelno je bolje koristiti vanjsku datoteku jer se ona može onda primijeniti na više dokumenata.
Uzmimo sljedeći XML-ovski zapis:

```
<zavod>
  <ime>Zavod za telekomunikacije</ime>
  <telefon>0038516129810</telefon>
</zavod>
```

DTD za gornji element vrste „zavod“ je sljedeći:

```
<!ELEMENT zavod (ime, telefon)>
<!ELEMENT ime (#PCDATA)>
<!ELEMENT telefon (#PCDATA)>
```

Oznaka `<!ELEMENT ... >` označava element koji se može nalaziti u dokumentu. Dalje, „zavod (ime, telefon)“ znači da element „zavod“ sadrži podelemente „ime“ i „telefon“ u tom poretku. Elementi „ime“ i „telefon“ mogu imati bilo kakav sadržaj (oznaka „`#PCDATA`“).
Kad bismo htjeli izraziti da element „zavod“ može sadržavati ili element „ime“ ili element „telefon“, napisali bismo sljedeće:

```
<!ELEMENT zavod (ime | telefon)>
```

Da želimo naglasiti da element „zavod“ koji sadrži elemente „ime“ i „telefon“ u bilo kojem redoslijedu ova dva elementa, pišemo sljedeće:

```
<!ELEMENT zavod( (ime, telefon) | (telefon, ime))>
```

Nedostatak ovog pristupa je gomilanje izraza ako imamo puno elemenata s proizvoljnim redoslijedom.
Pojavi li se `EMPTY` kao element, to znači da je element prazan. `EMPTY` se uvijek odnosi na krajnje elemente (lišće). `ANY` pak označava da element može sadržavati bilo koji drugi element deklariran DTD-om, što se rijetko koristi u praksi jer ne nosi baš neku korisnu informaciju o strukturi. `ANY` se zbog toga češće koristi tijekom razvoja DTD-a prije negoli su definirana sva pravila.
Uzmimo sljedeći ispis:

```
<predmet ime="ILJ" godina="3" smjer="Telekomunikacije">
  <student ime="Ana" prezime="Anić" status=redovni"/>
  <student ime="Marko" prezime="Marić" status=redovni"/>
</predmet>
```

DTD za ove elemente glasi:

```
<!ELEMENT predmet (student+)>
<!ATTRLIST predmet
   ime          ID          #REQUIRED
   godina       CDATA       #REQUIRED
   smjer        CDATA       #REQUIRED>
<!ELEMENT student EMPTY>
<!ATTRLIST student
   ime          CDATA       #REQUIRED
   prezime      CDATA       #REQUIRED
   status       CDATA       #REQUIRED>
```

Ovdje se pojavio operator brojnosti +. U DTD pojavljuju se sljedeći operatori brojnosti:

- ? – označava 0 ili 1 pojavljivanje,
- * – označava 0 ili više pojavljivanja i
- + – označava 1 ili više pojavljivanja.

Ako element nema oznaku brojnosti, znači da se pojavljuje točno jednom.
!ATTRLIST služi za definiranje atributa. Prvi se napiše ime elementa kojemu pripada lista atributa, pa slijede ime, vrsta i vrijednost atributa. Najvažnije ugrađene vrste atributa su:

- CDATA - znakovni niz (String)
- ID - jedinstveno ime u XML dokumentu
- IDREF - referenca na drugi element s ID atributom koji nosi istu vrijednost kao IDREF atribut
- IDREFS - niz IDREF-ova
- (v1|...|vn) - nabrajanje (enumeracija)
- ENTITY - ime vanjskog entiteta
- ENTITIES - lista entiteta (ENTITY) razdijeljenih razmacima
- NMTOKEN - riječ bez razmaka (vrijednosti koje su ispravna XML imena:
  - prvi znak mora biti ili slovo ili podcrta, dok se ostali znakovi
  - biraju iz skupa slova, brojeva i podcrta; pri tome imena ne smiju sadržavati razmake)
- NMTOKENS - lista riječi (NMTOKEN) razdijeljenih razmacima.

Ovdje valja uočiti da ne postoji eksplicitna specifikacija broja pa njih valja interpretirati kao znakovne nizove.
Vrste vrijednosti atributa su sljedeće:

- #REQUIRED - atribut se mora pojaviti kadgod je njegova vrsta elementa u XML dokumentu.
- #IMPLIED - pojavljivanje ovog atributa je opcionalno
- #FIXED - svaki element mora sadržavati ovaj atribut, a on uvijek ima onu vrijednost koja je navedena u DTD dokumentu nakon
- ključne riječi #FIXED
- „value“ - specificira podrazumijevanu vrijednost atributa. Ako se u XML dokumentu pojavi specifična vrijednost, ona „nadjačava“ ovu podrazumijevanu vrijednost.

Sljedeći primjer prikazuje korištenje referenci:

```
<family>
   <person id="nodeAAA" mother="nodeMather" father="nodeFather">
     <name>Node AAA</name>
   </person>
   <person id="nodeBBB" mother="nodeMather">
     <name>Node BBB</name>
   </person>
   <person id="nodeMather" children="nodeAAA nodeBBB">
     <name>Node Mather</name>
   </person>
   <person id="nodeFather" children="nodeAAA nodeBBB">
     <name>Node Father</name>
   </person>
```

```
</family>
```

Ovom primjeru odgovara sljedeća DTD struktura;

```
<!ELEMENT family  (person*)>
<!ELEMENT person  (name)>
<!ELEMENT name    (#PCDATA)>
<!ATTRLIST person

id                ID     #REQUIRED
mother            IDREF  #IMPLIED
father            IDREF  #IMPLIED
children          IDREFS #IMPLIED
```

Uzmimo sljedeći primjer;

```
<?xml version="1.0" encoding="UTF-16">
<!DOCTYPE email SYSTEM „email.dtd">
<email>
   <head>
      <from name="Perica" address="pero@fer.hr"/>
      <to name="Berislav" address=bero@fer.hr/>
      <subject>hej</subject>
   </head>
   <body>Bero... gdje je sir?!?
   </body>
</email>
```

Za gornji primjer koristimo sljedeću DTD strukturu:

```
<!ELEMENT email   (head, body)>
<!ELEMENT head    (from, to+, cc*, subject)>
<!ELEMENT from    EMPTY>
<!ATTRLIST from
   name     CDATA       #IMPLIED
   address  CDATA       #REQUIRED>
<! ATTRLIST to
   name     CDATA       #IMPLIED
   address  CDATA       #REQUIRED>
<! ATTRLIST cc
   name     CDATA       #IMPLIED
   address  CDATA       #REQUIRED>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body    (text, attachment*)>
<!ELEMENT text    (#PCDATA)>
<!ELEMENT attachment EMPTY>
<! ATTRLIST attachment
   encoding (mime|binhex)      „mime"
   file           CDATA #REQUIRED>
```

Ako je u XML dokumentu oznaka `<?xml version="1.0" encoding="UTF-8" standalone="no"?>`, ona označava ima li dokument deklaracije koje se nalaze izvan dokumenta.
Oznaka `<!DOCTYPE email SYSTEM „email.dtd">` u gornjem XML dokumentu označava DTD deklaraciju koja se sastoji od imena vršnog elementa (ovdje „email") te DTD-a (ovdje „email.dtd") koji je ovdje zadan kao vanjski dokument.
Element „head" sadrži „from" element, najmanje jedan „to" element, nula ili više „cc" elemenata te „subject" element; sve zadano tim redoslijedom. U „from", „to" i „cc" elementima nije potreban „name" atribut dok je „address" atribut obvezan. „body" element sadrži „text" element, a mogu ga slijediti niz „attachment" elemenata. Atribut

„encoding“ elementa „attachment“ mora imati vrijednost ili „mime“ ili „binhex“, pri čemu je „mime“ podrazumijevana vrijednost.
Zanimljivo je da se DTD može interpretirati kao prošireni Backus-Naurov zapis (ili forma - EBNF). Npr. deklaracija `<!ELEMENT email (head, body)>` je ekvivalentna pravilu:

```
email ::= head body
```

što označava da se email sastoji od `head`-a iza kojeg odmah slijedi `body`.
Također, DTD podržava rekurziju jer je moguća, primjerice, sljedeća defincija binarnog stabla:

```
<!ELEMENT bintree (bintree root bintree) | emptytree>
```

Binarno stablo je prazno stablo ili stablo sa lijevim podstablom, korijenom i desnim podstablom.

3.3 XML Schema

Za razliku od DTD-a XML Schema omogućava znatno bogatiji izričaj u strukturiranju XML dokumenata, a sintaksa je u potpunosti temeljena na XML-u. Znatno je poboljšana čitljivost dokumenta te mogućnost višestrukog korištenja. XML Schema omogućava stvaranje novih vrsta proširivanjem i ograničavanjem postojećih shema. Dok je DTD ograničen na znakovne nizove kao jedinu vrstu podataka, XML Schema ima znatno širi raspon.
XML schema je element sa sljedećom otvarajućom oznakom:

```
<xsd:schema
  xmlns:xsd=http://www.w3.org/2000/10/XMLSchema
  version="1.0">
```

Ovaj element koristi shemu XML Scheme sa W3C webske stranice. Naše sheme ćemo temeljiti uvijek na ovoj osnovnoj. Prefiks „xsd“ označava imenik sheme. Ako ga preskočimo u „xmlns“ atributu, onda koristimo elemente iz podrazumijevanog imenika:

```
<schema xmlns=http://www.w3.org/2000/10/XMLSchema version="1.0">
```

Najvažniji sadržaj sheme su definicije elemenata i vrste atributa koji se definiraju podatkovim vrstama. Sintaksa elemenata je sljedeća:

```
<element name=". . ."/>
```

te slijedi opcinalan niz atributa:

- vrsta: `type=". . ."`
- brojevna ograničenja:
  - `minOccurs="x"`, gdje je `x` nula ili prirodni broj
  - `maxOccurs="x"`, gdje je `x` nula, prirodni broj ili beskonačno.

Očito je da su `minOccurs` i `maxOccurs` generalizacije operatora brojnosti `?`, `*` i `+` iz DTD-a. Kad brojnosti nisu ekplicitno navedene, `minOccurs` i `maxOccurs` imaju podrazumijevanu vrijednost 1. Evo nekoliko primjera:

```
<element name="email"/>
<element name="head" minOccurs="1" maxOccurs="1"/>
<element name="to" minOccurs="1"/>
```

Sintaksa atributa je sljedeća:

```
<attribute name=". . ."/>
```

Mogu imati nekoliko opcionalnih atributa:

- vrsta: `type=". . ."`
- pojavnost (odgovara `#OPTIONAL` i `#IMPLIED` vrijednostim u DTD-u): `use="x"`, gdje je `x optional` ili `required`.
- podrazumijevana vrijednost (odgovara `#FIXED` i podrazumijevanim vrijednostima u DTD-u): `use="x" value=". . ."`, gdje je `x` podrazumijevana vrijednot ili `fixed`.

Evo nekoliko primjera:

```
<attribute name="id" type="ID" use="required"/>
<element name="speaks" type="Language" use="default" value="en"/>
```

XML Schema ima bogati skup podatkovnih vrijednosti.

- numeričke vrste: `Integer`, `Short`, `Byte`, `Long`, `Decimal`, `Float`, itd.
- znakovne vrste: `string`, `ID`, `IDREF`, `CDATA`, `Language`, itd.
- datum i vremenske vrste: `time`, `Date`, `Month`, `Year`, itd.

A tu su i korisnički definirane podatkovne vrste. Razlikujemo osnovne podatkovne vrste (engl. *simple data types*) koje ne mogu korstiti elemente niti atribute i složene podatkovne vrste (engl. complex data types) koje mogu koristiti elemente i atribute. Složene vrste se definiraju iz postojećih podatkovnih vrsta korištenjem eventualno nekog atributa i nekod od navedenih konstrukata:

- `sequence` - slijed postojećih podatkovnih elemenata gdje je bitan redoslijed,
- `all` – skup elemenata čiji redoslijed nije bitan i
- `choice` – skup elemenata od kojih se izabere jedan.

Evo primjera:

```
<complexType name="lecturerType">
<sequence>
   <element name="firstname" type="string" minOccurs="0"
      maxOccurs="unbounded"/>
   <element name="lastname" type="string"/>
</sequence>
<attribute name="title" type="string" use="optional"/>
</complexType>
```

Element u XML dokumentu koji će biti vrste „lecturerType“ mora imati atribut „title“, mora sadržavati nula ili više „firstname“ elemenata i točno jedan „lastname“ element. Postojeće podatkovne vrste mogu se proširivati novim elementima ili atributima. Proširit ćemo gornji primjer:

```
<complexType name="extendedLecturerType">
 <extension base="lecturerType">
   <sequence>
      <element name="email" type="string" minOccurs="0"
            maxOccurs="1"/>
   </sequence>
   <attribute name="rank" type="string" use="required"/>
 </extension>
</complexType>
```

U ovom primjeru `lecturerType` je proširen „email“ elementom i „rank“ atributom. Ovako izgleda rezultirajuća podatkovna vrsta:

```
<complexType name="extendedLecturerType">
 <sequence>
   <element name="firstname" type="string" minOccurs="0"
```

```
 maxOccurs="unbounded"/>
   <element name="lastname" type="string"/>
   <element name="email" type="string" minOccurs="0" maxOccurs="1"/>
 </sequence>
<attribute name="title" type="string" use="optional"/>
<attribute name="rank" type="string" use="required"/>
</complexType>
```

Postoji hijerarhijski odnos između originalne i proširene vrste tako da su instance proširene vrste također instance originalne vrste (mogu sadržavati dodatne informacije, ali to ne smeta).
Osim proširivanja postojeće podatkovne vrste mogu se i ograničiti zadavanjem uvjeta na neke vrijednosti. Primjerice, mogu se dodati atributi „type“ i „use“ ili istaknuti vrijednosti `minOccurs` i `maxOccurs`. Ograničenja nisu nipošto suprotan proces od proširivanja jer se ograničenjima ne brišu niti elementi niti atributi. I dalje vrijedi hijerarhijski odnos da su instance ograničenih vrsta također instance originalnih vrsta. Ograničene vrste zadovoljavaju najmanje ograničenja originalne vrste, ali i neka dodatna. Ograničimo naš prethodni primjer:

```
<complexType name="restrictedLecturerType">
   <restriction base="lecturerType">
     <sequence>
      <element name="firstname" type="string" minOccurs="1"
            maxOccurs="2"/>
     </sequence>
   <attribute name="title" type="string" use="required"/>
</restriction>
</complexType>
```

Istaknuta su dodana ograničenja.
I osnovne podatkovne vrste također mogu biti izvedene iz postojećih podatkovnih vrsta. Možemo, npr. definirati vrstu `dayOfMonth` koja prihvaća vrijednosti od 1 do 31 na sljedeći način:

```
<simpleType name="dayOfMonth">
   <restriction base="integer">
      <minInclusive value="1"/>
      <maxInclusive value="31"/>
   </restriction>
</simpleType>
```

Moguće je definirati podatkovnu vrstu nabrajanjem svih mogućih vrijednosti. Npr. `dayOfWeek` je zadana kako slijedi:

```
<simpleType name="dayOfWeek">
   <restriction base="string">
      <enumeration value="Mon"/>
      <enumeration value="Tue"/>
      <enumeration value="Wed"/>
      <enumeration value="Thu"/>
      <enumeration value="Fri"/>
      <enumeration value="Sat"/>
      <enumeration value="Sun"/>
   </restriction>
</simpleType>
```

Sljedeći primjer je XML schema za `email` pa je možemo usporediti sa DTD strukturom koju smo već objasnili.

```
<element name="email" type="emailType"/>
<complexType name="emailType">
```

```
   <sequence>
      <element name="head" type="headType"/>
      <element name="body" type="bodyType"/>
   </sequence>
</complexType>
<complexType name="headType">
    <sequence>
      <element name="from" type="nameAddress"/>
      <element name="to" type="nameAddress" minOccurs="1"
            maxOccurs="unbounded"/>
      <element name="cc" type="nameAddress" minOccurs="0"
      maxOccurs="unbounded"/>
      <element name="subject" type="string"/>
   </sequence>
</complexType>
<complexType name="nameAddress">
   <attribute name="name" type="string" use="optional"/>
   <attribute name="address" type="string" use="required"/>
</complexType>
<complexType name="bodyType">
 <sequence>
      <element name="text" type="string"/>
      <element name="attachment" minOccurs="0"
            maxOccurs="unbounded">
 <complexType>
   <attribute name="encoding" use="default" value="mime">
   <simpleType>
      <restriction base="string">
         <enumeration value="mime"/>
         <enumeration value="binhex"/>
      </restriction>
    </simpleType>
   </attribute>
  <attribute name="file" type="string" use="required"/>
 </complexType>
</element>
</sequence>
</complexType>
```

Primijetite da su neke podatkovne vrste definirane posebno sa zadanim imenima, a neke unutar drugih vrsta i anonimno (vrste za „attachment“ element i „encoding“ atribut). Općenito, ako se vrsta koristi samo jednom, onda se može definirati anonimno za lokalnu uporabu, ali ovakav pristup brzo zakaže ako se umnoži gniježđenje.

4 ODABIR ELEMENATA POMOĆU XPath-a

Da bismo iz baza podataka mogli odabirati podatke i manipulirati njima, razvijeni su jezici za pretraživanje i upite (poput SQL-a). Slično nam treba i za XML-ove dokumente pa su i ovdje razvijeni brojni jezici poput XQL-a, XML-QL-a i XQueryja.
XPath je jezik za identifikaciju određenih dijelova XML dokumenata, ali nije XML. Omogućava pisanje izraza koji upućuju na, recimo, prvi element „osoba“ u određenom dokumentu, ili sedmog potomka trećeg elementa „osoba“, ili atribut „ID“ prvog elementa „osoba“ koji sadrži znakovni niz „Pero Perić“, ili sve naredbe za obradu „xml:stylesheet“ u prologu dokumenta, itd. XPath upućuje na čvor pomoću položaja, relativnog položaja, vrste, sadržaja i još nekih kriterija. XSLT pomoću XPath-ovih izraza pronalazi i bira određene elemente ulaznog dokumenta i kopira ih u izlazni dokument ili ih dalje obrađuje. XPointer pomoću XPath-ovih izraza identificira određenu točku u XML dokumentu ili njegov određeni dio na koji upućuje neka XLink veza. XML Schema pomoću XPath-ovih izraza definira ograničenja jedinstvenosti i identiteta.
Podsjetimo da je XML-ov dokument zapravo stablo koje se sastoji od čvorova. Neki čvorovi sadrže jedan ili više drugih čvorova. Postoji točno jedan korijenski čvor koji sadrži sve druge čvorove. XPath je jezik za biranje čvorova i skupova čvorova u tom stablu, a razlikuje nekoliko vrsta čvorova: korijenski čvor, čvorove elemenata, čvorove teksta, čvorove atributa, čvorove komentara, čvorove naredbi za obradu i čvorove imenika. Čvor-korijen stabla sadrži cijeli dokument, uključujući korijenski element i sve komentare i naredbe za obradu smještene prije početne oznake korijenskog elementa ili iza njegove završne oznake.
Koristit ćemo sljedeći primjer:

```
<?xml version="1.0"?>
<?xml-stylesheet type="application/xml" href="people.xsl"?>
<!DOCTYPE osobe [
 <!ATTLIST homepage xlink:type CDATA #FIXED "simple"
      xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
 <!ATTLIST osoba id ID #IMPLIED>
]>
<osobe>

  <osoba rođen="1912" umro="1954" id="p342">
    <ime_i_prezime>
      <ime>Alan</ime>
      <prezime>Turing</prezime>
    </ ime_i_prezime >
    <!—Je li u Turingovo vrijeme postojao izraz znanstvenik
      računarstva?-->
    <zanimanje>znanstvenik računarstva</zanimanje>
    <zanimanje>matematičar</zanimanje>
    <zanimanje>kriptograf</zanimanje>
    <homepage xlink:href="http://www.turing.org.uk/"/>
  </osoba>

  <osoba rođen="1918" umro="1988" id="p4567">
    <ime_i_prezime>
      <ime>Richard</ime>
      <srednje_ime>&#x50;</srednje_ime>
      <prezime>Feynman</prezime>
    </ime_i_prezime>
    <zanimanje>fizičar</zanimanje>
    <hobi>sviranje bongo bubnjeva</hobi>
  </osoba>
```

```
</osobe>
```

U ovom primjeru čvor-korijen sadrži naredbu za obradu „xml-stylesheet“ i korijenski element „osobe“. XPath-ov model podataka ne obuhvaća sve u dokumentu. Deklaracija XML-a, deklaracija vrste dokumenta (DOCTYPE) i razni dijelovi DTD-a ne mogu se adresirati XPath-om, iako XPath registrira atribute kojima DTD daje podrazumijevane vrijednosti. Element „homepage“ ima atribut „xlink:type“ kojemu je vrijednost dodijelio DTD.
Atributi „xmlns“ i „xmlns:prefiks“ ne smatraju se atributskim čvorovima. Čvorovi imenika pridružuju se svakom čvoru elementa za koji vrijedi neka deklaracija, a ne samo onom elementu za koji je deklariran imenik.

4.1 Lokacijski putovi

Glavni koncept XPath-a je izraz za lokacijski put (engl. *location path*) koji specificira skup čvorova iz stabla XML dokumenta. Taj skup može biti prazan ili sadržavati jedan ili više čvorova. Lokacijski put sastoji se od uzastopnih koraka lokacije (engl. *location steps*). Svaki korak lokacije izračunava se u odnosu na određeni čvor u dokumentu koji nazivamo kontekstnim čvorom (engl. *context node*).
Najjednostavniji lokacijski put je onaj koji bira čvor-korijen dokumenta, a označava ga kosa crta /. Kosa crta je apsolutni lokacijski put jer uvijek označava isto, tj. uvijek označava čvor-korijen dokumenta. Apsolutni putovi počinju od korijena stabla, a označavaju se tako da počinju oznakom /. Ponavljamo, korijen dokumenta se u hijerarhiji XML dokumenta nalazi jednu razinu iznad korijenskog elementa.
Drugi najjednostavniji put je ime jednog elementa. Taj put bira sve elemente čije ime zadamo, a koji su potomci kontekstnog čvora. Npr. put `zanimanje` upućuje na sve potomke kontekstnog čvora nazvane `zanimanje`. Koji su to elementi ovisi o kontekstnom čvoru pa ovo predstavlja, tzv. relativni lokacijski put. Npr. ako je kontekstni čvor Alen Turingov element „osoba“, onda lokacijski put `zanimanje` upućuje na ova tri tako nazvana potomka tog elementa:

```
<zanimanje>znanstvenik računarstva</zanimanje>
<zanimanje>matematičar</zanimanje>
<zanimanje>kriptograf</zanimanje>
```

No, ako je kontekstni čvor Richard Feymanov element „osoba“ iz našeg primjera, onda lokacijski put `zanimanje` upućuje na jedinog njegovog tako nazvanog potomka:

```
<zanimanje>fizičar</zanimanje>
```

Ako je kontekstni čvor Feynmanov ili Turingov elelment potomak „ime_i_prezime“ koji je potomak elementa „osoba“, onda taj put ne označava baš ništa jer nijedan od elemenata „ime_i_prezime“ nema potomaka nazvanih „zanimanje“.
Da bismo izabrali određeni atribut nekog elementa, pišemo znak `@` i iza njega ime željenog atributa. Npr. XPath-ov izraz `@rođen` odabire atribut kontekstnog čvora nazvanog „rođen“.
Mada korijenski čvor i čvorovi elemenata i atributa predstavljaju većinu posla kojeg treba obaviti s XML dokumentima, valja reći ponešto i o ostalim vrstama čvorova. Čvorovi imenika rijetko se eksplicitno obrađuju, a čvorovi teksta, naredbi za obradu i komentara imaju specijalne uzorke kojima se ispituju. To su: `comment()`, `text()` i `processing-instruction()`. S obzirom da komentari i čvorovi teksta nemaju imena, koraci lokacija `comment()` i `text()` odgovaraju svim čvorovima komentara i teksta koji su potomci tekućeg kontekstnog čvora. Svaki komentar čini zaseban čvor. Svaki čvor teksta sadrži najveći mogući kontinuirani dio teksta koji ne sadrži nijednu oznaku. Ako korak lokacije `procesing-instrucion()` nema argumenata, on bira sve naredbe za obradu koji su potomci tekućeg kontekstnog čvora. Ako argument postoji, biraju se samo naredbe koje su potomci tekućeg kontekstnog čvora i cilj im je zadan

kao argument unutar zagrada. Npr. XPath-ov izraz `procesing-instrucion('xml-stylesheet')` bira sve naredbe za obradu koje su potomci kontekstnog čvora i čiji je cilj `xml-stylesheet`.
Postoje tri znaka koji je upotrebljavaju kao skraćenice: `*`, `node()` i `@*`. Zvjezdica (`*`) odgovara svim čvorovima elemenata bez obzira na njihova imena. Znak `node()` odgovara ne samo svim vrstama elemenata nego i korijenskom čvoru te čvorovima teksta, naredbi za obradu, imenika, atributa i komentara. `@*` odgovara svim čvorovima atributa. Skraćenicama možemo dodati prefiks imenika pa će se podudaranje ispitivati samo unutar tog imenika.
Često je potrebno ispitivati podudaranje više vrsta elemenata ili atributa (ali ne svih njih). Npr. trebamo elemente „zanimanje“ i „hobi“, ali ne i elemente „ime_i_prezime“, „osoba“ ili „osobe“. Lokacijske putove i njihove korake možemo kombinirati pomoću vertikalne crte (`|`) koja označava da je dovoljno utvrditi podudaranje bilo kojeg od tako povezanih elemenata. Npr. `zanimanje|hobi` odgovara elementima „zanimanje“ i „hobi“. Izraz `ime|srednje_ime|prezime` odgovara elementima „ime“, „srednje_ime“ i „prezime“. Izraz `@id|@xlink:type` odgovara atributima „id“ i „xlink:type“. `*|@*` odgovara svim elementima i atributima, ali ne i čvorovima teksta, komentara, niti naredbi za obradu.

4.2 Složeni lokacijski putovi

Sve dosad navedene XPath-ove izraze možemo kombinirati s kosom crtom i tako sa čvora koji se podudario s jednim korakom lokacije prijeći na druge čvorove u hijerarhiji. Nadalje, točka upućuje na kontekstni čvor, dvije točke na roditeljski čvor, a dvije kose crte na potomke kontekstnog čvora.
Korištenjem kose crte `/` dobijemo složeni put. Svaki korak je relativan u odnosu na onaj koji mu prethodi. Ako put počinje simbolom `/`, onda se prvi dio puta računa u odnosu na korijenski čvor. Npr. `/osobe/osoba/ime_i_prezime/ime` izraz počinje od korijenskog čvora, zatim bira sve njegove potomke nazvane „osoba“, onda sve potomke čvorova „osoba“, onda sve potomke čvorova „osoba“ koji su nazvani „ime_i_prezime“ čiji je naziv „ime“. Primijenjeno na naš primjer, to nam daje dva elementa:

```
<ime>Alan</ime>
<ime>Richard</ime>
```

Da bismo naznačili samo tekstualni sadržaj ta dva čvora, moramo poći korak dalje. XPath-ov izraz `/osobe/osoba/ime_i_prezime/ime/text()` bira znakovne nizove „Alan“ i „Richard“. Oba ova izraza počinju znakom `/` pa predstavljaju apsolutne lokacijske putove koji počinju od korijena. Relativni lokacijski putovi računaju se od kontekstnog čvora. Npr. izraz `osoba/@id` bira sve atribute onih potomaka kontekstnog čvora koji su nazvani „osoba“.
Pomoću dvostruke kose crte (`//`) potomak se bira među svim potomcima kontekstnog čvora i iz samog kontekstnog čvora. Simbol `//` na početku XPath-ova izraza bira potomke među svim čvorovima dokumenta. Npr. izraz `//ime_i_prezime` bira sve elemente „ime_i_prezime“ u dokumentu. Izraz `//@id` bira sve atribute „id“ svih elemenata u dokumentu. Izraz `osoba//@id` bira sve atribute elemenata „osoba“ i sve tako nazvane atribute njihovih potomaka.
Simbol dvije točke (`..`) označava roditeljski čvor tekućeg čvora. Npr. izraz `//@id` označava sve atribute „id“ u dokumentu. Stoga `//@id/..` označava sve elemente dokumenta koji imaju atribut „id“. Izraz `//drugo_ime/../ime` identificira sve elemente „ime“ koji su braća (potomci istog roditeljskog čvora) elemenata „drugo_ime“ u dokumentu.
Točka (`.`) označava kontekstni čvor.

XPath u pravilu može upućivati na više čvorova. katkad nam to odgovara, ali ponekad hoćemo suziti skup čvorova i odabrati samo dio čvorova koji odgovaraju izrazu. Svaki korak lokacijskog puta može, ali ne mora imati predikat koji sužava skup čvorova na tom mjestu izraza. Predikat sadrži logički izraz koji se izračunava za svaki čvor iz popisa kontekstnog čvora. Ako je rezultat izraza netočan, taj čvor briše se iz popisa, inače ostaje na popisu. Npr. želimo pronaći sve elemente „zanimanje“ čija je vrijednost „fizičar“. To radi izraz `//zanimanje[. = "fizičar"]`. Umjesto navodnika oko znakovnog niza mogu se staviti polunavodnici (`' '`). To se često radi kad se XPath-ov izraz pojavljuje unutar vrijednosti atributa danog u navodnicima. Kad bismo htjeli sve elemente „osoba“ koji imaju potomka „zanimanje“ čija je vrijednost „fizičar“, upotrijebili bismo izraz `//osoba[zanimanje = "fizičar"]`. Da tražimo elemente „osoba“ čiji atribut „id“ ima vrijednost „p4567“, stavili bismo `@` ispred iemna atributa, kao u izrazu `//osoba[@id = "p4567"]`.
Pored znaka jednakosti XPath podržava i sve druge relacijske operatore, među kojima su <, >, <=, >= i !=, te logičke operatore `and` i `or` za logičko kombiniranje izraza. Npr. izraz `//osoba[@rođena <= 1920 and @rođena >= 1910]` odgovara svim onim elementima „osoba“ u dokumentu čiji atribut „rođena“ ima vrijednost između 1910 i 1920. Svi koraci lokacijskog puta mogu imati predikate. Npr. izraz `//osobe/osoba[@rođena<1950]/ime_i_prezime[ime = "Alan"]` najprije bira sve elemente potomke „osobe“ korijenskog elementa. Zatim od tih elemenata bira sve elemente „osoba“ čiji atribut „rođena“ ima numeričku vrijednost manju od 1950. Na kraju, iz te grupe elemenata bira sve potomke nazvane „ime_i_prezime“ koji imaju potomka „ime“ čija je vrijednost „Alan“.
Uzmimo i sljedeći primjer:

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE library PUBLIC "library.dtd">
<library location="Bremen">
   <author name="Henry Wise">
<book title="Artificial Intelligence"/>
<book title="Modern Web Services"/>
<book title="Theory of Computation"/>
   </author>
   <author name="William Smart">
<book title="Artificial Intelligence"/>
   </author>
   <author name="Cynthia Singleton">
<book title="The Semantic Web"/>
<book title="Browser Technology Revised"/>
   </author>
</library>
```

Evo nekoliko primjera adresiranja XPathom:

- adresiranje svih elemenata „author“.

```
/library/author
```

Ovaj izraz za lokacijski put adresira sve elemente „author“ koji su potomci čvora „library“ koji se nalazi odmah ispod korijena. Korištenjem slijeda $/t_1/ \ldots /t_n$, gdje svaki t_{i+1} je čvor-dijete čvora t_i, definiramo put koji prođe cijelo stablo.

- alternativno rješenje prethodnog primjera je:

```
//author
```

Ovdje `//` označava da nam trebaju svi elementi u dokumentu i sve treba provjeriti jesu li vrste „author“. U našem primjeru imamo isti rezultat pretraživanja, ali općenito to ne mora biti tako.

- adresiranje atributskih čvorova unutar čvorova „library“ elementa;

```
/library/@location
```

- adresiranje svih „title“ atributskih čvorova unutar „book“ elemenata bilo gdje u dokumentu, a koji imaju vrijednost “Artificial Intelligence”;

```
//book/@title="Artificial Intelligence"
```

- adresiranje svih knjiga naslova („title“) “Artificial Intelligence”.

```
//book[@title="Artificial Intelligence"]
```

Izraz unutar uglatih zagrada nazivamo i filtrom jer ograničava skup adresiranih čvorova. Za razliku od prethodnog adresiranja ovdje adresiramo elemente „book“ koje zadovoljavaju uvjet da je „title“ upravo „Artificial Intelligence“. Prethodni primjer adresirao je atributske čvorove „book“ elemenata.

- adresiranje prvog čvora elementa „author“ u XML dokumentu;

```
//author[1]
```

- adresiranje zadnjeg „book“ elementa unutar prvog čvora elementa „author“ u dokumentu;

```
//author[1]/book[last()]
```

- adresiranje svih čvorova „book“ elementa bez „title“ atributa;

```
//book[not @title]
```

4.3 Neskraćeni lokacijski putovi

Osim dosad navedenih skraćenih lokacijskih putova XPath pruža i neskraćenu sintaksu za pisanje lokacija koje je opširnija, jasnija i fleksibilnija od skraćenih lokacijskih putova. Svaki korak lokacije u svom putu ima dva obvezna dijela: osovinu (engl. *axis specifier*) i test čvora (engl. *node test*) te jedan opcionalni dio – predikate. Osovina pokazuje smjer u kojem treba tražiti sljedeće čvorove počevši od kontekstnog čvora. Test čvora pokazuje koje čvorove treba uključiti duž osovine, a predikati dalje sužavaju taj skup čvorova u skladu s određenim izrazom. Predikati filtriraju čvorove pa nalaze specifičnije skupove čvorova. Npr. `[1]` odabire prvi čvor, `[position()=last()]` odabire zadnji čvor, `[position() mod 2 = 0]` odabire samo parne čvorove itd.
U skraćenom lokacijskom putu osovina i test su kombinirani, a u neskraćenom su razdvojeni dvjema dvotočkama (`::`). Npr. skraćeni put lokacije `osobe/osoba/@id` sastoji se od tri koraka. Prvi korak bira čvorove elemenata „osobe“ duž osovine potomaka. Drugi korak bira čvorove elemenata „osoba“ duž osovine potomaka. Treći korak bira čvorove atributa „id“ duž osovine atributa. U neskraćenoj sintaksi isti lokacijski put glasio bi `child::osobe/child::osoba/attribute::id`.
Ovaj neskraćeni oblik je preopširan i rijetko se koristi, ali on nudi jedini način pristupanja većini osovina duž kojih XPath može birati čvorove. Skraćena sintaksa omogućava kretanje duž osovina potomaka, osovine roditelja, vlastite osovine (engl. self), osovine atributa i osovine potomaka. Neskraćena osovina dodaje još osam osovina: `ancestor` (predak – sve iznad konteksta), `following-sibling` (braća ispred konteksta), `preceding-sibling` (braća iza konteksta), `following` (svi sljedeći čvorovi), `preceding` (svi prethodni čvorovi), `namespace` (imenik), `descendant` (potomak – sve ispod konteksta), `ancestor-of-self` (kontekst i sve iznad konteksta) .

5 TRANSFORAMACIJA XML DOKUMENATA

5.1 Jezik XQuery

XQuery je upitni jezik za odabir odabir i transformaciju dokumenata u XML-u. Njega je standardizirao W3C konzorcij. Koristi kao upitni jezik u bazama podataka temeljnima na XML-u ili kod upita koji uključuju nekoliko XML dokumenata u datotečnom sustavu. XQuery ima bogatu kolekciju mogućnosti:

- odabir informacija temeljenih na kriterijima,
- filtriranje neželjenih informacija,
- pretraživanje informacija u jednom ili više dokumenata,
- spajanje podataka iz više dokumenata ili kolekcije dokumenata,
- sortiranje, grupiranje i slaganje podataka,
- transformacija i rekonstrukcija podataka iz XML-a u neki drugi XML ili neku drugu strukturu,
- izvršavanje aritmetičkih operacija nad brojevima i datumima i
- manipuliranje nizom podataka.

XQuery zapravao proširuje mogućnosti XPatha tj. svaki XPath izraz je ujedno i XQuery izraz. XQuery u svojim izrazima koristi XPath, ali treba paziti na inačice XPatha jer XQuery koristi niačicu 2 XPatha. Dok za razliku od njega XSLT koristi inačicu 1 XPatha.
Za sve primjere koristiti ćemo dokument `studij.xml` koji izgleda ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<tel-studij>
  <godina r-br="3">
    <predmet>Informacija, logika i jezici</predmet>
    <predmet>Komunikacijske mreže</predmet>
  </godina>
  <godina r-br="4"/>
</tel-studij>
```

5.1.1 Adresiranje dokumenta

Adresiranje dokumenta u XQueryu se koristi funkcijom `doc` koja kao parametar ima naziv dokumenta. Rezultat te funkcije je da se u kontekst stavlja pročitani dokument i nad njim je moguće izvršavati izraze. Najčešće se koriste XPathovi izrazi:

```
doc("studij.xml")/tel-studij/godina
```

Ovo je jednostavan upit koji pročita dokument `studij.xml` i nad njim izvrši XPathov izraz `/tel-studij/godina`. Rezultat izvođenja su dva elementa `godina`:

```
<godina r-br="3">
  <predmet>Informacija, logika i jezici</predmet>
  <predmet>Komunikacijske mreže</predmet>
</godina>
<godina r-br="4"/>
```

5.1.2 Stvaranje elemenata

Svaki XML-ov element je ujedno i ispravan XQuery upit koji stvara elemente. Sljedeći upit je ispravan:

```
<godina r-br="3">
```

```
  <predmet>Informacija, logika i jezici</predmet>
  <predmet>Komunikacijske mreže</predmet>
</godina>
```

Rezultat izvršavanja tog upita je isti taj element:

```
<godina r-br="3">
  <predmet>Informacija, logika i jezici</predmet>
  <predmet>Komunikacijske mreže</predmet>
</godina>
```

Kod stvaranja elemenata možemo kombinirati i različite izraze. Npr. možemo stvoriti element predmeti u koji ćemo pomoću funkcija iz XQuerya u njega ubaciti odabrane elemente. Sljedeći primjer to radi:

```
<predmeti> {
    doc("studij.xml")//predmet
  } </predmeti>
```

Ono što je u vitičastim zagradama to su XQueryjevi izrazi koji odabrane elemente stavljaju u element predmeti. Rezultat je sljedeći:

```
<predmeti>
  <predmet>Informacija, logika i jezici</predmet>
  <predmet>Komunikacijske mreže</predmet>
</predmeti>
```

Drugi način stvaranja elementa je da se naziv elementa dobije iz XQuery izraza. Najjednostavnije primjer toga je:

```
element { "predmet" } {}
```

Ključna riječ element na početku označava da će se stvoriti novi element. Rezultat prvih vitičastih zagrada je niz znakova koji će biti ime elementa, a rezultat drugih zagrada je sadržaj tog elementa. Rezultat ovog upita je:

```
<predmet/>
```

Jedan složeniji upit bi bio ovakav:

```
element { concat( "godina-",doc("studij.xml")//godina[1]/@r-br ) }
{
  doc("studij.xml")//godina[1]/predmet
}
```

U prvim vitičastim zagradama se poziva funkcija `concat` koja spaja niz znakova `godina-` s vrijednošću atributa `r-br` prvog elementa `godina` u dokumentu `studij.xml`. U drugim vitičastim zagradama je izraz koji dohvaća eve elemente `predmet` u prvom elementu `godina` u dokumentu `studij.xml`. Rezultat je sljedeći:

```
<godina-3>
  <predmet>Informacija, logika i jezici</predmet>
  <predmet>Komunikacijske mreže</predmet>
</godina-3>
```

U drugim vitičastim zagradama možemo ugnijezditi i druge izraze. To pokazuje sljedeći primjer:

```
element { concat( "godina-",doc("studij.xml")//godina[1]/@r-br ) }
{
  <predmeti> {
   doc("studij.xml")//godina[1]/predmet
  } </predmeti>
}
```

Rezultat je:

```
<godina-3>
  <predmeti>
    <predmet>Informacija, logika i jezici</predmet>
    <predmet>Komunikacijske mreže</predmet>
  </predmeti>
</godina-3>
```

5.1.3 Stvaranje atributa

Ako želimo kopirati atribut iz nekog drugog elementa onda nakon stvaranja elementa u vitičastim zagradama trebamo staviti izraz koji odabire atribut:

```
<predmeti>{ doc("studij.xml")//godina[1]/@r-br }</predmeti>
```

Ovdje se stvara novi element `predmeti` i u njemu se kopira atribut `r-br` iz prvog elementa `godina` u dokumentu `studij.xml`. Rezultat je:

```
<predmeti r-br="3"/>
```

Ako želimo kopirati samo vrijednost nekog atributa i staviti ju u drugi atribut onda unutar navodnika atributa stavimo vitičaste zagrade s izrazom koji vraća vrijednost atributa:

```
<predmeti godina="{ doc("studij.xml")//godina[1]/@r-br }"/>
```

Rezultat je:

```
<predmeti godina="3"/>
```

Ako želimo izračunati ime atributa onda moramo koristiti konstruktor atributa (slično konstruktoru elementa). Ključna riječ konstruktora atributa je `attribute` iza koje slijede dva para vitičastih zagrada. U prvom paru treba biti izraz čiji je rezultat ime atributa, a u drugom paru je rezultat vrijednost atributa:

```
<predmet>{
  attribute {
    concat("godina-", doc("studij.xml")//godina[1]/@r-br )
  }
  { "vrijednost"}
}</predmet>
```

Rezultat je:

```
<predmet godina-3="vrijednost"/>
```

5.1.4 Izrazi FLWOR

Izrazi FLWOR nastali su po prvim slovima klauzula koje se koriste (*for*, *let*, *where*, *order by* i *return*). Oni se koriste za složene upite koji omogućuju

- pisanje čitljivijih i strukturiranijih upita,
- spajanje podataka iz više dokumenata,
- stvaranje elemenata i atributa,
- evaluaciju funkcija na varijablama i
- sortiranje rezultata.

U izrazi FLWOR može se koristiti više klauzula `let` i `for` i to u bilo kojime redoslijedu. Nakon toga slijedi klauzula `where` pa je onda moguće staviti klauzulu `order by` i na kraju je obavezna klauzula `return`. Izrazi FLWOR moraju imati barem jednu klauzulu `let` ili `for`.

Izraz FLWOR može biti ili cijeli upit ili može biti dio izraz u drugim izrazima npr. izraz FLWOR može biti stavljen u klauzuli `return` drugog izraza FLWOR ili može biti u parametar poziva neke XQuery funkcije.
Klauzule u izrazima FLWOR imaju sljedeće značenje:

- `let` – koristi se za postavljanje vrijednosti varijable (varijable počinju sa znakom $),
- `for` – postavlja petlju po čvorovima, a ostatak izraza FLWOR se izvršava u svakoj iteraciji,
- `where` – filtrira čvorove koji ispunjavaju uvjet u izrazu,
- `order by` – sortira rezultate i
- `return` – stvara čvorove i slaže odgovore.

Pogledajmo sada jedan izraz s klauzulom `let`:

```
let $godina := doc("studij.xml")//godina[1]
return
  <godina r-br="{$godina/@r-br}">
    <predmeti> {$godina/predmet} </predmeti>
  </godina>
```

Klauzula `let` definira varijablu `$godina` kojoj se dodjeljuje vrijednost izraza koji slijedi iza znakova `:=`. U ovom slučaju će varijabla `$godina` imati vrijednost elementa godina koji je prvi unutar elementa koji ga sadrži, a to je prvi element godina u dokumentu. U klauzuli `return` koristiti se definirana varijabla. Ovdje se stvara element `godina` s atributom `r-br` i njemu se dodjeljuje vrijednost izraza `$godina/@r-br`. Nakon toga se stvara element `predmeti` čiji je sadržaj rezultat izraza `$godina/predmet`. Rezultat ovog upita je:

```
<godina r-br="3">
  <predmeti>
    <predmet>Informacija, logika i jezici</predmet>
    <predmet>Komunikacijske mreže</predmet>
  </predmeti>
</godina>
```

Pogledajmo sada jedan upit koji ima definirane dvije varijable.

```
let $godina := doc("studij.xml")//godina[1],
    $rbr := $godina/@r-br, $predmeti := $godina/predmet
return
  <godina r-br="{$rbr}">
    <predmeti> {$predmeti} </predmeti>
  </godina>
```

Ovdje su definirane varijable `$godina`, `$rbr` i `$predmeti`. Varijable su odvojene zarezom. Rezultat evaluiranja ovog upita je isti kao i kod prošlog upita.
Klauzula `for` koristi se kada se želi napraviti petlje po elementima izraza. Prvo se stavlja ključna riječ `for` onda dolazi ime varijable pa ključna riječ `in` i na kraju izraz. Petlja će se izvršiti za svaki rezultat izraza i trenutni rezultat će biti spremljen u varijablu. Varijabla iz klauzule `for` može se koristiti u ostatku izraza FLWOR. Pogledajmo jedan primjer s klauzulom `for`:

```
for $godina in doc("studij.xml")//godina
let $rbr := $godina/@r-br
return
  <godina r-br="{$rbr}">
    <predmeti> {$godina/predmet} </predmeti>
  </godina>
```

Rezultat evaluiranja je:

```
<godina r-br="3">
  <predmeti>
    <predmet>Informacija, logika i jezici</predmet>
    <predmet>Komunikacijske mreže</predmet>
  </predmeti>
</godina>
```

```
<godina r-br="4">
  <predmeti/>
</godina>
```

Klauzula `where` se koristi za filtriranje rezultata. Ovdje se izvršava prvo klauzula `for` i za svaku iteraciju se izvršava `let` pa onda `where`. Ako klauzula `where` daje istinu onda se izvrši `return` i tako dok se ne izvrše sve iteracije. U klauzuli `where` mogu se koristiti varijable iz klauzula `for` i `let`. Pogledajmo primjer:

```
for $predmet in doc("studij.xml")//predmet
let $rbr := $predmet/parent::godina/@r-br
where $rbr >= 3
return
  <predmet godina="{$rbr}"> {$predmet/text()} </predmet>
```

Rezultat je:

```
<predmet godina="3">Informacija, logika i jezici</predmet>
```

```
<predmet godina="3">Komunikacijske mreže</predmet>
```

Klauzula `order by` koristi se za sortiranje. Nakon što se izvrše iteracije klauzule `for` onda se vrši sortiranje definirano u klauzuli `order by`. Pogledajmo primjer:

```
for $predmet in doc("studij.xml")//predmet
let $rbr := $predmet/parent::godina/@r-br,
    $imePredmeta := $predmet/text()
order by $imePredmeta descending
return
  <predmet godina="{$rbr}"> {$imePredmeta} </predmet>
```

Rezultat je:

```
<predmet godina="3">Komunikacijske mreže</predmet>
```

```
<predmet godina="3">Informacija, logika i jezici</predmet>
```

5.1.5 Izvođenje XQuerya u Javi

Za izvršavanje XQuery upita potrebna nam je knjižnica Qizx/open. Primjer koda koji ju koristi je sljedeći:

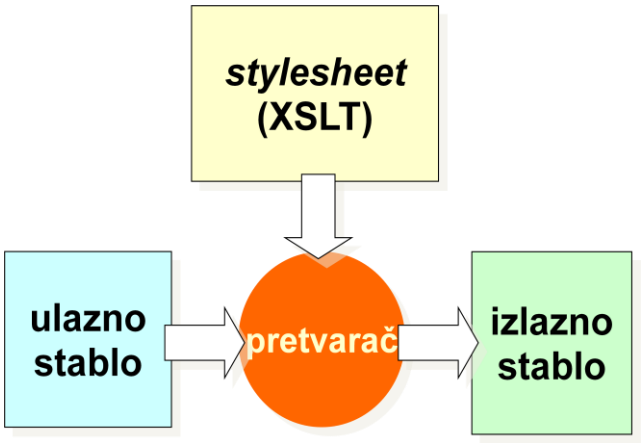
```
File currentDir = new File(System.getProperty("user.dir"));
XQuerySessionManager sm =
   new XQuerySessionManager(currentDir.toURL());
XQuerySession session = sm.createSession();

String script = "ovdje dolazi skripta";
Expression expr = session.compileExpression(script);
ItemSequence results = expr.evaluate();
XMLSerializer serializer = new XMLSerializer();
serializer.setIndent(2);

while (results.moveToNextItem()) {
      Item result = results.getCurrentItem();

      if (!result.isNode()) {
            System.out.println(result.getString());
      } else {
```

```
            String xmlForm =
              serializer.serializeToString((Node) result);
            System.out.println(xmlForm);
        }
}
```

Prvo što moramo napraviti je objekt iz klase `XQuerySessionManager`. On kao parametar prima direktorij u kojem se nalaze dokumenti. Nakon toga se pozivom metode `createSession` stvara sjednica. Sjednica može stvarati izraze tj. upite (klasa `Expression`) tako da se pozove metoda `compileExpression`. Ona kao parametar ima tekst koji predstavlja upit. Ako je upit sintaksno ispravan onda će rezultat biti objekt klase `Expression`. Pokretanje upita vrši se pozivom metode `evaulate` u objektu klase `Expression`. Rezultat je objekt klase `ItemSequence`. Pošto rezultata izvođenja upita može biti više to je onda spremljeno u objekt klase `ItemSequence`. Da bismo dobili pojedini rezultat koristimo metodu `getCurrentItem` koja vraća trenutni rezultat. On je objekt klase `Item`. Ako želimo sljedeći rezultat prvo se moramo pomaknuti na sljedeći rezultat pomoću metode `moveToNextItem`. Ove dvije metode (`getCurrentItem` i `moveToNextItem`) sličen su metodama koje koristimo kod iteratora.
Rezultat je objekt klase `Item`. Pomoću metode `isNode` provjeravamo da li je rezultat element. Ako nije onda ga možemo ispisati jer je u tom slučaju rezultat teksualni čvor ili atribut. Ako je rezultat čvor onda ga možemo ispisati pomoću objekta klase `XMLSerializer`.

5.2 Jezik XSLT

Akronim XSLT znači (Extensible Stylesheet Language Transformation). XSLT je jezik za transformaciju dokumenata koristeći pretvarač (XSLT *processor*) koji kao ulaz ima stil (XSLT) i dokument koji treba pretvoriti, a rezultat je novi dokument (slika 5.1). Jezik XSLT je zapravo XML za razliku od XQuerya koji nije XML.
Kod Weba XSLT se koristi za pretvaranje XML dokumenata u dokumente XHTML ili HTML koji se mogu prikazati u pregledniku. XSLT pretvarač može biti ili na poslužiteljskoj ili na klijentskoj strani. Kada je XSLT pretvarač na poslužitelju onda se XML koji se može nalaziti u datotekama ili nekoj XML bazi podataka pretvara u HTML i HTML se šalje pregledniku. Kada je XSLT pretvarač na klijentu onda se u samom XML-u mora nalaziti i procesorska naredba koja kaže pregledniku koju XSLT transformaciju mora učitati i primijeniti na tom dokumentu da bi ga mogao transformirati u HTML i onda ga prikazati.



Slika 5.1. Prikaz procesa transformacije dokumenata

XSLT ima sljedeće mogućnosti:

- odabire čvorove pomoću XPatha,
- petlje po čvorovima (for),
- sortiranje čvorova,
- filtriranje ,
- grananja if i choose,
- kopiranje elemenata,
- varijable,
- predlošci s parametrima, …

Pretpostavimo da želimo transformirati sljedeći XML:

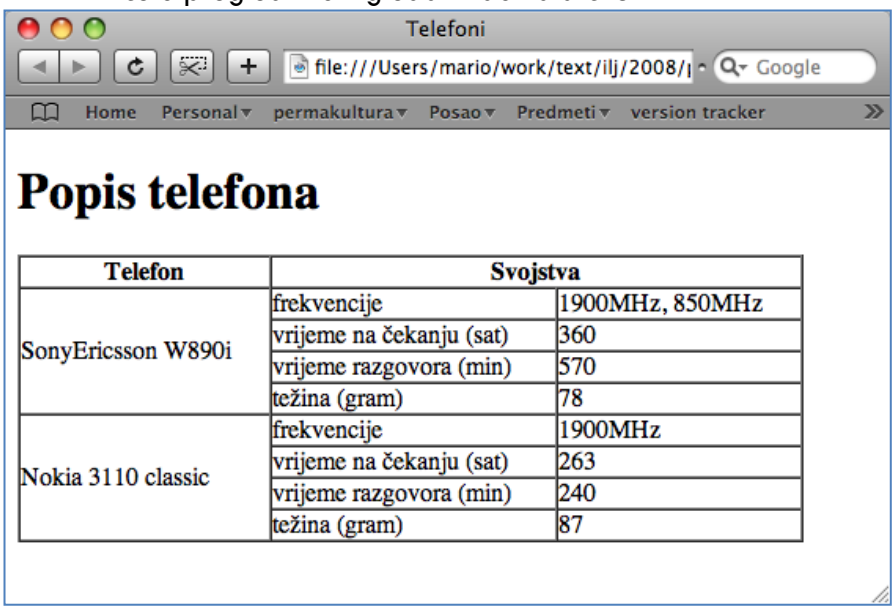
```
<?xml version="1.0" encoding="UTF-8"?>
<pokretniTelefoni>
    <telefon>
        <model>SonyEricsson W890i</model>
        <frekvencija jedinica="Hz">1900</frekvencija>
        <frekvencija jedinica="Hz">850</frekvencija>
        <vrijemeNaCekanju jedinica="sat">360</vrijemeNaCekanju>
        <vrijemeRazgovora jedinica="min">570</vrijemeRazgovora>
        <tezina jedinica="gram">78</tezina>
    </telefon>
    <telefon>
        <model>Nokia 3110 classic</model>
        <frekvencija jedinica="Hz">1900</frekvencija>
        <vrijemeNaCekanju jedinica="sat">263</vrijemeNaCekanju>
        <vrijemeRazgovora jedinica="min">240</vrijemeRazgovora>
        <tezina jedinica="gram">87</tezina>
    </telefon>
</pokretniTelefoni>
```

Taj dokument želimo transformirati u XHTML koji možemo prikazati u pregledniku. Nakon što znamo kako izgleda XML koji želimo transformirati potrebno je vidjeti kako treba izgledati XHTML. Pretpostavimo da željeni XHTML izgleda ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>Telefoni</title>
    <meta content="text/html; charset=utf-8"
          http-equiv="Content-Type" />
  </head>
  <body>
    <h1>Popis telefona</h1>
    <table border="1" cellpadding="0" cellspacing="0" width="500">
      <tr>
        <th scope="col">Telefon</th>
        <th colspan="2" scope="col">Svojstva</th>
      </tr>
      <tr>
        <td rowspan="4">SonyEricsson W890i</td>
        <td>frekvencije</td>
        <td>1900Hz, 850Hz</td>
      </tr>
      <tr>
        <td>vrijeme na čekanju (sat)</td>
        <td>360</td>
      </tr>
      <tr>
        <td>vrijeme razgovora (min)</td>
        <td>570</td>
```

```
      </tr>
      <tr>
        <td>težina (gram)</td>
        <td>78</td>
      </tr>
      <tr>
        <td rowspan="4">Nokia 3110 classic</td>
        <td>frekvencije</td>
        <td>1900Hz</td>
      </tr>
      <tr>
        <td>vrijeme na čekanju (sat)</td>
        <td>263</td>
      </tr>
      <tr>
        <td>vrijeme razgovora (min)</td>
        <td>240</td>
      </tr>
      <tr>
        <td>težina (gram)</td>
        <td>87</td>
      </tr>
    </table>
  </body>
</html>
```

Prikazani XHTML će u pregledniku izgledati kao na slici 5.2.



Slika 5.2. Prikaz XHTML-a u pregledniku

Osnovno načelo rada transformacije se odvija u sljedećim fazama:

1. učitava se ulazno stablo
2. učitava se XSLT (popis predložaka)
3. kreira se prazno izlazno stablo (sadrži samo korijen)
4. odabire se korijenski čvor u ulaznom stablu
5. traži se predložak za njega

Svaki dokument XSLT mora imati deklariran prostor imena (`http://www.w3.org/1999/XSL/Transform`) u kojem su elementi XSLT-a. Korijenski element svakog XSLT dokumenta je `stylesheet`. Pogledajmo primjer:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output encoding="UTF-8" indent="yes" method="xml"
                version="1.0"/>
...
</xsl:stylesheet>
```

U ovom primjeru je definiran prostor imena `xsl` tako da svaki element koji je u tom prostoru je zapravo element XSLT-a. Unutar korijenskog elementa `stylesheet` nalaze se pravila za transformaciju. U ovom promjeru vidimo element `output` koji definira da će izlazni dokument biti kodiran u formatu UTF-8 (atribut `encoding`), da će dokument biti lijepo formatiran s uvlačenjem (atribut `indent`) i da će izlazni dokument biti XML (atribut `method`) verzije 1.0 (atribut `version`). Osim elementa `output` unutar korijenskog elementa se stavljaju predlošci.

5.2.1 Predlošci

Predlošci su elementi XSLT-a koji imaju pravila odabira temeljena na XPathu. Kada je neki predložak odabran na temelju konteksta onda se izvršavaju pravila unutar predloška. Unutar predloška se mogu nalaziti elementi koji su dijelovi XSLT-a ili elementi koji su dio izlaznog dokumenta. Elementi izlaznog dokumenta ne rade transformacije, već samo dodaju te elemente u izlazni dokument. Elementi XSLT-a se izvršavaju po pravilima koja su definirana u standardu.
Pogledajmo jedan predložak:

```
<xsl:template match="/">
    <html>
        <head>
            <title>Telefoni</title>
            <meta content="text/html; charset=utf-8"
                  http-equiv="Content-Type"/>
        </head>
        <body>
            <h1>Popis telefona </h1>
            <table border="1" cellpadding="0" cellspacing="0"
                   width="500">
                <tr>
                    <th scope="col">Telefon</th>
                    <th colspan="2" scope="col">Svojstva</th>
                </tr>
                <xsl:apply-templates select="pokretniTelefoni"/>
            </table>
        </body>
    </html>
</xsl:template>
```

Predložak se nalazi u elementu `template`. On ima atribut `match` čija vrijednost je izraz XPath. Kada je neki kontekst odabran ona pretvarač prolazi kroz sve predloške i pokušava izvršiti izraz XPath. Ako izraz daje rezultat onda se izvršava taj predložak, a ako ne daje rezultat onda se traži neki drugi predložak. Ako za niti jedan predložak nije moguće naći izraz koji daje rezultat onda se provjeravaju ugrađeni predlošci.
Prikazani predložak u izrazu XPath (`/`) odabire korijen dokumenta i on se može primijeniti samo na kontekst korijena dokumenta. Na početku je korijen dokumenta kontekst koji se odabire pa je ovaj predložak taj koji će biti odabran za izvršavanje. Kada se krenu promjenjivati pravila ovog predloška prvo što imamo u predlošku su elementi XHTML-a koji se šalju u izlazni dokument i generiraju dio izlaznog stabla. Prvi element koji je iz XSLT-a je element `apply-templetes`. Taj element pokreće

pretraživanje i izvršavanje drugih predložaka postavljajući novi kontekst za odabir predložaka. U ovom promjeru je pomoću atributa `select` odabran novi kontekst koji se odnosi na elemente `pokretniTelefoni`. Dakle, za svaki element `pokretniTelefoni` će se pokušati pronaći predložak koji odabire taj kontekst i onda će se pokrenuti izvršavanje pravila u tom predlošku. Nakon završetka izvršavanja tog novog predloška (ili više njih) nastavlja se izvršavanje ovog predloška. Ovakav način izvršavanja je sličan pozivanju metoda u programskom jeziku Java ili funkcija u programskom jeziku C.

5.2.2 Ugrađeni predlošci

Kada niti jedan predložak, koji je specificiran u dokumentu XSLT, ne odabire trenutni kontekst onda se pokušava pronaći neki ugrađeni predložak koji odabire taj kontekst. Ako nema niti jednog takvog predloška onda se prekida potraga za predloškom.
Pretpostavimo da za naš primjer nemamo predložak koji odabire kontekst u kojem je element `pokretniTelefoni`, ali da imamo predložak koji odabire element `telefon`. U tom slučaju predložak koji odabire kontekst elementa `pokretniTelefoni` nije nađen pa se traži u ugrađenim predlošcima. Sljedeći predložak je ugrađeni predložak:

```
<xsl:template match="*">
  <xsl:apply-templates select="*"/>
</xsl:template>
```

Ovaj ugrađeni predložak odabire bilo koji element pa tako u ovom slučaju odabire `pokretniTelefoni` i njegova pravila se krenu izvršavati. Pošto je jedini element u njemu `apply-templates` onda se on izvrši. On ovdje odabire bilo koji element unutar konteksta (`pokretniTelefoni`), a to su u ovom slučaju dva elementa `telefon`, i za svakog od njih pokušava se naći predložak. Pošto smo pretpostavili da postoji takav predložak onda se on izvršava.

5.2.3 Sadržaj elementa

Pogledajmo sada kako izgleda predložak koji odabire element `telefon`:

```
<xsl:template match="telefon">
    <tr>
        <td rowspan="4">
            <xsl:value-of select="model"/>
        </td>
        <td>frekvencije</td>
        <td>
            <xsl:apply-templates select=".//frekvencija"/>
        </td>
    </tr>
    <tr>
        <td>vrijeme na čekanju
          (<xsl:value-of select="vrijemeNaCekanju/@jedinica"/>)
        </td>
        <td>
            <xsl:value-of select="vrijemeNaCekanju"/>
        </td>
    </tr>
    <tr>
        <td>vrijeme razgovora
          (<xsl:value-of select="vrijemeRazgovora/@jedinica"/>)
        </td>
        <td>
            <xsl:value-of select="vrijemeRazgovora"/>
```

```
        </td>
    </tr>
    <tr>
        <td>težina (<xsl:value-of select="tezina/@jedinica"/>)
        </td>
        <td>
            <xsl:value-of select="tezina"/>
        </td>
    </tr>
</xsl:template>
```

U ovom promjeru vidimo korištenje elemenata `value-of`. Taj element na mjestu na kojem se nalazi stavlja rezultat odabira pomoću XPatha koji je definiran atributom `select`. Ovaj element se koristi samo za odabir vrijednosti atributa ili za odabir jednostavnog elementa. Pod pojmom jednostavni element misli se na element koji unutar sebe ima samo jedan tekstualni čvor tj. nema druge elemente. Ako je odabran atribut onda se na mjesto elementa `value-of` stavlja vrijednost atributa, a ako je odabran jednostavni element onda se stavlja tekst iz odabranog elementa.
Ako se želi kopirati element koji ima podčvorove (druge elemente) i atribute onda se koristi sljedeći element XSLT-a:

```
<xsl:copy-of select="izraz">
```

Element `copy-of` odabire čvorove XPathovim izrazom u atributu `select` i na njegovo mjesto stavlja odabrane čvorove u izlazni dokument zajedno sa svim podčvorovima i atributima.
Postoji i element copy u XSLT-u. Primjer:

```
<xsl:copy>
  <dodatni-element1/>
  <dodatni-element2/>
</xsl:copy>
```

Element `copy` kopira trenutni element (element konteksta) u izlazno stablo bez podčvorova, a unutar njega se mogu dodavati čvorovi. U ovom primjeru su stvoreni elementi `dodatni-element1` i `dodatni-element2`.

5.2.4 Grananje

U XSLT-u postoje elementi koje služe za grananje. To su:

- element `if` i
- element `choose`.

Pogledajmo primjer predloška za element `frekvencija` koji koristi `if`:

```
<xsl:template match="frekvencija">
    <xsl:value-of select="."/>
    <xsl:value-of select="@jedinica"/>
    <xsl:if test="position()!=last()">
        <xsl:text>, </xsl:text>
    </xsl:if>
</xsl:template>
```

Element `if` ima u atributu `test` izraz XPath koji može vratiti `true` ili `false`. Ako je `true` onda se izvršavaju pravila unutar elementa `if`, a ineče se ne izvršavaju.
Drugi element za grananje je `choose` koji se koristi na sljedeći način:

```
<xsl:choose>
  <xsl:when test="izraz">
    ...
  </xsl:when>
```

```
  <xsl:otherwise>
    ...
  </xsl:otherwise>
</xsl:choose>
```

Element `coose` mora unutar sebe imati element `when` koji ima atribut test s izrazom koji se provjerava. Ako je izraz `true` onda se izvršavaju pravila unutar elementa `when`, a ako nije onda se izvršavaju pravila unutar elementa `otherwise`.

5.2.5 Korištenje XSLT-a u Javi

Primjer korištenja XSLT-a u Javi je sljedeći:

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build("telefoni.xml");

XSLTransformer transformer =
  new XSLTransformer("telefoni_xhtml.xsl");
Document doc2 = transformer.transform(doc);


XMLOutputter out =
  new XMLOutputter(Format.getPrettyFormat());
// ispis na standardni izlaz
out.output(doc2, System.out);
// ispis u datoteku
out.output(doc2, new OutputStreamWriter(
   new FileOutputStream("telefoni.html"), "UTF-8"));
```

Ovdje se koristi JDOM za pokretanje transformacije XSLT nad nekim dokumentom. Prvo što je potrebno je da se učita neki dokument u memoriju. Nakon toga se stvara objekt klase `XSLTransformer` koji je zapravo pretvarač. U konstruktoru mu se šalje referenca na dokument XSLT (može biti tok podataka iz koje se čita dokument, može biti XML dokument u memoriji ili može biti niz znakova koji predstavlja put do datoteke iz koje se čita dokument XSLT). Nakon toga se transformacija pokreće pozivom metode `transform`. Parametar je dokument koji se treba transformirati. Rezultat je novi dokument koji možemo spremiti u datoteku. Posebnu pažnju treba obratiti na format zapisa u datoteku. Java datoteke snima u formatu koji je definiran na operacijskom sustavu. Ako želimo spremiti u drugom formatu onda je potrebno koristiti klasu `OutputStreamWriter`. U primjeru je ona korištena da bi se u datoteku spremili podaci u formatu UTF-8.

6 Literatura

[1] Elliotte Rusty Harold, *XML Bible*, 1999., Wiley
[2] XML standard (1.5.2010.) http://www.w3.org/TR/REC-xml
[3] XML Tutorial (1.5.2010.) http://www.totheriver.com/learn/xml/xmltutorial.html
[4] Brett McLaughlin, Mike Loukides, Java and XML, 2000., O'Reilly Media
[5] Lars Vogel, JAXB Tutorial (1.4.2012.) http://www.vogella.de/articles/JAXB/article.html
[6] Wolfgang Laun, A JAXB Tutorial (1.4.2012.) http://jaxb.java.net/tutorial/
[7] DOM standard (1.5.2010.) http://www.w3.org/DOM/
[8] Knjižnica JDOM (1.5.2010.) http://www.jdom.org/
[9] Projekt SAX (1.5.2010.) http://www.saxproject.org/
[10] XSchema standardi (1.5.2010.) http://www.w3.org/2001/XMLSchema
[11] XPath standard (1.5.2010.) http://www.w3.org/TR/xpath/
[12] XQuery standard (1.5.2010.) http://www.w3.org/TR/xquery/
[13] XQuery Tutorial, (1.5.2010.) http://www.w3schools.com/xquery/default.asp
[14] Knjižnica Qizx/open (1.5.2010.) http://www.xmlmind.com/qizx/
[15] Priscilla Walmsley, XQuery, 2007., O'Reilly Media
[16] XSL Transformations (XSLT) Standard (1.5.2010.) http://www.w3.org/TR/xslt
[17] XSLT Tutorial, (1.5.2010.) http://www.w3schools.com/xsl/default.asp