In diesem Kapitel
- Einleitung
- Syntax einer Datei
- XML aus Strukturierten Daten
- Eingabe, Bearbeitung aus Ausgabe als XML Dokument
  - *Datenbank*
  - *CSV und Tab Dateien*
  - *Excel*
- XML Hierarchien aus einfachen Daten
- Apache POI – Zugriff auf MS Formate und Verbindung mit XML
  - *Excel Daten mit POI HSSF lesen*
  - *Word Dokumente mit POI HDF lesen*
- Google Interface
- Zusammenfassung

4. *Daten in XML umwandeln*

4.1. Einleitung

Die meisten Informationen stehen ziemlich roh zur Verfügung und können analysiert werden:
- Im Web (Web Mining), eventuell in Suchmaschinen
- in Word Dokumenten
- in normalen Text Dateien (Text Mining)
- aus strukturierten Dateien (beispielsweise mit einer DTD)
- als CSV Dateien
- in Excel Spreadsheets
- vielleicht sogar in einer Datenbank (Data Mining).

Wenn wir diese "Daten" (strukturierte und unstrukturierte, wie beispielsweise Text) in XML umformen möchten, stehen wir vor einem kaum allgemein lösbaren Problem, den unstrukturierten Daten.

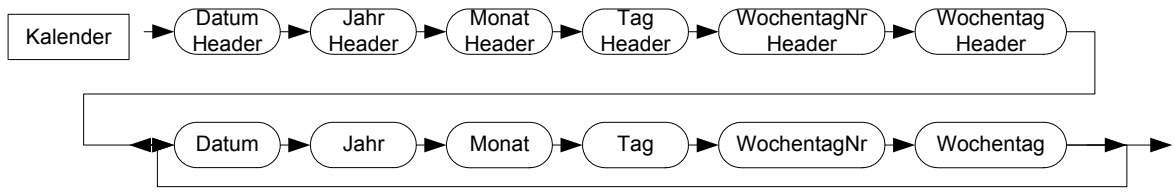
Viele Datenbanken sind in der Lage, XML Ausgaben zu generieren, was ja auch nicht gerade schwierig ist (einfach den Tabellennamen, dann die Spaltennamen mit den Daten dann schon brav wieder alles schliessen).

4.2. Beschreibung der Syntax einer Datei

Wir können den Inhalt einer Datei mithilfe einer Syntax beschreiben:

- gemäss BNF kann eine Syntax als Syntaxgraph / Baum beschrieben werden.
- Wir können dies genau so gut mit DTD machen.
- Die üblichen Operatoren sind + (mindestens 1-mal), * (0...), ?(0 oder 1)

Beispiel:



In unserem Beispiel ist der Aufbau der Datei sehr einfach:

Datum	Jahr	Monat	Tag	WochentagNr	Wochentag
01.01.2003	2003	1	1	2	Mittwoch
02.01.2003	2003	1	2	3	Donnerstag
03.01.2003	2003	1	3	4	Freitag
04.01.2003	2003	1	4	5	Samstag
05.01.2003	2003	1	5	6	Sonntag
06.01.2003	2003	1	6	0	Montag

Wir haben eine Headerzeile und beliebig viele Datenzeilen. Ein Parser für diese Syntax hat folgende Struktur:

```
//Header
 Read DatumHeader
 Read JahrHeader
 Read MonatHeader
 Read TagHeader
 Read WochentagNrHeader
 Read WochentagHeader
 while (not EOF) {      // * Operator entspricht while (+:until)
      Read Datum
      Read Jahr
      Read Monat
      Read Tag
      Read WochentagNr
      Read Wochentag
}
```

Wie würde eine Beschreibung dieser Syntax mit Hilfe einer DTD aussehen?
Schematisch:

```
<!ELEMENT Kalender (Header, Daten+)>
<!ELEMENT Header ...>
<!ELEMENT Daten ...>
```

4.3. XML aus strukturierten Daten

Für den Kalender ergeben sich verschiedene Ausgabeformate, die eventuell sinnvoll sein könnten:

```
<Kalender>
	<Kalendertag>
		<Datum>01.01.2003</Datum>
		<Jahr>2003</Jahr>
		<Monat>1</Monat>
		<Tag>1</Tag>
		<WochentagNr>2</WochentagNr>
		<Wochentag>Mittwoch</Wochentag>
	</Kalendertag>
	<Kalendertag>
		<Datum>02.01.2003</Datum>
		<Jahr>2003</Jahr>
		<Monat>1</Monat>
		<Tag>2</Tag>
		<WochentagNr>3</WochentagNr>
		<Wochentag>Donnerstag</Wochentag>
	</Kalendertag>
	<Kalendertag>
		<Datum>03.01.2003</Datum>
		<Jahr>2003</Jahr>
		<Monat>1</Monat>
		<Tag>3</Tag>
		<WochentagNr>4</WochentagNr>
		<Wochentag>Freitag</Wochentag>
	</Kalendertag>
	...
```

mit folgendem DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Datum (#PCDATA)>
<!ELEMENT Jahr (#PCDATA)>
<!ELEMENT Kalender (Kalendertag+)>
<!ELEMENT Kalendertag (Datum, Jahr, Monat, Tag, WochentagNr, Wochentag)>
<!ELEMENT Monat (#PCDATA)>
<!ELEMENT Tag (#PCDATA)>
<!ELEMENT Wochentag (#PCDATA)>
<!ELEMENT WochentagNr (#PCDATA)>
```

und dem XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
	<xs:element name="Kalender">
		<xs:complexType>
			<xs:sequence>
			<xs:element ref="Kalendertag" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Kalendertag">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Datum"/>
				<xs:element ref="Jahr"/>
				<xs:element ref="Monat"/>
				<xs:element ref="Tag"/>
				<xs:element ref="WochentagNr"/>
				<xs:element ref="Wochentag"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Datum" type="xs:string"/>
	<xs:element name="Jahr" type="xs:string"/>
	<xs:element name="Monat" type="xs:string"/>
```

```
      <xs:element name="Tag" type="xs:string"/>
      <xs:element name="WochentagNr" type="xs:string"/>
      <xs:element name="Wochentag" type="xs:string"/>
</xs:schema>
```

Wenn wir die Daten "vereinfachen" und die Jahreszahl und den Monat nicht in jeder Zeile wiederholen, erhalten wir eine äquivalente Darstellung, die aber von keiner Datenbank mehr verstanden würde.

Wir verlassen die *Datenebene* und bewegen uns auf der *Informationsebene*.

Die Syntaxbeschreibung wird deutlich komplexer, der Speicherbedarf sinkt!

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Kalender (JahrHeader, MonatHeader, WochentagHeader, Jahr+)>

<!ELEMENT MonatHeader (#PCDATA)>
<!ELEMENT JahrHeader (#PCDATA)>
<!ELEMENT WochentagHeader (#PCDATA)>

<!ELEMENT Jahr (#PCDATA | Monat)*>
<!ELEMENT Monat (#PCDATA | Wochentag)*>

<!ELEMENT Wochentag (#PCDATA | WochentagNr | Wochentag | Datum)*>
<!ELEMENT WochentagNr (#PCDATA)>
<!ELEMENT Datum (#PCDATA)>
```

```
<?xml version="1.0"?>
<Kalender>
      <JahrHeader>Jahr</JahrHeader>
      <MonatHeader>Monat</MonatHeader>
      <WochentagHeader>Wochentag</WochentagHeader>
      <Jahr>2003
            <Monat>1
                  <Wochentag>
                        <WochentagNr>0</WochentagNr>
                        <Wochentag>Montag</Wochentag>
                        <Datum>1-1-2003</Datum>
…
                  </Wochentag>
            </Monat>
            <Monat>1
…
      </Jahr>
      <Jahr>2004
…
```

4.4. Datenstrukturen

Die Datenstruktur, die unseren Daten angepasst ist, hängt von der Beschreibungsebene ab:

- auf der Datenebene haben wir lauter identisch aufgebaute Datensätze.
  Die Anzahl ist variabel. Eine mögliche Datenstruktur ist eine Liste für den Kalender und eine eigene Klasse für den Datensatz.
  Eingabe:
  wir können die Daten zeilenweise einlesen, ein Datenrecord-Objekt kreieren und der Liste hinzufügen.
- Auf der Informationsebene ist die Struktur ähnlich, das Datenrecord-Objekt aber anders.
  Eingabe:
  hier muss die Position der Daten abgefragt werden, um "Gruppenbrüche" feststellen zu können (Gruppenbruch: ein neuer Monat, ein neues Jahr).

Die Ausgabe als XML ist in beiden Fällen nicht besonders schwierig.

4.5. Eingabe, Bearbeitung und Ausgabe als XML Dokument

4.5.1. Datenbank

Falls die Daten in einer DB stehen, wenden wir JDBC (evtl. mit JDBC-ODBC) an. Der Einfachheit halber verwende ich Cloudscape, weil diese DB mit der J2EE Referenz-Implementierung von Sun mitgeliefert wird.

Sie finden zwei Beispiele im ZIP:

1) lesen der Daten aus einer Datei im CSV Format,
   eintragen der Datenelemente als String Arrays in eine `LinkedList`
   kreieren der DB Tabelle und einfügen der Daten.
2) Lesen der Daten aus der DB Tabelle und Ausgabe in eine XML Datei.

Teil 1:

```
private LinkedList readData() {
      LinkedList liste = new LinkedList();
      String[] record = new String[6];
      String header, data;
      /* Header Datum;Jahr;Monat;Tag;WochentagNr;Wochentag
       * 1           2    3 4 5   6
       * 01.01.2003;2003;1;1;2;Mittwoch
       */
       try {
            BufferedReader br = new BufferedReader(new
                        FileReader("Kalender.csv"));
            // Header
            header=br.readLine();
            // Daten
            while ((data=br.readLine())!=null) {
                  record = splitRecord(data);
                  liste.add(record);
            }
       } catch (FileNotFoundException fnfe) {
            fnfe.printStackTrace();
       } catch (IOException ioe) {
            ioe.printStackTrace();
       }
      return liste;
}
```

```
	private String[] splitRecord(String pData) {
		String data[] = new String[6];
		String temp;
		temp=pData.trim();
		int iEnd;
		// 01.01.2003;2003;1;1;2;Mittwoch
		for (int i=0; i<5; i++) {
			iEnd = temp.indexOf(';');
			data[i] = temp.substring(0,iEnd);
			temp = temp.substring(iEnd+1);
		}
		// am Schluss steht kein ';'
		data[5]=temp;
		return data;
	}
...
```

Eintragen in die Datenbank Tabelle "Kalender":

```
		String[] zeile = new String[6];
		//    "(01.01.2003,  2003,  , 1, 1, 4, 'Mittwoch')",
		Iterator it = rohDaten.iterator(); //LinkedList
		while(it.hasNext()) {
			zeile = (String[])it.next();
				iRowCount += stmt.executeUpdate(
			"INSERT INTO Kalender VALUES " +
			"('"+zeile[0]+"',"+zeile[1]+","+zeile[2]+","+
			zeile[3]+","+zeile[4]+","+"'"+zeile[5]+"')");
		}
```

Teil 2: Generieren des XML-Dokuments

```
	try    {
	  System.out.println("[KalenderAbfragen]executeQuery(\"SELECT ...\"
");
	  ResultSet result = stmt.executeQuery(
		"SELECT Jahr, Monat, Tag, Datum, WochentagNr, Wochentag " +
		"FROM Kalender"); // Order by : bei String-Datum problematisch!

		/* XML Ausgabe
		 *  <Kalender>
		 *    <Kalendertag>
		 *    <Jahr>2003</Jahr> <Monat>1</Monat><Tag>1</Tag>
		 *    <WochentagNr>2</WochentagNr>
		 *    <Wochentag>Mittwoch</Wochentag>
		 *    <Datum>01.01.2003</Datum>
		 *   </Kalendertag>
		 */
		BufferedWriter bw = new BufferedWriter(new
				FileWriter("KalenderAusgabe.xml"));

		bw.write("<?xml version=\"1.0\"?>\n\r");
		bw.write("<Kalender>");
		while(result.next()) {  // Daten zeilenweise lesen
			bw.write("<Kalendertag>\n\r");
			bw.write("<Jahr>\n\r");
			iJahr = result.getInt("Jahr");
			bw.write(iJahr+"</Jahr>\n\r");
…
			bw.write("</Kalendertag>\n\r");
		}
		bw.write("</Kalender>\n\r");
		bw.flush();
		bw.close();
	} // end try
```

4.5.1.1. Attribute

Als alternatives Ausgabeformat könnten die Jahreszahl und eventuell weitere Daten jeweils als Attribut definiert werden. Die Generierung der Ausgabe unterscheidet sich jedoch kaum: lediglich die Definition der Ausgabezeichenkette muss etwas modifiziert werden.

Schematisch:

```
<?xml version="1.0" encoding="UTF-8"?>
   <Kalendertag Jahr="2003" Monat="1" Tag="1" Wochentag="2">
      <Wochentag>Mittwoch</Wochentag>
      <Datum>01.01.2003</Datum>
   </Kalendertag>
   <Kalendertag Jahr="2003" Monat="1" Tag="2" Wochentag="3">
      <Wochentag>Donnerstag</Wochentag>
      <Datum>02.01.2003</Datum>
   </Kalendertag>
</Kalender>
```

4.5.2. CSV und Tab Dateien

Aus der Lösung für die Datenbank können Sie leicht eine Lösung für CSV Dateien und mit einigen kleineren Änderungen eine Lösung für Tab Dateien erstellen.

4.5.3. Excel

Auf Excel kann man auf der einen Seite über ODBC zugreifen, also genau so wie auf eine Datenbank. Dabei muss man den Datenbereich markieren und diesem einen eindeutigen Namen geben. Über diesen Namen kann dann der ODBC Treiber auf Excel (mit SQL) zugreifen. Beispiele dazu finden Sie in den JDBC Unterlagen.

Alternativ kann man mit Apache POI auf MS Office Dokumente zugreifen. Das schauen wir uns genauer an.

4.6. XML Hierarchien aus einfachen Daten

Wie wir oben gesehen haben, kann man an Stelle der Tabellenstruktur auch ein hierarchisch strukturiertes Dokument ausgeben, mit einer komplexeren DTD. Auf diese Ausgabe verzichte ich. Interessanter wäre ein allgemeines Ausgabeprogramm, welches eine DTD oder ein XML Schema liest und daraus eine gewünschte Ausgabe konstruiert.

Schauen wir uns die Daten genauer an, dann stellen wir fest, dass wir immer dann eine Hierarchiestufe bilden können, falls ein Wert an der selben Stelle in aufeinanderfolgenden Zeilen steht, wobei die Spalten ebenfalls "hierarchisch" geordnet sein müssen:

```
<jahr>2003</jahr><monat>Jan</monat><woche>1</woche><datum>1.1.2003</datum>
<jahr>2003</jahr><monat>Jan</monat><woche>1</woche><datum>2.1.2003</datum>
```

kann zusammengefasst werden:

```
<jahr>2003<monat>Jan<woche>1
<datum>1.1.2003</datum>
                        </woche>
                  </monat>
</jahr>
```

4.7. Apache POI – Zugriff auf MS Formate und Verbindung mit XML

Die Ausgabe von Daten in ein MS Format aus Java ist nicht einfach. Apache startete das POI Projekt, welches verschiedene Teilprojekte (für Excel, Word) umfasst. Teile des POI Projekts sind in Apache Cocoon eingeflossen. Hier beschränken wir uns auf POI.

4.7.1. Excel Daten mit POI HSSF lesen

Die Extraktion der Daten geschieht mit dem Programm `POIKalender.java`

```
try {
      fs = new POIFSFileSystem(new
                        FileInputStream("Test.xls"));
      HSSFWorkbook wb = new HSSFWorkbook(fs);
      HSSFSheet sheet = wb.getSheetAt(0); //Kalender
      HSSFRow row;
      HSSFCell cell;
      int first = sheet.getFirstRowNum();
      int last = sheet.getLastRowNum();

      String jahr, datum, wochentag;
      Date date;
      int  monat, tag, wochentagNr;
      short firstCell, lastCell;

      short s[] = new short[6];
      for (int i=0;i<6;i++) s[i]=(short)i;

      // Header
      row = sheet.getRow(0);
      firstCell = row.getFirstCellNum();
      lastCell = row.getLastCellNum();

      for (short sx=(short)(firstCell); sx<lastCell; sx++) {
            cell =row.getCell(sx);
            inhalt=cell.getStringCellValue();
            System.out.print(inhalt+ " ");
      }
      System.out.println();

      // Daten
      for (short sx=(short)(first+1); sx<=last; sx++) {
            row = sheet.getRow(sx);
            firstCell = row.getFirstCellNum();
            lastCell = row.getLastCellNum();

            cell = row.getCell((short)0);
            java.util.Date datum_1 =
                        (Date)cell.getDateCellValue();
            cell = row.getCell((short)1);
            int jahr_1 = (int)cell.getNumericCellValue();
            cell = row.getCell((short)2);
            int monat_1 = (int)cell.getNumericCellValue();
            cell = row.getCell((short)3);
            int tag_1 = (int)cell.getNumericCellValue();
            cell = row.getCell((short)4);
            int wo_1 = (int)cell.getNumericCellValue();
            cell = row.getCell((short)5);
            String wtag = cell.getStringCellValue();
      }
```

4.7.2. Word Dokumente mit POI HDF lesen

Dieses Projekt ist noch nicht so weit. Teile der entwickelten Software sind in Cocoon eingeflossen.
Bei Beiden POI Formaten muss man beachten, dass

1) Microsoft Formate laufend ändert
2) POI sich auf die 97-er Formate eingestellt hat
3) MS angekündigt hat, vermutlich in Zukunft XML als Standard Speicherformat zu verwenden....

Was man aber jederzeit kann, ist ein C++/C# Programm schreiben und in Java einbinden.
Die Formatierung eines Word Dokuments als XML / XHTML können Sie auch am Beispiel TestWord.xml (generiert mit XML Spy) erkennen.

4.8. Google Interface

Wenn Sie einen Schritt weiter gehen wollen, und Web Dokumente analysieren und klassifizieren wollen, bietet das Google Java Interface ein guter Startpunkt. Sie können Abfragen starten und die Antwort recht einfach parsen. Google bietet in absehbarer Zeit (in den USA heute) ein XML Interface und einen News Dienst an, analog zum jetzigen "News Neu " Dienst.

Startpunkt: http://www.google.com/apis/
Von dort können Sie das Java Archiv herunterladen. Zudem benötigen Sie einen Google Key. Dieser wird Ihnen per email zugestellt.

Der erste Test ist im Readme. Ein passendes Build.xml finden Sie im ZIP.

Sie können beispielsweise aus Java heraus Google Abfragen starten:

```
    String clientKey = args[0];
    String directive = args[1];
    String directiveArg = args[2];

    // Report the arguments received
    System.out.println("Parameters:");
    System.out.println("Client key = " + clientKey);
    System.out.println("Directive  = " + directive);
    System.out.println("Args       = " + directiveArg);

    // Create a Google Search object, set our authorization key
    GoogleSearch s = new GoogleSearch();
    s.setKey(clientKey);
...
...
    System.err.println("Usage: java " +
                       GoogleAPIDemo.class.getName() +
                       " <client-key>" +
                       " (search <query> | cached <url> | spell
<phrase>)");
```

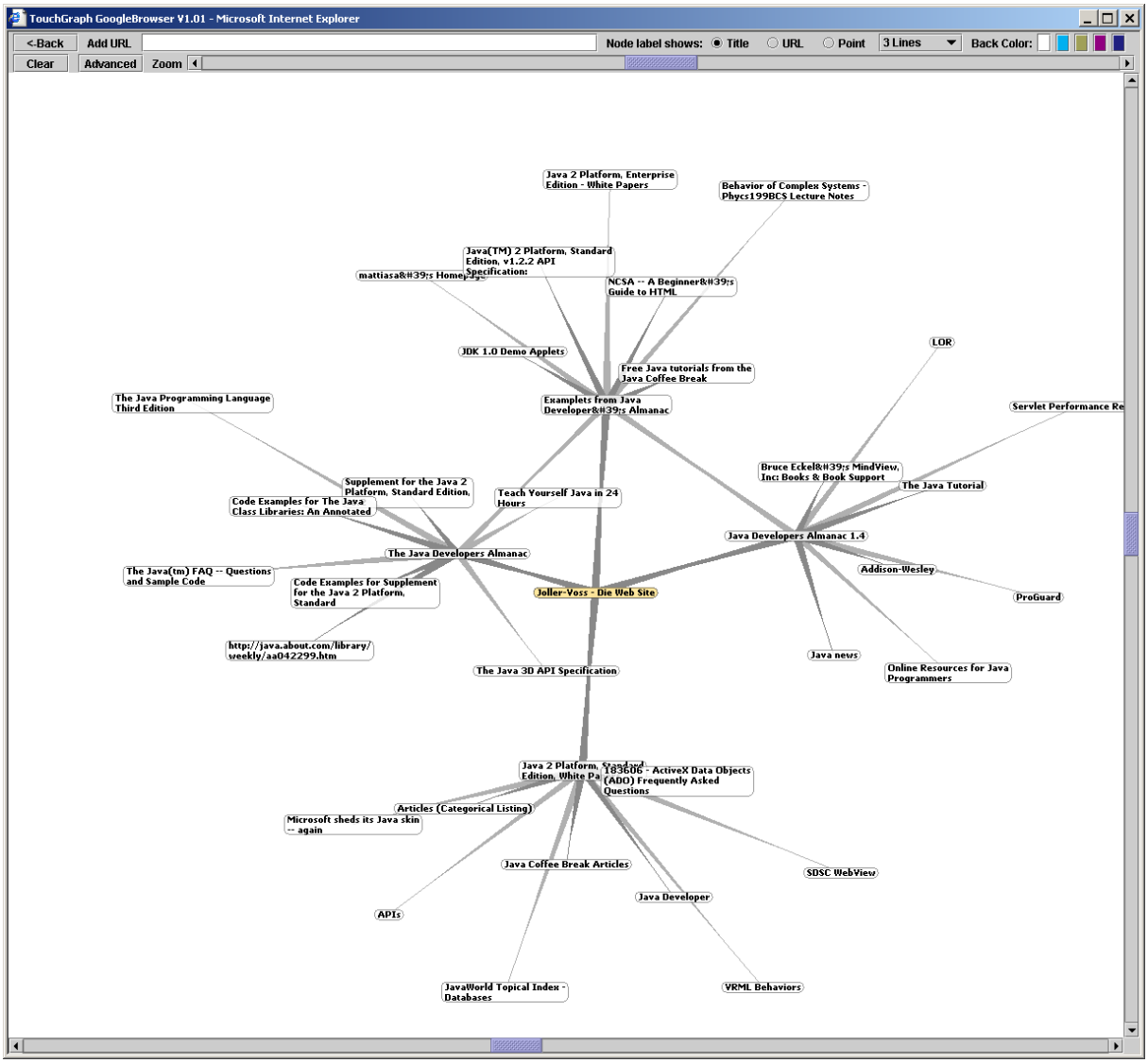
Das Ergebnis können Sie leicht als XML Ausgeben, da das Ergebnis klar strukturiert ist:

```
compile:
    [javac] Compiling 1 source file to C:\googleapi\build

run:
     [java] Parameters:
     [java] Client key = zeig ich nicht
     [java] Directive  = search
     [java] Args       = Google API
     [java] Google Search Results:
     [java] ======================
     [java] {
     [java] TM = 0.167939
     [java] Q  = "Google API"
     [java] CT = ""
     [java] TT = ""
     [java] CATs =
     [java]   {
     [java]   <EMPTY>
     [java]   }
     [java] Start Index = 1
     [java] End   Index = 10
     [java] Estimated Total Results Number = 168000
     [java] Document Filtering = true
     [java] Estimate Correct = false
     [java] Rs =
     [java]   {
     [java]
[
URL  = "http://www.google.com/apis/"
Title = "<b>Google</b> Web APIs - Home"
Snippet = "<b>Google</b> Web APIs (beta). Home All About <b>Google</b>
           <b>Google</b> Web APIs Overview Download<br> Create Account
           Getting Help <b>API</b> Terms FAQs Reference Find on this site:
           <b>...</b>  "
Directory Category = {SE="",
           FVN="Top/Computers/Internet/Searching/
           Search_Engines/Google/Web_APIs"}
Directory Title = "<b>Google</b> Web APIs"
Summary = "Official site. Includes developer's kit, terms and conditions,
           and FAQ."
Cached Size = "12k"
Related information present = true
Host Name = ""
],
     [java]
     [java]   [
     [java]   URL  = "http://api.google.com/"
     [java]   Title = ""
     [java]   Snippet = ""
     [java]   Directory Category = {SE="", FVN=""}
     [java]   Directory Title = ""
     [java]   Summary = ""
     [java]   Cached Size = ""
     [java]   Related information present = true
     [java]   Host Name = ""
     [java]   ],
```

Soweit das Demo Programm. Da Google seine Dienste auch als Web Services anbietet, werden wir später noch mal darauf zurück kommen. Das Extrahieren der einzelnen Felder ist besonders einfach, weil das Google API entsprechende Methoden zur Verfügung stellt (siehe Beispiel im ZIP).

Eine Anwendung wäre der Aufbau einer Verknüpfungstabelle, wie Sie beispielsweise von TouchGraph erstellt wird :



(http://www.touchgraph.com/TGGoogleBrowser.html). Ein Schritt weiter geht der OIModeller (http://kaon.semanticweb.org/Members/motik/oimodeller), bei dem Ontologien verarbeitet und bearbeitet werden.

4.9. Zusammenfassung

Die Umwandlung von Daten (und Informationen) in XML Dokumente ist nicht besonders schwierig. Schwierig sind allgemeinere Aufgaben, wie etwa die Formatierung der Ausgabe aufgrund einer DTD oder eines Schemas. Aber dafür gibt es Apache Projekte und Java XML.

4. DATEN IN XML UMWANDELN 1