Yvonne Barth
Geboren am 15.11.1978
Matrikel-Nummer: 412026

FACHHOCHSCHULE **HOCHSCHULE FÜR**
STUTTGART **TECHNIK**

STUTTGART UNIVERSITY OF APPLIED SCIENCES

Studiengang Vermessung und Geoinformatik

Diplomarbeit

Als PG nach der SPO-2000

Ausgeführt für die Diplomprüfung am Ende des Wintersemesters 2003/04

Nutzung von Servlets, JavaServer Pages, XML und XSL zur SVG-basierten Visualisierung raumbezogener Daten

Erstprüfer und Betreuer: Prof. Dr.-Ing. Franz-Josef Behr

Zweitprüfer: Prof. Dr.-Ing. Wolfgang Huep

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Zum Thema

In Bayern werden sämtliche Liegenschaften, dazu gehören die Grundstücke und die Gebäude, im Liegenschaftskataster beschrieben und graphisch dargestellt. Bisher wurden die zur graphischen Darstellung verwendeten Flurkarten in analoger Form geführt:

? Maßstab 1: 1000 im bebauten Gebiet

? Maßstab 1 : 2500 überwiegend in fränkischen Gebieten

? Maßstab 1 : 5000 überwiegend im ländlichen Gebiet

Bis Ende des Jahres 2003 soll die Flurkarte für Bayern in digitaler Form flächendeckend vorliegen. In Gebieten, in denen es die Digitale Flurkarte (DFK) derzeit noch nicht gibt, wird weiterhin die analoge Flurkarte verwendet. Den beschreibenden Teil des Liegenschaftskatasters bildet das Liegenschaftsbuch.

In das Liegenschaftskataster wird jedem Einsicht gewährt, soweit nicht Interessen des öffentlichen Wohls entgegen stehen. Auszüge aus dem Liegenschaftsbuch und aus der amtlichen Flurkarte werden vom zuständigen Vermessungsamt auf Antrag erstellt und können sowohl auf Papier als auch in digitaler Form bezogen werden. Für Fachanwender, also Geschäfts- und Behördenkunden, besteht zudem die Möglichkeit die Daten der DFK über das Internet zu erhalten[1].

Personenbezogene Daten, dazu gehören zum Beispiel die Eigentümerdaten des Liegenschaftskatasters, unterliegen dem Datenschutz. Deshalb muss für die Einsicht in diese personenbezogenen Daten sowie für Auskünfte und Auszüge aus dem Liegenschaftsbuch ein berechtigtes Interesse nachgewiesen werden.

Die Aufgabe der vorliegenden Diplomarbeit besteht in der Erstellung eines Programms, das die von der Vermessungsverwaltung beziehbaren Daten der DFK in Vektorgrafiken umwandeln kann. Diese Grafiken sollen mit Hilfe eines Plug-Ins[2] in einem Internetbrowser darstellbar sein. Das Programm soll auf einem Server betrieben werden und somit von verschiedenen Clients aus anwendbar sein.

Ein weiterer Teil der Diplomarbeit besteht in der Verwendung von verschiedenen aktuellen Technologien, wie Servlets, JavaServer Pages, XML und XSL.

1.2 Die gesetzten Ziele

Die Daten der Digitalen Flurkarte werden im DFK-Format weitergegeben und können mit Hilfe eines Geoinformationssystems (GIS), das dieses Format unterstützt, angezeigt werden. Da das DFK-Format zum einen aber nicht von allen Systemen unterstützt wird und zum anderen die Kosten für ein GIS sehr

---

1 Bezugsquelle: http://www.geodaten.bayern.de

2 Ein Programm, das in ein anderes Programm eingefügt wird, um zusätzliche Funktionen zur Verfügung zu stellen

hoch sind, haben Privatpersonen und kleinere Gemeinden oftmals keine Möglichkeit sich die digitalen Daten des Liegenschaftskatasters in Form einer Grafik anzeigen zu lassen. Dies soll sich mit der vorliegenden Diplomarbeit ändern.

Die Erstellung des Programms zur Visualisierung von DFK-Daten lässt sich in folgende Teilbereiche untergliedern:

1. Unter Verwendung von Servlets und JavaServer Pages muss eine DFK-Datei vom Client auf den Server geladen werden. Diese Datei ist zuvor vom Benutzer auszuwählen. Wird eine gültige DFK-Datei angegeben, so muss diese nach erfolgtem Upload serverseitig, zeilenweise eingelesen werden. Die so erhaltenen Informationen sind in entsprechenden Objekten und diese wiederum in passende Objektlisten zu speichern.
2. Die Informationen, die in den Objektlisten gespeichert sind, werden in verschiedene HTML[3]- und in eine XML[4]-Datei ausgegeben.
3. Es ist eine XSL[5]-Datei zu schreiben, mit der die zuvor erstellte XML-Datei in eine SVG[6]-Datei transformiert werden kann. Durch die generierte SVG-Datei können die Informationen der DFK-Datei graphisch als Karte angezeigt werden.
4. Die Karte soll interaktive Funktionen besitzen, die der Benutzer über ein Menü ausführen kann. Durch diese interaktiven Funktionen, die in JavaScript erstellt werden, soll die Möglichkeit bestehen die Karte individuell anzupassen. Verschiedene Maßstäbe sollen ausgewählt, der Ausschnitt verschoben und die Originalansicht wiederhergestellt werden können. Ebenen werden nach Wunsch ein- bzw. ausgeblendet, zudem soll eine Auswahl zwischen schwarz-weißer oder farbiger Darstellung möglich sein. Weiterhin soll zu Objekten, die mit der Maus ausgewählt werden, innerhalb der zuvor generierten HTML-Dateien Zusatzinformationen angezeigt werden.

Die verwendeten Technologien Servlets, JavaServer Pages, HTML, XML, XSL, SVG und JavaScript werden im Kapitel „Die verwendeten Werkzeuge und Programmiersprachen“ genau definiert und anhand von Beispielen erläutert.

1.3 Die Vorraussetzungen

Um das im Rahmen der Diplomarbeit erstellte Programm DFKviewer möglichst breit nutzbar zu machen, wurde versucht, die Anforderungen auf der Clientseite an Hard- und Software möglichst gering zu halten.

Zum Anzeigen der Karte benötigt der Benutzer nur einen gängigen Internetbrowser und ein spezielles Plug-In. Bei dem Plug-In handelt es sich um den SVG Viewer von Adobe, durch ihn wird die im Browser integrierte Script-Engine durch die Script-Engine des SVG Viewers ersetzt. Dies ist erforderlich, da die Script-Engine des Browsers nur das Document Object Model[7] (DOM) von HTML-Dokumenten unter-

---

[3] Hypertext Markup Language

[4] Extensible Markup Language

[5] Extensible Stylesheet Language

[6] Scalable Vector Graphics

[7] Programmiersprachenunabhängig kann auf HTML- und XML-Dokumente, somit auch auf SVG-Dokumente, zugegriffen und deren Inhalte dynamisch verändert werden

stützt, nicht aber das DOM des SVG-Formates, das benötigt wird, um die interaktiven Funktionen, die mit JavaScript geschrieben sind und auf dem DOM basieren, auszuführen.

1.4 Die verwendeten Schreibweisen

In diesem Dokument werden Begriffe, die zum ersten Mal verwendet werden entweder direkt im Text erläutert oder über eine zugehörige Fußnote näher definiert.

Programmierbeispiele, Variablen- und Funktionsnamen werden durch die Verwendung einer anderen Schriftart vom restlichen Text abgehoben.

```
// Schriftart für Quellcode
```

1.5 Zum Aufbau

Im Anschluss an die Einleitung, in der die Diplomarbeit in ihren Grundzügen vorgestellt wird, folgt das Kapitel „Die verwendeten Werkzeuge und Programmiersprachen“, in dem jeweils eine Vorstellung und Einführung zu den verwendeten Technologien gegeben wird. Im nächsten Kapitel „DFK - Das Datenaustauschformat der bayrischen Vermessungsverwaltung“ wird der Inhalt einer DFK-Datei erläutert. Den Hauptteil bildet das Kapitel „Das Programm DFKviewer“ in dem detailliert beschrieben wird, wie aus den DFK-Daten eine interaktive Karte generiert wird. Den Abschluss bildet ein Resümee.

1.6 Das Ergebnis

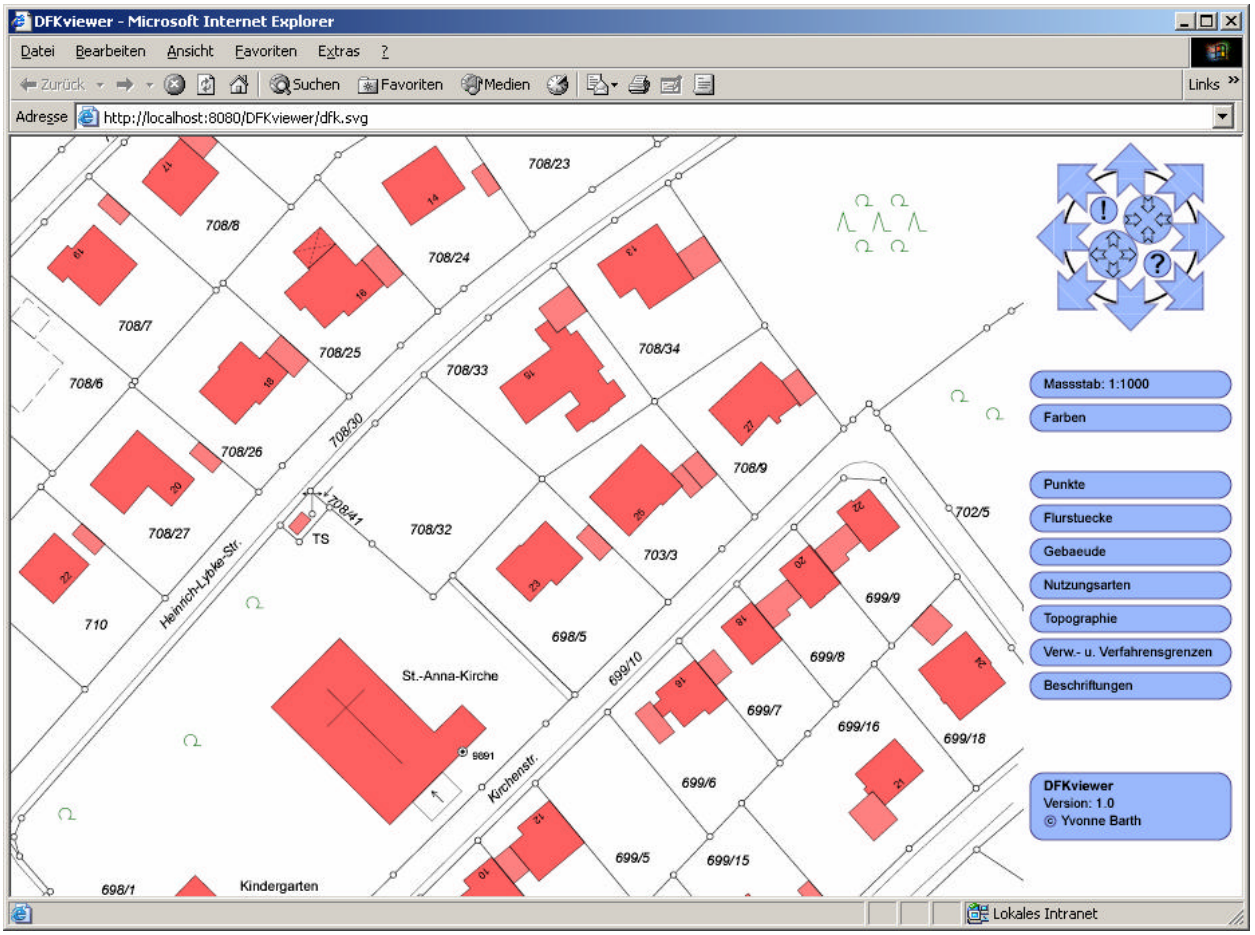

Abbildung 1: DFKviewer

Abbildung 1 zeigt das Ergebnis der Umsetzung einer DFK-Datei in eine interaktive Karte. Die Visualisierung erfolgt dabei im Internet Explorer mit Hilfe des SVG Viewers von Adobe. Über das integrierte Menü kann der angezeigte Kartenausschnitt beliebig vergrößert, verkleinert oder auch verschoben werden, wobei der aktuelle Maßstab stets angezeigt wird. Es können die Maßstäbe 1:100, 1:250, 1:333, 1:500, 1:1000, 1:1500, 1:2500 und 1: 5000 direkt ausgewählt werden. Über den Button mit dem Ausrufezeichen kann der ursprüngliche Maßstab mit der gesamten Karte wiederangezeigt werden. Wird das Fragezeichen angeklickt, so werden die Vordaten der DFK-Datei angezeigt. Über das Menü besteht die Möglichkeit verschiedene Ebenen aus der Karte ein- bzw. auszublenden. Bei den Punkten zum Beispiel die Ebenen für Trigonometrische Punkte, Katasterfestpunkte und Grenzpunkte, bei den Flurstücken die Ebenen für Flurstücksgrenzen und Flurstücksnummern. Dadurch kann die Karte nach individuellen Wünschen gestaltet werden. Zudem ist es durch Anklicken des Buttons Farben möglich, von der farbigen Darstellung zur schwarz-weißen Darstellung zu wechseln. Weiterhin lassen sich durch Anklicken eines vorhandenen Punktes in der Karte zusätzliche Informationen zu diesem Punkt anzeigen. Abbildung 2 zeigt beispielsweise die Zusatzinformationen für den Punkt 300 der Flurkarte 30940549 an.

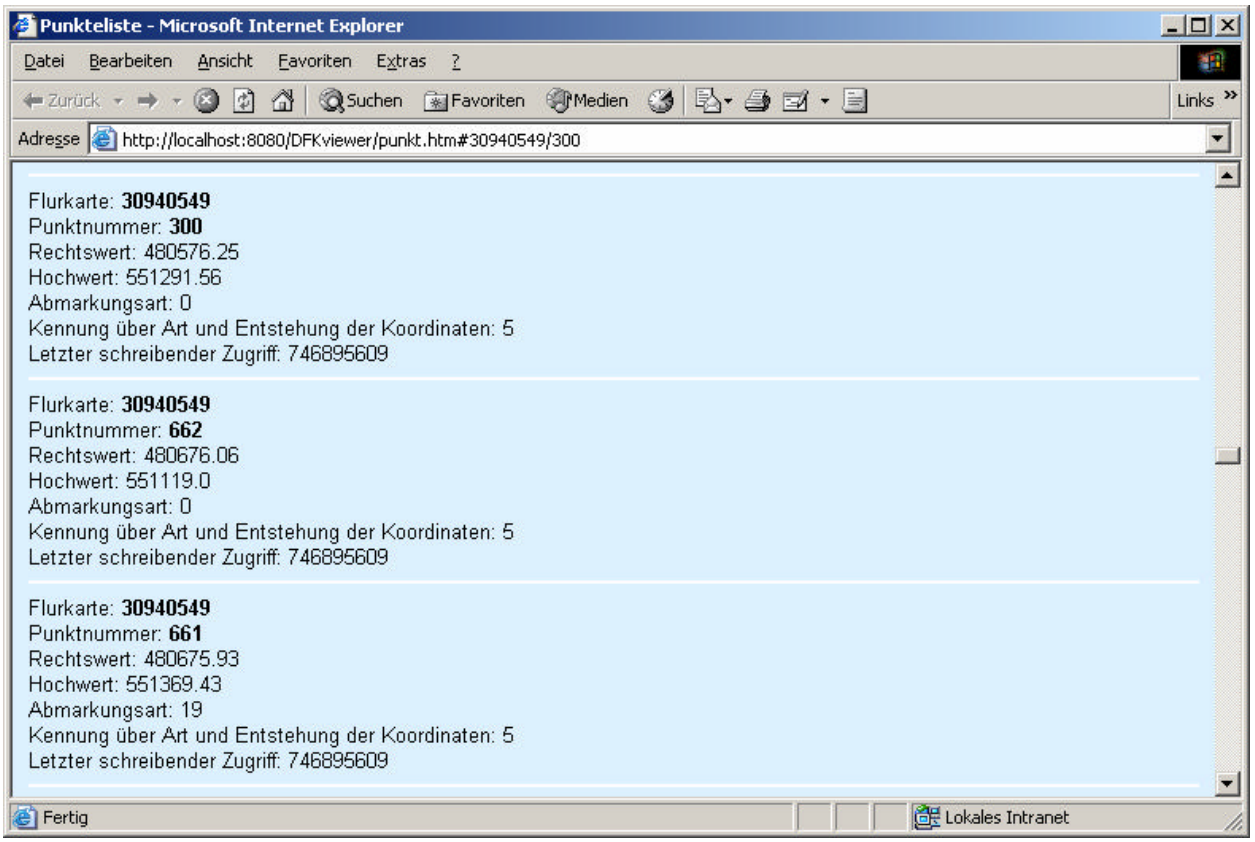

Abbildung 2: Zusatzinformationen für Punkte im DFKviewer

2 Die verwendeten Werkzeuge und Programmiersprachen

2.1 Servlets

Damit der Zweck von Servlets[8] verstanden werden kann, müssen zuerst ein paar Begriffe erläutert werden. Das Anfordern einer Webseite mittels Eingabe der URL oder über das Klicken eines Hyperlinks ist eine dynamische Interaktion mit dem Medium Internet. Eine Interaktion besteht aus einer Aktion und einer Reaktion. In diesem Zusammenhang bedeutet dies, dass auf die Aktion „Aufruf einer Webseite“ eine Reaktion „Senden einer Webseite“ folgen sollte. Dynamik bedeutet, dass durch eine Interaktion mit dem Anwender in Abhängigkeit von Parametern und Zuständen die Inhalte der übertragenen Informationen bestimmt werden. Ein Beispiel für Dynamik ist die Eingabemaske einer beliebigen Suchmaschine. Zunächst werden die entsprechenden Suchbegriffe eingegeben. Anschließend wird eine Interaktion ausgelöst, indem die Abfrage abschickt wird. Hat der entfernte Server die Formulardaten erhalten, erstellt er, basierend auf der Anfrage, eine Lösungsmenge und sendet diese Informationen zurück. Weitere Beispiele für benutzerdefinierte Inhalte sind personalisierte Webseiten, sowie zeitabhängige Informationen, wie das Wetter oder Börsenkurse.

Das Common Gateway Interface (CGI) war eine der ersten Techniken, mit der dynamische Inhalte erstellt werden konnten. Es kommt für kleinere Applikationen, die auf dem Server laufen und vom Client gesteuert werden, in Frage. Bei der Verwendung von CGI gibt ein Webserver Anfragen über eine Schnittstelle an ein externes Programm weiter, wobei die Ausgaben des Programms anschließend nicht in eine statische Datei geschrieben, sondern zurück an den Client gesendet werden. Der Nachteil von CGI ist, dass für jede Anfrage, die auf ein CGI-Programm zugreift, serverseitig ein neuer Prozess gestartet werden muss.

Um eine wesentlich neuere Technologie zur Erstellung dynamischer Inhalte handelt es sich bei Servlets. Servlets sind kleine Programme oder Anwendungen, die in der Programmiersprache Java geschrieben sind und im Gegensatz zu Java-Applets nicht auf einem Client, sondern auf einem Server ausgeführt werden. Der Grund dafür ist meistens, dass diese Applikationen auf serverseitige Datenbanken zurückgreifen. Im Gegensatz zu CGI, das mehrere Prozesse benötigt, um mit mehreren Anfragen umgehen zu können, können bei Servlets alle Anfragen durch separate Threads[9] innerhalb desselben Prozesses ausgeführt werden. Dadurch ist der Ablauf bei der Verwendung von Servlets wesentlich schneller als wenn die CGI-Schnittstelle verwendet wird.

Für die Verwendung von Servlets sprechen auch folgende Argumente:

? **Portabilität/Plattformunabhängigkeit**: Wie Java selbst sind Servlets unter dem Gesichtspunkt der Portabilität entwickelt worden. Sie sind auf allen Plattformen verwendbar, die Java unterstützen und arbeiten mit allen wichtigen Webservern zusammen. Dadurch besteht zum Beispiel die Möglichkeit,

---

[8] Umfassendes Referenzwerk zu Servlets: Java Servlet Programmierung von Jason Hunter

[9] Teil eines Programms, der unabhängig vom übrigen Programm ausgeführt werden kann. So kann beispielsweise bei einem Textverarbeitungsprogramm gedruckt werden, während der Anwender weiter am Dokument arbeitet.

Servlets unter Windows zu erstellen, sie später aber problemlos auf einem UNIX-Server zu betreiben. Servlets können also, wenn sie einmal geschrieben sind, überall eingesetzt werden.

? **Leistungsfähigkeit**: Bei der Verwendung von Servlets wird eine sehr hohe Leistungsfähigkeit erzielt, da das komplette Application Programmer Interface (API) von Java genutzt werden kann. Es besteht also ein sehr großer Bestand an fertigen Bibliotheken.

? **Effizienz**: Ein wichtiger Vorteil, der für Servlets und gegen externe CGI-Programme spricht, ist die gute Performance. Sie ergibt sich daraus, dass die Java Virtual Machine[10] (JVM) im Server integriert ist und daher keine externen Programme gestartet werden müssen. Dies ist, wie bereits erwähnt, bei der CGI-Lösung ein großes Problem, da jede Anfrage, die ein externes Programm aufruft, einen externen Prozess startet. Bei einer Anfrage am Tag, bei der die Antwortzeit nicht so wichtig ist, spielt das keine große Rolle, wollen jedoch viele Clients bedient werden, fällt die Antwortzeit ins Gewicht. Bei Servlets läuft permanent die JVM im Hintergrund. Dabei wird jede Verbindung durch ein Thread-Objekt gehandhabt, dessen Erzeugung und Speicherverbrauch wesentlich optimaler ist als die Verwendung externer Programme.

? **Datenaustausch und Kommunikation mit dem Web-Server**: Ein CGI-Programm kann mit anderen CGI-Programmen nur mühsam Daten austauschen. Jedes Programm läuft unabhängig von anderen und verwaltet eigene Zustände. Da Servlets aber in einem gemeinsamen Maschinen-Kontext laufen, können sie Daten teilen. Zwecks Optimierung lassen sich beispielsweise Datenbankverbindungen oder vorberechnete Daten gemeinsam nutzen. Diese jedes Mal neu zu erstellen würde sonst erst einige Sekunden in Anspruch nehmen.

? **Sicherheit**: Dank der Mechanismen von Java zum Exception-Handling können Servlets sicher mit Fehlern umgehen. Führt ein Servlet eine unzulässige Operation aus, so wird eine Exception ausgelöst, die durch den Server sicher abgefangen und behandelt werden kann.

? **Eleganz**: Servlets werden wie alle Java-Programme objektorientiert programmiert. Viele Routineaufgaben können mit Hilfe von Methoden und Klassen der Servlet-API erledigt werden.

? **Erweiterbarkeit und Flexibilität**: Die Servlet-API ist so angelegt, dass sie jederzeit erweitert werden kann.

Folgendes Listing zeigt ein Beispiel für ein einfaches Servlet:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HalloWelt extends HttpServlet
{
  public void doGet (HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException
  {
    response.setContentType ("text/html");
```

[10] Laufzeitumgebung für Java-Programme, führt den bereits kompilierten Java-Code (Byte-Code) aus

```
    String besucher = request.getParameter("name");
    if (besucher == null) besucher ="Welt";
    PrintWriter out = response.getWriter ();
    out.println ("<html>");
    out.println ("<body>");
    out.println ("Hallo "+besucher);
    out.println ("</body></html>");
    out.close ();
  }
  public void doPost (HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException
  {
    doGet(request, response);
  }
}
```

Die Klassen `HttpServletRequest` und `HttpServletResponse` liegen zusammen mit der Klasse `HttpServlet` im Paket `javax.servlet.http`. Das Paket `javax.servlet` beinhaltet die Klasse `ServletException`. Diese Klasse wird benötigt, da die Methode `getWriter()` eine Ausnahme auslösen kann, die aufgefangen werden muss. Im Paket `java.io` liegt die Klasse `Writer`. Da Methoden dieser Klassen im Servlet verwendet werden, müssen die entsprechenden Pakete wie folgt zu Beginn eingebunden werden:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Das Objekt `response` der Klasse `HttpServletResponse` stellt die Rückmeldung vom Servlet zum Client dar. Es ist für die Ausgabe zuständig, die das Servlet liefert und zum Client schickt. Über das Objekt `request` der Klasse `HttpServletRequest` lassen sich alle Anfrageparameter erfragen, es beinhaltet also die vom Browser gesendeten Informationen.

Mit

```
response.setContentType ("text/html");
```

wird dem Browser mitgeteilt, um welches Ausgabeformat es sich handelt, anschließend muss der Datenstrom noch speziell formatiert werden. Da etwas vom Servlet zum Browser geschickt werden soll, muss das Objekt `response` der Klasse `HttpServletResponse` verwendet werden, welche die Methode `setContentType()`enthält. Das Ausgabeformat wird dabei mit Hilfe des MIME-Types definiert, in diesem Fall `text/html`. Dem Browser wird somit also mitgeteilt, dass es sich bei der Ausgabe um HTML handelt. Der MIME-Type sollte immer gesetzt werden, wenn andere Typen als `text/plain`[11] gesendet werden.

[11] Der Standard-MIME-Type in den meisten Server-Installationen zur Kennzeichnung reiner Textdateien

Wird das Servlet über die URL http://localhost:8080/servlet/HalloWelt?name=Yvonne aufgerufen, so kann über

```
String besucher = request.getParameter("name");
```

der Inhalt des übergebenen Parameters `name`, in diesem Fall also Yvonne, erhalten werden. Da hier Informationen vom Browser benötigt werden, wird das Objekt `request` der Klasse `HttpServletRequest` verwendet, welche die Methode `getParameter()` enthält. Der Wert des Parameter `name` wird der Variablen `besucher` zugewiesen. Dadurch wird die Variable nicht mehr mit dem Standardwert Welt belegt, sondern verwendet den übergebenen Namen:

```
if (besucher == null) besucher ="Welt";
```

Anschließend wird durch den Rückgabewert der Methode `getWriter()`, die über das Objekt `response` aufgerufen wird, eine Referenz auf ein Writer-Objekt erhalten.

```
PrintWriter out = response.getWriter ();
```

Durch die Anweisungen

```
out.println(...);
```

werden die HTML-Tags in den Ausgabestrom `out` geschrieben und als Antwort auf die Anforderung an den Webbrowser übermittelt.

Als Ergebnis erhält der Benutzer dann je nach eingegebener URL folgende Ergebnisse:

Abbildung 3: Servlet ohne übergebenen Parameter

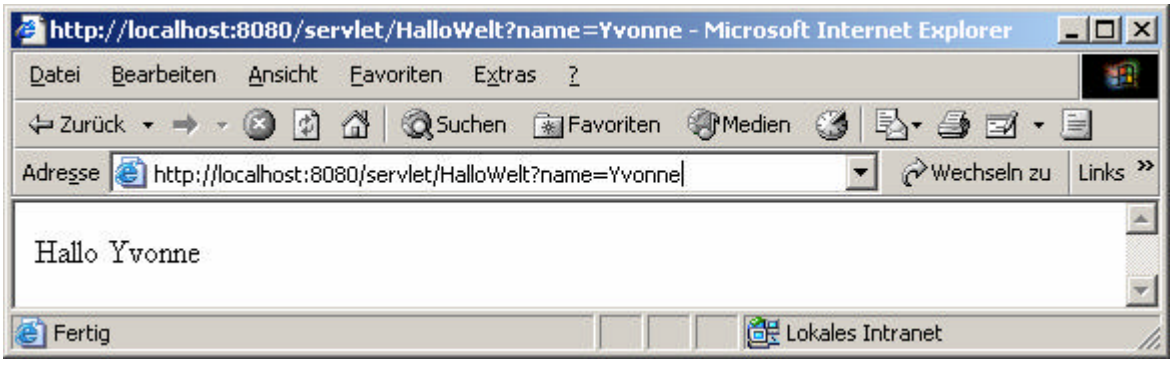

Abbildung 4: Servlet mit übergebenem Parameter

Der HTML-Code, der dynamisch erzeugt wird, sieht im ersten Fall ohne übergebenen Parameter wie folgt aus:

```
<html>
<body>
  Hallo Welt
</body>
</html>
```

Im zweiten Fall mit übergebenen Parameter dagegen so:

```
<html>
<body>
  Hallo Yvonne
</body>
</html>
```

2.2 JSP - JavaServer Pages

Servlets sind Server-Programme, die Web-Seiten erstellen, indem sie die HTML-Anweisungen mit `println()` oder ähnlichem in den Ausgabestrom senden. Aus diesem Grund fällt die gewünschte Trennung zwischen Informationen und Visualisierung schwer. Ändert sich das Erscheinungsbild, so muss das Programm umgebaut werden. In vielen dynamischen Programmen stecken oft nur ein oder zwei Zeilen Dynamik, der Rest ist statischer HTML-Code. In der Regel ist der Programmierer nicht gleichzeitig auch der Designer und dieser möchte mit HTML-Editoren wie DreamWeaver oder Microsoft FrontPage arbeiten, ohne dabei selbst Programmcode zu schreiben.

Beim Konzept der JavaServer Pages[12] (JSP) wird deshalb anders verfahren. Während bei Servlets Java-Klassen für die Ausgabe des HTML-Codes zuständig sind, ist eine JSP-Seite eine HTML-Seite mit Java-Code, ähnlich wie bei JavaScript. Nun kann der Designer die Visualisierung der Informationen noch nachträglich anpassen. Das ist sehr wichtig, da so eine Trennung zwischen Informationen und Visualisierung erfolgt.

JSP-Skripte werden vom Server automatisch in Servlets übersetzt. Der Server kann JSP-Seiten von normalen HTML-Seiten unterscheiden und kompiliert mit Hilfe eines JSP-Compilers[13] daraus ein Servlet und stellt es dar. Der Übersetzungsvorgang von einer JSP-Seite in ein Servlet muss nur einmal getätigt werden, danach benutzt der Servlet-Container[14] direkt die übersetzte Klasse.

Neben dem Inhalt der HTML-Datei lassen sich drei unterschiedliche JSP-Konstrukte in eine Seite einbinden:

1. **Skript-Elemente**:
   Sie enthalten Java-Programmcode, der direkt in das Servlet wandert. Es gibt unterschiedliche Typen für Vereinbarungen, Ausdrücke und Anweisungsfragmente (Skriplets)

---

[12] Einführung in JSP: JavaServer Pages für DUMMIES von Sahin Türeyenler

[13] Übersetzt JSP-Seiten in Servlets

- **Vereinbarungen**: Sie werden eingesetzt, um Java-Variablen und Methoden in einer JSP-Seite zu deklarieren.

  Beispiel: Vereinbarung einer globalen Variable
  ```
  <%! int globalerZaehler = 0; %>
  ```

- **Ausdrücke**: Sie geben den Wert des Ausdrucks in das Ausgabedokument zurück. Dabei kann auf bekannte Objekte und ihre Methoden zurückgegriffen werden. Dabei ist allerdings zu berücksichtigen, dass ein String als Rückgabewert geliefert wird. Wichtig ist, dass hinter dem Ausdruck kein Semikolon stehen darf.

  Beispiel: Ausgabe des aktuellen Datums
  ```
  <%= new java.util.Date() %>
  ```

- **Anweisungen**: Zwischen ihnen kann beliebiger Java-Quellcode eingebettet werden.

  Beispiel: eine if-Abfrage
  ```
  <% if (uhr.getHours() < 12) %>
  ```

- **Kommentare**: In einer JSP-Seite können zwei Arten von Kommentaren eingesetzt werden. Einmal Kommentare, die nicht in HTML-Text umgewandelt werden und folglich auch nicht zum Browser wandern. Sie haben das folgende Format:

  ```
  <%-- Kommentar --%>
  ```

  Der zweite Typ Kommentar ist ein HTML-Kommentar, der auf der Client-Seite sichtbar ist. Er hat folgendes Format:

  ```
  <!-- Kommentar -->
  ```

2. **Direktiven**:
   Direktiven steuern die Struktur und die Verarbeitung der Seite.

   Beispiel: Einbindung einer externen Datei
   ```
   <%@ include file=”einzubindendeDatei.jsp” %>
   ```

3. **Aktionen**:
   Es handelt sich um das Nutzen von vorgefertigten Komponenten wie Beans[15], Einbinden von externen Seiten und Weiterleitung an andere Seiten.

   Beispiel: Weiterleitung an eine andere Seite
   ```
   <jsp:forward page =“zielseite“>
   ```

---

[14] Programm zum Starten von Servlets

[15] Ein auf Java basierendes Komponentenmodell zur Softwareentwicklung, d.h. eine Architektur für die Definition und Wiederverwendung von Softwarekomponenten. Einzelne Beans dienen als Basisbausteine aus denen durch Zusammensetzen größere Softwareteile erstellt werden.

Die Vorteile, die im Kapitel über Servlets aufgeführt wurden, gelten natürlich auch für JSP-Seiten, da diese ja in Servlets umgesetzt werden. In der folgenden Tabelle wird JSP mit alternativen Skriptsprachen wie PHP und ASP verglichen.

Tabelle 1: Vergleich von JSP mit anderen Skriptsprachen

	PHP	ASP	JSP
Plattformunabhängigkeit	+	-	+
Skalierbarkeit	-	-	++
Sicherheit	-	-	+
Anfälligkeit des Servers für Viren	gering bei Unix-Systemen	Hoch, weil immer Windows	gering bei Unix-Systemen
Wiederverwendbare, plattformunabhängige Komponenten	bedingt	-	++
Sprachumfang	-	+	++
Umfang der API	-	+	++
Benutzerdefinierte Tags	-	-	++
Separation Präsentation/Logik	-	+	++
Lizenzfreie Entwicklung	+	-	+
Leistungsfähigkeit	+	+	+
Erlernbarkeit	+	+	-

Zum Abschluss dieses Kapitels folgt nun ein vollständiges JSP-Beispiel:

```
<html>
<body>
<h1>JSP-Beispiel:</h1>
<%! java.util.Date uhr = new java.util.Date(); %>
<p>
<% if (uhr.getHours() < 12) { %>
<!-- es ist vor 12 Uhr -->
Einen wundersch&ouml;nen Guten Morgen!<br>
<% } else if (uhr.getHours() < 17) { %>
<!-- es ist zwischen 12 und 17 Uhr -->
Einen erfolgreichen Tag!<br>
<%}else {%>
```

```
<!-- es ist nach 17 Uhr -->
Einen erholsamen Abend!<br>
<%}%>
</p>
<br>
</body>
</html>
```

Abhängig von der Uhrzeit zu der die JSP-Seite ausgeführt wird, wird eines der folgenden Ergebnisse erhalten:

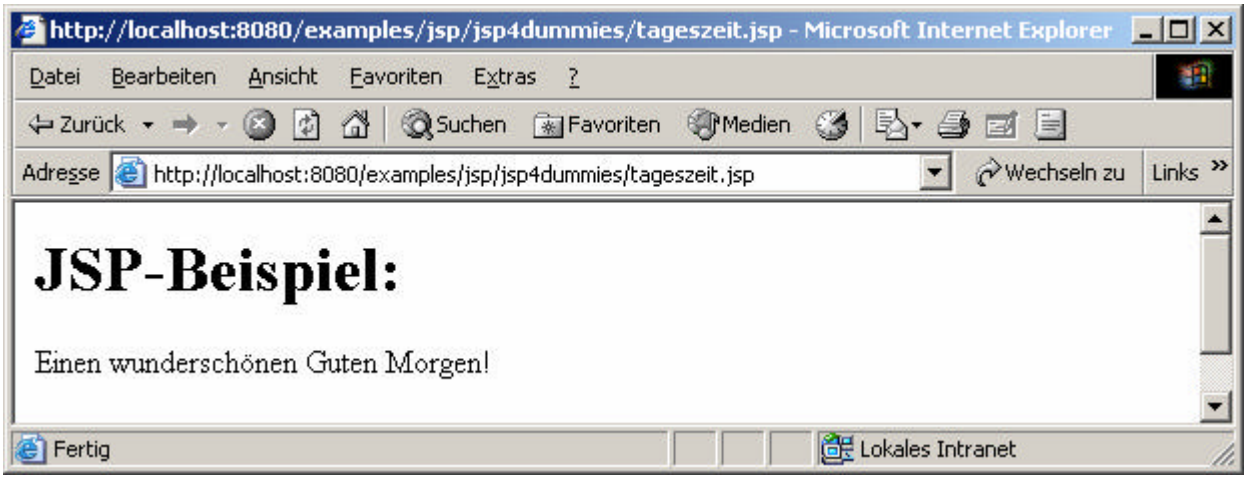

Abbildung 5: JSP, Ergebnis bei Aufruf vor 12 Uhr

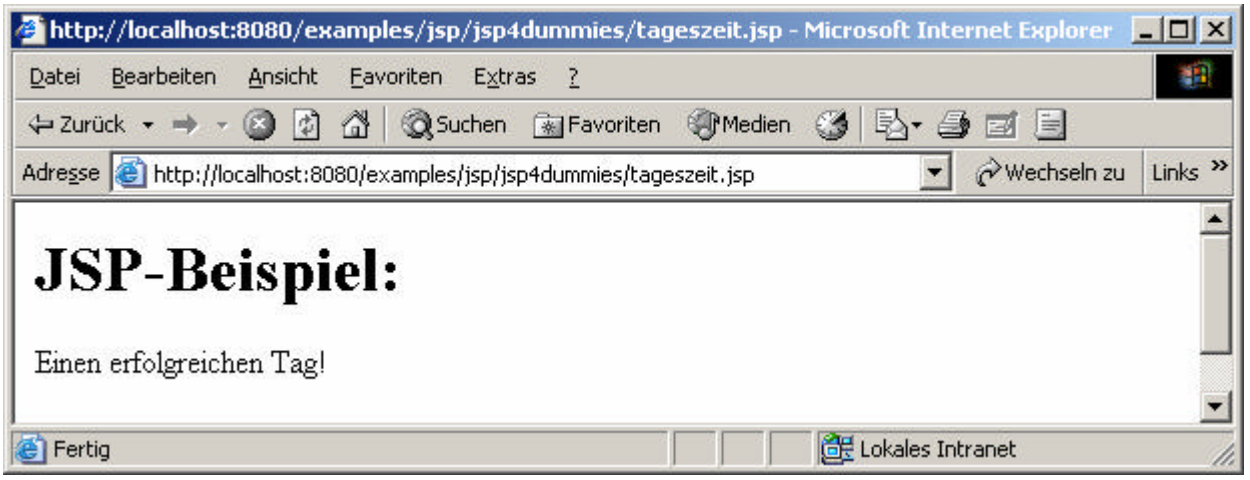

Abbildung 6: JSP, Ergebnis bei Aufruf zwischen 12 und 17 Uhr

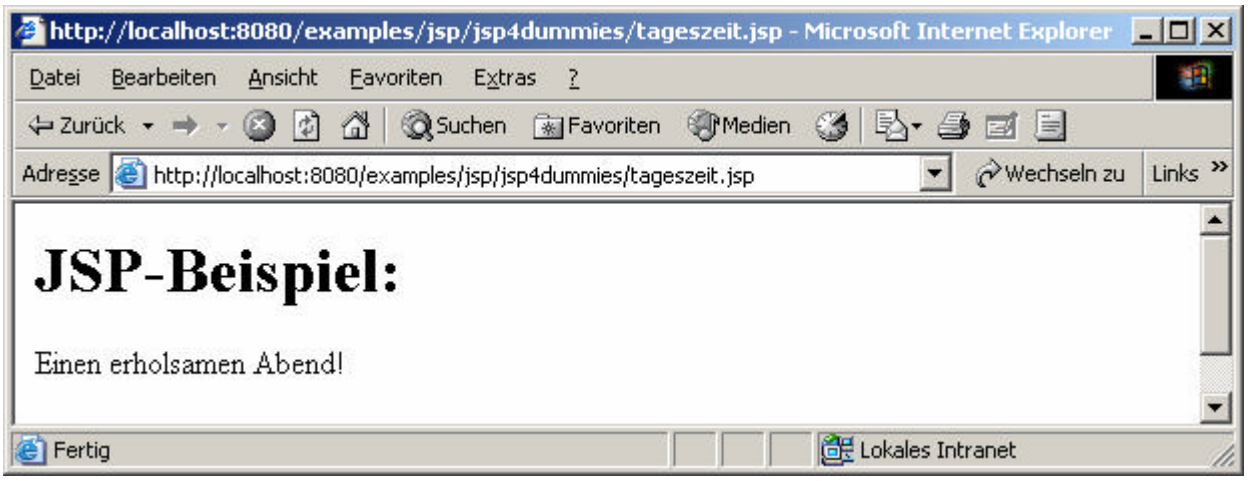

Abbildung 7: JSP, Ergebnis bei Aufruf nach 17 Uhr

2.3 Tomcat

Alle Java-Anwendungen, egal ob es sich hierbei um Applikationen, Applets oder Servlets handelt, müssen als kompilierte Dateien zur Ausführung bereitstehen, da die JVM nur solche verarbeiten kann. Bei den dynamischen Skript-Elementen von JSP handelt es sich ebenfalls um Java; daher gilt auch hier, dass eine Ausführung nur dann stattfindet, wenn eine kompilierte Version der angeforderten Datei vorliegt. Genau wie bei Servlets reicht somit bei JSP eine bereits installierte JVM nicht aus, um die Ausführung der Datei zu gewährleisten. Es wird eine JSP-Engine benötigt. Diese besteht aus einem JSP-Compiler und einem Servlet-Container. Der JSP-Compiler hat die Aufgabe, noch nicht kompilierte, angefragte JSP-Seiten in Servlets zu übersetzen. Der Servlet-Container hingegen ist für die Initialisierung, den Aufruf und die Beendung eines Servlets verantwortlich, d.h. er verwaltet den gesamten Lebenszyklus eines Servlets. Der Container führt alle Servlets innerhalb der gleichen JVM aus, daher können sie auf gemeinsam genutzte Daten zugreifen, während die JVM private Daten voreinander effektiv schützt. Statt von einer JSP-Engine wird oft auch von einem JSP-Container gesprochen. Beide Begriffe bedeuten jedoch das gleiche.

Der Konvertierungsvorgang lässt sich in folgende Schritte einteilen:

1. Der Webserver übergibt eine Anforderung aufgrund der Endung .jsp an den JSP-Container
2. Die angeforderte JSP-Seite wird „geparst“, d.h. HTML- und Java-Teil werden getrennt
3. Ein entsprechendes Servlet wird im Quelltext erstellt
4. Dieses wird anschließend kompiliert (.java nach .class) und ist damit als Bytecode vorhanden
5. Erstellter Bytecode wird durch JVM geladen und ausgeführt
6. Das Ergebnis wird als HTML-Seite an den aufrufenden Webbrowser geschickt

Die offiziell von SUN unterstützte Referenzimplementation eines JSP-Containers wurde von der Apache Software Foundation (ASF) innerhalb des Jakarta-Projektes[16] entwickelt und Ende 1999 erstmalig unter dem Namen Tomcat vorgestellt. Bestehend aus einem Webserver und einem JSP-Container können mit ihm sowohl statische, als auch dynamische Inhalte dargestellt werden.

Vorraussetzung für die Installation von Tomcat ist ein bereits installiertes Java Development Kit (JDK), es ist unter http://java.sun.com/ als kostenloser Download erhältlich. Das JDK stellt die Basis aller Java-Anwendungen dar. Da später Java-Programme ausführt und kompiliert werden sollen, wird sowohl die Laufzeitumgebung, als auch der Java-Compiler benötigt.

Anschließend kann mit der Installation von Tomcat begonnen werden. Bei der vorliegenden Diplomarbeit wurde die Version 4.0.2 verwendet, welche die Servlet-Spezifikation 2.3 und die JSP-Spezifikation 1.2 erfüllt. Die jeweils aktuelle Version ist bei http://jakarta.apache.org/ zu finden. Dort steht wahlweise der komplette Quellcode oder eine bereits vorkompilierte Binary-Version kostenlos zum Download zur Verfügung. Um Zeit zu sparen sei hier die Binary-Version empfohlen.

1. Zuerst muss ein Doppelklick auf die Datei jakarta-tomcat-402.exe (oder die entsprechende Version) erfolgen, anschließend führt der Installationsassistent durch die Installation. Es besteht nur an zwei Punkten die Möglichkeit Einfluss auf die Installation zu nehmen.
2. Der erste Punkt ist erreicht, wenn die Aufforderung erfolgt die Installationsoptionen auszuwählen. Die Auswahl sollte wie in folgender Abbildung dargestellt getroffen werden.

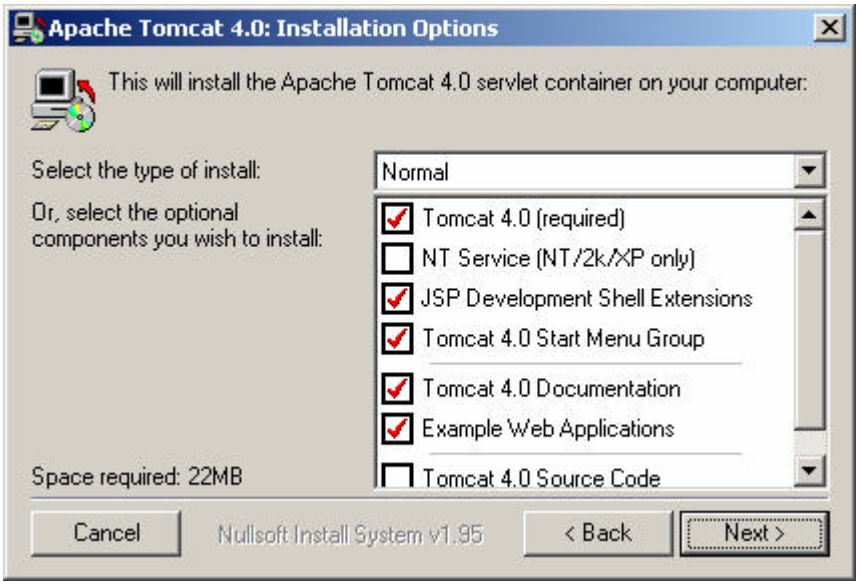

Abbildung 8: Optionen bei der Installation von Tomcat

3. Der zweite Punkt ist die Stelle, an der das Installationsverzeichnis des JSP-Containers festgelegt werden kann. Es ist z.B. C:\entwicklung\Tomcat4.0.2 zu wählen.
4. Nachdem die Installation beendet ist, wurden auf dem Computer innerhalb des Tomcat-Ordners folgende Verzeichnisse angelegt:

[16] Informationen unter: http://www.jakarta.apache.org

- bin Skripte zum Starten und Stoppen des Servers
- conf Wichtige Konfigurationsdateien
- doc Dokumentation zur Installation und zum Start des Servers
- lib Enthält Module zur Verbindung mit anderen Webservern
- logs enthält die log-Dateien des Servers
- src enthält den Quellcode aller Servlet und JSP-Klassen
- webapps Standardverzeichnis für alle Webapplikationen
- work enthält dynamisch erzeugte .java- und .class-Dateien

Tomcat kann nun gestartet werden, indem über das Startmenü Start/Programme/Apache Tomcat 4.0/Start Tomcat aufrufen wird. Es erscheint ein DOS-Fenster, das den erfolgreichen Start anzeigt:

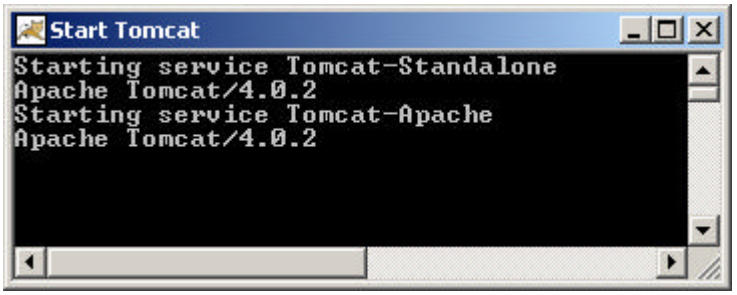

Abbildung 9: DOS-Fenster, das den erfolgreichen Start von Tomcat anzeigt

Wird im Webbrowser als URL entweder http://localhost:8080 oder die lokale IP http://127.0.0.1:8080 eingegeben und die Anfrage ausgeführt, so wird als Ergebnis die Tomcat-Startseite mit einigen Beispielen zu JSP und Servlets erhalten, deren korrekte Ausführung ein Beweis für eine erfolgreiche Installation ist.

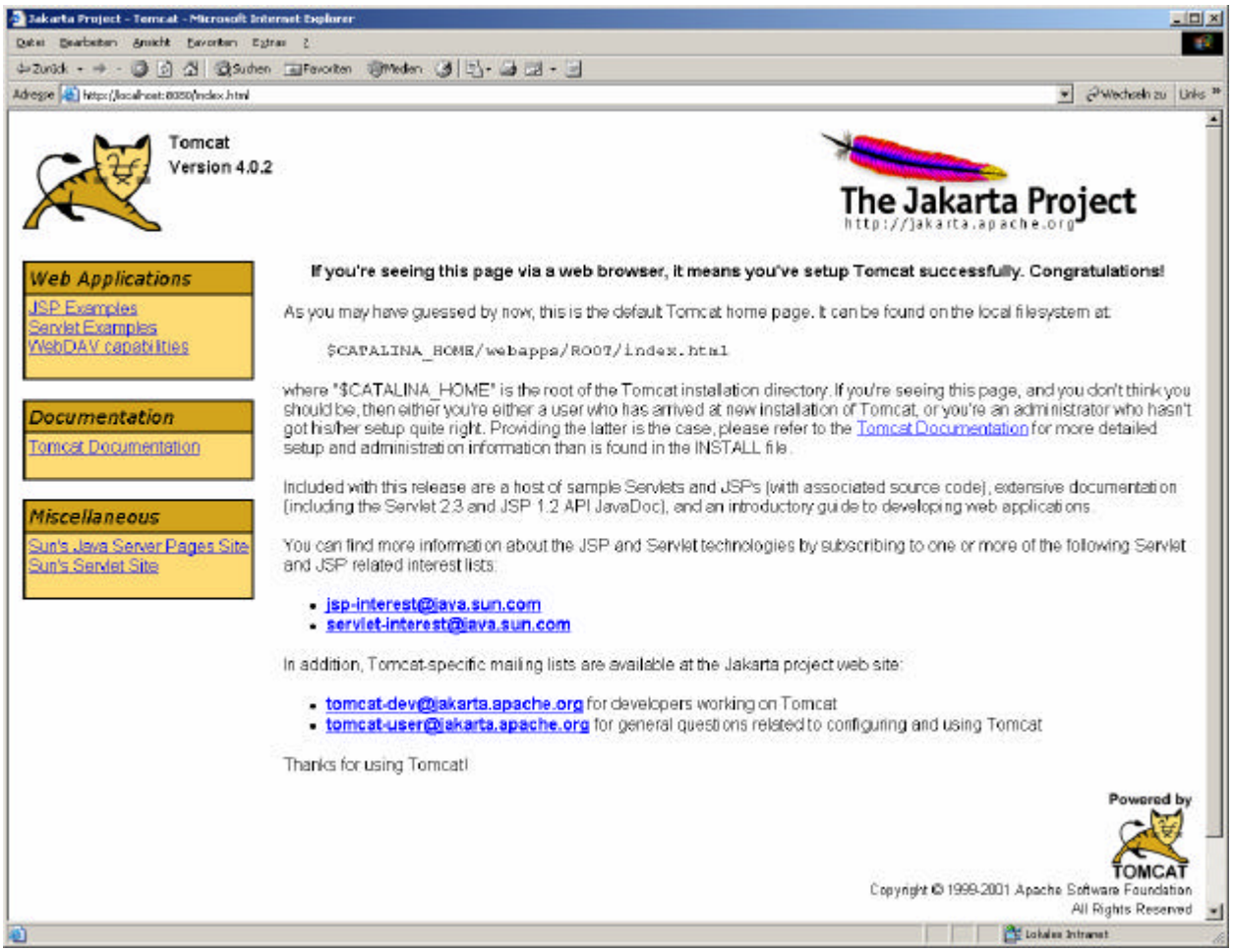

Abbildung 10: Tomcat-Startseite

Um Tomcat anzuhalten wird über das Startmenü Start/Programme/Apache Tomcat 4.0/Stop Tomcat aufgerufen. Das DOS-Fenster, das sich beim Starten des Servers geöffnet hat, wird geschlossen, und damit wird auch die JSP/Servlet Unterstützung beendet.

Es besteht zusätzlich die Möglichkeit Tomcat durch eine spezielle Adaptersoftware mit dem Webserver Apache zu kombinieren. Dies ist sinnvoll, wenn viele statische Seiten verwendet werden, da Tomcat statische Seiten nicht mit der gleichen Geschwindigkeit wie Apache anzeigen kann. Apache kümmert sich dann um die statischen Anfragen und leitet die dynamischen an Tomcat weiter.

Zusätzlich zum Tomcat wird noch ein Text-Editor wie TextPad oder Ultraedit[17] benötigt, mit dem Quelltexte erstellt und editiert werden können. Selbstverständlich muss auch ein Webbrowser, wie z.B. Microsoft Internet Explorer oder Netscape Navigator, vorhanden sein, mit dem die Ergebnisse anzeigt werden können.

[17] http://www.ultraedit.com

2.4 HTML - Hypertext Markup Language

Die Hypertext Markup Language[18] (HTML) wurde von Tim Berners-Lee, dem Begründer des World Wide Web, entwickelt und im Zuge des Web-Booms zum erfolgreichsten und verbreitetsten Dateiformat der Welt. Mit HTML können Texte strukturiert werden, wobei aber auch die Möglichkeit besteht, Grafiken und multimediale Inhalte in Form einer Referenz einzubinden und in den Text zu integrieren. Überschriften, Textabsätze, Listen und Tabellen können erstellt, Verweise auf beliebige andere Web-Seiten oder Datenquellen im Internet erzeugt werden. Auch Formulare können in den Text integriert werden. Weiterhin werden Schnittstellen für Erweiterungssprachen wie Cascading Style Sheets[19] (CSS) oder JavaScript angeboten, mit deren Hilfe HTML-Elemente nach Wunsch gestaltet und formatiert werden können bzw. eine Interaktion mit dem Anwender realisiert werden kann. Das World Wide Web Consortium[20] (W3C) das für die Standardisierung von HTML zuständig ist, ist bemüht, HTML als einfache, reine Text-Strukturierungssprache zu etablieren. In der Praxis dient HTML heute aber oftmals auch als Basis für das Erstellen von Web-Seiten-Layouts.

HTML ist ein so genanntes Klartext-Format, d.h. HTML-Dateien bestehen ausschließlich aus ASCII[21]-Zeichen. Deshalb können sie mit jedem beliebigen Texteditor bearbeitet werden, der Daten als reine Textdateien abspeichern kann. Daneben gibt es eine Vielzahl mächtiger Programme, die auf das Editieren von HTML spezialisiert sind, doch das ändert nichts an der entscheidenden Eigenschaft, dass HTML nicht an irgendein bestimmtes, kommerzielles Software-Produkt gebunden ist. Da HTML ein Klartextformat ist, lässt es sich auch hervorragend mit Hilfe von Programmen generieren. Serverseitig können hierfür beispielsweise JSP, PHP oder CGI verwendet werden.

Der Inhalt von HTML-Dateien steht in HTML-Elementen, diese werden durch so genannte Tags markiert. Fast alle HTML-Elemente werden durch ein einleitendes und ein abschließendes Tag eingeschlossen. Der Inhalt dazwischen ist der Gültigkeitsbereich des entsprechenden Elements. Tags werden in spitzen Klammern notiert.

```
<h1>Die ist Kapitel 1</h1>
```

Das Beispiel zeigt eine Überschrift. Während das einleitende Tag `<h1>` signalisiert, dass eine Überschrift 1. Ordnung folgt, markiert das abschließende Tag `</h1>` das Ende der Überschrift. Im Gegensatz zum einleitenden Tag beginnt ein abschließendes Tag mit einem Schrägstrich `"/"`.

Es gibt auch einige Elemente, die keinen Inhalt haben und deshalb nur aus einem Tag bestehen, welches das Endtag bereits mit einschließt.

```
Es folgt ein Zeilenumbruch<br>
Neue Zeile
```

---

[18] Einführung unter: http://selfhtml.teamone.de/html/index.htm
Spezifikation unter: http://www.w3.org/TR/html4/

[19] Siehe Kapitel 2.5

[20] Gremium zur Standardisierung das Internet betreffender Techniken, wie z.B. HTML, XML und CSS. Beschäftigen sich auch mit der Weiterentwicklung von Standards oder mit der Entwicklung neuer Standards.

[21] American Standard Code for Infomation Interchange
Dateien, die im ASCII-Format erstellt wurden, enthalten keinerlei Formatierung, lassen sich aber von jedem Computer lesen und darstellen.

Am Ende der ersten Zeile wird durch das Tag `<br>`, ein manueller Zeilenumbruch eingefügt.

Zu Beginn einer HTML-Datei steht die Dokumenttyp-Angabe. Hier wird die verwendete HTML-Version angegeben, damit sich eine auslesende Software, etwa ein Web-Browser, an dieser Angabe orientieren kann. Hinter jeder Dokumenttyp-Angabe stecken so genannte Dokumenttyp-Definitionen (DTD). Dort ist geregelt, welche Elemente ein Dokument vom Typ HTML enthalten darf, welche Attribute zu einem Element gehören, und ob die Angabe dieser Attribute Pflicht ist oder freiwillig usw.. Eine Dokumenttyp-Angabe sieht wie folgt aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
```

Hinter der startenden spitzen Klammer folgt ein Ausrufezeichen, dahinter die Angabe `DOCTYPE HTML PUBLIC`, was bedeutet, dass Bezug auf die öffentlich verfügbare HTML-DTD genommen wird. Die folgende Angabe, die in Anführungszeichen steht, bezeichnet den Namen der DTD. Das W3C ist `der` Herausgeber der DTD. Die `Angabe DTD HTML 4.01 Transitional` bedeutet, dass in der folgenden Datei der Dokumenttyp HTML in der Sprachversion 4.01 und deren Variante Transitional verwendet wird. `EN` ist ein Sprachenkürzel und steht für Englisch. Die Angabe bezieht sich auf die Sprache in der die Element- und Attributnamen der Tagsprache definiert wurden und nicht auf den Inhalt der Datei. Da die Namen von HTML-Elementen und -Attributen auf der englischen Sprache basieren, muss also immer `EN` verwendet werden,. Über die darauf folgend angegebene Web-Adresse kann eine auslesende Software die DTD aufrufen.

Der gesamte übrige Inhalt der Datei wird in die Tags `<html>` bzw. `</html>` eingeschlossen. Das `html`-Element wird auch als Wurzelelement einer HTML-Datei bezeichnet. Hinter dem einleitenden HTML-Tag folgt das einleitende Tag für den Kopf `<head>`. Zwischen diesem Tag und seinem Gegenstück `</head>` werden die Kopfdaten der HTML-Datei notiert. Die wichtigste dieser Angaben ist der Titel der Datei, der durch die Tags `<title>` bzw. `</title>` eingeschlossen wird. Unterhalb davon folgt der Textkörper, markiert durch `<body>` bzw. `</body>`. Dazwischen wird dann der eigentliche Inhalt der Datei notiert, der im Browser angezeigt werden soll, also Text mit Überschriften, Verweisen, Grafikreferenzen usw. .

Für den Aufbau einer kompletten HTML-Datei ergibt sich also folgender Aufbau:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Titel der Datei</title>
  </head>
  <body>
    <h1>Dies ist Kapitel 1</h1>
    Es folgt ein Zeilenumbruch<br>
    Neue Zeile
  </body>
</html>
```

Das Ergebnis sieht im Internet Explorer wie folgt aus:

Abbildung 11: HTML-Dokument

2.5 CSS - Cascading Style Sheets

Bei Cascading Style Sheets[22] (CSS) handelt es sich um eine HTML-Ergänzungssprache, mit der HTML-Elemente exakt formatiert werden können. Formate können zentral für eine HTML-Datei angegeben werden. Dafür wird innerhalb einer HTML-Datei ein Bereich für CSS-Formate definiert:

```
<html>
  <head>
    <title>Titel der Datei</title>
    <style type="text/css">
    <!--
      /* Hier werden die Formate definiert */
      h1 {color:red; font-weight:bold;}
    -->
    </style>
  </head>
  <body>
  </body>
</html>
```

Durch `<style...>` ... `</style>` wird der Bereich für Stylesheet-Formatdefinitionen festgelegt. Im einleitenden `<style>`-Tag muss der MIME-Type der Stylesheet-Sprache angeben werden. Für CSS ist das die Angabe `type="text/css"`. Zwischen dem einleitenden Tag und dem abschließenden `</style>` können zentrale CSS-Formate definiert werden.

[22] Einführung unter: http://selfhtml.teamone.de/css/index.htm
Spezifikation unter: http://www.w3.org/TR/REC-CSS2/

Formate können auch zentral in einer separaten CSS-Datei definiert werden, die im HTML-Dokument wie folgt referenziert wird:

```
<html>
  <head>
    <title>Titel der Datei</title>
    <link rel="stylesheet" type="text/css" href="formate.css">
  </head>
  <body>
  </body>
</html>
```

Mit `<link...>` wird die CSS-Datei referenziert, welche die CSS-Formatdefinitionen enthält. Innerhalb des `<link>`-Tags müssen die Angaben `rel="stylesheet" type="text/css"` stehen. Beim Attribut `href=` wird die gewünschte Datei angegeben. Bei der referenzierten Datei muss es sich um eine reine Textdatei handeln, welche die Endung `.css` haben sollte. Die Datei darf nichts anderes als Definitionen zentraler Formate und Kommentare enthalten.

Der Vorteil bei der Verwendung einer separaten CSS-Datei liegt darin, dass einheitliche Formate für verschiedene HTML-Dateien nicht in jeder Datei wiederholt aufgeführt werden müssen. Stattdessen können die Formate in einer separaten Textdatei definiert und später in jeder gewünschten HTML-Datei eingebunden werden. Wenn Angaben in der separaten Datei geändert werden, wirken sich die Änderungen einheitlich auf alle Dateien aus, in denen die separate CSS-Datei eingebunden ist.

Unabhängig davon, ob Formate zentral definiert oder ob eine separate CSS-Datei referenziert wird, können auch einzelne HTML-Elemente formatiert werden:

```
<html>
  <head>
    <title>Titel der Datei</title>
  </head>
  <body>
    <h1 style="color:red;font-weight:bold">...</h1>
  </body>
</html>
```

Durch das Attribut `style` im einleitenden Tag eines Elements können CSS-Formate für dieses eine Element notiert werden.

Die Definitionen in den aufgeführten Beispielen haben zur Folge, dass eine Überschrift der 1. Ordnung nun fett und in rot geschrieben wird.

2.6 JavaScript

Durch JavaScript[23] können Web-Seiten, die mit Hilfe von HTML strukturiert und mit CSS gestaltet sind, dahingehend erweitert werden, dass sie auf dynamische Interaktionen durch den Benutzer reagieren können. Zum Beispiel erlaubt JavaScript es, Maus- und Tastatureingaben des Anwenders zu verarbeiten und darauf mit Bildschirmausgaben oder dynamischen Änderungen innerhalb der angezeigten Web-Seite zu reagieren.

Beispiel: In HTML können unter anderem Formulare definiert werden. Solche Formulare können Eingabefelder, Auswahllisten, Buttons usw. enthalten. Der Anwender kann ein Formular ausfüllen und über das Web absenden. Doch HTML erlaubt z.B. nicht, die Eingaben des Anwenders vor dem Absenden des Formulars auf Vollständigkeit und Plausibilität zu prüfen. Mit JavaScript hingegen ist dies möglich.

All diese Dinge laufen im Web-Browser des Anwenders ab, während eine Web-Seite am Bildschirm angezeigt wird. Es ist keine zusätzliche Kommunikation zwischen Web-Browser und Web-Server erforderlich. Aus Sicherheitsgründen ist JavaScript in seinen Möglichkeiten stark beschränkt. Nicht möglich ist es beispielsweise, eine Datei auf dem Rechner des Anwenders auszulesen oder diese zu löschen, da die Möglichkeiten von JavaScript auf das Umfeld der Web-Seite eingeschränkt sind, in die das jeweilige Script eingebettet ist.

Ebenso wie CSS können JavaScripts direkt innerhalb von HTML-Dateien geschrieben oder wahlweise als separate Datei eingebunden werden. Damit bezieht sich ein JavaScript-Code auf die Seite, in die er eingebunden ist und kann auf das unmittelbare Umfeld und die Elemente dieser Seite zugreifen.

Das folgende Listing zeigt die Verwendung von JavaScript innerhalb einer HTML-Datei:

```
<html>
  <head>
    <title>JavaScript innerhalb einer HTML-Datei</title>
  </head>
  <body>
    <h1>JavaScript innerhalb einer HTML-Datei</h1>
    <script type=”text/javascript”>
    <!--
      document.write("Hallo Welt");
    //-->
    </script>
  </body>
</html>
```

Innerhalb einer HTML-Datei muss JavaScript immer in einem HTML-Kommentars stehen, also zwischen `<!--` und `//-->`.

[23] Einführung unter: http://selfhtml.teamone.de/javascript/index.htm

Das Ergebnis zeigt Abbildung 12:

Abbildung 12: JavaScript innerhalb einer HTML-Datei

Ebenfalls ist die Einbindung einer externen JavaScript-Datei möglich, welche die Endung .js tragen muss. Die Einbindung in eine HTML-Datei wird auf folgende Weise erzielt:

```
<html>
  <head>
    <title>Einbindung einer externen JavaScript-Datei</title>
    <script src="JavaScript.js" type="text/javascript"></script>
  </head>
  <body>
    <h1> Einbindung einer externen JavaScript-Datei </h1>
  </body>
</html>
```

In der Datei JavaScript.js könnte folgendes stehen:

```
document.write("Hallo Welt")
```

Das Ergebnis nach dem Aufruf der HTML-Datei im Browser, ist in Abbildung 13 dargestellt.

Abbildung 13: Einbindung einer externen JavaScript-Datei

Die Ausgabe „Hallo Welt“ steht jetzt ganz am Anfang, da die Java-Script-Datei bereits im Head-Bereich der HTML-Datei eingebunden wurde. Diese Position sollte also dann ausgewählt werden, wenn Funktionen schon beim Einlesen des Body-Teiles zur Verfügung stehen sollen.

Zusammen mit den Elementen eines Formulars ermöglicht JavaScript die Gestaltung von interaktiven Webseiten.

Es folgt einfaches Beispiel für eine interaktive Webseite:

```
<html>
  <head>
    <title>JavaScript und Formulare</title>
  </head>
  <body>
    <form name="meinFormular">
      <input type="text" name="meinTextfeld" value="" size=30><p>
      <input type="button" name="Taste" value="Bitte anklicken!"
      onclick="var vorname=prompt('Bitte geben Sie ihren Vornamen
      ein!');
      meinFormular.meinTextfeld.value='Guten Morgen, ' +vorname">
    </form>
  </body>
</html>
```

Die HTML-Seite enthält ein Fomular `<form name="meinFormular">...</form>` mit dem Namensattribut '`meinFormular`". Es besteht aus zwei Elementen, einem Knopf und einem Textfeld. Drückt man den Knopf (`INPUT TYPE="button"`), ruft der Eventhandler `onclick` ein JavaScript auf:

```
"var vorname=prompt('Bitte geben Sie ihren Vornamen ein!');
 meinFormular.meinTextfeld.value='Guten Morgen, ' +vorname"
```

Die vordefinierte Funktion `prompt('Bitte geben Sie ihren Vornamen ein!')` erzeugt ein Dialogfenster, das den übergebenen Text und ein Textfeld zur Eingabe durch den Benutzer enthält. Wählt man im Dialogfenster den Knopf "OK", wird der eingegebene Text der Variablen '`vorname`" übergeben und im Textfeld des Formulars angezeigt.

Das Ergebnis ist in den Abbildungen 14 bis 16 zu sehen.

Abbildung 14: JavaScript mit Formularen, Teil 1

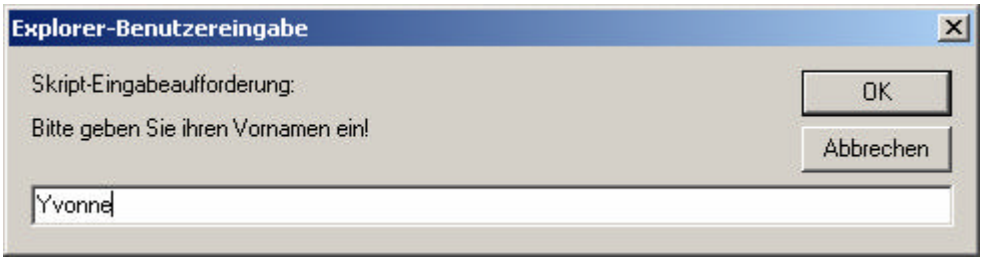

Abbildung 15: JavaScript mit Formularen, Teil 2

Abbildung 16: JavaScript mit Formularen, Teil 3

2.7 XML - Extensible Markup Language

Die Extensible Markup Language[24] (XML) ist eine Auszeichnungssprache zur Erstellung strukturierter Dokumente. Die Sprache wurde von einer Arbeitsgruppe des W3C entwickelt und 1998 standardisiert. XML ähnelt HTML dahingehend, dass beide Auszeichnungssprachen Tags enthalten, die den Inhalt des Dokumentes beschreiben. Im Gegensatz zu HTML, das festlegt wie der Inhalt einer Web-Seite dargestellt wird und welche Interaktionsmöglichkeiten der Benutzer hat, beschreibt XML die Bedeutung des Inhalts. So könnte ein Tag `<koordinate>` darauf hinweisen, dass die folgenden Daten eine Koordinate bezeichnen. XML wird als erweiterbar (extensible) bezeichnet, weil die Möglichkeit besteht, eigene Auszeichnungstags zu erstellen.

Eine XML-Datei besteht aus einem Prolog und den eigentlichen Daten:

```
<?xml version="1.0"?>
<koordinatenliste>
  <koordinate>
    <punktnummer datum="15.11.78">1</punktnummer>
    <rechtswert>3500000</rechtswert>
    <hochwert>4500000</hochwert>
  </koordinate>
  <koordinate>
```

[24] Einführung unter: http://selfhtml.teamone.de/xml/index.htm
Spezifikation unter: http://www.w3.org/TR/REC-xml

```
    <punktnummer>2</punktnummer>
    <rechtswert>3600000</rechtswert>
    <hochwert>4600000</hochwert>
  </koordinate>
</koordinatenliste>
```

Durch die erste Zeile, die den Prolog enthält, wird die Datei als XML-Datei gekennzeichnet. Daher erkennt der XML-Parser[25], dass der folgende Teil von ihm verarbeitet werden soll. Des weiteren wird die Version angegeben. Anschließend folgen die Tags durch die der Inhalt der Daten beschrieben wird.

2.8 XSL - Extensible Stylesheet Language

2.8.1 Überblick über die XSL-Sprachfamilie

Die Extensible Stylesheet Language[26] (XSL) stellt einen auf XML basierenden Standard dar, mit dem XML-Dokumente transformiert und formatiert werden können. Genau genommen handelt es sich bei XSL aber nicht um eine einzige Sprache, sondern um eine ganze Sprachfamilie, die durch mehrere eigenständige Sprachen gebildet wird und für die es jeweils eigenständige Spezifikationen gibt. Zur XSL-Sprachfamilie gehören XSL Formatting Objects (XSL-FO), XSL Transformations (XSLT) und die XML Path Language (XPath).

XSL-FO ist der jüngste Standard der XSL-Familie, da die Empfehlung für die Version 1.0 erst im Oktober 2001 vom W3C verabschiedet wurde. XSLT 1.0 und XPath 1.0 liegen dagegen bereits seit November 1999 als Empfehlungen vor. Die entsprechenden Arbeitsgruppen des W3C sind schon mit den nächsten Versionen für diese Standards beschäftigt. Für XSLT 2.0 und Xpath 2.0 existieren bereits erste Arbeitsentwürfe.

Eine längere Geschichte, die bis 1997 zurückreicht, hat XSL-FO. Noch vor der Verabschiedung der offiziellen XML-Empfehlung im Februar 1998 begann die Arbeit an einer Formatierungssprache für XML. Wie XML sich von der Standard Generalized Markup Language (SGML) ableitet, so sollte sich XSL von der Document Style Semantics und Specification Language (DSSSL) ableiten, der Formatierungssprache für SGML. Von Anfang an war XSL-FO als XML-Anwendung konzipiert, um die für XML entwickelten Parser wiederverwenden zu können. XSL-FO wird verwendet, um die Formatierung für die Darstellung von XML-Dokumenten auf dem Bildschirm festzulegen. Es werden Bereiche der darzustellenden Seite definiert und dazugehörige Eigenschaften festgelegt. Dies geschieht mittels Formatierungsobjekten, deswegen auch der Name XSL Formatting Objects.

XSLT dagegen dient der Transformation der Struktur eines XML-Dokuments in eine andere Struktur. So kann zum Beispiel ein XML-Dokument in SVG umgewandelt werden, um es auf diese Weise in einem Browser anzeigen zu können. Das gleiche XML-Dokument könnte aber auch nach HTML transformiert werden.

---

[25] Zerlegt ein XML-Dokument und überprüft es auf Gültigkeit

[26] Einführung in XSL: Das Einsteigerseminar - XSL von Franz-Josef Herpers und Thomas J. Sebestyen

Bei XPath handelt es sich um eine Sprache, mit der bestimmte Teile eines XML-Dokuments selektiert werden können. Dies wird bei jedem Transformationsvorgang benötigt, denn um bestimmte Elemente eines XML-Dokumentes transformieren zu können, müssen diese Elemente zunächst angesteuert werden. XPath ist daher eng mit XSLT verbunden. Es wird aber keine XML-Syntax verwendet, sondern eine Art Pfadangabe, um bestimmte Elemente in einem XML-Dokument auszuwählen.

2.8.2 XSLT - Extensible Stylesheet Language Transformations

2.8.2.1 Vorraussetzungen für die Programmierung mit XSLT

Um ein XML-Dokument mittels XSLT zu verarbeiten, wird ein XSLT-Prozessor benötigt. Dieser wendet ein XSLT-Dokument auf ein XML-Dokument an und erzeugt ein Ergebnisdokument. Der beschriebene Vorgang wird als Transformation bezeichnet.

Ein sehr bekannter kommandozeilenorientierter XSLT-Prozessor ist SAXON. Er wurde von Michael Kay einem Mitglied der XSLT-Arbeitsgruppe des W3C entwickelt und ist unter http://saxon.sourceforge.net zu erhalten. Nach erfolgter Installation, muss ein DOS-Fenster geöffnet und in das Verzeichnis in dem SAXON installiert ist gewechselt werden. Anschließend kann eine Eingabe wie in Abbildung 17 dargestellt erfolgen.

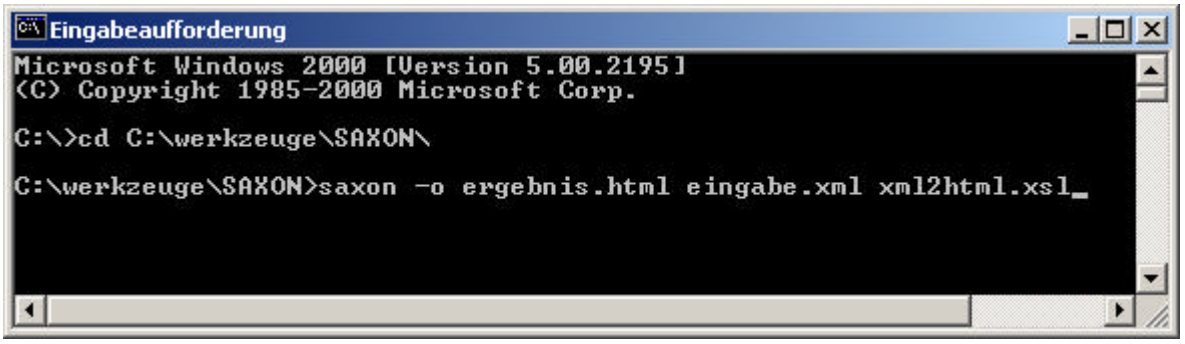

Abbildung 17: SAXON - ein kommandozeilenorientierter XSLT-Prozessor

Durch die Angabe `-o` wird festgelegt, dass das Ergebnis nicht im DOS-Fenster angezeigt wird, sondern in eine Datei, in diesem Fall ergebnis.html geschrieben wird. Der zweite angegebene Dateiname spezifiziert die XML-Datei, die transformiert werden soll. Als letztes folgt der Name der XSLT-Datei, die auf die XML-Datei angewendet werden soll.

Zum besseren Verständnis für die nachfolgenden Kapitel ist noch zu sagen, dass ein XSLT-Prozessor eine XML-Datei gemäß dem Document Object Model (DOM) betrachtet. Mit Hilfe des DOM können Inhalte von XML- und HTML-Dokumenten strukturiert werden. Das Dokument wird als Baum betrachtet, dieser wiederum hat Verzweigungen, die als Knoten bezeichnet werden. Die Bausteine eines Dokuments erscheinen in dem Baum in der Reihenfolge, in der sie auch innerhalb der Datei auftreten. Das erste Element im Dokument wird als Wurzelelement bezeichnet. Es werden verschiedene Knotentypen unterschieden. Elemente werden durch Elementknoten, Attribute durch Attributknoten und Textinhalte von Elementen von Textknoten repräsentiert.

2.8.2.2 Struktur einer XSLT-Datei

Das Wurzelelement einer XSLT-Datei ist immer das `<xsl:stylesheet>`-Element. Über dessen Attribut `version` wird die genutzte XSLT-Version angegeben. Außerdem wird ein Namensraum für die XSLT-Elemente deklariert. Darüber erkennt der XSLT-Prozessor, wann er es mit Anweisungen zu tun hat, die von ihm zu verarbeiten sind. Das zugewiesene Namensraumpräfix ist `xsl` und wird als Platzhalter für den eigentlichen Identifikator des XSLT-Namensraums verwendet. Dieser Identifikator lautet für den XSLT-Namensraum immer `http://www.w3.org/1999/XSL/Transform.` Im Zusammenhang sieht dies wie folgt aus:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 ...
</xsl:stylesheet>
```

Innerhalb des `<xsl:stylesheet>`-Elements folgen dann die eigentlichen XSLT-Anweisungen, immer mit vorangestelltem Namensraumpräfix `xsl.`

2.8.2.3 Festlegung von Ausgabeformaten

Über das Element `<xsl:output>`, das direkt unterhalb des `<xsl:stylesheet>`-Elements notiert werden muss, können Festlegungen bezüglich des Ausgabeformates des Ergebnisbaums getroffen werden. Wird kein explizites Ausgabeformat angegeben, so nimmt der XSLT-Prozessor standardmäßig XML als Ausgabeformat an.

Für eine Ausgabe nach HTML sieht dies wie folgt aus:

```
<xsl:output method="html"/>
```

Ist das Zielformat SVG, so ist die Angabe etwas umfangreicher:

```
<xsl:output method="xml" encoding="iso-8859-1"
 doctype-public="-//W3C//DTD SVG 1.0//EN"
 doctype-system="http://www.w3.org/TR/2001/REC-SVG-
 20010904/DTD/svg10.dtd"/>
```

Als Ausgabeformat wird `xml` festgelegt, da SVG eine XML-basierte Sprache ist. Das Attribut `encoding` gibt an welcher Zeichensatz verwendet werden soll. In diesem Fall der Zeichensatz `iso-8859-1`, der die meisten Zeichen der westeuropäischen Sprachen umfasst. Die Angabe von `doctype-public` und `doctype-system` bewirkt, dass in der zu erstellenden SVG-Datei ein Tag mit Namen <!DOCTYPE> eingefügt wird, in dem auf die zu verwendende DTD verwiesen wird. Über `doctype-public` wird der Name der DTD angegeben und unter `doctype-system` die URL unter der die DTD abgelegt ist.

2.8.2.4 Verwendung von Template Rules

Über das Element `<xsl:template>` wird in XSLT eine sogenannte Template Rule[27] für bestimmte Elemente des Quelldokuments definiert. Die Elemente, für die diese Regel gelten soll, werden durch das Attribut `match` ausgewählt. Der Attributwert des `match`-Attributs ist ein XPath-Ausdruck, der dazu dient durch den Quellbaum zu navigieren. Zum Verständnis genügt es, wenn man sich einen solchen Ausdruck als eine Pfadangabe durch den Quellbaum vorstellt. Ein Schrägstrich deutet dabei jeweils eine Verzweigung in eine tiefere Ebene an. Eine Template Rule hat zur Folge, dass alles was innerhalb steht ausgegeben und alle Anweisungen des XSL-Namensraums ausgeführt werden.

```
<xsl:template match="/daten/ergebnis">
  Dieser Text wird direkt ins Ausgabedokument geschrieben
  <xsl: ... />
</xsl:template>
```

2.8.2.5 Schleifen und Bedingungen

In XSLT besteht die Möglichkeit durch das Tag `<xsl:for-each>` Schleifen zu verwenden. Um die Knotenmenge auszuwählen, auf welche die Schleife anzuwenden ist, steht das Attribut `select` zur Verfügung. Für jeden der beinhalteten Knoten führt der XSLT-Prozessor die Anweisungen, welche innerhalb der Schleife stehen, aus.

```
<xsl:for-each select="ergebnis">
  <xsl:value-of select="@id"/>
  <xsl:value-of select="wert"/>
</xsl:for-each>
```

Im vorangegangenen Listing sind noch die zwei Anweisungen innerhalb der Schleife zu erläutern. Die Anweisung `<xsl:value-of select="@id"/>` hat zu Folge, dass der Inhalt des Attributes `id` in das Ergebnisdokument ausgegeben wird, während mit der zweiten Anweisung `<xsl:value-of select="wert"/>` der Inhalt des Elements Wert in das Ergebnisdokument geschrieben wird. Um ein Attribut auszugeben muss innerhalb von `select` das Zeichen @ und der Name des Attributs angegeben werden, beim Auslesen des Inhalts eines Elements dagegen nur der Elementname.

Wie in Programmiersprachen auch, kann mit XSLT eine Bedingung erzeugt werden. Dazu steht das Tag `<xsl:if>` zur Verfügung. Über dessen Attribut `test` kann die Bedingung angegeben werden, die erfüllt sein muss, damit die Anweisungen innerhalb des `<xsl:if>`-Elementes ausgeführt werden.

```
<xsl:if test=" ... ">
  <xsl: ... />
</xsl:if>
```

[27] rule (englisch): Regel

2.8.2.6 Variablen und Attribute

Variablen werden mittels `<xsl:variable>` erstellt. Durch das Attribut `name` wird der Variablenname angegeben. Der Inhalt der Variablen wird durch die Anweisung `<xsl:value-of>` festgelegt.

```
<xsl:variable name="h">
  <xsl:value-of select=" ... "/>
</xsl:variable>
```

Zu der Verwendung von Variablen innerhalb von XSLT ist zu sagen, dass sie eher den Charakter von Konstanten haben, da ein Wert zwar zugewiesen werden kann, später aber innerhalb einer Template Rule keine Möglichkeit besteht ihn wieder zu ändern.

In XSLT können Attribute zu einem Element hinzugefügt werden. Dafür wird das Tag `<xsl:attribute>` verwendet. Über dessen Pflichtattribut `name` wird der Name des zu erzeugenden Attributs festgelegt. Der Wert des Attributs wird, wie bei der Erstellung von Variablen, durch `<xsl:value-of>` festgelegt.

```
<xsl:attribute name="height">
  <xsl:value-of select="$h"/>
</xsl:attribute>
```

Soll das zu erstellende Attribut als Wert den Inhalt einer Variablen zugewiesen bekommen, so muss wie im obigen Listing das Zeichen $ und der Name der Variablen innerhalb des `select`-Attributs angegeben werden.

2.8.2.7 Beispiel zu Template Rules und Schleifen

Als Ausgangsdatei sei folgende XML-Datei gegeben:

```
<?xml version="1.0"?>
<daten>
  <ergebnis id="Q1"><wert>100</wert></ergebnis>
  <ergebnis id="Q2"><wert>150</wert></ergebnis>
  <ergebnis id="Q3"><wert>55</wert></ergebnis>
  <ergebnis id="Q4"><wert>122</wert></ergebnis>
</daten>
```

Das Wurzelelement ist `<daten>`. Dieses Tag besitzt weitere vier Elementknoten mit Namen `<ergebnis>`, die jeweils wieder einen Knoten `<wert>` enthalten. Innerhalb von `<wert>` werden Umsatzzahlen (in Millionen Euro) angegeben. Jeder Knoten `<ergebnis>` repräsentiert ein Quartal eines Jahres. Jedes Tag besitzt dazu das Attribut `id`, das jeweils den Inhalt Q1, Q2, Q3 und Q4 enthält.

Eine XSL-Datei, welche die vorliegende Datei in eine HTML-Datei transformiert, könnte wie folgt aussehen:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="daten">
```

```
  <html><body>
  <h2>Das Umsatzergebnis des letzten Jahres in Millionen Euro</h2>
  <table border="2">
  <tr><td><b>Quartal</b></td><td><b>Umsatz</b></td></tr>
  <xsl:for-each select="ergebnis">
  <tr><td>
  <xsl:value-of select="@id"/>
  </td><td>
  <xsl:value-of select="wert"/>
  </td></tr>
  </xsl:for-each>
  </table>
  </body></html>
</xsl:template>
</xsl:stylesheet>
```

In der ersten Zeile wird über das Element `<xsl:stylesheet>` die XSLT-Datei eingeleitet. Das Ausgabeformat wird in der zweiten Zeile durch die Angabe `<xsl:output method="html"/>` als HTML festgelegt. In der nächsten Zeile `<xsl:template match="daten">` wird eine Vorlage für das Tag `<daten>` erstellt, dadurch wird der folgende XSLT-Code nur angewendet, wenn das entsprechende Tag in der XML-Datei gefunden wird. Anschließend wird über das HTML-Tag `<H2>` die Überschrift erzeugt und mit `<table>` die Tabelle geöffnet. Darauf folgend wird die erste Tabellenzeile erstellt, die zur Beschriftung dient. Über die Zeile `<xsl:for-each select="ergebnis">` wird für jedes vorhandene Tag `<ergebnis>` die folgende Schleife durchlaufen. Über `<tr>` wird eine neue Tabellenzeile geöffnet, anschließend wird die linke Tabellenzeile erzeugt `<td><xsl:value-of select="@id"/></td>`, wobei über `<xsl:value-of select="@id"/>` der Inhalt des Attributs `id` des Tags `<ergebnis>` ausgelesen und als Wert zwischen `<td>` und `</td>` eingesetzt wird. Darauf folgend wird die rechte Tabellenzeile erzeugt, wobei `<xsl:value-of select="wert"/>` den Inhalt des Tags `<wert>` ausliest und wieder zwischen `<td>` und `</td>` einsetzt. Abschließend wird noch die Tabelle, die HTML- und die XSL-Datei beendet.

Die durch Transformation erzeugte HTML-Datei hat folgenden Inhalt:

```
<html><body>
  <h2>Das Umsatzergebnis des letzten Jahres in Millionen Euro</h2>
  <table border="2">
  <tr><td><b>Quartal</b></td><td><b>Umsatz</b></td></tr>
  <tr><td>Q1</td><td>100</td></tr>
  <tr><td>Q2</td><td>150</td></tr>
  <tr><td>Q3</td><td>55</td></tr>
  <tr><td>Q4</td><td>122</td></tr>
  </table>
</body></html>
```

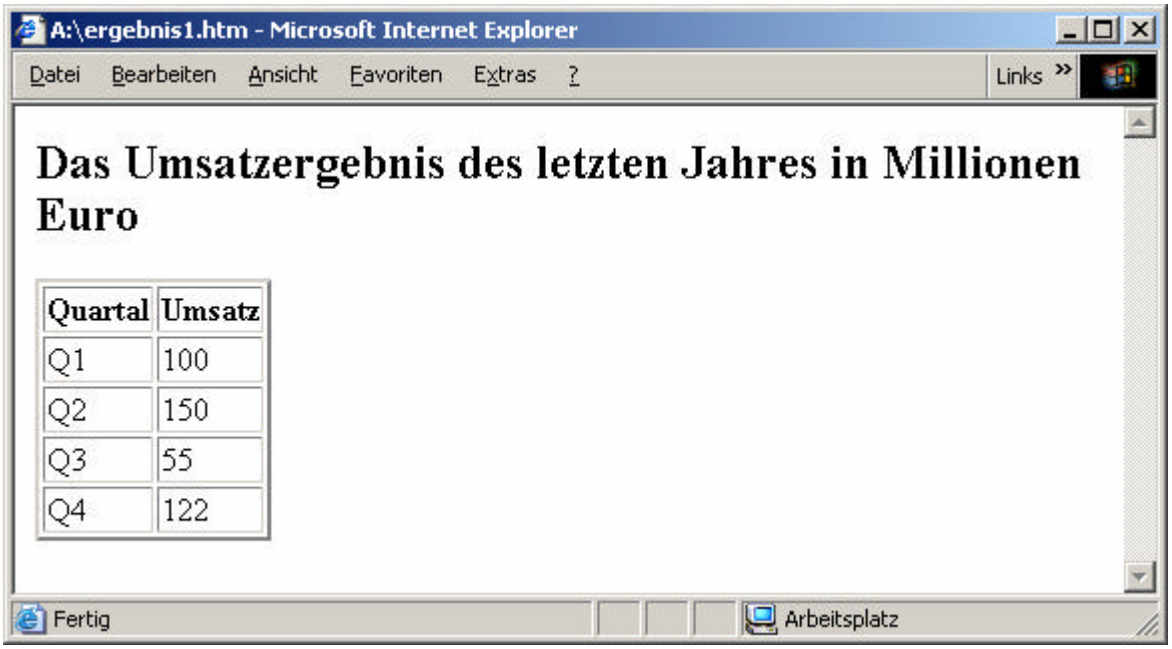

Abbildung 18: Transformation einer XML-Datei in eine HTML-Datei

2.8.2.8 Beispiel zu Bedingungen, Variablen und Attributen

Um den Vorteil von XSLT deutlich zu machen, wird die gleiche XML-Datei nun mit Hilfe der folgenden XSL-Datei in eine SVG-Datei transformiert:

```
<xsl:stylesheet version="1.0"
 xmlns="http://www.w3.org/2000/svg"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="iso-8859-1"
 doctype-public="-//W3C//DTD SVG 1.0//EN"
 doctype-system="http://www.w3.org/TR/2001/REC-SVG-
 20010904/DTD/svg10.dtd"/>
<xsl:template match="daten">
<svg>
<xsl:for-each select="ergebnis">
<xsl:variable name="quartal">
<xsl:value-of select="@id"/>
</xsl:variable>
<rect width="15" style="fill:blue">
<xsl:variable name="h">
<xsl:value-of select="wert"/>
</xsl:variable>
<xsl:attribute name="height">
<xsl:value-of select="$h"/>
</xsl:attribute>
<xsl:attribute name="x">
<xsl:if test="$quartal='Q1'">30</xsl:if>
```

```
<xsl:if test="$quartal='Q2'">60</xsl:if>
<xsl:if test="$quartal='Q3'">90</xsl:if>
<xsl:if test="$quartal='Q4'">120</xsl:if>
</xsl:attribute>
<xsl:attribute name="y">
<xsl:value-of select="300-$h"/>
</xsl:attribute>
</rect>
</xsl:for-each>
</svg>
</xsl:template>
</xsl:stylesheet>
```

Das Tag `<xsl:output>` legt als Ausgabeformat XML fest, da SVG eine XML-basierte Sprache ist. Wie im vorangegangenen Beispiel wird in der nächsten Zeile `<xsl:template match="daten">` eine Vorlage für das Tag `<daten>` erstellt, dadurch wird der folgende XSLT-Code nur angewendet, wenn ein entsprechendes Tag in der XML-Datei gefunden wird. Die Vektorgrafik wird anschließend über das Tag `<svg>` eingeleitet. Darauf folgend wird eine Schleife durchlaufen `<xsl:for-each select="ergebnis">`, was zur Folge hat, dass der Code innerhalb der Schleife auf jedes Tag `<ergebnis>` angewendet wird. Dann wird das Attribut `id` aus dem `Tag <ergebnis>` ausgelesen und in der Variablen `quartal` gespeichert. Anschließend wird eine Säule über das Tag `<rect>` erstellt. Über das Attribut `width` wird die Breite des Rechtecks angegeben und innerhalb des `style`-Attributs die Farbe gesetzt. Es fehlt nun noch die x-Koordinate und die Höhe des Rechtecks. Deswegen wird der Umsatzwert, der in `<wert>` steht, ausgelesen und in der Variablen `h` gespeichert. Der Umsatzwert soll der Höhe der Säule entsprechen. Damit ist die Höhe der Säule bekannt und es kann über `<xsl:attribut>` das Attribut `height` zum Tag `<rect>` hinzugefügt werden. Anschließend wird das Attribut `x` zum Tag `<rect>` hinzugefügt. Jede Säule muss gegenüber der vorhergehenden versetzt sein, damit sie nicht überschrieben wird. Dies wird durch `<xsl:if>` gelöst, wodurch der Wert des Attributs `id` geprüft und für jeden anderen Wert von `id` eine andere Zahl für die x-Koordinate angegeben wird. Als letztes wird noch die y-Koordinate gesetzt, an der die Säulen beginnen.

Die SVG-Datei die durch die Transformation entsteht, sieht dann wie folgt aus:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
<rect width="15" style="fill:blue" height="100" x="30" y="100"/>
<rect width="15" style="fill:blue" height="150" x="60" y="50"/>
<rect width="15" style="fill:blue" height="55" x="90" y="145"/>
<rect width="15" style="fill:blue" height="122" x="120" y="78"/>
</svg>
```

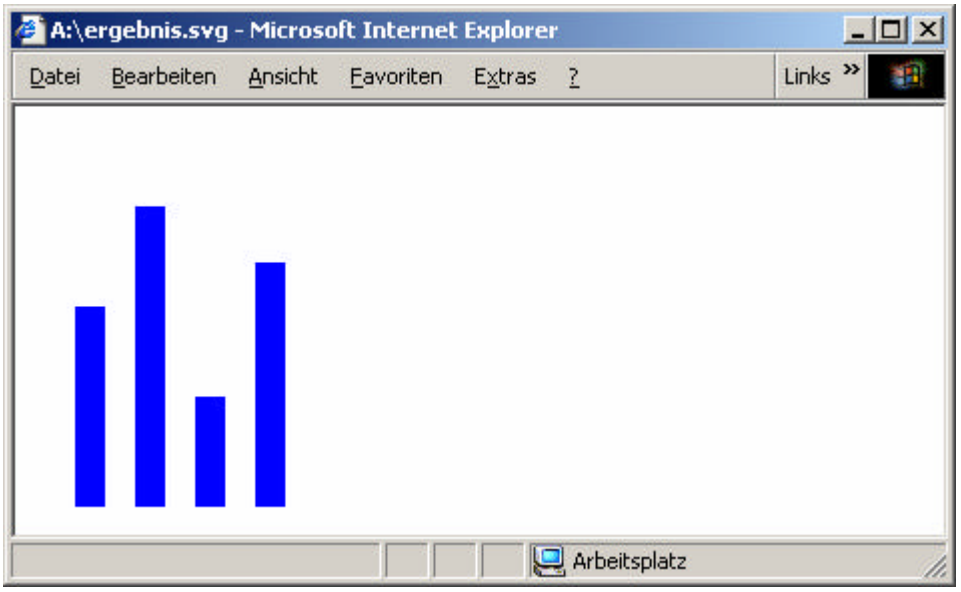

Abbildung 19: Transformation einer XML-Datei in eine SVG-Datei

2.9 XML und Java[28]

Das Java 2 Developement Kit Standard Edition 1.4 bietet umfangreiche XML-Unterstützung. Es verfügt über zwei XML-Parser und einen XSLT-Prozessor. XML-Parser haben die Funktion eine XML-Datei einzulesen und auf syntaktische Korrektheit zu prüfen.

Nachdem XML im Februar 1998 standardisiert wurde kamen zunächst verschiedene XML-Parser heraus. Zum Beispiel XP von James Clark und Xerces von Apache. Beide verfügen über eine XML-API, um sie in eigene Projekte einzubinden. Jeder Parser verfügt aber über eine andere API, die untereinander nicht oder nur teilweise kompatibel sind, daher musste eine einheitliche API entwickelt werden, die innerhalb von Java verwendet werden kann, dies führte zu JAXP.

XSLT-Prozessoren werden wie bereits erwähnt dazu genutzt um XML-Dateien in andere Sprache, wie SVG oder HTML zu formen. Eine der ersten XSLT-Prozessoren war XP von James Clark, dabei handelt es sich um ein separates Programm, das installiert werden muss. Ein anderer bekannter XSLT-Prozessor ist SAXON von Michael Kay. Von Apache gibt es Xalan. Wie auch bei den XML-Parsern verfügen die XSLT-Prozessoren jeweils über eine eigene API, die in eigenen Projekten eingesetzt werden kann. Auch hier sind sie untereinander nicht kompatibel, was bedeutet, das ein Java-Code der zum Beispiel für SAXON geschrieben wurde nicht für Xalan eingesetzt werden kann. Dies führt wieder zur Notwendigkeit einer einheitlichen API, wiederum der JAXP.

Die Java API for XML Processing (JAXP) wurde von SUN Microsystems entwickelt und wurde in der Version 1.0 im März 2000 vorgestellt. Im März 2001 kam die Version 1.1 heraus. Bisher musste JAXP neben Java separat installiert werden, ab der Java-Version 1.4 ist es aber bereits komplett enthalten und muss nicht mehr extra installiert werden.

---

[28] Einführung in XML und Java: Das Einsteigerseminar – Java/XML von Michael Seeboerger-Weichselbaum

Die JAXP besteht aus zwei Teilen, dem Parsen von XML-Dokumenten und dem Formen von XML-Dokumenten mit XSLT. Beinhaltet sind zwei XML-Parser, der DOM-Parser und der SAX-Parser. Der Unterschied zwischen beiden liegt darin, dass der DOM-Parser im Gegensatz zum SAX-Parser nicht nur die syntaktische Korrektheit der XML-Datei überprüft, sondern die XML-Datei intern als Dokumentenbaum abbildet. Dadurch ist es möglich, einzelne Tags über XPath anzusprechen und auf deren Inhalt zuzugreifen. Möchte man also ein XML-Dokument mittels einer XSLT-Datei transformieren, so muss dieser XML-Parser verwendet werden. Der Nachteil ist, dass bei besonders großen XML-Dateien das Parsen und das Bilden des Dokumentenbaums viel Zeit und Speicherplatz beansprucht.

Das folgende Beispiel zeigt das Parsen und die Transformation einer XML-Datei unter Verwendung der JAXP. Die Erläuterung des Codes erfolgt als Kommentar innerhalb des Programms:

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class JDOMtransform
{
 // Definition von zwei Objekten XMLdocument und XSLTdocument der
 // Klasse Document, die das eingelesene XML- und XSLT-Dokument
 // repräsentieren
 static Document XMLdocument;
 static Document XSLTdocument;

 public static void main(String args[])
 {
  JDOMtransform jdomtransform = new JDOMtransform();
 }

 public JDOMtransform()
 {
  transforming();
 }

 public void transforming()

 {
  // XML-Datei und XSLT-Datei werden geladen
  File XMLdatei = new File("C:/Diplomarbeit/ergebnis.xml");
  boolean XMLladen = XMLdatei.canRead();
  File XSLTdatei = new File("C:/Diplomarbeit/xml2svg.xsl");
  boolean XSLTladen = XSLTdatei.canRead();

  // es wird geprüft, ob beide Dateien lesbar sind
```

```
if (XMLladen==true && XSLTladen==true)
{
 try
 {
  // Parsen der XML- und XSLT-Datei
  DocumentBuilderFactory factory =
  DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = factory.newDocumentBuilder();
  XMLdocument = builder.parse(XMLdatei);
  XSLTdocument = builder.parse(XSLTdatei);
  System.out.println("\nDas Formen startet");

  // der XSLT-Processor, der die Formung vornehmen wird, wird
  // erzeugt
  TransformerFactory tfactory = TransformerFactory.newInstance();
  // StreamSource bereitet die beiden Datenquellen für das Formen
  // vor
  StreamSource xsltsource = new StreamSource(XSLTdatei);
  StreamSource xmlsource = new StreamSource(XMLdatei);
  // der XSLT-Processor wird aktiviert
  // es wird die Datenquelle übergeben, die die XSLT-
  // Informationen enthält
  Transformer transformer = tfactory.newTransformer(xsltsource);

  // Speichern des Ergebnisses wird vorbereitet
  FileWriter Ausgabestrom = new
  FileWriter("C:/Diplomarbeit/ergebnis.svg");
  BufferedWriter output = new BufferedWriter(Ausgabestrom);
  // result enthält das Ergebnis der Formung, da dieses als Datei
  // ausgegeben werden soll,wird an den Konstruktor das Objekt
  // output der Klasse BufferedWriter übergeben
  StreamResult result = new StreamResult(output);

  // Formung wird vorgenommen
  // die XML-Datenquelle und das Objekt, das das Ergebnis speichert,
  // werden übergeben
  transformer.transform(xmlsource, result);
  // Ergebnis in einen String umwandeln, damit der String als Ascii-
  // Datei ausgegeben werden kann
  String out = result.toString();
  // bei toString() erscheint am Ende der Name der Klasse, das @-
  // Symbol und der hexadezimale Wert des Hash Codes des Objekts,
  // daher wird der String out von Position 0 an nur bis zu der
  // Position, an der das erste Mal javax auftaucht in die Datei
  // geschrieben
  output.write(out,0,out.indexOf("javax"));
```

```
    output.close();
    System.out.println("\nFormen beendet");
   }
   // Fehler, der ausgeworfen wird, wenn beim Parsen ein Fehler in der
   // Datei entdeckt wird
   catch (SAXParseException error)
   {
    System.out.println("\n+++Parse Error+++"+ "\nZeile: " +
    error.getLineNumber() + "\nDatei: " + error.getSystemId());
    System.out.println("\n" + error.getMessage());
   }
   // Fehler bei der Parserkonfiguration
   catch (ParserConfigurationException pce)
   {
    pce.printStackTrace();
   }
   // signalisiert einen allgemeinen Ein-/Ausgabefehler
   catch (IOException e)
   {
    System.out.println("IO-Fehler:\n"+e);
   }
   // Fehler, der von der Klasse TranformerFactory ausgeworfen werden
   // kann, wenn ein Objekt über newInstance() erzeugt wird
   catch (TransformerFactoryConfigurationError te)
   {
    System.out.println("TransformerFactory-Fehler:\n"+te);
   }
   // Fehler, der ausgeworfen werden kann, wenn ein Objekt der Klasse
   // Transformer erzeugt wird, indem die Methode newTransformer() der
   // Klasse TransformerFactory genutzt wird
  catch (TransformerConfigurationException tc)
  {
   System.out.println("Transformer-Fehler:\n"+tc);
  }
  // Fehler, der von der Methode transform() der Klasse Transformer
  // ausgeworfen werden kann
   catch (TransformerException tx)
   {
    System.out.println("Transform-Fehler:\n"+tx);
   }
   // Klasse Throwable dient als Basisklasse für Errors und
   // Exceptions.
   catch (Throwable t)
   {
```

```
    t.printStackTrace();
   }
  }
  else
  {
   System.out.println("Datei existiert nicht!");
  }
 }
}
```

2.10 SVG - Scalable Vector Graphics

2.10.1 Einführung

Mit Scalable Vector Graphics[29] (SVG) können komplexe zweidimensionale Vektorgraphiken erstellt werden. Zudem ist ein hohes Maß an Interaktivität, wie z.B. Zoomen und Verschieben des Bildausschnittes möglich. SVG kann, da es textbasiert ist, mit jedem Texteditor erstellt und, da es XML-strukturiert ist, in einer einfachen XML-Datei gespeichert werden. Wie jedes andere XML-Dokument auch, lassen sich SVG-Grafiken mit Hilfe von Scriptsprachen wie JavaScript, JSP, PHP und ASP generieren. Mit XSLT können Transformationen nach SVG durchgeführt werden.

Gegenüber Rasterdaten wie GIF[30], JPEG[31] und TIFF[32] hat SVG entscheidende Vorteile:

? Die Dateien sind klein, da sie aus Text bestehen

? SVG-Grafiken lassen sich ohne Qualitätsverlust skalieren

? Es wird eine hohe Farbtiefe unterstützt

? Animationen sind möglich, führen aber nicht zu größeren Dateien

? Das Document Object Model (DOM) wird unterstützt

2.10.2 Entstehung

Im April 1998 reichten Adobe, IBM, Netscape und SUN beim W3C ein Konzept für die Precision Graphics Markup Language (PGML) ein. PGML ist eine Sprache, die stark an Post-Script angelehnt ist und Vorteile bei den Gestaltungsmöglichkeiten und Animationen von Grafiken besitzt. Bereits einen Monat später brachten die Firmen Hewlett Packard, Microsoft, Macromedia und Visio einen Vorschlag zur Vector Markup Language (VML) ein, wobei hier die Schwerpunkte auf technischen Aspekten, wie beispielsweise der Einbindung mathematischer Funktionen liegen. Des weiteren zeigten PRP und Orange PCSL

---

[29] Einführung in SVG: SVG – Scalable Vector Graphics von Iris Fibinger
Spezifikation unter: http://www.w3.org/TR/SVG11/

[30] Graphics Interchange Format

[31] Joint Photographic Expert Group

[32] Tagged Image File Format

im Juni 1998 mit ihrer Hyper Graphics Markup Language (HGML) interessante Aspekte bezüglich Kopieren, Transformieren und Einfügen von Bildausschnitten.

Aufbauend auf diesen verschiedenen Modellen, als Kombination der jeweiligen Vorteile und mit weiteren Verbesserungen und Ergänzungen, entstand im Oktober 1998 SVG. Die erste offizielle Empfehlung des W3C liegt seit September 2001 vor. Seit Januar 2003 existiert bereits ein zweiter offizieller Sandard, die Version 1.1. Die vom W3C ins Leben gerufene Arbeitsgruppe, die SVG-Working Group, arbeitet momentan an Version 1.2. Zu ihr gehören namhafte Firmen der Computerindustrie, darunter Adobe, Apple, Autodesk, Canon, Corel, HP, IBM, Kodak, Macromedia, Microsoft, Netscape/AOL, Quark, Sun und Xerox. Aus der Liste der teilnehmenden Firmen wird deutlich, dass großes Interesse an der Weiterentwicklung von SVG besteht und dass SVG auch in der Zukunft von Bedeutung sein wird.

2.10.3 Der SVG Viewer von Adobe

Auch mit dem SVG Viewer bekundet Adobe echtes Interesse an der Weiterentwicklung von SVG. Er setzt bereits einen Großteil aller SVG-Features um. Ab der Version 3.0 besitzt der SVG Viewer eine eigene Script-Engine, die Scripts anstelle des Browsers verarbeitet. Damit beseitigt Adobe das Problem, dass einige Browser keine in XML eingebundenen Scripts umsetzen. Nützlich ist auch der eingebaute kleine Debugger. In der Statuszeile des Browserfensters wird die Position des ersten Programmierfehlers im Dokument angezeigt. Der Hauptvorteil gegenüber anderen Viewern liegt aber in der softwareeigenen Funktionalität. Dazu gehören Zoom- und Verschiebemöglichkeiten, sowie eine Suchfunktion mit der in einer SVG-Grafik nach bestimmten Texten gesucht werden kann. Abbildung 20 zeigt das Kontextmenü des SVG Viewers von Adobe, das mit einem Klick der rechten Maustaste innerhalb der SVG-Grafik aufgerufen werden kann.

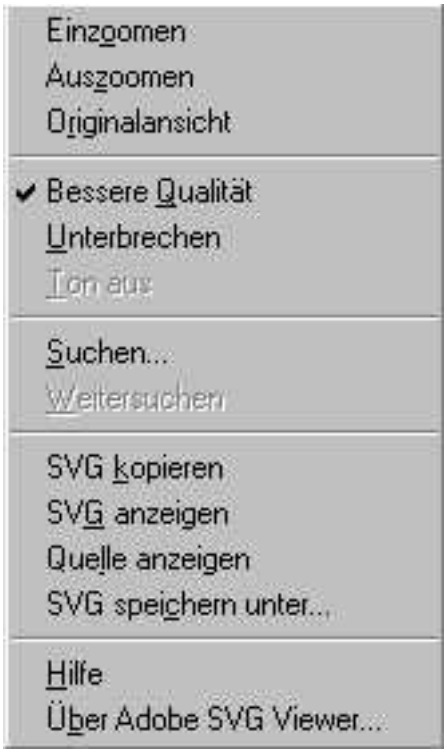

Abbildung 20: Kontextmenü des SVG-Viewers von Adobe

Außer über das Kontextmenü lässt sich der Bildausschnitt auch unter Verwendung der in Tabelle 2 aufgeführten Tastenkombinationen verändern.

Tabelle 2: Tastenkombinationenfür den SVG-Viewer von Adobe

Ausschnitt dynamisch verschieben	Alt-Taste gedrückt halten und den Ausschnitt mit der linker Maustaste verschieben
Dynamisches Einzoomen	Strg-Taste gedrückt halten und mit linker Maustaste klicken
Dynamisches Auszoomen	Strg- und Shift- Taste gedrückt halten und mit linker Maustaste klicken

Die neueste Version des SVG-Viewers ist unter http://www.adobe.com/svg/viewer/install/main.html kostenlos erhältlich. Die momentan aktuelle Version 3.0 wird von den Plattformen Windows (95/98, ME, NT, 2000, XP) und MacOS (8.x, 9.x, 10.1) unterstützt. Releases für Solaris 8 und Red Hat-Linux sind als Betaversionen unter http://www.adobe.com/svg/viewer/install/old.html zu bekommen. Der SVG Viewer 3.0 ist ein Plug-In für Netscape Navigator (ab Version 4.0), Netscape 6.x (hier ist eine manuelle Installation erforderlich), Internet Explorer (Version 4.0 - 5.5), RealPlayer 8 und Opera 5.x.

2.10.4 Aufbau eines SVG-Dokuments

Das Grundgerüst eines SVG-Dokumentes hat folgenden grundsätzlichen Aufbau:

```
<?xml version=”1.0” encoding=”iso-8859-1” standalone=”no”>
<!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 20010904//EN”
“http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd”>
<svg width=”200” height=”100” xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink>
<!-- *** Hier folgt die eigentliche Grafik *** -->
</svg>
```

Die erste Zeile stellt den XML-Prolog dar. Dieser Prolog muss immer zu Beginn einer SVG-Datei stehen, damit der Parser weiß, dass es sich hierbei um ein XML-Dokument handelt. Der XML-Prolog kann zusätzlich die Attribute `encoding` und `standalone` beinhalten. Mit `encoding` wird der Zeichenstandard angegeben, der innerhalb des Dokuments benutzt wird. In diesem Fall wird die ISO-8859-1 verwendet, was bedeutet, dass auch Sonderzeichen wie deutsche Umlaute im Quelltext verwendet werden können. Mit dem Attribut `standalone` wird eine Aussage darüber getroffen, ob das vorliegende Dokument in Abhängigkeit zu einem anderen Dokument steht. Wird das Attribut auf `no` gesetzt, so ist diese Abhängigkeit vorhanden. In den meisten Fällen ist das andere Dokument eine DTD.

Da in diesem Beispiel eine entsprechende DTD vorliegt, kann auf diese in der nächsten Zeile über `DOCTYPE` verwiesen werden. Zusätzlich kann angegeben werden, ob die referenzierte DTD öffentlich zugänglich (`PUBLIC`) oder auf einem privaten Rechner (`SYSTEM`) abgelegt ist. Im Anschluß folgt der Name der DTD und ihre URL.

Als nächstes muss das SVG-Element folgen, indem sich alle übrigen Inhalte befinden. Es wird auch als Root-Element bezeichnet, da es das erste Element in einer SVG-Datei ist. Zusätzlich kann es nochmals innerhalb des Dokumentes verwendet werden, um SVG-Grafiken zu schachteln. Wichtig sind die Attribute `width` und `height`, mit denen die Ausmaße des SVG-Fensters innerhalb des Browser-Fensters festgelegt werden können.

Die Namensraumdeklarationen

```
xmlns=http://www.w3.org/2000/svg
xmlns:xlink=http://www.w3.org/1999/xlink
```

sind optional. Durch Namensraumdeklarationen können SVG-fremde Inhalte miteingebunden werden. Unter Verwendung des xlink-Namensraumes können zum Beispiel Links zu bestimmten Stellen innerhalb und auch außerhalb des Dokumentes erzeugt werden.

Des weiteren kann in dem SVG-Element das Attribut `zoomAndPan` auf `disable` gesetzt werden, mit dem Ergebnis, dass die erstellte Grafik nicht mehr unter Verwendung des Kontextmenüs verschoben, vergrößert oder verkleinert werden kann.

2.10.5 Koordinatensystem und ViewBox

Im Gegensatz zu den geodätischen Koordinatensystemen, bei denen der Ursprung links unten liegt, die positve y-Achse nach rechts zeigt und die positive x-Achse nach oben ausgerichtet ist, verhält es sich mit dem Koordinatensystem in SVG anders. Der Koordinatenursprung liegt in der linken oberen Ecke, die positive x-Achse geht nach rechts und die positive y-Achse nach unten. Um räumliche Koordinaten zum Beispiel Gauß-Krüger-Koordinaten im SVG-System darstellen zu können, müssen die Koordinaten zuerst transformiert werden. Dabei müssen die Rechtswerte um den kleinsten vorhandenen Rechtswert reduziert werden, die Hochwerte dagegen vom größten vorhandenen Hochwert abgezogen werden, um die entgegengesetzte Ausrichtung der Achse für die Hochwerte zu berücksichtigen.

Abbildung 21: Koordinatensystem in SVG

In SVG können gezielt Bildausschnitte einer Grafik mit Hilfe eines Rechtecks, das den Bildausschnitt definiert, erstellt werden. Dazu steht das Attribut `ViewBox` zur Verfügung, das innerhalb des SVG-Elements eingefügt werden kann. Angegeben wird hierbei der x- und y-Wert der linken oberen Ecke, sowie die Höhe und die Breite des Rechtecks. Der grafische Inhalt dieses Rechtecks wird dann auf die gesamte Größe des Sichtbereiches skaliert. Abhängig von den Höhen- und Breitenangaben der ViewBox im Verhältnis zu denen des SVG-Elementes ergeben sich Skalierungsfaktoren für die x- und für die y-Richtung. Zusätzlich lässt sich steuern, ob der abgebildete Ausschnitt verzerrt werden darf, oder ob er proportional dargestellt werden muss. Wird die proportionale Darstellung gewählt, so kann weiterhin entschieden werden, ob die Grafik auf das Rechteck skaliert wird, oder ob überstehende Teile abgeschnitten werden.

Die Definition einer ViewBox innerhalb des SVG-Elements kann wie folgt aussehen:

```
<svg width="200" height="100" viewBox="50 50 30 30">
```

Der erste Wert gibt hierbei die x-Koordinate der linken oberen Ecke der ViewBox an, der zweite die entsprechende y-Koordinate. Der dritte beinhaltet die Breite, der vierte die Höhe der ViewBox.

Abbildung 22 veranschaulicht noch einmal die benötigten Angaben.

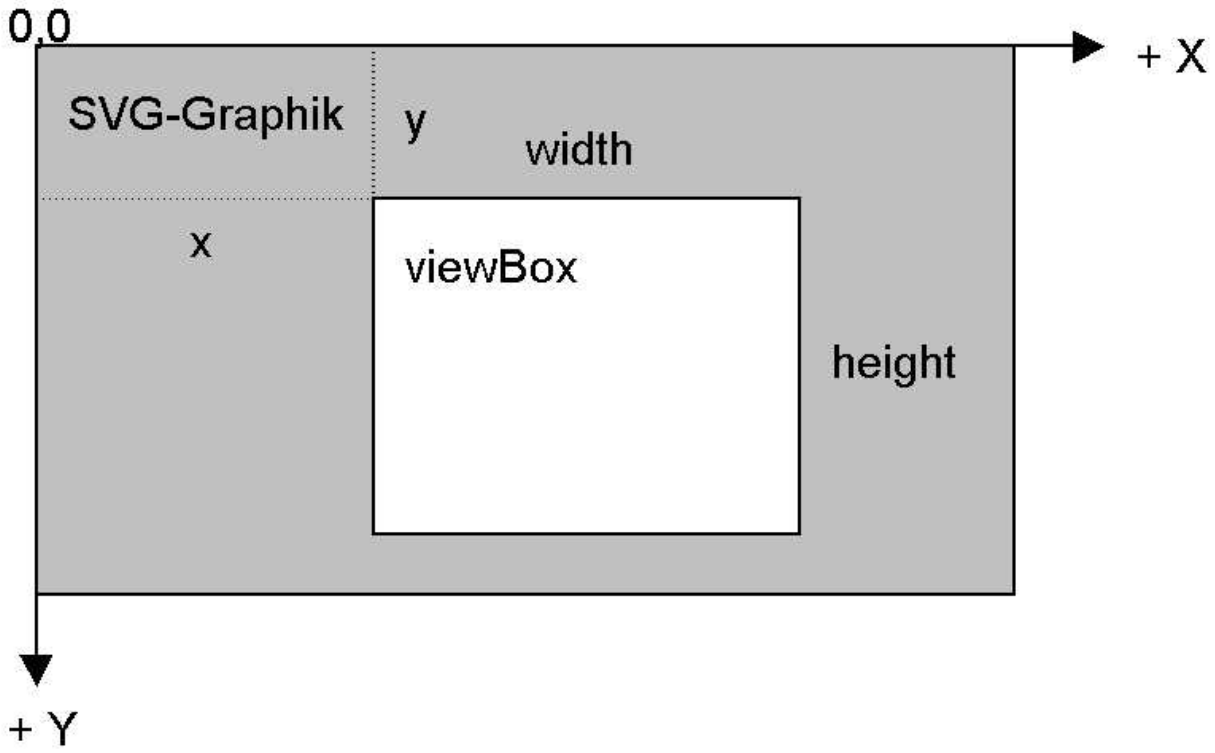

Abbildung 22: Die ViewBox in SVG

Die Seitenverhältnisse von Bildausschnitten lassen sich innerhalb der ViewBox mit dem Attribut `perserveAspectRatio` steuern. Dabei gibt es folgende Möglichkeiten:

? `perserveAspectRatio="none"`
Die Grafik wird verzerrt innerhalb des Sichtbereiches dargestellt, so dass dieser komplett ausgefüllt ist.

? `perserveAspectRatio="... slice"`
Der Sichtbereich wird komplett ausgefüllt, die Grafik aber nicht verzerrt dargestellt, stattdessen werden überstehende Teile abgeschnitten.

? `perserveAspectRatio="... meet"`
Ist die Grafik zu groß wird sie proportional verkleinert, so dass sie komplett innerhalb des Sichtbereiches angezeigt werden kann. Der Sichtbereich ist dann nicht komplett ausgefüllt.

An der Stelle, die in den obigen Möglichkeiten mit Punkten gekennzeichnet sind, wird die Lage des Bildausschnittes innerhalb des Sichtbereiches gesteuert. Zur Auswahl stehen die in Abbildung 23 aufgeführten Angaben. Wird zum Beispiel `meet` in Kombination mit `xMinyMin` angegeben, so wird die Grafik in x-Richtung links und in y-Richtung oben ausgerichtet. Dabei bleibt die Grafik komplett erhalten.

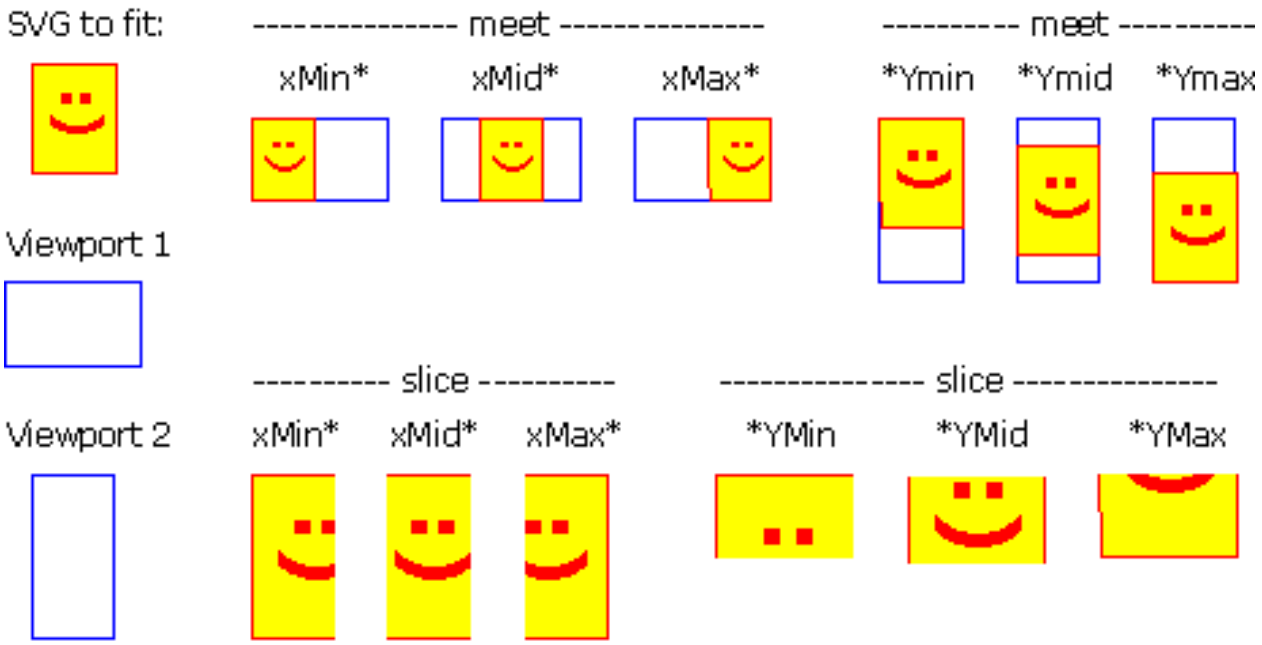

Abbildung 23: Mögliche Werte des Attributes perserveAspectRatio

2.10.6 Rechtecke, Kreise, Ellipsen und Linien

Mit SVG können einfache Grundformen wie Rechtecke, Kreise, Ellipsen und Linien schnell erstellt werden, da es für Grundformen vordefinierte Tags gibt.

Um ein Rechteck zu erzeugen, wird das Tag `<rect>` verwendet. Durch die Attribute `x` und `y` werden die Koordinaten für die linke obere Ecke des Rechtecks festgelegt. Standardmäßig ist für beide 0 eingestellt. Durch die Pflichtattributen `width` und `heigth` wird die Breite bzw. die Höhe des Rechteckes angegeben. Zusätzlich können die Attribute `rx` und `ry` benutzt werden um die Ecken des Rechtecks abzurunden. Formatierungen, wie die Strichdicke, die Strichfarbe und die Füllfarbe können durch die Attribute `stroke-width`, `stroke` und `fill` angegeben werden.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<rect x="20" y="20" width="100" height="100"
stroke-width="2" stroke="black" fill="yellow"/>
<rect x="140" y="20" rx="15" ry="15" width="100" height="100"
stroke-width="4" stroke="blue" fill="none"/>
</svg>
```

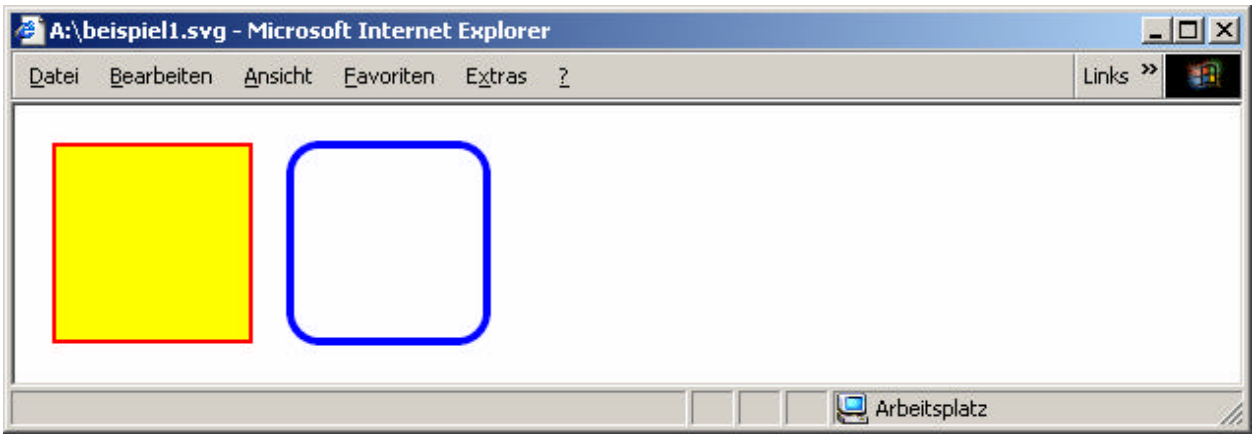

Abbildung 24: Rechtecke mit SVG

Für einen Kreis steht das Tag `<circle>` zur Verfügung. Durch die Attribute `cx` und `cy` können die Koordinaten für den Kreismittelpunkt gesetzt werden. Werden sie nicht angegeben, so wird für beide Attribute 0 eingestellt. Das Attribut `r` hingegen muss gesetzt werden, um den Radius des Kreises festzulegen.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<circle cx="70" cy="70" r="50" stroke-width="4" stroke="blue"
fill="yellow"/>
</svg>
```

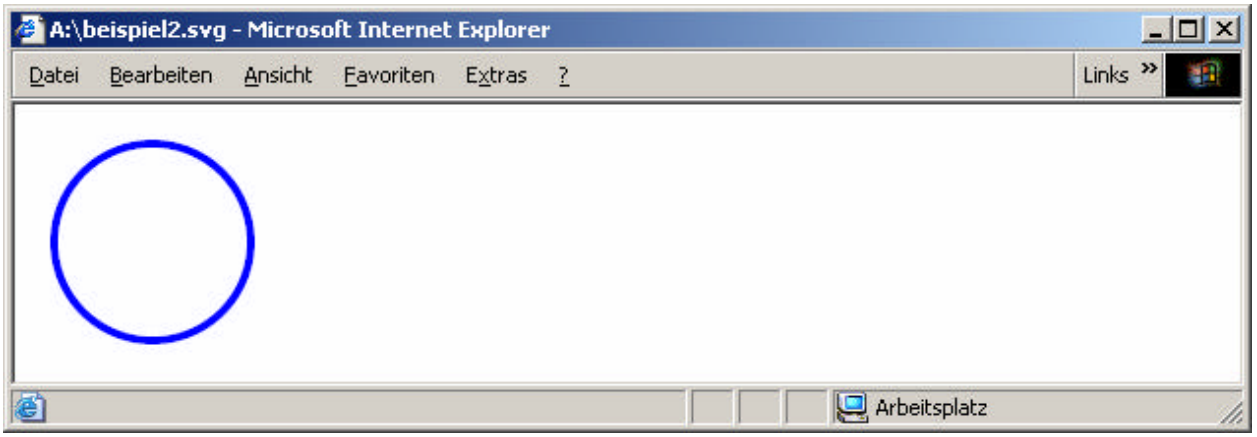

Abbildung 25: Kreis mit SVG

Das Tag `<ellipse>` dient zum Zeichnen einer Ellipse. Durch `cx` und `cy` werden die Koordinaten für den Ellipsenmittelpunkt angegeben. Die Pflichtattribute `rx` und `ry` legen den Radius in x- bzw. y- Richtung fest.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<ellipse cx="120" cy="70" rx="100" ry="40" stroke-width="3"
stroke="red" fill="yellow"/>
</svg>
```

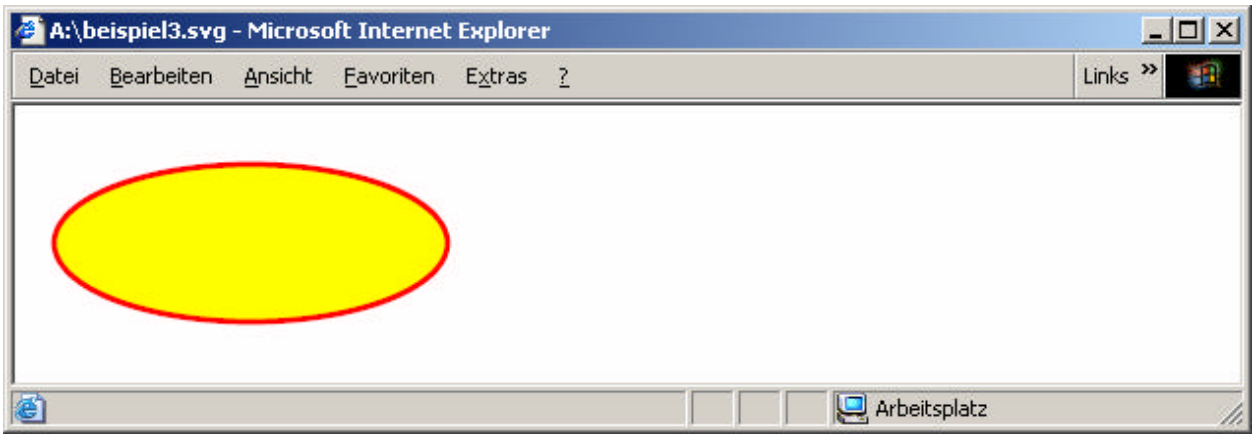

Abbildung 26: Ellipse mit SVG

Linien werden mit dem Tag `<line>` gezeichnet. Durch die Attribute `x1` und `y1` wird der Anfangspunkt der Line festgelegt. Standardmäßig ist 0 eingestellt. Mit `x2` und `y2` wird der Endpunkt der Linie festgesetzt. Auch hier ist 0 der Standardwert. Durch das Attribut `stroke-dasharray` kann eine gepunktete oder eine gestrichelte Linie erzeugt werden. Dabei geben die aufgeführten Zahlenwerte die Länge eines Striches bzw. die Länge der Lücke zwischen zwei Strichen an. Das Attribut `stroke-dasharray` kann auch für die Umrandung eines beliebigen Elementes verwendet werden.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<line x1="20" y1="20" x2="300" y2="20" stroke-width="10"
stroke="yellow"/>
<line x1="20" y1="50" x2="300" y2="50" stroke-width="10" stroke="red"
stroke-dasharray="20 5"/>
<line x1="20" y1="80" x2="300" y2="80" stroke-width="10" stroke="blue"
stroke-dasharray="20 5 10"/>
</svg>
```

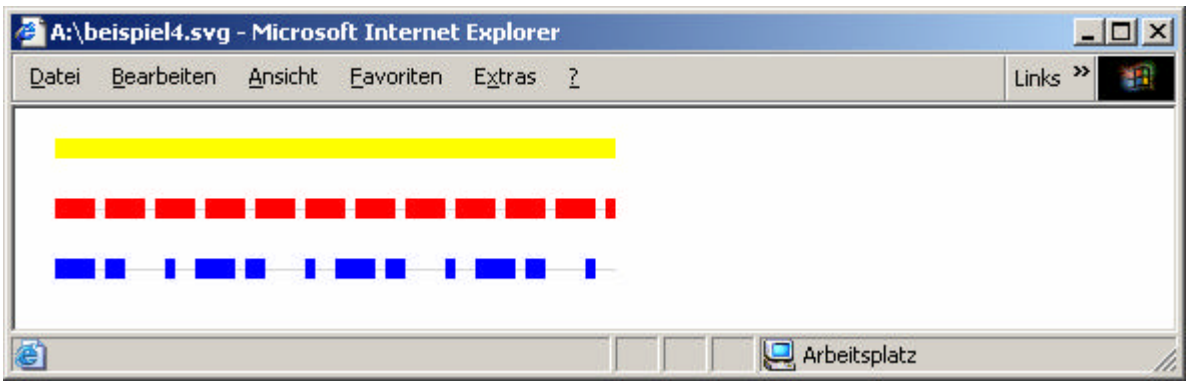

Abbildung 27: Linien mit SVG

Um Grafikelementen eine Farbe zuzuordnen, wurden in den bisherigen Beispielen benannte Farbwerte wie zum Beispiel yellow, red, blue und black verwendet. Hierbei ist man in der Farbwahl stark eingeschränkt. Es gibt jedoch noch eine weitere Möglichkeit Farben zu definieren, bei der Farbwerte entsprechend des RGB-Schemas angegeben werden. Für den Rot-, Grün-, und Blaukanal wird jeweils ein Wert zwischen 0 und 255 gewählt. Auch möglich ist eine Angabe in Prozent, die zwischen 0% und 100% liegt. Für die Farbe blau sieht dies wie folgt aus:

```
rgb(0,0,255)
rgb(0%,0%,100%)
```

Als Alternative ist auch die Hexadezimalschreibweise möglich:

```
#0000ff
```

2.10.7 Polylinie und Polygon

Eine Polylinie wird mit dem Tag `<polyline>` gezeichnet, ein Polygon mit `<polygon>`. Innerhalb des Attributs `points` wird eine Punkteliste angegeben, die das Polygon bzw. die Polylinie beschreibt. Die x- und y-Werte werden jeweils durch ein Komma getrennt. Zwischen zwei Koordinatenpaaren steht ein Leerzeichen. Für die einzelnen Koordinatenwerte können hierbei auch Gleitkommazahlen angegeben werden. Der Unterschied zwischen der Polylinie und dem Polygon besteht darin, dass beim Polygon im Gegensatz zur Polylinie der Pfad automatisch geschlossen wird. D.h., dass der Endpunkt automatisch mit dem Anfangspunkt verbunden wird.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<polyline points="20,20 20,80 60,80 60,120 120,120" stroke-width="2"
stroke="rgb(0,0,0)" fill="none"/>
<polygon points="140,20 140,80 180,80 180,120 240,120"
stroke-width="2" stroke="rgb(0,0,0)" fill="none"/>
</svg>
```

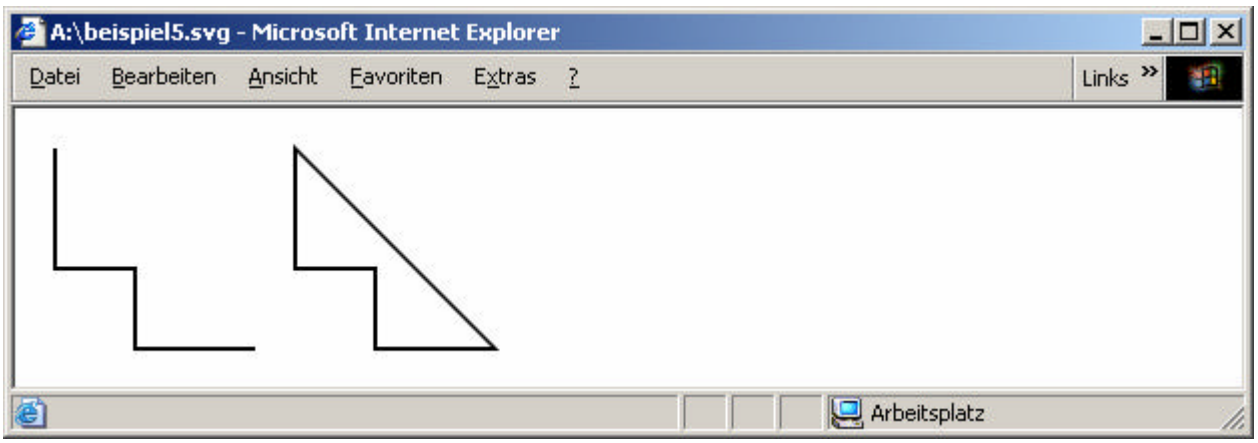

Abbildung 28: Polyline und Polygon mit SVG

2.10.8 Beliebige Pfade

Die Grundformen in SVG decken bereits einen großen Teil der möglichen Gestaltungsmittel ab. Bisher können aber noch keine Pfade erstellt werden, die sowohl aus Eck- als auch aus Kurvenpunkten bestehen. Hierfür gibt es das Tag `<path>`. Über dessen Attribut `d` werden die Knotenpunkte in ihrer Lage und Beschaffenheit genau beschrieben. Hierzu werden verschiedene Punktbefehle verwendet, von denen die Wichtigsten in der folgenden Tabelle beschrieben sind:

Tabelle 3: Punktbefehle und Beschreibungen für das path-Element in SVG

Punktbefehl	Beschreibung
`M[X][Y]`	moveto: damit wird die Position des Startpunktes angegeben, oder es wird von einem Punkt zum folgenden Punkt übergegangen, ohne dass eine Linie gezeichnet wird
`m[?x][?y]`	
`Z`	closepath: hiermit wird der Endpunkt mit dem Startpunkt verbunden, ohne dass der Startpunkt noch einmal extra angegeben werden muss. D.h. der Pfad wird automatisch geschlossen
`z`	
`L[X][Y]`	lineto: erzeugt eine direkte Linie zu dem Punkt mit den angegebenen Koordinaten
`l[?x][?y]`	
`H[X]`	horizontal lineto: zeichnet eine horizontale Linie zu dem Punkt, dessen x-Koordinate angegeben wurde
`h[?x]`	
`V[Y]`	vertical lineto: zeichnet eine vertikale Linie zu dem Punkt, dessen y-Koordinate angegeben wurde
`v[?y]`	

A[RX][RY] [X-Rotation] [Ellipsenbogen] [Richtung des Ellipsenbogens] [X][Y]	elliptical arc: zeichnet eine elliptische Kurve zum Punkt [X][Y] mit den Radien RX und RY. Durch die Angabe der X-Achsenrotation kann die Ellipse gedreht werden. Der Parameter für den Ellipsenbogen kann zwei unterschiedliche Werte annehmen: „0“, falls der kleine und „1“, falls der große Ellipsenbogen ausgewählt werden soll. Durch die Richtung des Ellipsenbogens kann gewählt werden, ob die Ellipse gegen den Uhrzeigersinn (0), oder im Uhrzeigersinn (1) verlaufen soll
a[RX][RY] [X-Rotation] [Ellipsenbogen] [Richtung des Ellipsenbogens] [?x][?y]	

Die Verwendung von Groß- und Kleinbuchstaben hat bei den vorgestellten Punktbefehlen folgende Bedeutung. Großbuchstaben kennzeichnen absolute Koordinaten, die auf den Nullpunkt bezogen sind, Kleinbuchstaben dagegen kennzeichnen absolute Koordinaten, die sich auf den vorangegangenen Punkt beziehen.

Zum besseren Verständnis folgen nun zwei Beispiele:

1.Beispiel: Verwendung von moveto, lineto, horizontal lineto und vertical lineto

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<path d="M20,80 L70,30 V80 H150" stroke-width="3"
stroke="rgb(0%,0%,100%)" fill="none" />
</svg>
```

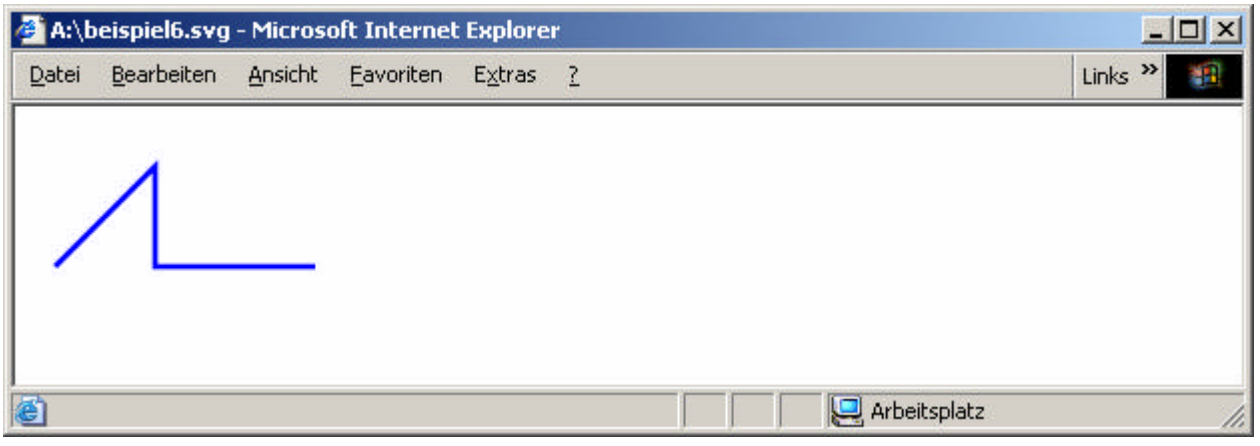

Abbildung 29: Verwendung der Punktbefehle moveto, lineto, horizontal und vertical lineto

2.Beispiel: Verwendung von elliptical arc und closepath

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<path d="M50,20 A100 35 0 1 1 50 120 Z" stroke-width="3"
stroke="#0000ff" fill="none" />
<path d="M350,20 A100 35 -30 1 0 350 120" stroke-width="3"
stroke="#0000ff" fill="none" />
</svg>
```

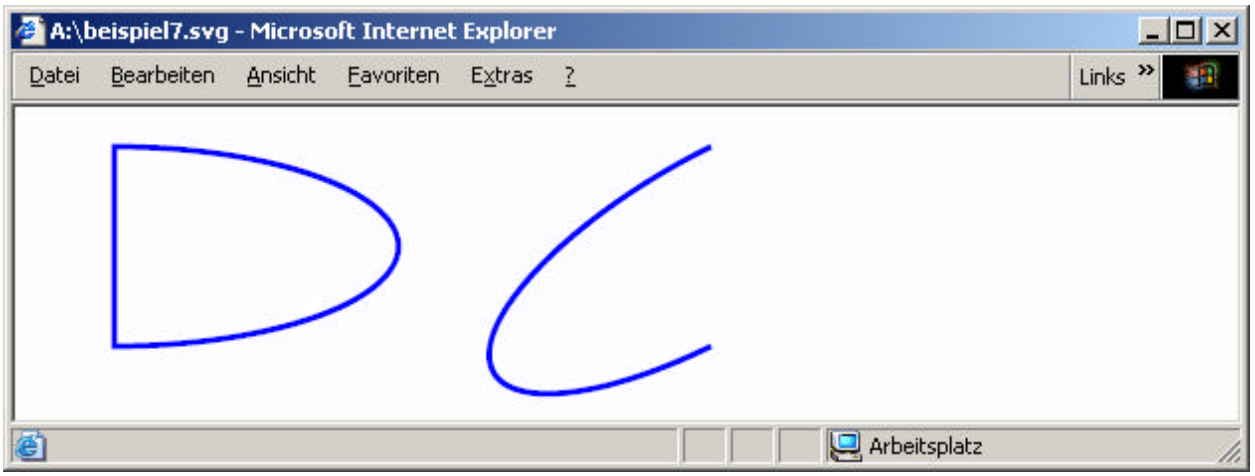

Abbildung 30: Verwendung der Punktbefehle elliptical arc und closepath

Abbildung 31[33] zeigt noch einmal die Auswirkungen der Parameter Ellipsenbogen und Richtung des Ellipsenbogens des Punktbefehls elliptical arc:

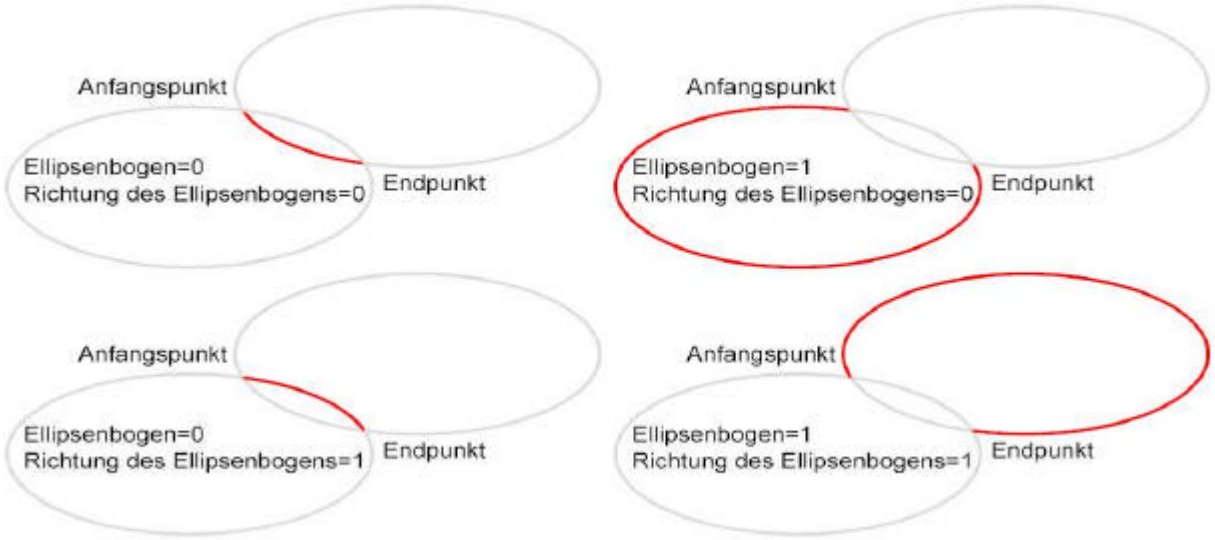

Abbildung 31: Die Parameter der elliptischen Kurve

[33] Quelle: Diplomarbeit „Visualisierung von ALK-Daten mittels SVG“ von Thomas Mailänder

2.10.9 Texte

Um Texte in SVG zu erzeugen wird das Tag `<text>` verwendet. Die Position für einen Text sollte explizit angegeben werden, da für die Attribute `x` und `y` standardmäßig der Wert 0 voreingestellt ist. Diese Werte beziehen sich auf den Startpunkt der Grundlinie, die sich unterhalb des Textes befindet, daher würde sich der Text sonst außerhalb des sichtbaren Bereiches befinden. Mit Hilfe des Attributs `text-anchor` kann der Text zentriert, links- oder rechtsbündig ausgegeben werden. Die Schriftart kann durch das Attribut `font-family`, die Schriftgröße durch `font-size` angegeben werden. Unter Verwendung von `<tspan>` können mehrzeilige Texte erstellt und durch die Transformation `transform="rotate(...)"` auch gedreht werden.

Folgendes Listing zeigt einige der vielen Möglichkeiten, die es bei der Gestaltung von Texten in SVG gibt: ⁇

```
<?xml version="1.0"?>
<svg width="100%" height="100%">
<text x="40" y="20" text-anchor="start">links</text>
<text x="40" y="40" text-anchor="middle">mitte</text>
<text x="40" y="60" text-anchor="end">rechts</text>
<text x="100" y="20" font-family="Arial">Arial</text>
<text x="100" y="40" font-family="Courier">Courier</text>
<text x="180" y="20" font-size="10pt">10 Punkt</text>
<text x="180" y="60" font-size="20pt">20 Punkt</text>
<text>
<tspan x="300" dy="20">Dieser Text</tspan>
<tspan x="300" dy="20">ist mehrzeilig</tspan>
</text>
<g transform="rotate(45,400,20)">
<text x="400" y="20">gedreht</text>
</g>
</svg>
```

Abbildung 32: Texte in SVG

2.10.10 Gruppierungen und Transformationen

Im letzten Beispiel wurde für die Drehung des Textes das Tag `<g>` verwendet, welches dazu dient mehrere Grafikelemente, die innerhalb dieses Tags aufgeführt sind, zu einer Einheit zusammenzufassen. Dies wird benötigt, um beispielsweise Transformationen oder Formatierungen auf das ganze Gebilde anzuwenden. Generell werden Transformationen mit dem Attribut `transform` gekennzeichnet. Mit Hilfe von `transform` können die Grundtransformationen Skalierung, Rotation, Translation und Neigung durchgeführt werden.

Skalierung: hierbei können Grafikelemente verkleinert oder vergrößert werden. Der Befehl dafür lautet `scale()`. Angegeben werden die Skalierungsfaktoren für die x- und y-Werte.

Rotation: damit kann ein Grafikelement gedreht werden. Im Gegensatz zu den meisten Grafikprogrammen wird hier bei positiven Winkelwerten im Uhrzeigersinn gedreht. Der Winkel wird in Grad angegeben. Der Befehl dazu lautet `rotate()`. Angegeben wird der Rotationswinkel und optional die Koordinatenwerte für den Rotationsmittelpunkt.

Translation: wird benötigt um ein Grafikelement zu verschieben. Der Befehl lautet `translate()`. Angegeben werden hierbei die Verschiebungswerte für die x- und optional für die y-Richtung.

Neigung: dabei wird der Winkel zwischen den beiden Koordinatenachsen geändert. Bei `skewX()` wird die x-Achse bei positivem Neigungswinkel gegen den Uhrzeigersinn gedreht, bei `skewY()` hingegen die y-Achse im Uhrzeigersinn gedreht.

Die verschiedenen Transformationen können untereinander auch kombiniert werden. Zur Veranschaulichung folgt ein Beispiel, in dem die verschiedenen Transformationen durchgeführt werden.

```
<?xml version="1.0"?>
<svg width="100%" height="100%">

<style type="text/css">
<![CDATA[
rect {stroke-width:3;stroke:red;fill:yellow}
text {font-size:12}
]]>
</style>

<g>
<rect x="0" y="0" width="150" height="40"/>
<text x="12" y="25">Ohne Transformation</text>
</g>

<g transform="translate(75,45)">
<rect x="0" y="0" width="150" height="40"/>
<text x="12" y="25">Translation</text>
</g>

<g transform="rotate(45,30,70),translate(30,70)">
<rect x="0" y="0" width="150" height="40"/>
<text x="5" y="25">Rotation und Translation</text>
```

```
</g>
<g transform="translate(0,210),scale(3,1)">
<rect x="0" y="0" width="150" height="40"/>
<text x="5" y="25">Translation und Skalierung</text>
</g>
<g transform="translate(270,0),skewX(30)">
<rect x="0" y="0" width="150" height="40"/>
<text x="5" y="25">Translation und Neigung X</text>
</g>
<g transform="translate(270,60),skewY(30)">
<rect x="0" y="0" width="150" height="40"/>
<text x="5" y="25">Translation und Neigung Y</text>
</g>
</svg>
```

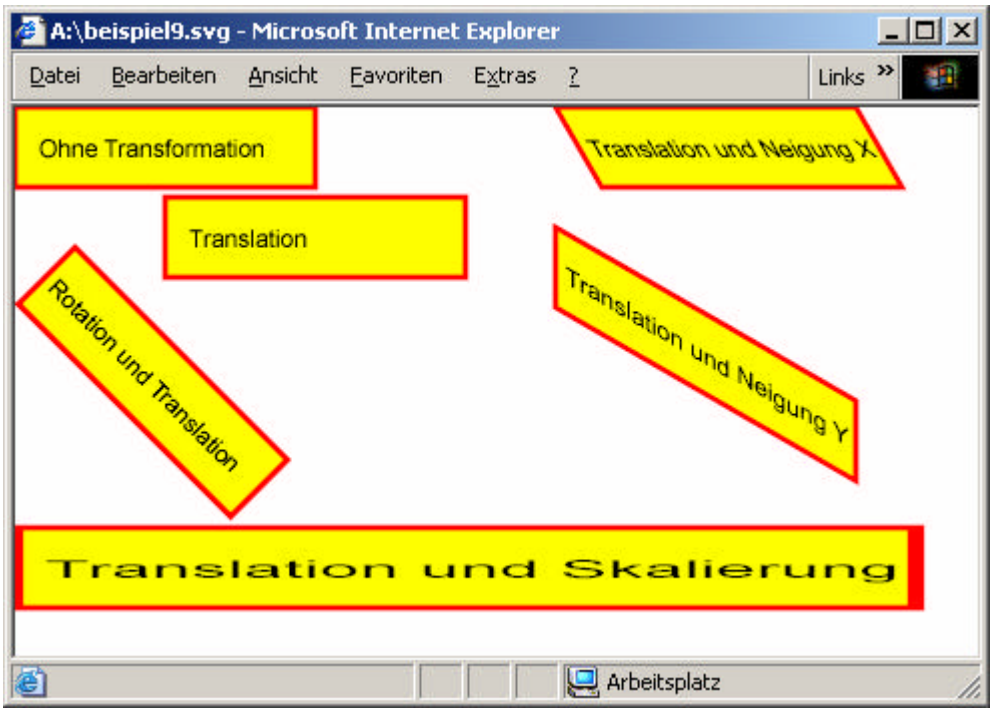

Abbildung 33: Transformationen in SVG

2.10.11 Cascading Style Sheets innerhalb von SVG

Im vorangegangenen Beispiel wurden Formatierungen, wie die Strichstärke, die Strichfarbe und die Füllfarbe nicht über Attribute dem jeweiligen Element zugeordnet, stattdessen wurde zu Beginn des SVG-Dokumentes ein CSS-Bereich eingefügt. Der Bereich folgt direkt auf das SVG-Element und sieht folgendermaßen aus:

```
<style type="text/css">
<![CDATA[
```

```
]]>
</style>
```

Für das Style-Tag wird der MIME-Type als Attribut `type="text/css"` angegeben. Da der CSS-Teil nicht vom XML-Parser ausgewertet werden soll, wird er durch <![CDATA[und]]> umschlossen. Der XML-Parser gibt dann den Inhalt unverändert weiter, so dass dieser vom CSS-Parser ausgewertet werden kann.

Um die erstellten Formatdefinitionen gezielt auf bestimmte Elemente zu übertragen, gibt es folgende Möglichkeiten:

? **Elementname:** weist man einem Elementnamen ein Format zu, so erhalten alle Elemente des gleichen Typs die angegebene Gestaltung, soweit sie nicht durch weitere Formatangaben in Form von Attributen ersetzt werden.

  Beispiel:

  ```
  rect{stroke-width:3;stroke:red;fill:yellow}
  ```

  Somit werden alle Rechtecke, die keine eigenen Formatangaben besitzen, mit der Strichbreite 3, der Strichfarbe rot und der Füllfarbe gelb gezeichnet:

  ```
  <rect x="0" y="0" width="150" height="40"/>
  ```

? **Klassenname**: der Vorteil bei der Verwendung von Klassennamen liegt darin, dass einem Element mehrere Stylesheets zugewiesen werden können. Auch möglich ist es dem Klassennamen einen bestimmten Elementnamen voranzustellen, um dieses Stylesheet nur für gleichnamige Elemente zugänglich zu machen.

  Beispiel:

  ```
  .style1 {stroke:red;fill:yellow}
  .style2 {stroke-width:3}
  rect.style3 {stroke-width:3;stroke:red;fill:yellow}
  ```

  Über das Attribut `class` lassen sich die so erstellten Stylesheets, unter Verwendung des Klassennamens, den Elementen zuweisen:

  ```
  <rect class=”style1 style2” x="0" y="0" width="150" height="40"/>
  <rect class=”style3” x="0" y="0" width="150" height="40"/>
  ```

? **ID-String:** Einem Element kann über das Attribut `id`, ein zuvor definiertes Stylesheet zugewiesen werden. Genau wie auch bei den Klassennamen kann dem ID-String ein Elementname vorangestellt werden.

  Beispiel:

  ```
  #style1 { stroke-width:3;stroke:red;fill:yellow }
  circle#style2 { stroke-width:2;stroke:blue;fill:red }
  ```

  Die Zuweisung zu einem Element funktioniert dann wie folgt:

  ```
  <rect id=”style1” x="0" y="0" width="150" height="40"/>
  <circle id=”style2” x="0" y="0" width="150" height="40"/>
  ```

3 DFK - Das Datenaustauschformat der bayerischen Vermessungsverwaltung

Die Digitale Flurkarte (DFK) ist neben dem Automatisierten Liegenschaftsbuch (ALB) der zweite Bestandteil des amtlichen Grundstücks- und Bodeninformationssystems (GRUBIS) und enthält die parzellenscharfe Darstellung der Liegenschaften, also aller Flurstücke und Gebäude. Außerdem werden Nutzungsartengrenzen und wichtige topographische Gegenstände dargestellt.

Zum Austausch von Digitalen Flurkarten wird das DFK[34]-Schnittstellenformat verwendet. Die Angaben der Digitalen Flurkarte werden hierbei in Punkte, Linien, Texte und Symbole unterteilt. Die Daten der Digitalen Flurkarte sind entweder als Komplettdatenaustausch, d.h. als vollständiger Datenbestand oder als Differenzdatenaustausch, ab einem vom Empfänger vorgegebenen Zeitpunkt, zu erhalten. Differenzdaten sind dabei nur sinnvoll, wenn sie an einen Komplettdatenaustausch anbinden.

Der Inhalt einer DKK-Datei besteht aus verschiedenen Datensätzen, wobei jeder Datensatz der DFK mit einer Kennung beginnt, die sich aus einem Vorzeichen und einer Kennzahl zusammensetzt. Das Vorzeichen gibt Auskunft darüber, ob es sich um einen Lösch (-) oder Einfügeblock (+) handelt, während die Kennzahl den Typ des Datensatzes angibt.

Im folgenden Abschnitt werden die Inhalte der verschiedenen Datensätze beschrieben:

Startsatz: der erste Datensatz in einem Block ist immer der Startsatz. In ihm werden allgemeine Angaben zum Datenaustausch und zum Block angegeben. Er ist in folgende Felder aufgegliedert:
Feld 1: Kennung
Feld 2: Kennung der datenabgebenden Stelle
Feld 3: Kennung für das Datenaustauschverfahren, KG bedeutet zum Beispiel Komplettdatenaustausch - Digitale Flurkarte und DG Differenzdatenaustausch - Digitale Flurkarte
Feld 4: beim Komplettdatenaustausch Zeitpunkt der Datenausspielung aus der Datenbank, bei Differenzdaten Zeitpunkt der Fortführung
Feld 5: Minimaler Rechtswert des Rechteckfensters
Feld 6: Minimaler Hochwert des Rechteckfensters
Feld 7: Maximaler Rechtswert des Rechteckfensters
Feld 8: Maximaler Hochwert des Rechteckfensters

Beispiel:

```
-.9999$00077$DG$.712506343$.37917233$.34545178$.37961126$.34575976
```

[34] Beschreibung unter: http://www.geodaten.bayern.de/datri.html
Testdaten unter: http://www.geodaten.bayern.de/download/dos/testdaten-dos.html

Gebietssatz: ein Block repräsentiert ein bestimmtes Gebiet aus der Digitalen Flurkarte. Die Grenzen dieses Gebiets werden durch ein Umfangspolygon festgelegt. Alle vorhandenen Gebietssätze definieren dieses Umfangspolygon. Ein Gebietssatz setzt sich wie folgt zusammen:
Feld 1: Kennung
Feld 2: Rechtswert eines Punktes des Umfangspolygons
Feld 3: Hochwert eines Punktes des Umfangspolygons

Beispiel:

```
-.9998$.37917233$.34545178
```

Koordinatensatz: durch einen Koordinatensatz wird ein Grenz-, Gebäude-, Katasterfest- und ein sonstiger Punkt ausgetauscht, er beinhaltet die punktbezogenen Daten des Koordinatenarchivs und ist in folgende Felder gegliedert:
Feld 1: Kennung
Feld 2: Angabe der Flurkarte innerhalb derer sich der Punkt befindet
Feld 3: Punktnummer
Feld 4: Rechtswert des Punktes, wird in cm angegeben, die Meridiankennziffer wird weggelassen
Feld 5: Hochwert des Punktes, wird in cm angegeben, aber ohne die 1000 km-Stelle
Feld 6: Abmarkungsart, beinhaltet die Kennzahlen (Symbol) der Punkte der Vermessungsverwaltung.
Feld 7: Kennung über Art und Entstehung der Koordinaten

Tabelle 4: Art und Entstehung von Koordinaten

Kennung	Art der Koordinaten	Entstehung durch	JSP
0	d-Koordinaten	Homogenisierung	fehlt / vorläufig
1	d-Koordinaten	exakten numerischen Ansatz	fehlt / vorläufig
4	d-Koordinaten	Homogenisierung	endgültig
5	d-Koordinaten	exakten numerischen Ansatz	endgültig
7	g-Koordinaten	Zuordnung bei Homogenisierung	endgültig
8	g-Koordinaten	exakten numerischen Ansatz	endgültig

Die Angabe d-Koordinate in Tabelle 4 bedeutet, dass es sich um genaue Koordinaten handelt, g-Koordinate dagegen steht für Koordinaten, die nur eine Genauigkeit im Dezimeterbereich besitzen.

Feld 8: Letzter schreibender Zugriff

Beispiel:

```
+.9990$20053900$5466$.37927233$.34560740$11$8$.712506343
```

Liniensatz: dient zum Austausch der Linienelemente der Digitalen Flurkarte und wird in folgende Felder unterteilt:
Feld 1: Kennung
Feld 2: Flurkarte - Linienanfang
Feld 3: Punktnummer - Linienanfang
Feld 4: Flurkarte - Linienende
Feld 5: Punktnummer – Linienende

Beispiel:

```
-.6000$20053900$5466$20053900$9242
```

Bogensatz: wird zum Austausch der Bogenelemente der Digitalen Flurkarte verwendet. Der Datensatz ist in folgende Felder gegliedert:
Feld 1: Kennung
Feld 2: Flurkarte - Bogenanfang
Feld 3: Punktnummer – Bogenanfang
Feld 4: Flurkarte - Bogenende
Feld 5: Punktnummer - Bogenende
Feld 6: Bogenradius in cm. Position des Mittelpunktes wird durch das zusätzliche Vorzeichen festgelegt. Ein + bedeutet Mittelpunkt rechts, ein – Mittelpunkt links, jeweils vom Bogenanfang aus gesehen.

Beispiel:

```
+.6001$30940544$.414$30940544$.375$-..540
```

Textsatz / Symbolsatz: dient zum Austausch von Textelementen/Symbolen der digitalen Flurkarten
Feld 1: Kennung
Feld 2: Flurkarte
Feld 3: Rechtswert der tatsächlichen Text-/Symbolposition
Feld 4: Hochwert der tatsächlichen Text-/Symbolposition
Feld 5: Differenz im Rechtswert zum Text- / Symbolbezugspunkt
Feld 6: Differenz im Hochwert zum Text- / Symbolbezugspunkt
Feld 7: Richtungswinkel des Textes / Symbols in gon mit drei Nachkommastellen
Feld 8: Textinhalt / Kennzahl (Symbol), beinhaltet beim Textsatz den Inhalt des auszutauschenden Textes oder beim Symbolsatz die Kennzahl (Symbol).

Beispiel:

```
+.4450$20053905$.37924627$.34560569$+....0$+....0$358.000$160
+.4500$20053905$.37926663$.34553478$+....0$+....0$.85.685$St.1497
```

Schlusssatz: Der letzte Datensatz in jedem Block ist der Schlusssatz. Er beinhaltet nur die Kennung.

Beispiel:

```
-....1
```

4 Das Programm DFKviewer

4.1 Einlesen einer DFK-Datei und Bilden von Objekten

4.1.1 Dateiupload vom Client zum Server

Wird innerhalb des erstellten Programms DFKviewer der Hyperlink „zum Programm“ ausgewählt, so wird eine Darstellung wie in Abbildung 34 erhalten.

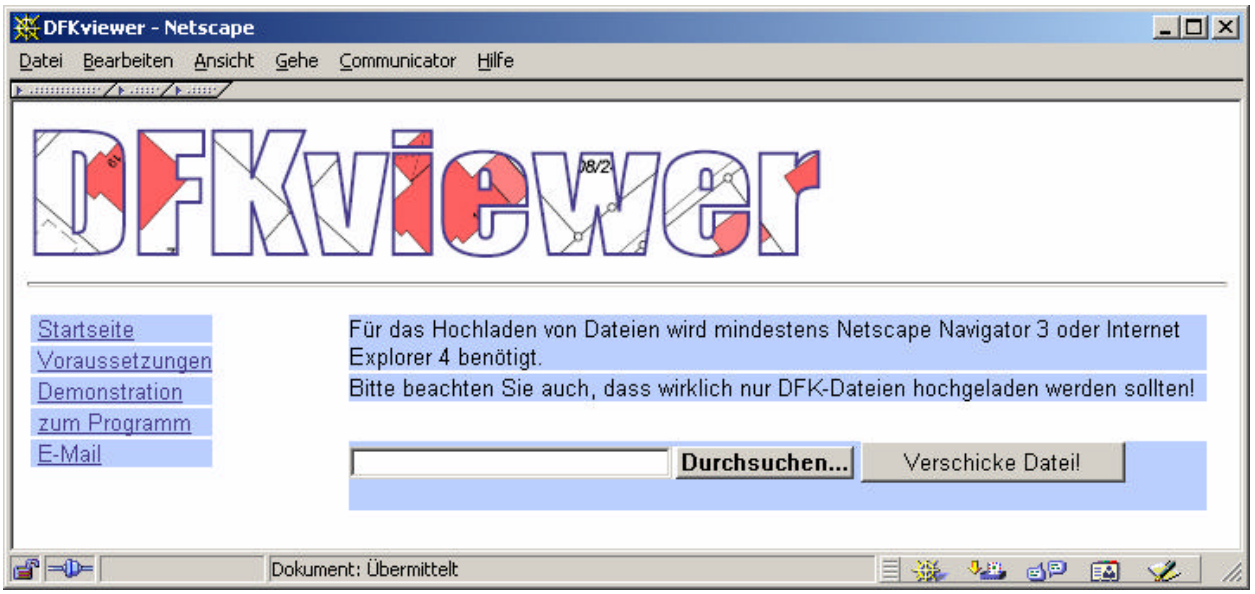

Abbildung 34: Upload einer DFK-Datei mit dem DFKviewer

Innerhalb des weißen Textfeldes kann nun eine Datei angegeben oder zur Erleichterung über den Durchsuchen-Button im Dateisystem ausgewählt werden. Wird anschließend der Button „Verschicke Datei“ gedrückt, so wird die zuvor angegebene Datei vom Client auf den Server übertragen und kann dort verarbeitet werden.

Der Teil für die Clientseite ist hierbei einfach zu schreiben, da Dateien als Formulardaten gesendet werden können. Durch das HTML-Tag `<form>` wird ein Formular erzeugt, wobei das Attribut `enctype`, welches den Inhaltstyp angibt, auf `multipart/form-data` gesetzt werden muss. Des weiteren ist beim Upload einer Datei als Wert für das Attribut `method` die Angabe `POST` zu wählen. Über das Attribut `action` hingegen wird angegeben welches Servlet für die weitere Verarbeitung zuständig ist.

Die Angabe `<input type="file" name="fupload">` innerhalb des `<form>`-Tags hat zu Folge, dass das Textfeld zur Eingabe der Datei, sowie der Button zum Auswählen der Datei im Dateisystem erstellt wird.

Der zweite Button, über den die Anfrage abgeschickt werden kann, wird durch `<input type="submit" value="Verschicke Datei!">` erstellt, wobei über das Attribut `value` die Beschriftung für den Button angegeben wird.

Im Zusammenhang sieht dies wie folgt aus:

```
<form name="fileForm" enctype="multipart/form-data" method="POST"
action="/servlet/dateiauslesen">
<input type="file" name="fupload">
<input type="submit" value="Verschicke Datei!">
</form>
```

Der Teil für die Serverseite sieht etwas komplizierter aus. Aus der Perspektive des Servlets ist die Sendung nur ein Rohdatenstrom, der entsprechend dem Inhalttyp `multipart/form-data` formatiert wurde. Die Servlet-Api liefert aber keine Methoden die beim Parsen der Daten helfen. Zur Lösung des Problems existiert aber die Klasse Multipart Request[35] von Jason Hunter, die über eine import-Anweisung in das eigene Servlet eingebunden werden kann.

Die durch Klasse MultipartRequest bereitgestellten Funktionen können, wie im folgenden Listing gezeigt, eingesetzt werden. Die Erläuterung des Codes wird innerhalb der Kommentare vorgenommen.

```
import java.io.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.oreilly.servlet.MultipartRequest;

public class dateiauslesen extends HttpServlet

{
 public void doPost(HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException
 {
  res.setContentType("text/html");
  PrintWriter out = res.getWriter();

  try
  {
   String nachricht = "";
   // Erstellung eines Objekts multi der Klasse MultipartRequest als
   // Hilfe beim Lesen der Informationen
   // Übergeben wird die Anfrage, das Verzeichnis zum Speichern der
   // hochgeladenen Datei, sowie die maximale Dateigröße
   MultipartRequest multi = new MultipartRequest(req, "c:/temp", 1
   * 1024 * 1024);
```

```
// gibt die Namen aller hochgeladenen Dateien als Enumeration von
// String-Objekten zurück, ein Dateiname ist durch das
// Namensattribut im HTML-Formular festgelegt
Enumeration files = multi.getFileNames();
if (files.hasMoreElements()==true)
{
 String name = (String)files.nextElement();
 // gibt Namen der Datei im Dateisystem zurück, wie vom Benutzer
 // angegeben
 String filename = multi.getFilesystemName(name);
 // Kontrolle, ob Benutzer einen Dateinamen angegeben hat,
 // ansonsten zurück zum Dateiupload
 if (filename == null)
 {
  nachricht = "Sie haben keine Datei ausgewählt!";
  req.setAttribute("nachricht", nachricht);
  RequestDispatcher dispatcher =
  req.getRequestDispatcher("/DFKviewer/dateiupload.jsp");
  dispatcher.forward(req, res);
 }
 // gibt ein File-Objekt für die angegebene, auf dem Dateisystem
 // des Servers gespeicherte Datei zurück
 File f = multi.getFile(name);
 // Kontrolle, ob Datei im Upload enthalten war, ansonsten zurück
 // zum Dateiupload
 if (f == null)
 {
  nachricht = "Die gewählte Datei war nicht im Upload enthalten!";
  req.setAttribute("nachricht", nachricht);
  RequestDispatcher dispatcher =
  req.getRequestDispatcher("/DFKviewer/dateiupload.jsp");
  dispatcher.forward(req, res);
 }
 // gibt Länge der Datei als long in Byte zurück
 // Kontrolle, ob die gewählte Datei überhaupt existiert oder leer
 // ist, ansonsten zurück zum Dateiupload
 if (f.length() == 0)
 {
  f.delete();
  nachricht = "Die gewählte Datei existiert nicht, oder ist leer!";
  req.setAttribute("nachricht", nachricht);
  RequestDispatcher dispatcher =
```

[35] Quelle: http://www.servlets.com/cos/index.html

```
 req.getRequestDispatcher("/DFKviewer/dateiupload.jsp");
 dispatcher.forward(req, res);
}
// gibt den Inhaltstyp der gewählten Datei zurück
String type = multi.getContentType(name);
// Kontrolle, ob Inhaltstyp text/plain (wie im DFK-Format) ist,
// ansonsten zurück zum Dateiupload
if(!type.equals("text/plain"))
{
 f.delete();
 nachricht = "Der Inhaltstyp der gewählten Datei ist nicht
 identisch mit dem Inhaltstyp einer
 req.setAttribute("nachricht", nachricht);
 RequestDispatcher dispatcher =
 req.getRequestDispatcher("/DFKviewer/dateiupload.jsp");
 dispatcher.forward(req, res);
}
// Einlesen der ersten Zeile der Datei
// Kontrolle, ob hochgeladene Datei ein Komplettdatenaustausch  im
// DFK-Format ist, ansonsten zurück zum Dateiupload
String zeile;
FileReader eingabeStrom = new FileReader(f);
BufferedReader eingabe = new BufferedReader(eingabeStrom);
if ((zeile = eingabe.readLine()) != null)
{
 if (!(zeile.substring(0,6)).equals("+ 9999"))
 {
  eingabe.close();
  eingabeStrom.close();
  f.delete();
  nachricht = "Die gewählte Datei ist kein Komplettdatenaustausch
  im DFK-Format!";
  req.setAttribute("nachricht", nachricht);
  RequestDispatcher dispatcher =
  req.getRequestDispatcher("/DFKviewer/dateiupload.jsp");
  dispatcher.forward(req, res);
 }
 else
 {
  // Hier erfolgt das Auslesen der Datei und die weitere
  // Verarbeitung
  ...
 }
}
```

```
   }
  }
  catch (Exception e)
  {
   out.println("<PRE>");
   e.printStackTrace(out);
   out.println("</PRE>");
  }
 }
}
```

Wie im vorangegangenen Listing zu sehen ist, wird beim DFKviewer nach der Auswahl einer Datei durch den Benutzer zuerst überprüft, ob:

? überhaupt ein Dateiname angegeben worden ist

? die Datei im Upload enthalten war

? die gewählte Datei überhaupt existiert oder leer ist

? der Inhaltstyp text/plain, wie im DFK-Format ist

? die hochgeladene Datei ein Komplettdatenaustausch im DFK-Format ist

Werden diese Kriterien nicht erfüllt, so bekommt der Benutzer eine entsprechende Fehlermeldung angezeigt und erhält erneut die Möglichkeit eine Datei auszuwählen. Ist die Datei aber ein Komplettdatenaustausch im DFK-Format, so kann die Verarbeitung der Datei beginnen.

4.1.2 Bilden von Objekten und Speichern von Objekten in Objektlisten

Nach erfolgtem Upload wird nacheinander jeder einzelne Datensatz der DFK-Datei serverseitig eingelesen und abhängig von seiner Kennung in ein zum Datensatz passendes Objekt gespeichert.

Die Informationen aus dem Start- und den Gebietssätzen werden in ein Objekt der Klasse Vordaten aufgenommen. Da der Startsatz immer die Kennung 9999, sowie die Gebietssätze immer die Kennung 9998 haben, werden diese Kennungen als konstante Klassenvariablen gespeichert. Aus dem im Gebietssatz angegebenen Rechtswert und Hochwert eines Punktes des Umfangspolygons wird ein Objekt der Klasse Koordinate gebildet und anschließend in einem Objekt der Klasse Vektor, das beliebige Objekte aufnehmen kann, eingefügt. Für jeden weiteren Gebietssatz wird dann ein weiteres Element in den Vektor aufgenommen. Die Inhalte der übrigen Felder des Startsatzes werden in den entsprechenden Variablen der Klasse Vordaten abgelegt.

Tabelle 5 und 6 zeigen die Eigenschaften und Methoden der Klasse Vordaten, sowie der Klasse Koordinate.

Tabelle 5: Die Klasse Vordaten

Eigenschaften	Methoden
? Kennung für Startsatz ? Kennung für Gebietssatz ? Kennung der datenabgebenden Stelle ? Kennung für das Datenaustauschverfahren ? Zeitpunkt der Datenausspielung/Fortführung ? Minimaler Rechtswert des Rechteckfensters ? Maximaler Rechtswert des Rechteckfensters ? Minimaler Hochwert des Rechteckfensters ? Maximaler Hochwert des Rechteckfensters ? Umfangspolygon (Vector)	? Konstruktor ? get- und set-Funktion für das Umfangspolypon ? Ausgabe der Vordaten in eine HTML-Datei ? Ausgabe der Vordaten in eine XML-Datei

Tabelle 6: Die Klasse Koordinate

Eigenschaften	Methoden
? Rechtswert ? Hochwert	? Konstruktor ? get-Funktion für den Rechtswert und für den Hochwert

Für jeden Koordinatensatz wird ein eigenes Objekt der Klasse Punkt gebildet. Da Koordinatensätze alle die Kennung 9990 haben, reicht auch hier zum Speichen der Kennung wieder eine konstante Klassenvariable aus. Die übrigen Informationen werden auch hier wieder in die entsprechenden Variablen der Klasse Punkt gespeichert.

Tabelle 7 zeigt die in der Klasse Punkt enthaltenen Eigenschaften und Methoden.

Tabelle 7: Die Klasse Punkt

Eigenschaften	Methoden
? Kennung für Koordinatensatz ? Flurkarte ? Punktnummer ? Rechtswert ? Hochwert ? Abmarkungsart, d.h. Kennung Symbol ? Kennung für die Art und Entstehung der Koordinaten ? Letzter Zugriff	? Konstruktor ? get-Funktionen für den Rechtswert und für den Hochwert ? Ausgabe eines Punktes in eine HTML-Datei ? Ausgabe eines Punktes in eine XML-Datei

Ein erzeugtes Punkt-Objekt wird anschließend in eine Hashtable eines Objektes der Klasse Punktliste geschrieben. Eine Hashtable hat gegenüber Vector den Vorteil, dass Elemente über einen mitgespeicherten Schlüssel angesprochen werden können, was einen schnelleren Zugriff ermöglicht. Die Hashtable, in

die gespeichert werden soll, wird über das Feld Kennung Symbol im jeweiligen Datensatz ermittelt. Das Speichern in unterschiedlichen Listen ist notwendig, da so später die Punkte innerhalb der SVG-Datei durch das Tag <g> zu Gruppen zusammengefasst werden können. D.h. alle Trigonometrischen Boden- oder Hochpunkte können nacheinander innerhalb eines <g>-Tags ausgegeben werden, danach alle Katasterfestpunkte, wieder innerhalb eines eigenen <g>-Tags usw. Dies wäre nicht möglich, wenn man die Punkte in beliebiger Reihenfolge in einer einzigen Liste speichern würde. Durch die Gruppierungen ist es möglich, durch interaktive Funktionen, dem Benutzer die Möglichkeit zu geben z.B. alle Katasterfestpunkte auszublenden und alle Trigonometrischen Punkte einzublenden. Die Karte kann somit also nach den Wünschen des Benutzers gestaltet werden. Diese Aufteilung in verschiedene Listen wird bei den folgenden Klassen immer wieder aufgegriffen. Der Grund dafür ist jeweils der hier beschriebene.

In Tabelle 8 werden die Eigenschaften und Methoden der Klasse Punktliste angegeben.

Tabelle 8: Die Klasse Punktliste

Eigenschaften	Methoden
? Liste der Trigonometrischen Boden- und Hochpunkte (Hashtable) ? Liste der Katasterfestpunkte (Hashtable) ? Liste der Grenzpunkte - numerischer Grenznachweis (Hashtable) ? Liste der Grenzpunkte - graphischer Grenznachweis (Hashtable) ? Liste der sonstigen Grenzpunkte und der nicht abgemarkten Punkte (Hashtable)	? Konstruktor ? get- und set-Funktionen für alle Eigenschaften ? Ausgabe der Punkte in eine HTML-Datei ? Ausgabe der Punkte in eine XML-Datei

Auffallend beim DFK-Schnittstellenformat ist, dass es grundsätzlich keine feste Anordnung der graphischen Elemente gibt. Einzige Ausnahme davon sind die Grundrisselemente bei Gebäuden. D.h. die Linien und Bögen eines Gebäudes werden unmittelbar aufeinander folgend im Uhrzeigersinn übergeben. Dies gilt zum Beispiel aber nicht für Flurstücksgrenzen. Man erhält einzelne Linen und Bögen, die nicht einmal aufeinander folgen müssen. Linien und Bögen beinhalten folgende Informationen: Kennung, Flurkarte-Linienanfang, Punktnummer-Linienanfang, Flurkarte-Linienende, Punktnummer-Linienende. Die Kennung gibt in diesem Fall an, dass es sich um eine Flurstücksgrenze handelt. Es gibt aber nicht einmal einen Bezug zur Flurstücksnummer, die gesondert als Textelement, mit den Koordinaten, an denen der Text platziert werden soll, übergeben wird. Daher ist es nicht möglich aus den einzelnen Linien und Bögen eine zusammenhängende Flurstücksgrenze zu bilden. Aus diesem Grund habe ich eigene Klassen für Linien und Bögen gebildet. Für Gebäude, welche als Flächenelemente vorliegen, bestand dagegen die Möglichkeit, die einzelnen Linien und Bögen zu Umringen zusammenzusetzen und Gebäudeobjekte zu bilden.

Linien und Bögen besitzen eine Kennzahl, die im Bereich zwischen 6000 und 9899 liegt. Da jede Linie und jeder Bogen jeweils eine andere Kennzahl haben kann, je nachdem ob sie zu einer Flurstücksgrenze, Nutzungsartengrenze usw. gehören, muss hier die Kennung für jedes Objekt einzeln gespeichert werden. Eine einzige Klassenvariable ist also nicht mehr möglich, stattdessen wird eine Objektvariable benötigt.

Für jeden Liniensatz wird ein Objekt der Klasse Linie gebildet, für jeden Bogensatz ein Objekt der Klasse Bogen.

Beide Klassen sind in den Tabellen 9 und 10 dargestellt.

Tabelle 9: Die Klasse Linie

Eigenschaften	Methoden
? Kennung ? Flurkarte - Linienanfang ? Punktnummer - Linienanfang ? Rechtswert - Punkt Linienanfang ? Hochwert - Punkt Linienanfang ? Flurkarte - Linienende ? Punktnummer - Linienende ? Rechtswert - Punkt Linienende ? Hochwert - Punkt Linienende	? Konstruktor ? set-Funktionen für den Rechtswert und Hochwert des Anfangs- und des Endpunktes ? Ausgabe einer Linie in eine HTML-Datei ? Ausgabe einer Linie in eine XML-Datei

Tabelle 10: Die Klasse Bogen

Eigenschaften	Methoden
? Kennung ? Flurkarte - Bogenanfang ? Punktnummer - Bogenanfang ? Rechtswert - Punkt Bogenanfang ? Hochwert - Punkt Bogenanfang ? Flurkarte - Bogenende ? Punktnummer - Bogenende ? Rechtswert - Punkt Bogenende ? Hochwert - Punkt Bogenende ? Bogenradius	? Konstruktor ? set-Funktionen für den Rechtswert und Hochwert des Anfangs- und des Endpunktes ? Ausgabe eines Bogens in eine HTML-Datei ? Ausgabe einer Bogens in eine XML-Datei

Ein Punktobjekt wird anschließend in eine der Listen des Objektes der Klasse Linienliste geschrieben, ein Bogen in eine der Listen des Objektes der Klasse Bogenliste. Die Aufteilung in die entsprechenden Listen erfolgt in Abhängigkeit von der Kennung.

Tabelle 11 und 12 zeigen die Klassen Linienliste und Bogenliste.

Tabelle 11: Die Klasse Linienliste

Eigenschaften	Methoden
? Flurstücksgrenze - Linie (Vector) ? Gebäudedetail - Linie (Vector) ? Gebäudeinformation - Linie (Vector)	? Konstruktor ? get- und set-Funktionen für alle Eigenschaften ? Ausgabe der Linienlisten in eine HTML-Datei

? Nutzungsartengrenze - Linie (Vector) ? Topographie - Linie (Vector)	? Ausgabe der Linienlisten in eine XML-Datei

Tabelle 12: Die Klasse Bogenliste

Eigenschaften	Methoden
? Flurstücksgrenze - Bogen (Vector) ? Topographie - Bogen (Vector)	? Konstruktor ? get- und set-Funktionen für alle Eigenschaften ? Ausgabe der Bogenlisten in eine HTML-Datei ? Ausgabe der Bogenlisten in eine XML-Datei

Linien, welche die Kennung 6050, 6052, 6054 und 6100 haben, sowie Bögen mit der Kennung 6051, 6053 und 6101 gehören zu einem Gebäudegrundriss. Für sie wird ein Linien- bzw. Bogenobjekt gebildet. Alle Linien- und Bogenobjekte, die zu einem Gebäude gehören, werden in der richtigen Reihenfolge in ein Objekt der Klasse Vector geschrieben. Dieses Vektorobjekt wird dann jeweils in dem Vector des Objektes der Klasse Gebäudeliste abgelegt. Ob ein Linien- oder Bogensatz der letzte für ein bestimmtes Gebäude ist, oder ob noch weitere folgen, wird dadurch überprüft, indem der Endpunkt des aktuellen Datensatzes mit dem Startpunkt des ersten Datensatzes verglichen wird. Sind sie identisch, so ist der Umring für das Gebäude geschlossen. Sind sie es nicht, so müssen noch weitere Datensätze folgen.

Tabelle 13 zeigt die Eigenschaften und Methoden der Klasse Gebäudeliste.

Tabelle 13: Die Klasse Gebäudeliste

Eigenschaften	Methoden
? Gebäude (Vektor)	? Konstruktor ? get- und set-Funktionen für die Eigenschaft ? Ausgabe der Gebäude in eine HTML-Datei ? Ausgabe der Gebäude in eine XML-Datei

Textsätze haben eine Kennung zwischen 4000 und 5999 (ohne 4450 und 5450) oder zwischen 2000 und 3999 (ohne 2450 und 3450). Symbolsätze haben die Kennung 4450, 5450, 2450 oder 3450. Da die Kennung also auch hier unterschiedlich sein kann wird wieder eine Objektvariable und keine Klassenvariable zum Speichern der Kennung verwendet. Weil die Inhalte eines Textsatzes und eines Symbolsatzes zum Großteil gleich sind, habe ich eine Klasse Text_Symbol gebildet, welche die identischen Eigenschaften beinhaltet. Von dieser Klasse sind die Klassen Text und Symbol abgeleitet. Die Klasse Text speichert zusätzlich den Textinhalt, die Klasse Symbol dagegen die Kennung des Symbols.

In Tabelle 14 ist die Klasse Text_Symbol, von der die in Tabelle 15 und 16 dargestellten Klassen Text und Symbol abgeleitet sind, abgebildet.

Tabelle 14: Die Klasse Text_Symbol

Eigenschaften	Methoden
? Kennung ? Flurkarte ? Rechtswert der tatsächlichen Text/Symbolposition ? Hochwert der tatsächlichen Text-/Symbolposition ? Differenz im Rechtswert zum Text- / Symbolbezugspunkt ? Differenz im Hochwert zum Text- / Symbolbezugspunkt ? Richtungswinkel	? Konstruktor

Tabelle 15: Die Klasse Text

Abgeleitet von der Klasse Text_Symbol	
Eigenschaften	Methoden
? Textinhalt	? Konstruktor ? Ausgabe des Textes in eine HTML-Datei ? Ausgabe einer Textes in eine XML-Datei

Tabelle 16: Die Klasse Symbol

Abgeleitet von der Klasse Text_Symbol	
Eigenschaften	Methoden
? Kennung des Symbols	? Konstruktor ? Ausgabe des Symbols in eine HTML-Datei ? Ausgabe des Symbols in eine XML-Datei

Ein gebildetes Textobjekt wird abhängig von der Kennung in eine der Listen des Objektes der Klasse Textliste aufgenommen, ein Symbolobjekt dagegen abhängig von der Symbolkennung in ein Objekt der Klasse Symbolliste. Die Symbolkennung entspricht dabei nicht der Kennung zu Beginn des Symbolsatzes. Sie steht am Ende des Symbolsatzes und gibt an, welches Symbol verwendet werden soll. Dies kann zum Beispiel die Zahl 110 sein, was für Grünland – 7er Gruppe steht.

Tabelle 17 und 18 zeigen die Eigenschaften und Methoden der Klasse Textliste und der Klasse Symbolliste.

Tabelle 17: Die Klasse Textliste

Eigenschaften	Methoden
? Punktnummer/Beschriftung eines Landesgrenzsteins, Forstgrenzstein oder Grenzstein für Fischereirecht - Text (Vektor) ? Flurstücksnummer - Text (Vektor) ? Gebäudeinformation - Text (Vektor) ? Nutzungsartenbezeichnung - Text (Vektor) ? Topographie - Text (Vektor) ? Beschriftung1 - Text (Vektor) ? Beschriftung2 - Text (Vektor) ? Kartenrand - Text (Vektor)	? Konstruktor ? get- und set-Funktionen für die Eigenschaften ? Ausgabe der Texte in eine HTML-Datei ? Ausgabe der Texte in eine XML-Datei

Tabelle 18: Die Klasse Symbolliste

Eigenschaften	Methoden
? Flurstück – Symbol (Vector) ? Gebäudegrundriss - Symbol (Vector) ? Gebäudedetail - Symbol (Vector) ? Nutzungsartenbezeichnung - Symbol (Vector) ? Topographie - Symbol (Vector) ? Verwaltungs/Verfahrensgrenze – Symbol (Vector) ? Kartenrand - Symbol (Vector)	? Konstruktor ? get- und set-Funktionen für die Eigenschaften ? Ausgabe der Symbole in eine HTML-Datei ? Ausgabe der Symbole in eine XML-Datei

4.2 Erstellen der erforderlichen Dateien

4.2.1 Anlegen des Benutzerverzeichnisses und der benötigten Dateien

Nachdem die DFK-Datei ausgelesen und die gebildeten Objekte in die Objektlisten geschrieben wurden, wird erst einmal kontrolliert, ob bereits ein Ordner mit dem Namen DFK auf dem Server vorhanden ist. In diesem Ordner werden die Verzeichnisse für die einzelnen Benutzer angelegt. Ist der Ordner nicht vorhanden, so wird er erstellt. Existiert er bereits, so werden alle Ordner der einzelnen Anwender, die sich in ihm befinden, der Reihe nach durchgegangen. Sind sie veraltet, d.h. älter als eine Stunde, so werden erst die sich im Ordner befindenden Dateien und anschließend der Ordner selbst gelöscht, sind die Benutzerordner noch aktuell so bleiben sie erhalten. Anschließend wird für den aktuellen Benutzer innerhalb des DFK-Ordners ein eigener Ordner erstellt. Der Ordner bekommt als Namen die Anzahl der seit dem 1.1.1970, 00.00 Uhr, vergangenen Anzahl an Millisekunden. Diese Zahl ist über den Rückgabewert der Funktion `System.currentTimeMillis();` zu bekommen. Diese Art der Namensgebung ist sinnvoll, da so später sehr gut überprüft werden kann, ob ein Ordner veraltet ist und somit wieder gelöscht werden muss. In den neu erstellten Benutzerordner wird eine JavaScript-Datei kopiert, die später für die interaktiven Funktionen der Karte zuständig ist. Zusätzlich werden verschiedene HTML-Dateien, eine

XML-Datei und eine SVG-Datei neu angelegt, in die später die Ergebnisse der Verarbeitung der DFK-Datei geschrieben werden.

In der nächsten Abbildung ist der DFK-Ordner mit den jeweiligen Benutzerordnern zu sehen. Die darauf folgende Abbildung zeigt den Inhalt eines Benutzerordners. In diesem Fall wurden die Ergebnisse der Verarbeitung bereits in die entsprechenden Dateien geschrieben.

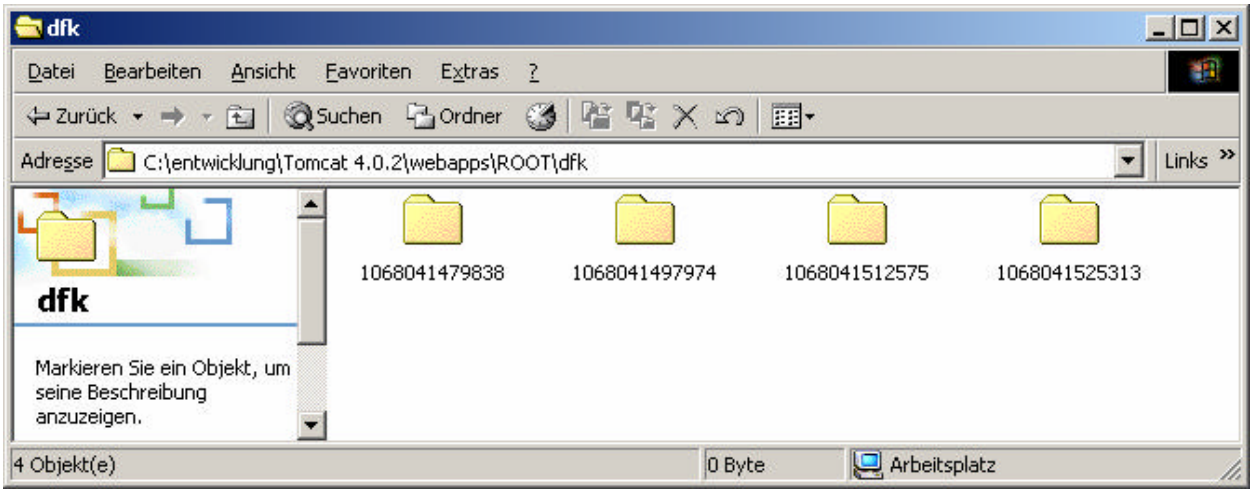

Abbildung 35: DFK-Ordner auf dem Server mit angelegten Benutzerverzeichnissen

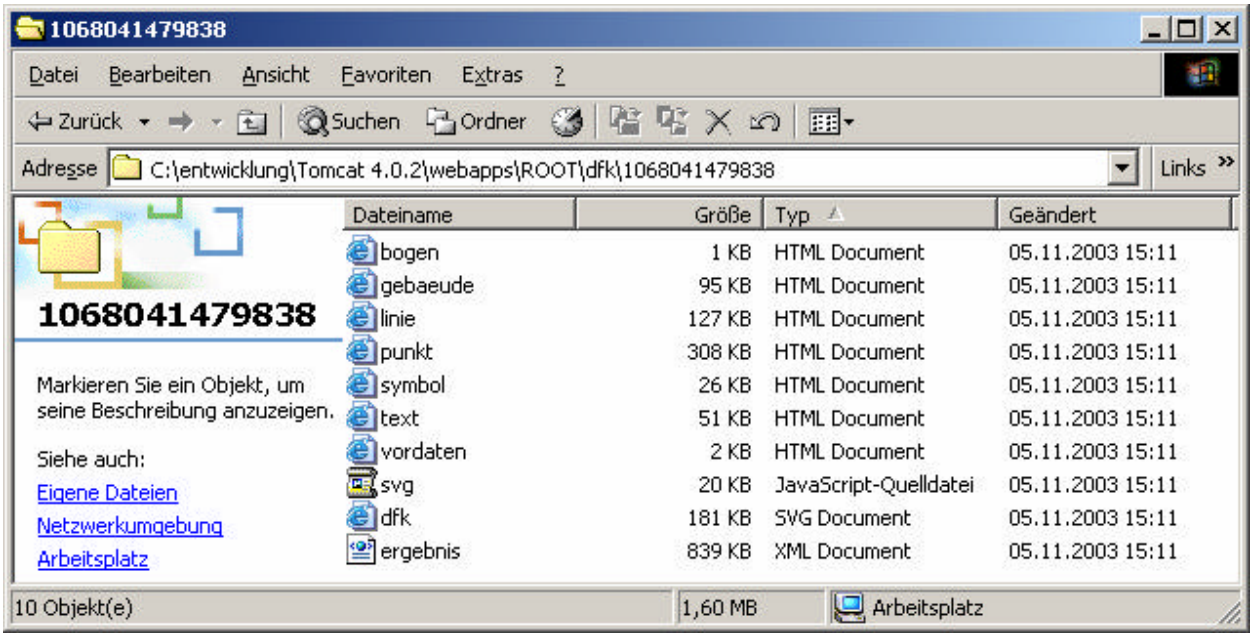

Abbildung 36: Inhalt eines Benutzerverzeichnisses auf dem Server

4.2.2 Generieren der HTML-Dateien

Die Informationen aus den Objektlisten werden nacheinander ausgelesen und in verschiedene HTML-Dateien geschrieben. Dazu wurden für die Klassen Vordaten, Punktliste, Linienliste, Bogenliste, Gebäudeliste, Textliste und Symbolliste jeweils eine Funktion zur Ausgabe der Listen in eine HTML-Datei geschrieben:

```
public void ausgabeHTML(dateien DFKdateien)
```

Des weiteren wurden für die Klassen Punkt, Linie, Bogen, Text und Symbol jeweils eine Funktion zur Ausgabe eines einzelnen Objektes implementiert:

```
public void ausgabeHTML(BufferedWriter bw)
```

Innerhalb der jeweiligen Funktion für die Ausgabe der Liste werden die einzelnen Objektlisten durchgegangen und für jedes Objekt die entsprechende Funktion zur Ausgabe eines einzelnen Objektes aufgerufen, welche dann das jeweilige Objekt in die entsprechende HTML-Datei schreibt.

Es werden 7 verschiedene HTML-Dateien generiert:

? vordaten.htm

? punkt.htm

? linie.htm

? bogen.htm

? gebäude.htm

? text.htm

? symbol.htm

Die Funktion für die Ausgabe der Linienliste sieht in stark gekürzter Version beispielsweise wie folgt aus:

```
public void ausgabeHTML(dateien DFKdateien)
{
 Linie linie;
 try
 {
  File ausgabeDatei = DFKdateien.getLinie_datei();
  FileWriter fw = new FileWriter(ausgabeDatei);
  BufferedWriter bw = new BufferedWriter(fw);
  bw.write("<html>"); bw.newLine();
  bw.write("<head>"); bw.newLine();
  bw.write("<title>Linienliste</title>"); bw.newLine();
  bw.write("<style type=\"text/css\">"); bw.newLine();
  bw.write("<!--"); bw.newLine();
  bw.write("  body { background-color:#bbcffe;font-family:Arial;font-
  size:10pt}"); bw.newLine();
  bw.write(" hr {color:#ffffff}"); bw.newLine();
  bw.write(" a:link {color:#483d8b;text-decoration:underline}");
  bw.newLine();
  bw.write(" a:visited {color:#483d8b;text-decoration:underline}");
  bw.newLine();
  bw.write(" a:hover {color:#5a5a5a;text-decoration:underline}");
  bw.newLine();
```

```
  bw.write(" a:active {color:#5a5a5a;text-decoration:none}");
  bw.newLine();
  bw.write("-->"); bw.newLine();
  bw.write("</style>"); bw.newLine();
  bw.write("</head>"); bw.newLine();
  bw.write("<body>"); bw.newLine();
  bw.write("<a href=\"#anfang\">zum Anfang</a><br><br><br>");
  bw.newLine();
  int anzahlLinien = flurstuecksgrenzeLinie_liste.size() +
  gebaeudedetailLinie_liste.size() +
  gebaeudeinformationLinie_liste.size() +
  nutzungsartengrenzeLinie_liste.size() +
  topographieLinie_liste.size();
  bw.write("<a name=\"anfang\"><hr>" +anzahlLinien +" Linien
  insgesamt</a><br>"); bw.newLine();
  bw.write("<br><a name=\"abschnittFS\"><hr><a
  href=\"#abschnittGD\">Gebäudedetail (Linie)</a> | <a
  href=\"#abschnittGI\">Gebäudeinformation (Linie)</a> | <a
  href=\"#abschnittNG\">Nutzungsartengrenze (Linie)</a> | <a
  href=\"#abschnittTOP\">Topographie (Linie)</a></a><br>");
  bw.newLine();
  bw.write("<br>* * * * * * * * * "
  +flurstuecksgrenzeLinie_liste.size() +" mal Flurstücksgrenze (Linie)
  * * * * * * * * *<br>"); bw.newLine();
  for(int i=0; i<flurstuecksgrenzeLinie_liste.size(); i++)
  {
   linie = (Linie)flurstuecksgrenzeLinie_liste.elementAt(i);
   linie.ausgabeHTML(bw);
  }
  // es folgt auf die gleiche Weise die Ausgabe für die Listen
  // Gebäudedetail (Linie)
  // Gebäudeinformation (Linie)
  // Nutzungsartengrenze (Linie)
  // Topographie (Linie)
  ...
  bw.write("<br><br><br><a href=\"#anfang\">zum Anfang</a>");
  bw.newLine();
  bw.write("</body>"); bw.newLine();
  bw.write("</html>"); bw.newLine();
  bw.close();
 }
  // hier folgt der catch-Block
  ...
}
```

Wie zu sehen ist wird in oben markierten for-Schleife jedes Element der Liste, welche die Flurstücksgrenzen enthält, durchgegangen und für jedes enthaltene Linienobjekt die Funktion `ausgabeHTML(bw)` der Klasse `linie` aufgerufen. Die eigentliche Ausgabe der Linie in die HTML-Datei erfolgt dort:

```
public void ausgabeHTML(BufferedWriter bw)
{
 try
 {
  bw.write("<hr>"); bw.newLine();
  bw.write("Kennung: " +this.kennung +"<br>"); bw.newLine();
  bw.write("Flurkarte Linienanfang: " +this.flurkarteAnf +"<br>");
  bw.newLine();
  bw.write("Punktnummer Linienanfang: " +this.punktnummerAnf +"<br>");
  bw.newLine();
  bw.write("Flurkarte Linienende: " +this.flurkarteEnde +"<br>");
  bw.newLine();
  bw.write("Punktnummer Linienende: " +this.punktnummerEnde +"<br>");
  bw.newLine();
 }
 // hier folgt der catch-Block
 ...
}
```

Ausgegeben werden jeweils die Kennung, die Flurkarten die Linienanfang und Linienende enthalten, sowie die zugehörigen Punktnummern.

Abbildung 37 zeigt die erstellte Datei Linie.htm.

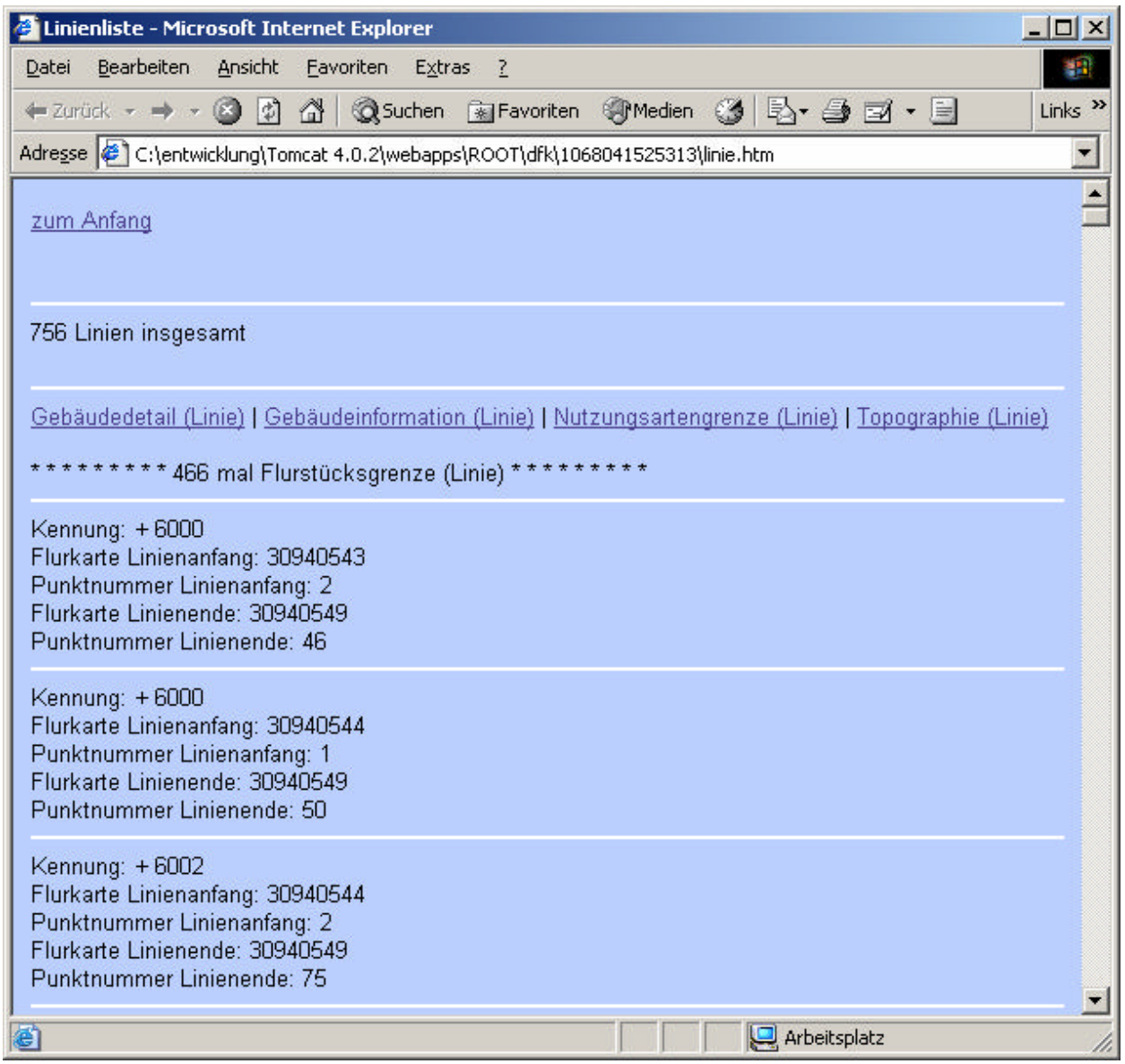

Abbildung 37: die generierte Datei Linie.htm

4.2.3 Generieren der XML-Datei

Die Daten aus den Objektlisten werden noch ein mal ausgelesen, diesmal aber in eine einzige XML-Datei geschrieben. Der Ablauf ist der gleiche, wie beim Schreiben der HTML-Dateien.

Für die Klassen Vordaten, Punktliste, Linienliste, Bogenliste, Gebäudeliste, Textliste und Symbolliste wurde jeweils eine Funktion zur Ausgabe der Listen in die XML-Datei geschrieben:

```
public void ausgabeXML(dateien DFKdateien)
```

Des weiteren wurden für die Klassen Punkt, Linie, Bogen, Text und Symbol jeweils eine Funktion zur Ausgabe eines einzelnen Objektes implementiert:

```
public void ausgabeXML(BufferedWriter bw)
```

Innerhalb der jeweiligen Funktion für die Ausgabe der Liste werden die einzelnen Objektlisten durchgegangen und für jedes Objekt die entsprechende Funktion zur Ausgabe eines einzelnen Objektes aufgerufen, die dann das jeweilige Objekt in die XML-Datei ausgibt.

Die Funktion für die Ausgabe der Punkteliste sieht in gekürzter Version beispielsweise folgendermaßen aus:

```
public void ausgabeXML(dateien DFKdateien)
{
 Enumeration enum;
 Punkt pkt;
 try
 {
  File ausgabeDatei = DFKdateien.getXML_datei();
  FileWriter fw = new FileWriter(ausgabeDatei,true);
  BufferedWriter bw = new BufferedWriter(fw);
  bw.write("<Punktliste>");
  bw.newLine();
  if(!TP_liste.isEmpty())
  {
   bw.write("<TP_liste>");
   bw.newLine();
   enum = TP_liste.elements();
   while (enum.hasMoreElements()==true)
   {
    pkt = (Punkt)enum.nextElement();
    pkt.ausgabeXML(bw);
   }
   bw.write("</TP_liste>");
   bw.newLine();
  }
  // es folgt auf die gleiche Weise die Ausgabe für die Listen
  // Katasterfestpunkte
  // Grenzpunkte (numerischer Grenznachweis)
  // Grenzpunkte (graphischer Grenznachweis)
  // sonstigen Grenzpunkte und der nicht abgemarkten Punkte
  ...
  bw.write("</Punktliste>");
  bw.newLine();
  bw.close();
 }
 // hier folgt der catch-Block
 ...
}
```

Zuerst wird der Knoten Punktliste geöffnet, danach durch eine if-Anweisung überprüft, ob die Liste für die Trigonometrischen Punkte überhaupt Elemente enthält. Ist dies der Fall, so wird der Knoten TP_liste geöffnet und die Liste innerhalb einer while-Schleife der Reihe nach durchgegangen. Für jedes erhaltene Punktobjekt wird die Funktion `ausgabeXML(bw)` aufgerufen, die den Punkt dann in die XML-Datei ausgibt. Nachdem die Liste abgearbeitet ist wird der Knoten TP_liste geschlossen. Anschließend werden auf die gleiche Art und Weise die übrigen Punktlisten für Katasterfestpunkte und für die verschiedenen Grenzpunkte abgearbeitet. Zum Schluss kann der Knoten Punktliste wieder geschlossen werden.

Die Funktion, die ein einzelnes Punktobjekt in die XML-Datei ausgibt sieht folgendermaßen aus:

```
public void ausgabeXML(BufferedWriter bw)
{
 try
 {
  bw.write("<Punkt>"); bw.newLine();
  bw.write("<flurkarte>" +this.flurkarte +"</flurkarte>");
  bw.newLine();
  bw.write("<punktnummer>" +this.punktnummer +"</punktnummer>");
  bw.newLine();
  bw.write("<rechtswert>" +this.rechtswert +"</rechtswert>");
  bw.newLine();
  bw.write("<hochwert>" +this.hochwert +"</hochwert>"); bw.newLine();
  bw.write("<kSymbol>" +this.kSymbol +"</kSymbol>"); bw.newLine();
  bw.write("<kArtEntstehungKoord>" +this.kArtEntstehungKoord
  +"</kArtEntstehungKoord>"); bw.newLine();
  bw.write("<letzterZugriff>" +this.letzterZugriff
  +"</letzterZugriff>"); bw.newLine();
  bw.write("</Punkt>"); bw.newLine();
 }
 // hier folgt der catch-Block
 ...
}
```

Zuerst wird ein Punktknoten geöffnet. Anschließend die Informationen innerhalb gleichnamiger Knoten ausgegeben:

- die Flurkarte innerhalb des Knotens `flurkarte`
- die Punktnummer innerhalb des Knotens `punktnummer`
- der Rechtswert innerhalb des Knotens `rechtswert`
- die Hochwert innerhalb des Knotens `hochwert`
- die Kennung des Symbols innerhalb des Knotens `kSymbol`
- die Kennung für die Art und Entstehung der Koordinaten innerhalb des Knotens `kArtEntstehungKoord`

? der Zeitpunkt des letzten Zugriffs innerhalb des Knotens `letzterZugriff`

Zum Abschluss wird der Knoten `Punkt` wieder geschlossen.

Werden die anderen Objektlisten, welche die Vordaten, Linien, Bögen, Gebäude, Texte und Symbole enthalten in die XML-Datei ausgegeben, so wird nicht wie bei den HTML-Dateien eine neue Datei angelegt, sondern die bereits verwendete Datei wieder geöffnet und darin weitergeschrieben.

Das Ergebnis zeigt die Abbildung 38:

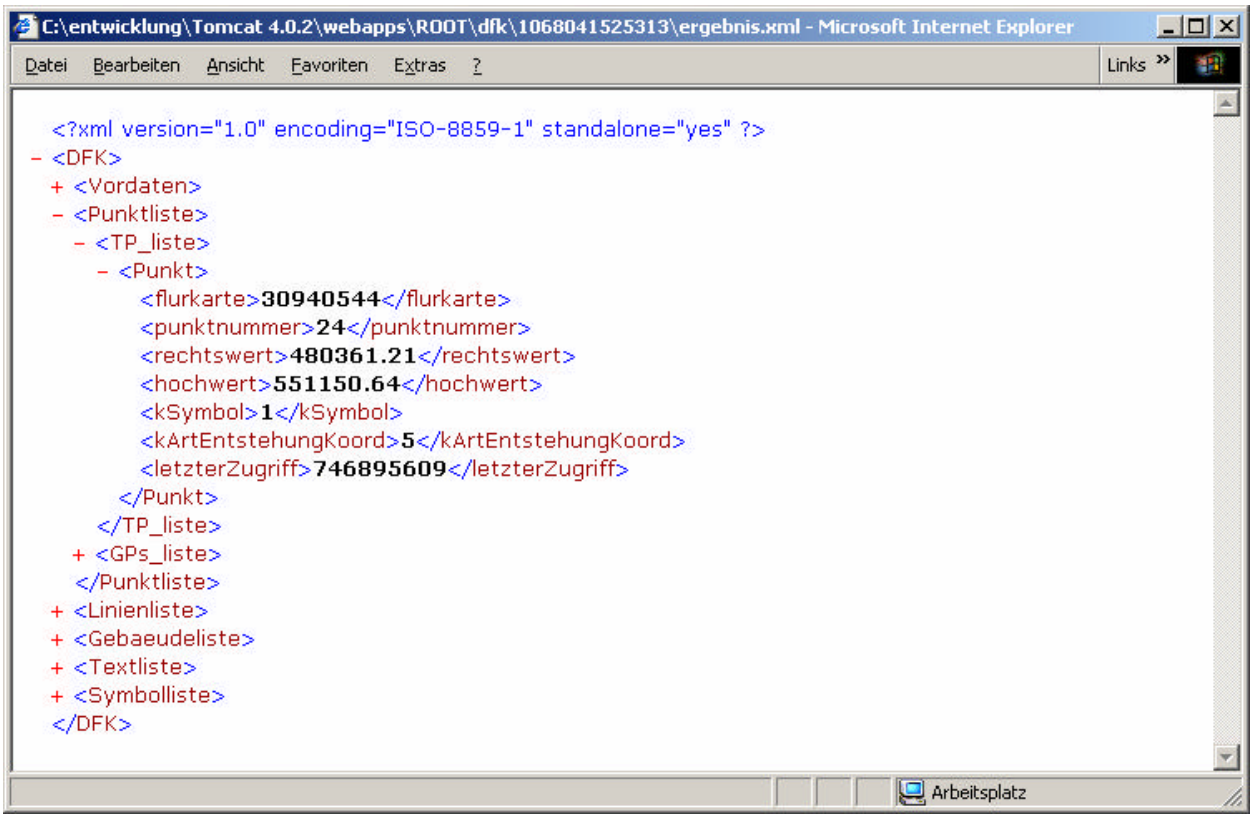

Abbildung 38: die generierte XML-Datei

Wie zu sehen ist werden nur Knoten für Listen angelegt, die auch Elemente besitzen. Es existiert zum Beispiel kein Knoten für die Bögen. Die einzelnen Knoten sind in der Abbildung geschlossen, da die Datei viel zu groß ist, um sie komplett anzuzeigen. Als Beispiel wurde der Knoten `TP_liste` geöffnet, da dieser nur einen Knoten `Punkt` beinhaltet.

4.2.4 Erstellen der XSLT-Datei zur Transformation der XML-Datei in eine SVG-Datei

Eine XSLT-Datei wird benötigt, um die zuvor generierte XML-Datei in eine SVG-Datei zu transformieren. Da die Datei einen Umfang von mehr als 2000 Zeilen hat, werden hier nur einige beispielhafte Inhalte vorgestellt.

Die Datei beginnt mit dem `<xsl:stylesheet>`-Element, wobei im `version`-Attribut die genutzte XSLT-Version angegeben wird. Außerdem werden die Namensräume für die verschiedenen Elemente deklariert. Das Ausgabeformat wird als XML festgelegt, da SVG auf XML basiert.

Daraufhin folgt die Deklaration einiger globaler Variablen, unter anderem:

```
<xsl:variable name="minX" select="/DFK/Vordaten/minRechtswert"/>
<xsl:variable name="minY" select="/DFK/Vordaten/minHochwert"/>
<xsl:variable name="maxX" select="/DFK/Vordaten/maxRechtswert"/>
<xsl:variable name="maxY" select="/DFK/Vordaten/maxHochwert"/>
```

Ihnen wird der minimale Rechtswert und Hochwert, sowie der maximale Rechtswert und Hochwert des Rechtecksfensters zugewiesen. Innerhalb dieses Rechtecksfensters befinden sich alle weiteren Elemente. Mit Hilfe der in diesen Variablen gespeicherten Werte können anschließend die benötigten Parameter für die ViewBox berechnet werden.

In folgendem Listing wird die Breite und die Höhe der ViewBox berechnet:

```
<xsl:variable name="viewBoxWidth" select="format-number(($maxX -
$minX) * $viewBoxRahmen, '#.00')"/>

<xsl:variable name="viewBoxHeight" select="format-number(($maxY -
$minY) * $viewBoxRahmen, '#.00')"/>
```

Bei `ViewBoxRahmen` handelt es sich auch um eine zuvor definierte Variable, die auf den Wert 1.1 gesetzt wurde. Der ermittelte Bereich in x- und y-Richtung wird mit diesem Wert multipliziert, damit auch Elemente welche direkt auf dem Rand der ViewBox liegen noch dargestellt werden können. Über die Angabe `format-number(..., '#.00')` wird erreicht, dass das Ergebnis der Berechnung auf zwei Nachkommastellen genau angegeben wird.

Anschließend wird durch `<xsl:template match="/DFK">` eine Template Rule erstellt. Da ein Element namens `DFK` nur einmal als Wurzelelement der XML-Datei existiert, wird der in der Template Rule enthaltene XSLT-Code genau einmal für die gesamte XML-Datei ausgeführt.

Danach wird der äußerste SVG-Bereich festgelegt. In SVG besteht die Möglichkeit neben dem äußersten SVG-Element noch weitere SVG-Objekte aufzuführen, was bei dieser Diplomarbeit dazu verwendet wurde um einen weiteren Bereich für die Kartendarstellung, sowie einen für das Menü der Karte zu definieren.

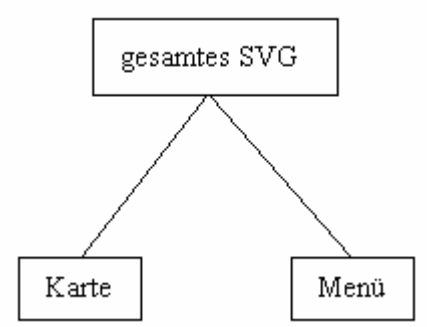

Abbildung 39: Aufteilung des gesamten SVG-Bereiches in die Karte und das Menü

Der äußerste SVG-Bereich umfasst den gesamten Bildschirmbereich und enthält Angaben, die sowohl das Zoomen und Verschieben des Kartenausschnittes über das Kontextmenü des SVG Viewers unterbinden, sowie die Anweisung Skripte in JavaScript nicht vom Browser, sondern von der Script-Engine des SVG

Viewers ausführen zu lassen. Des weiteren erfolgen hier schon sämtliche Formatdefinitionen für alle Elemente der Karte und des Menüs mittels Cascading Style Sheets. Zusätzlich werden hier auch die Festlegungen für die Symbole angegeben. Einerseits wurden Symbole zur Verwendung innerhalb der Karte erstellt, andererseits aber auch Symbole für die Schaltflächen des Menüs definiert. Die Symbole für die Karte wurden entsprechend dem Katolog der technischen Zeichnungen der Symbole und Punktausprägungen für Digitale Flurkarten (KtZ - DFK) angefertigt. Insgesamt wurden 30 der in diesem Katalog enthaltenen Symbole umgesetzt. Eine Liste der umgesetzten Symbole befindet sich in der Anlage 2.

Folgendes Listing zeigt beispielhaft die Definition des Symbols für Mischwald, 3er Gruppe:

```
<symbol id="SYM201" overflow="visible"
style="fill:none;stroke:green;stroke-width:0.2">
 <g transform="translate(0.0,-2.0)">
  <polyline points="-1.1,1.85 0.0,-1.85 1.1,1.85 2.5,1.85"/>
 </g>
 <g transform="translate(-3.25,2.85)">
  <path style="fill:white;stroke:green;stroke-width:0.2" d="M-
  0.7,1.0 A1.2 1.2 0 1 1 0.7 1.0 L2.1,1.0"/>
 </g>
 <g transform="translate(3.75,2.85)">
  <path style="fill:white;stroke:green;stroke-width:0.2" d="M-
  0.7,1.0 A1.2 1.2 0 1 1 0.7 1.0 L2.1,1.0"/>
 </g>
</symbol>
```

Wie zu sehen ist wird das Symbol aus den Grundelementen von SVG zusammengesetzt. Innerhalb des ersten `<g>`-Tags wird unter Verwendung des Elements `<polyline>` ein Nadelbaum erzeugt, innerhalb des zweiten und dritten `<g>`-Tag dagegen ein Laubbaum. Durch die Verschiebungsbeträge innerhalb der Transformation werden die einzelnen Baumsymbole an die gewünschte Stelle platziert.

Das erstellte Symbol ist in Abbildung 40 dargestellt.

Abbildung 40: Das Symbol für Mischwald

Über den Wert des Attributes `id` des Tags `<symbol>` kann das Symbol später referenziert und somit innerhalb der Karte verwendet werden.

Nachdem die Definitionen der Formate und die Festlegungen für die Symbole erfolgt sind, wird der SVG-Bereich für die Karte definiert. Die Breite und die Höhe des Kartenbereiches werden dabei in Zentimeter angegeben, damit daraus, in Kombination mit den Werten der zugehörigen ViewBox, jeweils der aktuelle Maßstab berechnet werden kann.

Die Parameter des Kartenbereiches, sowie die Anfangsparameter der ViewBox werden dabei mit Hilfe der zu Beginn der Datei definierten globalen Variablen festgelegt.

Folgendes Listing zeigt die Definition des Kartenbereiches:

```
<svg id="Karte" width="{$svgWidth}cm" height="{$svgHeight}cm"
 xmlns=http://www.w3.org/2000/svg
 viewBox="{$viewBoxX} {$viewBoxY} {$viewBoxWidth} {$viewBoxHeight}"
 preserveAspectRatio="xMidYMid meet">
```

Innerhalb des Kartenbereiches werden, wie in folgendem Listing dargestellt, verschiedene Template Rules für bestimmte Knoten der XML-Datei aufgerufen.

```
<xsl:apply-templates select="Gebaeudeliste"/>
<xsl:apply-templates select="Linienliste"/>
<xsl:apply-templates select="Bogenliste"/>
<xsl:apply-templates select="Textliste"/>
<xsl:apply-templates select="Symbolliste"/>
<xsl:apply-templates select="Punktliste"/>
```

Die Verwendung von `<xsl:apply-templates>` hat zur Folge, dass nach einer Template Rule gesucht wird, bei welcher der Wert des `match`-Attributes des `<xsl:template>`-Tags identisch ist mit dem Inhalt des `select`-Attributes von `<xsl:apply-templates>`.

Die Angabe von

```
<xsl:apply-templates select="Linienliste"/>
```

hat beispielsweise zur Folge, dass das Template, welches in folgendem Listing aufgeführt ist, ausgeführt wird.

```
<xsl:template match="Linienliste">
   <xsl:apply-templates select="gebaeudedetailLinie_liste"/>
   <xsl:apply-templates select="gebaeudeinformationLinie_liste"/>
   <xsl:apply-templates select="nutzungsartengrenzeLinie_liste"/>
   <xsl:apply-templates select="flurstuecksgrenzeLinie_liste"/>
   <xsl:apply-templates select="topographieLinie_liste"/>
</xsl:template>
```

Innerhalb dieses Templates werden, wieder über den Inhalt des `select`-Attributes, für verschiedene Elemente der XML-Datei, weitere Template Rules direkt aufgerufen.

Durch

```
<xsl:apply-templates select="nutzungsartengrenzeLinie_liste"/>
```

wird zum Beispiel folgendes Template aufgerufen.

```
<xsl:template match="nutzungsartengrenzeLinie_liste">
 <g id="layer_nutzungen" visibility="visible">
  <xsl:for-each select="Linie">
   <line class="nutzung">
    <xsl:variable name="rechtswert1"
```

```
    select="format-number(rechtswertAnf - $minX, '#.00')"/>
    <xsl:variable name="hochwert1"
    select="format-number($minY - hochwertAnf, '#.00')"/>
    <xsl:variable name="rechtswert2"
    select="format-number(rechtswertEnde - $minX, '#.00')"/>
    <xsl:variable name="hochwert2"
    select="format-number($minY - hochwertEnde, '#.00')"/>
    <xsl:attribute name="x1">
    <xsl:value-of select="$rechtswert1"/>
    </xsl:attribute>
    <xsl:attribute name="y1">
    <xsl:value-of select="$hochwert1"/>
    </xsl:attribute>
    <xsl:attribute name="x2">
    <xsl:value-of select="$rechtswert2"/></xsl:attribute>
    <xsl:attribute name="y2">
    <xsl:value-of select="$hochwert2"/></xsl:attribute>
   </line>
  </xsl:for-each>
 </g>
</xsl:template>
```

Durch die Angabe `<g id="layer_nutzungen" visibility="visible">` zu Beginn des Templates wird erreicht, dass alle folgenden Elemente zu einer Gruppe zusammengefasst werden. Das Attribut `visibility` ist für die gesamte Gruppe auf `visible` gesetzt, was zur Folge hat, dass alle Elemente der Gruppe sichtbar sind. Die darauf folgende Schleife `<xsl:for-each select="Linie">` sorgt dafür, dass der enthaltene XSLT-Code für jedes Element `Linie` der XML-Datei ausgeführt wird. Als nächstes wird durch `<line class="nutzung">` eine Linie innerhalb der zu erstellenden SVG-Datei erzeugt. Ihr wird die Darstellungsart `nutzung` zugewiesen, die innerhalb des Bereiches für die Cascading Style Sheets definiert ist.

Durch den Code

```
<xsl:variable name="rechtswert1"
select="format-number(rechtswertAnf - $minX, '#.00')"/>
<xsl:variable name="hochwert1"
select="format-number($minY - hochwertAnf, '#.00')"/>
<xsl:variable name="rechtswert2"
select="format-number(rechtswertEnde - $minX, '#.00')"/>
<xsl:variable name=" hochwert2"
select="format-number($minY - hochwertEnde, '#.00')"/>
```

werden verschiedene Variablen mit den Namen `rechtswert1`, `hochwert1`, `rechtswert2` und `hochwert2` erzeugt, deren Inhalte mit Hilfe der Inhalte der XML-Elemente `rechtswertAnf`, `hochwertAnf`, `rechtswertEnde` und `hochwertEnde` eines Linienknotens innerhalb der XML-

Datei berechnet werden. Genutzt werden auch hier wieder die globalen Variablen, welche zu Beginn der Datei definiert wurden.

Die Angaben

```
<xsl:attribute name="x1">
<xsl:value-of select="$rechtswert1"/>
</xsl:attribute>
<xsl:attribute name="y1">
<xsl:value-of select="$hochwert1"/>
</xsl:attribute>
<xsl:attribute name="x2">
<xsl:value-of select="$rechtswert2"/></xsl:attribute>
<xsl:attribute name="y2">
<xsl:value-of select="$hochwert2"/></xsl:attribute>
```

haben zur Folge, dass zu dem `<line>`-Tag neue Attribute hinzugefügt werden. Erstellt werden die Attribute `x1`, `y1`, `x2` und y2, deren Inhalte mit Hilfe der zuvor festgelegten Variablen angegeben werden. Diese Attribute sind notwendig, um den Anfangs- und den Endpunkt einer Linie festzulegen.

Durch die letzten Zeilen innerhalb des Templates wird noch die Linie, die Schleife, die Gruppierung und das Template selbst geschlossen.

Für jeden Elementtyp der XML-Datei existieren solche Templates innerhalb der XSLT-Datei. Da die anderen Templates komplizierter und vor allem wesentlich umfangreicher sind, als das hier vorgestellte, soll an dieser Stelle auf deren Abbildung und Erläuterung verzichtet werden.

Auf den Bereich für die Karte folgt noch der SVG-Bereich für das Menü, mit dem die Darstellung der erzeugten Karte modifiziert werden kann. Die einzelnen Funktionen sind ausführlich im Kapitel 4.3.3 beschrieben.

4.3 Die Ergebnisse

4.3.1 Darstellung der Ergebnisse des Umsetzens der DFK-Datei

Nachdem durch das Programm DFKviewer die komplette DFK-Datei eingelesen und verarbeitet wurde, sowie alle erforderlichen Dateien erstellt wurden, bekommt der Benutzer eine Zusammenfassung der Verarbeitung angezeigt. Diese kann wie folgt aussehen:

Abbildung 41: Darstellung der Ergebnisse im DFKviewer

Im oberen Teil sind alle generierten Dateien aufgelistet, welche durch Anklicken des entsprechenden Hyperlinks angezeigt werden können. Am Anfang stehen die HTML-Dateien, welche die Informationen zu den erzeugten Elementen beinhalten. Als letztes ist die XML-Datei aufgeführt, die als Ausgangsdatei für die Transformation zur Erzeugung der SVG-Datei verwendet wurde. In der Mitte der Abbildung befindet sich eine Übersicht, welche die Art und die Anzahl der verarbeiteten Objekte angibt. Unten ist der Link „zur Karte“ zu sehen, über den die erzeugte Karte, die auf der erstellten SVG-Datei basiert, aufgerufen werden kann.

4.3.2 Die erstellte Karte und das zugehörige Menü

4.3.2.1 Die erstellte Karte

Folgende Abbildung zeigt ein Beispiel für die Umsetzung einer DFK-Datei in eine zweidimensionale Vektorgrafik im SVG-Format.

Abbildung 42: Beispiel für eine erzeugte Karte

4.3.2.2 Aktionen beim Aufruf der Karte

Beim Öffnen der erstellten SVG-Datei wird eine JavaScript-Funktion ausgeführt. Diese Funktion speichert die ursprünglichen Parameter der ViewBox als globale Variablen und berechnet den aktuellen Maßstab. Anschließend wird der berechnete Maßstab in dem dafür vorgesehenen Feld des Menüs eingetragen.

4.3.2.3 Verschieben des Kartenausschnitts

Über das Menü der Karte kann der Kartenausschnitt in Nord-, Nordost- Ost-, Südost-, Süd-, Südwest-, West- und Nordwest-Richtung verschoben werden. Hierbei kann zwischen drei verschiedenen Verschiebungsbeträgen gewählt werden. Je nachdem in welchen Bereich der Schaltfläche geklickt wird, beträgt der Verschiebungsbetrag 10%, 50% oder 90% des Kartenauschnittes. Die einzelnen Abschnitte eines Pfeils werden optisch durch weiße Linien voneinander getrennt. Zusätzlich färben sich die Pfeile beim

Überfahren mit dem Mauszeiger unterschiedlich ein. Je heller der Pfeil wird, desto weiter verschiebt sich der Kartenausschnitt.

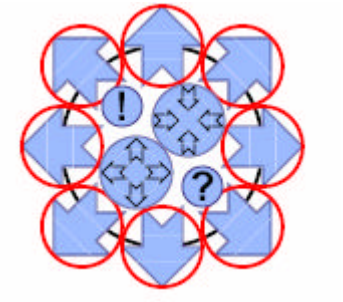

Abbildung 43: Schaltflächen zum Verschieben des Kartenausschnitts

4.3.2.4 Vergrößern und Verkleinern des Kartenausschnitts

Der Abbildungsmaßstab der Karte kann auf verschiedene Weise verändert werden. Die erste Möglichkeit ist, die entsprechende Schaltfläche für das Ein- bzw. Auszoomen anzuklicken. Dabei gibt es, wie auch bei den Schaltflächen für das Verschieben, drei verschiedene Stufen.

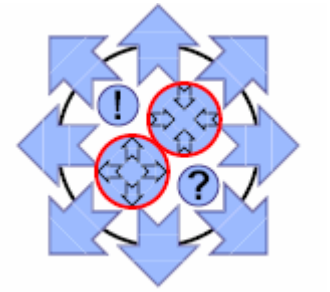

Abbildung 44: Schaltflächen zum Zoomen des Kartenausschnittes

Die zweite Möglichkeit, um den Abbildungsmaßstab zu verändern, besteht über das Popup-Menü, das erscheint sobald der Mauszeiger über die Anzeigefläche des aktuellen Maßstabes bewegt wird. Zur Auswahl stehen die Maßstäbe 1:100, 1:250, 1:333, 1:500, 1:1000, 1:1500, 1:2500 und 1:5000.

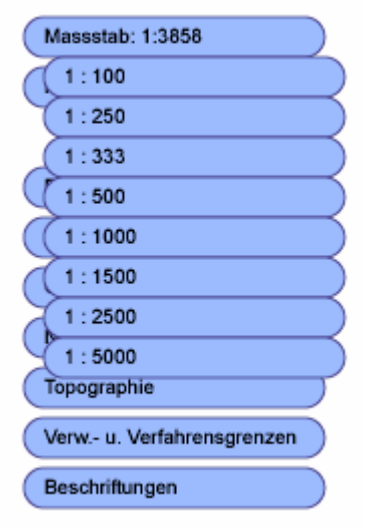

Abbildung 45: Popup-Menü für die Auswahl des gewünschten Maßstabs

Unabhängig davon, ob der Zoomvorgang über die Schaltflächen oder über das Popup-Menü ausgelöst wird, werden die Parameter für die ViewBox entsprechend der Auswahl neu gesetzt, der neue Maßstab aus den neuen Werten berechnet und anschließend in dem Feld für den aktuellen Maßstab angegeben.

4.3.2.5 Wiederherstellung des Originalausschnitts

Wird die Schaltfläche mit dem Ausrufezeichen angeklickt, so wird eine JavaScript-Funktion ausgeführt, welche die Parameter für die ViewBox auf die Orginalwerte setzt. Dies ist möglich, da diese Werte beim Aufruf der Karte als global Variablen gespeichert wurden.

Dadurch wird der Originalkartenausschnitt wiederhergestellt, der beim Öffnen des SVG-Dokumentes abgebildet wurde und der das gesamte Gebiet umfasst.

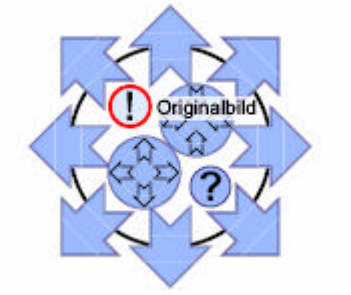

Abbildung 46: Schaltfläche zur Wiederherstellung des Originalbildes

4.3.2.6 Anzeigen der Vordaten

Wird die Schaltfläche mit dem Fragezeichen angeklickt, so werden die Vordaten angezeigt, welche die Informationen aus dem Start- und den Gebietssätzen enthalten. Über das Kreuz lässt sich das Fenster für die Vordaten wieder schließen.

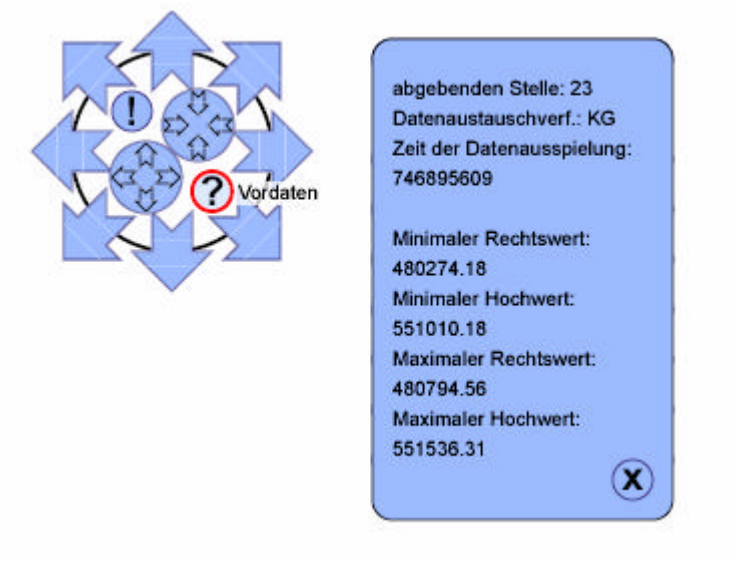

Abbildung 47: Schaltfläche zum Anzeigen der Vordaten

4.3.2.7 Ein- und Ausblenden von Ebenen

Da alle Elemente der Grafik innerhalb sogenannter `<g>`-Tags zu Gruppen zusammengefasst sind, lassen sich Eigenschaften jeweils für eine gesamte Gruppe ändern. Setzt man das Attribut `visibility` (Sichtbarkeit) auf `hidden` (versteckt), werden alle Grafikobjekte der entsprechenden Gruppe ausgeblendet. Um die Elemente wieder sichtbar zu machen, kann das Attribut wieder auf `visible` (sichtbar) gesetzt werden.

Abbildung 48: Die Schwarz-Weiß-Darstellung der Karte

Eine Schwarz-Weiß-Darstellung der Karte ist beispielsweise dann sinnvoll, wenn der Kartenausschnitt auf einem Drucker ausgegeben werden soll, der keine Farben wiedergibt. Durch Anklicken der entsprechenden Schaltfläche im Menü werden die Ebene mit der Farbfüllung für die Gebäude ausgeblendet. Zwar bleiben die Symbole weiterhin farbig, was sich jedoch bei einem Ausdruck in Schwarz-Weiß kaum störend auswirken dürfte.

Durch das Ein- und Ausblenden der einzelnen Ebenen lässt sich die Karte entsprechend den individuellen Wünschen des Nutzers anpassen. Besteht kein Interesse an der Darstellung der Gebäude, so können diese über das Menü ausgeblendet werden. Beim Überfahren der entsprechenden Schaltfläche mit der Maus erscheint ein Popup-Menü mit dem eine differenzierte Auswahl möglich ist. Bei den Gebäuden können beispielsweise die Ebenen für die Gebäudeinformationen, die Gebäudedetails und die Gebäudegrundrisse gesondert ein- und ausblendet werden.

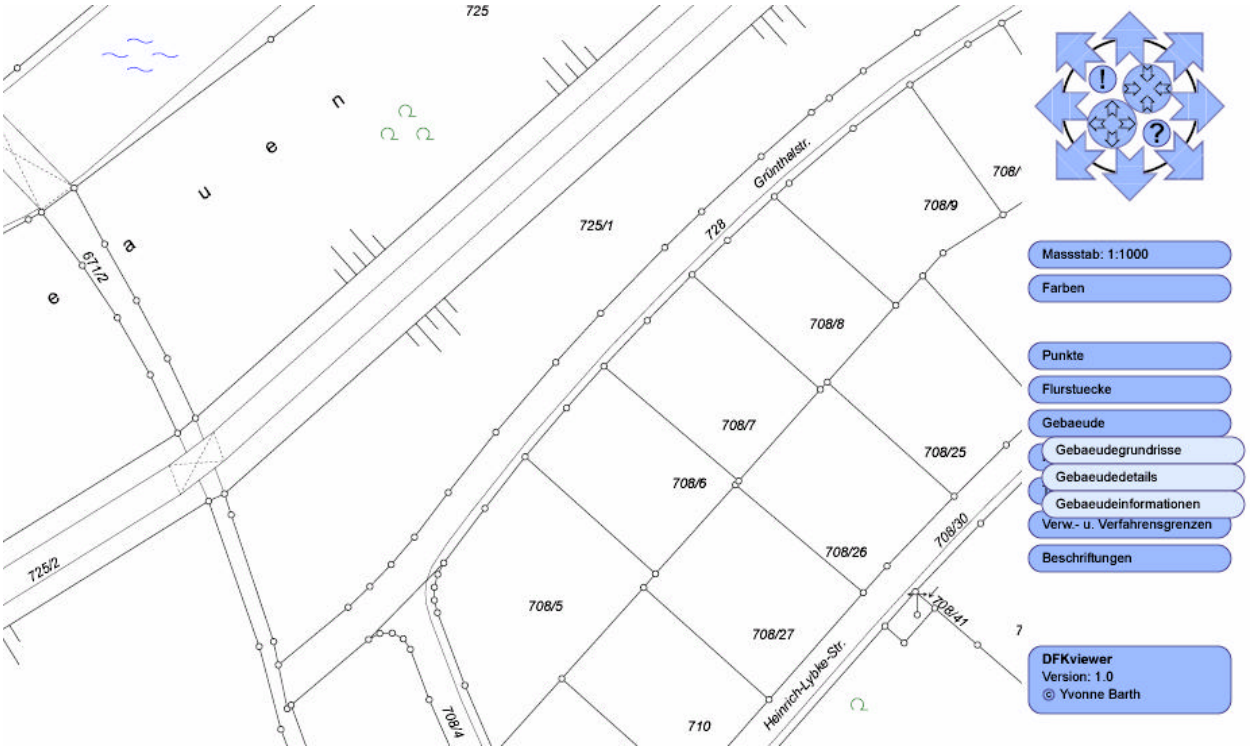

Abbildung 49: Die Darstellung mit ausgeblendeter Gebäudeebene

Beim Aufruf der Karte im Browser sind hierbei zunächst alle Ebenen sichtbar, wobei über das Menü für jedes Element der Karte die Möglichkeit zum Ein- und Ausblenden besteht.

5 Resümee

5.1 Zusammenfassende Schlussbetrachtung

Es wurde erstmalig eine servletbasierte Umsetzung von DFK nach SVG erstellt. Dabei wurden verschiedene aktuelle Technologien verwendet, die nach einer gewissen Einarbeitungszeit zu sehr guten Ergebnissen führten.

Die Anforderungen an die Hardware des Benutzers wurden dadurch gering gehalten, dass durch die Verwendung von Servlets und JavaServer Pages das Programm DFKviewer vollständig auf dem Server abläuft und deswegen keine Kapazitäten beim Benutzer verbraucht. Beim Anwender wird nur die fertige Karte angezeigt und die Interaktionen durchgeführt. Die Vorraussetzungen an die Hardware sollten also von allen modernen Rechnern erfüllt werden. Die Geschwindigkeit ist natürlich auch von der Größe der umzusetzenden DFK-Datei abhängig.

Durch die Verwendung von XSL könnte der DFKviewer ohne allzu großen Aufwand weiterentwickelt werden, um noch weitere Ausgabeformate außer SVG zu unterstützen. Dazu müsste einfach eine weitere XSLT-Datei geschrieben werden, mit der die erzeugte XML-Datei dann in das gewünschte Ausgabeformat transformiert wird. Denkbar wäre hier beispielsweise eine Ausgabe nach WML.

5.2 Danksagungen

An dieser Stelle möchte ich mich ganz besonders bei meinen Betreuern Herrn Prof. Dr.-Ing. Franz-Josef Behr und Herrn Prof. Dr.-Ing. Wolfgang Huep bedanken. Herr Prof. Dr.-Ing. Behr machte mich auf SVG und XSL aufmerksam, wodurch nach einigen Gesprächen das Thema dieser Diplomarbeit entstand. Während der Ausführung der Diplomarbeit stand er mir immer mit hilfreichen Tipps und Anregungen zur Seite, lies mir aber ansonsten die Möglichkeit selbstständig zu arbeiten.

Bedanken möchte ich mich auch bei Herrn Mailänder, der im letzten Semester seine Diplomarbeit zum Thema Visualisierung von ALK-Daten mittels SVG (Scalable Vector Graphics) erstellt hat und der mir erlaubt hat das Design seines Programms für meinen DFKviewer zu verwenden. Dadurch konnte ich in diesem Bereich Zeit sparen und mich den Transformationen mit XSLT widmen, was sonst wahrscheinlich den Zeitrahmen der Diplomarbeit gesprengt hätte.

Ein ganz spezieller Dank gilt auch meinen Eltern und meinem Freund Thomas, die mich während meines gesamten Studiums unterstützt haben.

Anhang 1: Die Dateien des Programmsystems DFKviewer

Die Dateien werden auf einer separaten CD-ROM abgegeben.

1. Definition der Seitenaufteilung

DFKviewer.jsp

2. Kopfbereich

kopf.jsp

3. Navigation durch die einzelnen Seiten

navigation.jsp

4. Startseite mit allgemeinen Informationen

startseite.jsp

5. Seite, welche die Voraussetzungen beschreibt

voraussetzungen.jsp

6. Seite, die ein Beispiel enthält

demonstration.jsp

7. Upload der DFK-Datei

dateiupload.jsp

8. Einlesen und Verarbeiten der DFK-Datei

dateiauslesen.java

9. Die Klasse Vordaten

Vordaten.java

10. Die Klasse Koordinate

Koordinate.java

11. Die Klasse Punkt

Punkt.java

12. Die Klasse Punktliste

Punktliste.java

13. Die Klasse Linie

Linie.java

14. Die Klasse Linienliste

Linienliste.java

15. Die Klasse Bogen

Bogen.java

16. Die Klasse Bogenliste

Bogenliste.java

17. Die Klasse Gebäudeliste

Gebäudeliste.java

18. Die Klasse Text_Symbol

Text_Symbol.java

19. Die Klasse Text

Text.java

20. Die Klasse Textliste

Textliste.java

21. Die Klasse Symbol

Symbol.java

22. Die Klasse Symbolliste

Symbolliste.java

23. Anlegen und Löschen des benötigten Verzeichnisses und der Dateien

dateien.java

24. CSS- Definitionen

formate.css

25. JavaScript-Funktionen für die Interaktionen der erstellten Karte

svg.js

26. Datei zur Transformation der XML-Datei in eine SVG-Datei

xml2svg.xsl

Anhang 2: Liste der umgesetzten Symbole

SYM1	Trigonometrischer Bodenpunkt jeder Art
SYM2	Trigonometrischer Hochpunkt jeder Art
SYM3	Katasterfestpunkt
SYM4	Pfeilerbolzen
SYM5	Mauerbolzen und Pegel
SYM10	Grenzstein
SYM52	Zugehörigkeitshaken
SYM60	Zuordnungspfeil (Flurstück)
SYM63	Pfeil für Treppe und Tiefgaragenzufahrt
SYM76	Gemarkungsgrenze, 3er Gruppe
SYM77	Gemarkungsgrenze, 2er Gruppe
SYM78	Gemarkungsgrenze, Einzelzeichen
SYM110	Grünland, 7er Gruppe
SYM130	Gartenland, 4er Gruppe
SYM200	Wald, Mischwald, 7er Gruppe
SYM201	Wald, Mischwald, 3er Gruppe
SYM210	Laubwald, 7er Gruppe
SYM211	Laubwald, 3er Gruppe
SYM212	Laubwald, Einzelzeichen
SYM220	Nadelwald, 7er Gruppe
SYM222	Nadelwald, Einzelzeichen
SYM242	Fliesspfeil (groß)
SYM250	Wasserfläche, 4er Gruppe
SYM251	Wasserfläche, Einzelzeichen
SYM271	Ödland/Unland, Einzelzeichen
SYM290	Böschungssymbol
SYM292	Böschungssymbol
SYM293	Böschungssymbol
SYM301	Kilometerstein, Kilometersäule
SYM309	Mast für Hochspannungsleitungen

Literaturverzeichnis

? **Hunter, Jason: Java Servlet Programmierung** (2002),
O'Reilly Verlag, ISBN 3-89721-282-X

? **Türeyenler, Sahin: Java ServerPages für Dummies** (2002),
mitp-Verlag, ISBN 3-8266-3009-2

? **Münz, Stefan und Nefzger Wolfgang: HTML 4.0 Handbuch** (1999),
Franzis' Verlag, ISBN 3-7723-7514-6

? **Seeboerger-Weichselbaum, Michael: Java/ XML** (2002),
bhv-Verlag, ISBN 3-8266-7209-7

? **Seeboerger-Weichselbaum, Michael: XSL** (2002),
bhv-Verlag, ISBN 3-8266-7208-9

? **Fibinger, Iris: SVG - Scalable Vector Graphics** (2002),
Markt + Technik Verlag, ISBN 3-8272-6239-9

Webseiten

? Fachhochschule Stuttgart - Hochschule für Technik
http://www.fht-stuttgart.de/

Servlets und JSP - JavaServer Pages

? JSP - Produktseite von SUN
http://java.sun.com/products/jsp/

? JSP - Portal
http://www.jsp-portal.de

? JSP - Forum
http://forum.java.sun.com/forum.jsp?forum=45

? Tag- und JavaBeans- Bibliotheken
http://jsptags.com/

? Links zum Thema JSP und Servlets
http://www.jspinsider.com

? Informationssammlung zu JSP
http://www.jspin.com

? FAQ - Liste (Frequently Asked Questions) zu JSP
http://www.jguru.com/faq/JSP

? Einführung zu JSP und Servlets
http://www.galileocomputing.de/openbook/javainsel3/javainsel_170020.html

? Klassen MultipartRequest und MultipartParser
http://www.servlets.com/cos/index.html

Tomcat

? Projekt Jakarta der Apache Software Foundation, beinhaltet Tomcat
http://www.jakarta.apache.org

HTML - Hypertext Markup Language

? W3C - Hypertext Markup Language
http://www.w3.org/MarkUp/

? SELFHTML - Hypertext Markup Language
http://selfhtml.teamone.de/html/index.htm

CSS - Cascading Style Sheets

? W3C - Cascading Style Sheets
http://www.w3.org/Style/CSS/

? SELFHTML - Cascading Style Sheets
http://selfhtml.teamone.de/css/index.htm

JavaScript

? SELFHTML - JavaScript
http://selfhtml.teamone.de/javascript/index.htm

XML

? W3C – Extensible Markup Language
http://www.w3.org/XML/

? SELFHTML - Extensible Markup Language
http://selfhtml.teamone.de/xml/index.htm

SVG - Scalable Vector Graphics

? Beispiele zu SVG
http://www.svg-site.de

? Unterschiede zwischen SVG und Flash
http://www.carto.net/papers/svg/comparison_flash_svg.html

? Interaktive SVG-Karte
http://www.carto.net.papers/svg/tuerlersee/

? SVG-Viewer von Adobe
http://www.adobe.com/svg/viewer/install/main.html

DFK - Datenaustauschformat der bayerischen Vermessungsverwaltung

? Beschreibung
http://www.geodaten.bayern.de/datri.html

? Testdaten
http://www.geodaten.bayern.de/download/dos/testdaten-dos.html

Erklärung

Die vorliegende Diplomarbeit wurde von mir zur Diplomprüfung im Wintersemester 2003/04 selbst verfasst und ohne fremde Hilfe angefertigt. Die benützten Hilfsmittel und Literaturquellen sind in den entsprechenden Verzeichnissen aufgeführt.

Stuttgart, den _____ _____

(Yvonne Barth)

Gesehen:

Erstprüfer und Betreuer: _____ _____

(Datum) (Prof. Dr.-Ing. Franz-Josef Behr)

Zweitprüfer: _____ _____

(Datum) (Prof. Dr.-Ing. Wolfgang Huep)