Seminar on XML-Based Markup Languages

SVG & SMIL
Überblick und Einführung

Georg Jahn
mail@gjahn.com

Betreuung:	Prof. Dr. Wolfgang May
Eingereicht am:	14.08.2013

Inhaltsverzeichnis

Anmerkung: Internetquellen und Formulierungen wie "*zum aktuellen Zeitpunkt*" beziehen sich auf August 2013.

1 SVG

Die stetig steigende Rechenkapazität in den Jahrzehnten vor der Jahrtausendwende führte zu einer ebenso ansteigenden Popularität der Vektorgrafik: Solche Grafiken enthalten nicht die Pixelabfolge wie sogenannte Rastergrafiken, sondern nur Beschreibungen der darzustellenden Figuren, und müssen deshalb für ihre Darstellung noch rechenzeitaufwändig gerastert werden. Da in den Achtzigern bestehenden Formate zum Austausch von Vektorgrafiken (*PS*[1], *WMF*[2], etc.) größtenteils inperformant oder proprietär waren, entwickelte sich die Idee, ein modernes, offenes und flexibles Format einzuführen.

So entstanden parallel zwei XML-basierte Sprachen, *PGML*[3] und *VML*[4], die von einflussreichen Firmen unterstützt wurden. Als beide zur Standardisierung beim W3C[5] eingereicht wurden, entschied sich das Gremium, beide Formate nicht einzeln fortzuführen, sondern vorteilhafte Aspekte beider Ansätze in einem neuen Standard fortzusetzen. So entstand die erste Version von *SVG*[6] (*Scalable Vector Graphics*), die ab 2001 offiziell vom W3C empfohlen wurde.

1.1 Überblick

SVG ist eine auf *XML 1.0*[7] basierende Sprache zur Beschreibung von Vektorgrafiken. Es kann in Form von SVG-Dokumenten mit der üblichen Endung `.svg` für sich alleine stehen oder über den XML-Namespace-Mechanismus in andere XML-Dokumente eingebunden werden. Um Speicherplatz zu sparen, existiert auch die mit *gzip* komprimierte Variante der SVG-Dokumente, die zumeist die Endung `.svgz` tragen.

Der vollständige Umfang der Sprache bietet u. a. folgende Elemente und Features:

Vektorelemente: Allgemeine Formen werden durch Pfade beschrieben, welche unter anderem Bézier-, Ellipsen- und kubische Kurvenabschnitte beinhalten können. Zur vereinfachten Handhabbarkeit und für eine höhere Abstraktion sind aber auch speziellere Primitive wie Rechtecke und Kreise direkt definierbar.

Textelemente: Schriftzüge können durch Referenzieren von externen Schriftarten erstellt werden. Texte können auch an Pfaden entlanglaufen.

Eingebundene Elemente: Externe Raster- wie Vektorgrafiken können referenziert werden und erscheinen dann als Teil der Vektorgrafik.

Styling: Für die meisten Elemente kann eine Vielzahl von Attributen festgelegt werden, um das genaue Aussehen (Füllung, Rahmen, etc.) zu bestimmen. Dabei können auch Farbverläufe und ein Opazitätskanal verwendet werden.

Transformationen: Beliebige lineare Transformationen der definierten Elemente, also insbesondere auch Rotationen und Scherungen sind möglich.

Clipping und Masken: Diese erweiterten Vektorgrafiktechniken erlauben es, Elemente mit anderen Elementen zu modifizieren und zu kombinieren.

Filtereffekte: Für den Rastervorgang steht eine Vielzahl an Optionen und Effekten, wie z. B. Unschärfe oder Schlagschatten, bereit; diese können für Teile der SVG-Grafik festgelegt werden.

Strukturierende Elemente: Zum Einbringen einer gewissen Semantik können Objekte gruppiert oder in Abhängigkeit von der Sprache des Benutzers ausgewählt werden.

Animationen: Angelehnt an die Syntax von *SMIL*[8] lassen sich die Elemente auch mit einfachen Animationen und Interaktivitäten versehen. Für komplexere Abläufe muss jedoch auf Skriptsprachen etc. zurückgegriffen werden.

1.1.1 Versionen

Zum aktuellen Zeitpunkt ist die Version *SVG 1.1*[9] am verbreitetsten. Zwar wurde bereits *SVG 1.2 Tiny*[10] definiert, da dass vollständige *SVG 1.2 Full*[11] vermutlich ausbleibt, konnte es sich jedoch nicht durchsetzen und wird wohl von den meisten Anwendern zugunsten von *SVG 2.0*[12] (voraussichtlich 2014) übergangen. Der modularisierte Aufbau von *SVG 1.1* erlaubt die Einführung von Profilen, die nur eine Teilmenge der Sprache definieren, um an die Möglichkeiten des Ausgabegeräts angepasst zu werden:

SVG 1.1 Tiny hat nur einen geringen Sprachumfang und ist damit für wenig leistungsfähige Geräte wie Mobiltelefone gedacht.

SVG 1.1 Basic definiert eine größere Teilmenge der Sprache und ist damit für mittelmäßig leistungsfähige Geräte, wie aktuelle Smartphones, gedacht.

SVG 1.1 Full enthält den kompletten Sprachumfang und ist für Computer gedacht. In der Realität ist dies das fast ausschließlich verwendete Profil.

1.1.2 Bezug zu anderen Standards

Da es sich um eine XML-basierte Sprache handelt, ist SVG zu vielen aus dem XML-Bereich bekannten Technologien kompatibel, dies verleiht der Sprache eine große Flexibilität. Die Kompatibilität mit *XML-Namespaces*[13] erlaubt es, zusätzliche Informationen in SVG-Dokumente zu integrieren und SVG direkt in andere XML-Dokumente einzubetten, wie es oft bei *XHTML*[14] verwendet wird. Der Standard *XLink*[15] wurde ausgewählt, um Links in SVG möglich zu machen. Weiterhin lässt sich SVG mit *CSS*[16] stylen und relevante, aus XHTML bekannte Styling-Eigenschaften wurden direkt übernommen. Auch in vielen weiteren Details erinnert SVG stark an XHTML mit ein paar aus SMIL übernommenen Elementen. SVG lässt sich ebenso auch mit *XSLT*[17] stylen und dadurch generieren, was eine Menge Anwendungsmöglichkeiten eröffnet, dazu folgen später ein paar Beispiele.

Schließlich beschreibt der SVG-Standard ein komplettes *Document Object Model*, sodass SVG auch Skriptsprachen zugänglich ist.

1.1.3 Grenzen und Nachteile

Das Format SVG ist nur für einzelne Grafiken gedacht, gesamte Dokumente aus mehreren Seiten müssen über mehrere SVG-Fragmente verteilt werden. Weiterhin ist der SVG 1.1 Standard mittlerweile ein bisschen älter und er unterstützt somit eine Menge Features nicht, die in aktuellen Grafikprogrammen Standard sind. So gibt es keine Ebenen (Layers), keinen Z-Index (Elemente werden in der Reihenfolge gezeichnet, in der sie vorkommen), keine variablen Pinselbreiten, etc. Eine oft vermisste Option sind automatische Textumbrüche in Schriftzügen, zwar wurden Modelle dafür in SVG 1.2 diskutiert, konnten sich aber noch nicht durchsetzen. Dazu muss wohl noch auf SVG 2.0 gewartet werden.

Auch sollte darauf geachtet werden, dass SVG zwar von vielen Viewern unterstützt wird, aber das längst nicht alle Viewer den kompletten SVG- Befehlssatz korrekt wiedergeben können. So scheitern viele Programme an Animationen, Farbverläufen, Opazität, Text an Pfaden und Filtereffekten.

1.2 Unterstützung und Dokumentation

Da SVG nun seit mehr als zehn Jahren existiert, konnte es sich weitgehend als ein IT-Standard durchsetzen, wird von vielen Firmen unterstützt und ist von verschiedenen Seiten gut dokumentiert worden. Die meisten Grafikprogramme wie *GIMP*[18], *Adobe Photoshop*, etc. können den kompletten SVG-Befehlssatz fehlerfrei rastern. Ebenso unterstützen die bekanntesten Office-Pakete das Einfügen von SVG-Grafiken; leider ist dies bei *OpenOffice* noch ein aktuell sehr neues Feature, dessen Implementation an vielen Ecken und Enden zu wünschen übrig lässt.

Besondere Wichtigkeit hat SVG im Bereich des Webs erhalten, ein großer Teil grafisch aufwendiger Webseiten sind durch SVG realisiert. Deshalb unterstützen seit einigen wenigen Jahren alle modernen Browser den Großteil des SVG-Sprachumfangs. Bekannteste Ausnahme ist der Internet Explorer vor Version 9, denn zuvor unterstützte Microsoft noch weiter den eigenen Standard VML, erst in Version 9 konnte der Browser SVG ohne zusätzliche Plugins darstellen. Ironischerweise stellte Microsoft bereits in Version 10 des Internet Explorers schließlich die Unterstützung für VML ein. Eine schöne Übersicht über die SVG-Fähigkeiten moderner Browser und ihrer Vorgänger ist auf `caniuse.com/svg` zusammengestellt. Die Website schätzt, dass zum aktuellen Zeitpunkt etwa 85 % der Internetnutzer mit einem SVG-fähigen Browser unterwegs sind.

1.2.1 Tools

Die beste frei verfügbare Authoring-Software für SVG ist wohl unumstritten *Inkscape*[19], mit dem fast der komplette SVG-Befehlssatz grafisch bearbeitet werden kann, größte Ausnahme sind nicht-statische Elemente wie Animationen. Andererseits greift Inkscape dem Standard in vielerlei Hinsicht voraus und speichert zusätzliche Informationen in die SVG, welche innerhalb von Inkscape zum Beispiel auch automatischen Textumbruch erlauben. Im kommerziellen Bereich ist vor allem *Adobe Illustrator*[20] zu nennen.

Auch einige Online-Editoren haben sich entwickelt, wie zum Beispiel *SVG-Edit*[21]. Insbesondere zu nennen ist der Side-By-Side-Editor *dabblet*[22], bei dem Code, Stylesheet und Ausgabe nebeneinander angezeigt werden, er wird später für Beispiele verwendet.

1.2.2 Dokumentation

Die beste ausführliche Referenz für SVG ist wohl die *W3C-Recommendation SVG 1.1* [9] selbst. Hier fehlt nur ein leichter Einstieg und ein Überblick über die praktischen Anwendungen. Für ein kurzes Tutorial möchte ich *W3Schools SVG* [23] oder aber Abschnitt 1.3 empfehlen. In Buchform verschafft *SVG Essentials* [24] einen gelungenen Überblick.

1.3 Praktische Einführung

Dieser Abschnitt führt auf praktische Weise in SVG ein. Dabei wird nicht auf alle Details, ja nicht einmal auf alle Sprachelemente eingegangen, es soll nur ein leichter Einstieg gewährleistet werden, da die verfügbaren Referenzen umfassend sind. Weiterhin sollen praktische Anwendungen angedeutet werden, wie vor allem im letzten Abschnitt über XSLT.

1.3.1 Das SVG-Fragment

Ein SVG-Fragment ist das Root-Node in einem SVG-Dokument und enthält alle grafischen und nicht-grafischen Elemente, die Teil der Vektorgrafik sind. Listing 1 zeigt, wie das SVG-Element mit seinen Attributen die wichtigsten Eigenschaften des Canvas festlegt und mit korrekter Doctype- und XML-Spezifikation zu einem validen SVG-Dokument wird. Als eine Datei mit der Endung `.svg` kann dies in SVG-Betrachtern angezeigt werden und führt zu der Ausgabe, wie sie in Abbildung 1 zu sehen ist.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
     "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3 <svg xmlns="http://www.w3.org/2000/svg" version="1.1" baseProfile="full"
     viewBox="0 0 1024 768" width="4cm" height="3cm">
4     <desc>Ein Kreis</desc>
5     <circle cx="512" cy="384" r="300"/>
6 </svg>
```

Listing 1: `circle.svg` pastie.org/8223121

Die erste Zeile legt fest, dass es sich um den Standard XML 1.0 handelt und legt das den Zeichensatz auf UTF8 fest, in der zweiten Zeile wird der spezielle Doctype von SVG und dessen DTD angegeben. Diese beiden Zeilen finden sich fast exakt so in jeder SVG-Datei wieder.

Das SVG-Element in der dritten Zeile deklariert den Default-Namespace als den Namespace von SVG, legt die Version fest und bestimmt das Profil (siehe dazu Abschnitt 1.1.1). Weiterhin wird hier die externe Größe der Vektorgrafik mit `width` und `height` festgelegt. Dabei können beliebige, auch aus CSS bekannte Einheiten (also zum Beispiel `px` für Pixel) verwendet werden. Das Attribut `viewBox` legt das Koordinatensystem fest, dass innerhalb des SVG genutzt werden soll (links, oben, rechts, unten), auch hierbei können beliebige Einheiten gewählt werden. Im Inneren wird lediglich mit `desc` eine Beschreibung der Grafik festgelegt und mit `circle` ein Kreis definiert.

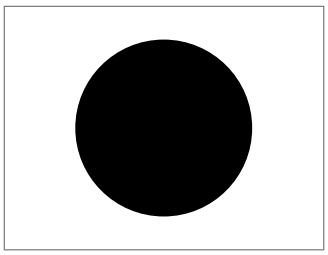

Abbildung 1: Die Darstellung von `circle.svg` im SVG-Betrachter. Die Datei ist ein valides SVG 1.1 Dokument, wie mit `validator.w3.org` überprüft werden kann.

1.3.2 Einbetten in XHTML

Durch die Verwendung von XML-Namespaces kann SVG auch direkt in andere XML-Dokumente eingebunden werden, besonders häufig wird dies im Web mit XHTML verwendet. Auf diese Weise wird die Vektorgrafik direkt als Element im XHTML-Dokument angezeigt. Listing 2 zeigt das Beispiel von oben in einem XHTML-Dokument.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
       "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
4      <head>
5          <title>Eingebundenes SVG</title>
6      </head>
7      <body>
8          <h1>Ein Kreis</h1>
9          <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
               baseProfile="full" viewBox="0 0 1024 768" width="400px"
               height="300px">
10             <circle cx="512" cy="384" r="300"/>
11         </svg>
12     </body>
13 </html>
```

Listing 2: `circle.html` pastie.org/8224178

1.3.3 SVG Primitive

Für die meisten Standard-Formen gibt es ein einfaches Element in SVG, komplexere Formen können mit Pfaden nachgezeichnet werden.

Rechteck: Über das `rect`-Element entsteht ein Rechteck, dass mit `x` und `y` positioniert werden kann und dessen Größe durch `width` und `height` bestimmt wird. Wenn die Attribute `rx` und `ry` gesetzt werden, wird das Rechteck entsprechend stark abgerundet.

Kreis, Ellipse: Hier stehen das `circle`- und das `ellipse`-Element zur Verfügung. Der Mittelpunkt wird über `cx` und `cy` positioniert und der Radius über `r` bzw. `rx` und `ry` festgelegt.

Linie: Das `line`-Element zeichnet eine Linie von `x1`, `y1` bis `x2`, `y2`.

Pfad: Das `path`-Element braucht nur das Attribut `d`, das die Pfadform festlegt. Es beinhaltet eine Liste von Einzelbuchstaben, die von Koordinatenpaaren gefolgt werden. So bedeutet `M 10 10`, die Zeichenmarke zur Position (10, 10) zu bewegen.

`M` $x\ y$	Die aktuelle Zeichenposition zu (x, y) bewegen.
`L` $x\ y$	Eine Linie von der aktuellen Zeichenposition nach (x, y).
`H` x	Eine horizontale Linie nach x ziehen.
`V` y	Eine vertikale Linie nach y ziehen.
`C` $x_1\ y_1\ x_2\ y_2\ x\ y$	Eine kubische Bézier-Kurve nach (x, y) ziehen, mit den durch (x_1, y_1) und (x_2, y_2) festgelegten Tangenten.
`A` $r_x\ r_y\ \alpha\ f_1\ f_2\ x\ y$	Ein elliptischer Bogenausschnitt nach (x, y) einer Ellipse mit Radien r_x, r_y, die um α rotiert wurde. Wenn $f_1 = 0$, wird so der kürzeste mögliche Bogenausschnitt gewählt, bei $f_1 = 1$ der längstmögliche. $f_2 = 1$ bedeutet, dass der Bogen in mathematisch positiver Richtung durchlaufen wird, bei $f_2 = 0$ entgegengesetzt.

Es gibt noch einige weitere Zeichenmodi und zu jedem Großbuchstaben (absolute Koordinaten) kann der jeweilige Kleinbuchstabe gewählt werden, sodass sich die folgenden Koordinaten auf die aktuelle Zeichenposition beziehen.

In Listing 3 ist ein Beispiel zu jedem der genannten Primitive, Abbildung 2 zeigt diese Vektorgrafik. Über den Link kann online damit experimentiert werden.

```svg
<svg ...>
    <rect x="20" y="20" width="50" height="50" rx="5"/>
    <circle cx="105" cy="45" r="25"/>
    <ellipse cx="75" cy="105" rx="55" ry="25"/>
    <line x1="140" y1="20" x2="140" y2="130"/>
    <path d="M 150 20 l 130 0 l -65 110 z"/>
</svg>
```

Listing 3: `primitives.svg` dabblet.com/gist/6200477

Abbildung 2: Die Darstellung von `primitives.svg`.

1.3.4 Styling

Damit die Elemente dann wirklich wie in Abbildung 2 erscheinen, muss ihr Aussehen noch genauer festgelegt werden. Dies geschieht entweder über verschiedene Attribute oder über das `style`-Attribut in CSS-Manier. Ein Rechteck mit dicker, roter Umrandung erreicht man also über zwei verschiedene Möglichkeiten:

```
<rect ... stroke-width="5px" stroke="red"/>
    <!-- oder aber ... -->
<rect ... style="stroke-width: 5px; stroke: red"/>
```

Farben können wie bei CSS als Hexadezimaldarstellung `#abcdef` angegeben werden, ansonsten sollten sich die Einheiten an der Einheit des Koordinatensystem des übergeordneten Elementes orientieren (beim `svg`-Element durch `viewBox` festgelegt). Es gibt eine große Vielzahl an Styling-Eigenschaften, aus CSS2 wurden zum Beispiel folgende Attribute übernommen: `font-*`, `letter-spacing`, `text-decoration`, `cursor`, `display`, `overflow`, `visibility`, `direction`, `unicode-bidi`, `word-spacing`, `clip`, `color`.

Besonders häufig benötigt man `fill` zum Einstellen der Füllfarbe, `fill-opacity` (Wertebereich $[0, 1]$) zum Bestimmen der zugehörigen Opazität; `stroke`, `stroke-opacity` beschreiben die Färbung der Linien und mit `stroke-width` und `stroke-dasharray` gibt man das genaue Aussehen vor.

Weiterhin kann das Styling auch durch eine eingebundene CSS-Datei erfolgen. Ist die SVG in XHTML eingebunden, so kann das zugehörige CSS auch die SVG formatieren. Dabei können die SVG-Elemente wie in HTML gewohnt auch mit einer `id` ausgezeichnet und angesprochen werden. Hiermit ergeben sich zum Beispiel interaktive Möglichkeiten:

```
<svg ...>
    <path id="bubblered" d="..."/>
    <path id="bubblegreen" .../>
    <path id="bubbleblue" .../>
</svg>
...
<style>
    #bubblered, #bubblegreen, #bubbleblue {
        stroke: black;
        stroke-width: 8;
    }

    #bubblered:hover { fill: #902020; }
    #bubblegreen:hover { fill: #209020; }
    #bubbleblue:hover { fill: #202090; }
</style>
```

Listing 4: `bubbles.html` dabblet.com/gist/6201066

1.3.5 Textelemente

Schriftzüge werden über das `text`-Element eingebunden. Dieses wird über `x` und `y` positioniert und wie aus CSS bekannt mit den `font-*` Eigenschaften formatiert. Listing 5 zeigt ein Beispiel dazu.

```
1  <text x="355" y="130" font-family="Arial" font-size="100"
       font-style="italic">
2      ?!
3  </text>
4
5  <defs>
6      <path id="liesl" .../>
7  </defs>
8  <text font-family="Arial" font-size="9">
9      <textPath xlink:href="#liesl">
10         Das Gänseliesel vor dem alten Rathaus ist als Brunnenfigur seit
               1901 das Wahrzeichen der Universitätsstadt Göttingen...
11     </textPath>
12 </text>
```

Listing 5: `svgtext.svg` dabblet.com/gist/6201198

In Zeile 8 bis 12 wird eine erweiterte Funktion genutzt: Mit `textPath` kann ein Text entlang eines Pfades fließen. Auf den Pfad wird dabei mit dem `href`-Attribut des XLink-Standards hingewiesen. Dieser wird in einem `def`-Bereich deklariert. In einem solchen Bereich können alle normalen SVG-Elemente genutzt werden, dabei werden sie jedoch nicht angezeigt, sondern können lediglich von außerhalb über ihre `id`-Eigenschaft angesprochen werden. Auch im nächsten Abschnitt wird eine Verwendung von `def` aufgezeigt.

1.3.6 Transformationen

Alle grafischen Elemente können beliebigen linearen zwei-dimensionalen Transformationen unterzogen werden. Inklusive von Translationen handelt es sich somit im Allgemeinen um eine sechsparametrige Transformation, die wiefolgt als Matrix-Vektor-Multiplikation verstanden werden kann:

$$\begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha & \gamma & \Delta_x \\ \beta & \delta & \Delta_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$$

Dabei bezeichnet (p_x, p_y) den ursprünglichen Punkt und (p'_x, p'_y) sein transformiertes Äquivalent. Die Parameter Δ_x und Δ_y der Matrix verursachen eine Translation, die restlichen eine Linearkombination der Koordinaten.

Solche und speziellere Transformationen können über das `transform`-Attribut angewendet werden, Listing 6 zeigt einige Beispiele auf: Mit `translate(`Δ_x, Δ_y`)` kann ein Element verschoben werden, `scale(`γ`)` skaliert ein Element um den Faktor γ. `rotate(`$\alpha[, c_x, c_y]$`)` rotiert ein Element um den Winkel α um das Zentrum (c_x, c_y). Die Schlüsselwörter

`skewX(`α`)` und `skewY(`α`)` verursachen eine Scherung um den Winkel α in X- bzw. Y-Richtung. Mehrere solcher Transformationen können hintereinander geschrieben werden (sie werden von rechts nach links ausgeführt) oder direkt eine Transformationsmatrix mit `matrix(`$\alpha, \beta, \gamma, \delta, \Delta_x, \Delta_y$`)` angegeben werden.

```
<defs>
    <path id="bubble" d="..."/>
</defs>

<use xlink:href="#bubble" transform="translate(0, 0)"/>
<use xlink:href="#bubble" transform="translate(220, 20) scale(0.8)"/>
<use xlink:href="#bubble" transform="translate(440, 0)
   rotate(70, 100, 100)"/>
<use xlink:href="#bubble" transform="translate(0, 200) skewX(15)"/>
<use xlink:href="#bubble" transform="translate(220, 200) skewY(15)"/>
<use xlink:href="#bubble" transform="matrix(0.9, -0.1, 0.1, 1.2, 440,
   200)"/>
```

Listing 6: `transform.svg` dabblet.com/gist/6204782

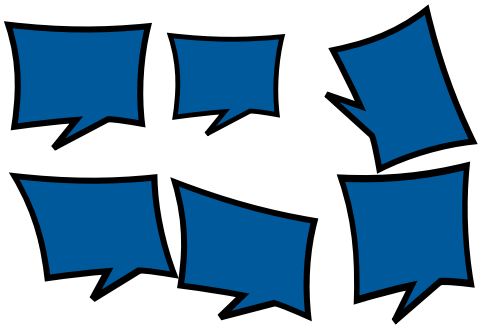

Abbildung 3: Die Darstellung von `transform.svg`.

In obigem Beispiel wurde auch das `use`-Tag zum ersten Mal verwendet: Es erzeugt eine exakte Kopie des mit `xlink:href` referenzierten Elements, dass zumeist in einer `defs`-Umgebung steht, um nicht selbst angezeigt zu werden. Natürlich kann das `transform`-Attribut auch auf andere Elemente, wie den Pfad selbst, Grafiken und Schriftzüge angewendet werden.

1.3.7 Externe Elemente

Externe Grafiken können mit dem `image`-Element eingebunden werden. Listing 7 veranschaulicht die erforderlichen Attribute.

```
<image x="100" y="50" width="290" height="100"
   xlink:href="Pfad/zum/Bild.png">
```

Listing 7: `images.svg` dabblet.com/gist/6204891

Als Pfade kommen Internet-URLs, absolute und relative Pfade infrage. Laut Standard müssen sich zumindest PNG-, JPG- und SVG-Dateien einbinden lassen, aber viele Viewer unterstützen auch weitere Dateitypen.

1.3.8 Gruppen und `switch`

Das Element `g` kann beliebige weitere grafische Elemente beinhalten und gruppiert diese. Dies hat den Vorteil, dass gemeinsame Style-Eigenschaften und Transformationen angewendet werden können und die Gruppe als ganzes referenziert werden kann. Auch fügt eine Gruppierung der Struktur der SVG einen semantischen Inhalt hinzu.

Aus einer `switch`-Gruppe wird nur das erste Tochterelement angezeigt, dessen Voraussetzungen, die mit Attributen spezifiziert werden, erfüllt sind. So kann zum Beispiel mit dem `systemLanguage`-Attribut die Internationalisierung einer SVG-Grafik erreicht werden, Listing 8 zeigt ein Beispiel.

```
<switch>
    <g systemLanguage="de">
        <image width="50" height="40"
            xlink:href=".../Flag_of_Germany.svg"/>
        <text x="60" y="25">Hallo Welt!</text>
    </g>
    <g systemLanguage="en">
        <image width="50" height="40"
            xlink:href=".../Flag_of_the_United_Kingdom.svg"/>
        <text x="60" y="25">Hello world!</text>
    </g>
    <g systemLanguage="fr">
        <image width="50" height="40"
            xlink:href=".../Flag_of_France.svg"/>
        <text x="60" y="25">Bonjour tout le monde!</text>
    </g>
</switch>
```

Listing 8: `groups.svg` dabblet.com/gist/6205193

1.3.9 Filtereffekte

Wie bereits erwähnt, bietet die SVG die Möglichkeit, Effekte für den Rastervorgang festzulegen. Dies soll nur mithilfe eines kurzen Beispiels, das einen Gaußschen Weichzeichner beim Rastern einsetzt, demonstriert werden. Solche Effekte eignen sich gut, um Schlagschatten zu Formen hinzuzufügen.

```
<defs>
    <path id="bubble" .../>

    <filter id="blur">
        <feGaussianBlur in="SourceAlpha" stdDeviation="8"/>
```

```
6       </filter>
7   </defs>
8
9   <use xlink:href="#bubble" filter="url(#blur)"
        transform="translate(5,5)"/>
10  <use xlink:href="#bubble"/>
```

Listing 9: `shadow.svg` dabblet.com/gist/6205280

1.3.10 XSLT

In den vorherigen Abschnitten wurde ein Großteil häufig benutzter Elemente von SVG vorgestellt. In diesem letzten Abschnitt soll nun noch praktische Anwendungen des Vorgestellten im Rahmen von XSLT aufgezeigt werden. Als Datengrundlage, die es zu visualisieren gilt, dient die XML-Version der Datenbank `mondial` [25].

Als erstes Beispiel soll ein Balkendiagramm, das die Bevölkerung der europäischen Länder visualisiert, erstellt werden. Dazu gibt sich das XSLT-Stylesheet in Listing 10 zwei Variablen vor, `plotwidth` und `plotheight`, welche die Größe des Diagramms spezifizieren. Es wird zunächst eine HTML-Struktur generiert, welche eine SVG der gewünschten Größe direkt einbindet. Zeile 12-19 bestimmen die größte Bevölkerungszahl unter den Ländern aus der Datenbank, dies wird zur Skalierung der Balken gebraucht. Danach kann ein `for-each`-Konstrukt in den Zeilen 24-33 je den Balken (ein `rect`-Element) und eine Beschriftung (ein per Transformation gedrehtes `text`-Element) gruppiert generieren. In der leicht erweiterten Version wird dies noch mit einer Beschriftung der genauen Bevölkerungszahl und einigen Hover-Effekten versehen, sodass sich insgesamt Abbildung 4 ergibt.

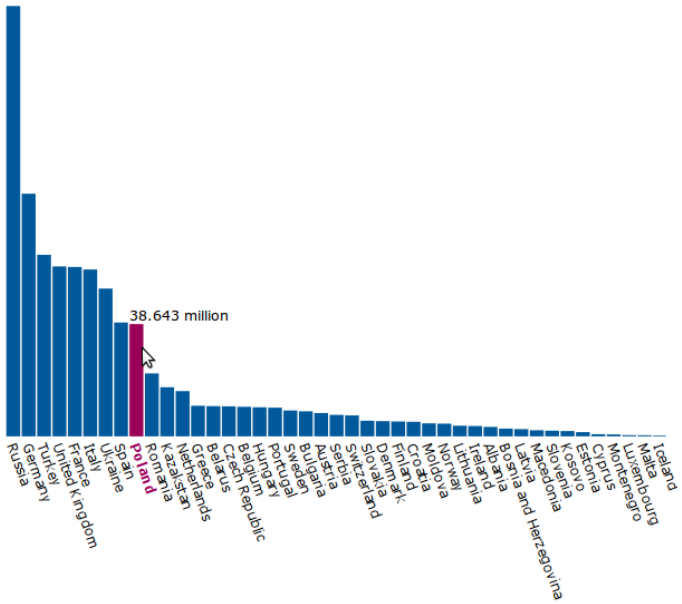

Abbildung 4: Die Darstellung von `mondial-europe.xml` mit `diagram.xsl`.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="...">
    <xsl:template match="/">
        <html> ...
        <body>
            <xsl:variable name="plotwidth" select="800"/>
            <xsl:variable name="plotheight" select="600"/>

            <svg ... width="{$plotwidth}px" height="{$plotheight}px">
                <xsl:variable name="maxpopulation">
                    <xsl:for-each select="mondial/country/population">
                        <xsl:sort data-type="number" order="descending"/>
                        <xsl:if test="position()=1">
                            <xsl:value-of select="."/>
                        </xsl:if>
                    </xsl:for-each>
                </xsl:variable>
                <xsl:variable name="scalex" select="($plotwidth) div
                    (count(mondial/country)+4)"/>
                <xsl:variable name="scaley"
                    select="(($plotheight)-200)-($scalex)) div
                    ($maxpopulation)"/>

                <xsl:for-each select="mondial/country">
                    <xsl:sort select="population" order="descending" />
                    <g>
                        <rect x="{($scalex)*position()}"
                            y="{(($maxpopulation)-population)*($scaley)+
                            ($scalex)}" width="{($scalex)*0.9}"
                            height="{population*($scaley)}"/>
                        <text x="0" y="0"
                            transform="translate({($scalex)*position()},
                            {($plotheight)-192}) rotate(70)" ...>
                            <xsl:value-of select="name"/>
                        </text>
                        ...
                    </g>
                </xsl:for-each>
            </svg>
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Listing 10: `diagram.xsl` pastie.org/8227596

Um noch ein komplexeres Anwendungsbeispiel zu demonstrieren, soll die Bevölkerungszahl nach Kontinenten aus `mondial.xml` summiert und in einem Kreisdiagramm dargestellt werden. Hierbei ergab sich schnell das Problem, dass XSLT keine trigonometrischen Funktionen unterstützt. Die erste Idee, diese über selbst-definierte Funktionen in einer Taylor-Reihe zu approximieren, scheiterte daran, dass XSLT erst ab Version 2.0 eigene Funktionen unterstützt, aktuelle Browser kommen aber nur mit XSLT 1.0 zurecht. So musste schließlich die Taylor-Approximation 12. Ordnung für Sinus und Cosinus mehrfach direkt in den Code übernommen werden, diese Stellen sind aber aus Listing 11 gekürzt worden.

Weitere Schwierigkeit ist in diesem Fall, dass nicht nur die Bevölkerungssumme für einen Kontinent, sondern auch die Bevölkerungssumme aller Kontinente davor bestimmt werden muss, um ein Tortenstück korrekt zu positionieren. Aus diesen beiden Angaben wird ein Winkel `alpha` und ein Winkel `beta` bestimmt, zwischen denen das Tortenstück mithilfe eines Pfades gezeichnet wird. In einer ausführlicheren Version, die über den Link heruntergeladen werden kann, werden noch einige Hover-Effekte mittels CSS hinzugefügt und es ergibt sich Abbildung 5.

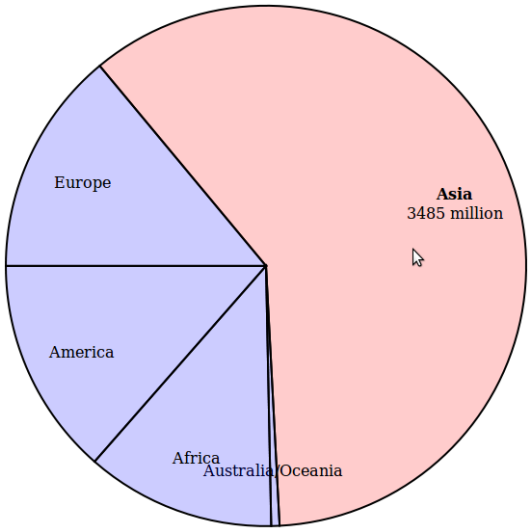

Abbildung 5: Die Darstellung von `mondial.xml` mit `piechart.xsl`.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="...">
3      <xsl:template match="/">
4          <html> ...
5          <body>
6              <xsl:variable name="plotwidth" select="600"/>
7
8              <svg ... width="{$plotwidth}px" height="{$plotwidth}px">
9                  <xsl:variable name="totalpopulation"
                       select="sum(mondial/country/population)"/>
10
```

```
11              <xsl:for-each select="mondial/continent">
12                  <g>
13                      <xsl:variable name="population"
                            select="sum(/mondial/country[encompassed[1]/
                            @continent=current()/@id]/population)"/>
14                      <xsl:variable name="populationsum"
                            select="sum(/mondial/country[encompassed[1]/
                            @continent=current()/preceding-sibling::continent
                            /@id]/population)"/>
15
16                      <xsl:variable name="alpha"
                            select="(($populationsum) div
                            ($totalpopulation) - 0.5)*3.14159*2"/> ...
17                      <xsl:variable name="beta"
                            select="((($population)+($populationsum)) div
                            ($totalpopulation) - 0.5)*3.14159*2"/> ...
18                      <xsl:variable name="gamma"
                            select="((($population)*0.5+($populationsum))
                            div ($totalpopulation) - 0.5)*3.14159*2"/> ...
19
20                      <path d="M {($plotwidth)*0.5} {($plotwidth)*0.5} L
                            {($plotwidth)*0.5*(1+($cosalpha)*0.9)}
                            {($plotwidth)*0.5*(1+($sinalpha)*0.9)} A
                            {($plotwidth)*0.5*0.9} {($plotwidth)*0.5*0.9} 0
                            {round(($population) div ($totalpopulation))} 1
                            {($plotwidth)*0.5*(1+($cosbeta)*0.9)}
                            {($plotwidth)*0.5*(1+($sinbeta)*0.9)} z"/>
21
22                      <text x="{($plotwidth)*0.5*(1+($cosgamma)*0.7)}"
                            y="{($plotwidth)*0.5*(1+($singamma)*0.7)+8}">
23                          <xsl:value-of select="name"/>
24                      </text>
25                  </g>
26              </xsl:for-each>
27          </svg>
28      </body>
29      </html>
30    </xsl:template>
31 </xsl:stylesheet>
```

Listing 11: piechart.xsl pastie.org/8227627

2 SMIL

Als die erste Version der *Synchronized Multimedia Integration Language* (*SMIL*)[8] 1998 zur W3C Recommendation wurde und einige kompatible Player erschienen, war der direkte Konkurrenz-Standard *Adobe Flash*[26] bereits seit einem Jahr auf dem Markt und erschien gerade in der dritten Version. An Anwendungsmöglichkeiten mangelte es nicht, wie die weite Bekanntheit von Flash-Dateien demonstriert, aber SMIL bekam aufgrund der stets etwas langsameren Entwicklung nur wenig Anhänger. Zwar unterstützen wenig später viele bekannte Video-Player das SMIL-Format, aber die Konkurrenz ist dem bereits voraus und kann aufgrund der Streaming-Fähigkeit bequem in den meisten Browsern betrachtet werden.

So wurde SMIL in der folgenden Zeit zumeist nur für kleinere Projekte angewendet, zum einen da es kaum Authoring-Software gab und zweitens weil sich die Sprache als zu unflexibel für viele komplexe Anwendungen herausstellte. Das erste Mal zur Anwendung im großen Stile kam SMIL bei der *HD DVD*, bei der die Menüstrukturen über SMIL realisiert wurden. Für diese Anwendung hatte die Sprache gerade die richtige Komplexität und hätte sich vermutlich bewährt, wenn sich nicht ein weiterer Nachfolger der DVD, die *Bluray*, durchgesetzt hätte. Seitdem erhält das Format wenig Beachtung und mittlerweile gibt es kaum noch SMIL-kompatible Software. Dennoch ist das Erbe der Sprache zum Beispiel in dem Animationsbefehlssatz von SVG zu sehen.

2.1 Überblick

SMIL ist eine XML-basierte Auszeichnungssprache zur Erstellung interaktiver multimedialer Präsentationen aller Art. Sie erlaubt es, externe Medien zu einem Ganzen zu komponieren, insbesondere unter Beachtung von zeitlicher Vorgaben und Interaktivitätsmöglichkeiten. Ähnlich wie SVG kann SMIL entweder in Form einer Datei mit der üblichen Erweiterung `.smil` auftreten oder aber über den Namespace-Mechanismus direkt in andere XML-Dokumente eingebunden werden. Zu den Features von SMIL zählen insbesondere:

Synchronisation: Im Fokus des Standards stehen die Möglichkeiten, die eingebundenen Komponenten in eine zeitliche Abfolge zu bringen und zu synchronisieren. Dabei können auch Benutzerinteraktionen in das Timing mit einbezogen werden.

Automatische Inhaltswahl: Anhand von Leistungsfähigkeit des Anzeigegeräts, Benutzersprache, etc. können automatisch die richtigen Inhalte ausgewählt werden.

Layouting: Jede SMIL-Datei definiert ein statisches Layout, in welches die Medien eingeordnet werden. Diese Vorgehensweise führt zu einer geringen Komplexität der Darstellungs-Software aber auch zu einer geringen Flexibilität des Formats.

Animationen: Attribute von Elementen können zwecks Animation zeitlich variiert werden. Da SMIL aber nur wenige Elemente enthält, bei denen dies sinnvoll ist, stellte sich dieser Ansatz erst im Kontext von SVG als erfolgreich heraus.

Übergänge: Sogenannte Transitions erlauben es, die Übergänge zwischen den einzelnen Medienkomponenten optisch ansprechend zu animieren.

2.1.1 Versionen

Die *Synchronized Multimedia Group* (*SYMM*) des W3C entwickelte seit Mitte der 90er an dem Standard und 1998 wurde die erste Version von SMIL offiziell Empfehlung des W3C. Wie bei SVG wurde in der zweiten Version die Verwendung des Namespace-Mechanismus hinzugefügt und eine Modularisierung der Komponenten der Sprache hinzugefügt. Dies führte im Falle von SMIL dazu, dass sich die einzelnen Implementation hinsichtlich der unterstützten Komponenten stark unterschieden.

Seit 2008 ist *SMIL 3.0* [27] die Empfehlung des W3C. Im Jahre 2012 wurde jedoch die SYMM geschlossen und es findet somit keine Weiterentwicklung des Standards mehr statt.

2.1.2 Nachteile und Konkurrenz

Eingie der Nachteile von SMIL wurden bereits erwähnt: Die fehlende Layout-Flexibilität und die langsame Entwicklung. Die Medien entwickelten sich um die Jahrtausendwende so schnell, dass man sich heute beim Betrachten einer SMIL-Präsentation unweigerlich an die Neunziger erinnert fühlt. Konkurrenzformate erkannten diese Trends schneller und bezogen entsprechende Technologien schnell ein. Die Kasten-Vorstellung, die SMIL vom Layout hat, und die Vorstellung, dass sich komplexe Benutzerführungen auch ohne imperative Programmierung und allein deklarativ gut beschreiben ließen, erwiesen sich als nicht zeitgemäß.

Weiterhin gab es eine Menge Konkurrenzformate, HTML zusammen mit JavaScript war allgemeiner und erlaubte quasi alle Features von SMIL zu modellieren. Aber auch viele deutlich spezialisierte Formate wie *MS PowerPoint*, *Flash* und erweiterte Video-Formate, die mehrere Ton- und Untertitelspuren enthalten konnten, führten dazu, dass SMIL wenig Anwendung fand.

2.1.3 Unterstützung und Dokumentation

Tools zum Bearbeiten von SMIL-Dokumenten sind fast gar nicht mehr zu finden, lediglich solche, die vermutlich eher zu Forschungszwecken als für die reale Anwendung gedacht waren. Offenbar kann wegen des *HD DVD*-Standards auch noch einige kommerzielle Software wie *Adobe Dreamweaver* in SMIL exportieren. Anders sieht es bei Abspiel-Software aus, neben dem SMIL-eigenen *Ambulant*-Player [28] können noch viele moderne Player wie der *RealPlayer* das Format abspielen.

Als Dokumentation ist vor allem die Empfehlung *SMIL 3.0* [27] des W3C und eine 2008 im Springer-Verlag veröffentlichte Referenz [29] zu nennen.

2.2 Praktische Einführung

Ziel dieses Abschnitts ist es, eine kurze Einführung in die Sprache zu geben. Dabei geht es nicht darum, jedes Konstrukte zu jedem Feature zu behandeln, sondern das Look&Feel der Sprache zu beschreiben und auf solche Elemente hinzuweisen, die sich als sinnvoll herausgestellt haben und in andere Sprachen übernommen wurden.

2.2.1 Ein einfaches SMIL-Dokument

Ähnlich wie eine HTML-Datei gliedert sich ein SMIL-Dokument in einen `head`- und einen `body`-Teil, die durch entsprechende Tags eingeschlossen werden. Im ersten Teil werden allgemeine Definitionen abgelegt, während sich im zweiten der wirkliche Ablauf und Inhalt befindet. Listing 12 zeigt eine einfache SMIL-Datei, die leicht abgeändert aus dem Paket um den Ambulant-Player übernommen ist.

```
1  <smil>
2      <head>
3          <layout>
4              <rootlayout width="248" height="300" backgroundcolor="blue" />
5              <region id="a" top="20" left="64" />
6              <region id="b" top="120" left="20"/>
7          </layout>
8      </head>
9      <body>
10         <par>
11             <img src="http://www.contentnetworking.com/smil/hello.jpg"
                   region="a" begin="0s" dur="6s"/>
12             <img
                   src="http://www.contentnetworking.com/smil/earthrise.jpg"
                   region="b" begin="2s" end="8s"/>
13             <audio src="http://www.contentnetworking.com/smil/hello.wav"
                   begin="4s"/>
14         </par>
15     </body>
16 </smil>
```

Listing 12: `helloworld.smil`

Wichtig ist, dass zunächst ein `layout` definiert wird, dass mit dem `rootlayout`-Element beschreibt, wie die Anzeigefläche aussehen soll. Die danach folgenden benannten `region`-Elemente beschreiben Stellen, an denen Medien eingebunden werden können, wie dies zum Beispiel auch in Zeile 11 und 12 geschieht. Dort werden mit dem `img`-Tag Bilder an die benannten Stellen eingebunden.

Sehr wichtig für SMIL ist das `par`-Element: Alle Tochterelemente werden parallel abgespielt, enthalten sie weitere Zeitangaben, so beziehen sich diese auf einen synchronen Startzeitpunkt. Im Gegensatz dazu werden die Tochterelemente des `seq`-Elements sequentiell abgespielt. Es ist zu beachten, dass ein Bild-Element keine inhärente zeitliche Länge

besitzt, sondern z. B. mit dem `dur`-Attribut mit einer solchen versehen wurde. Im Gegensatz dazu braucht für das `audio`-Element nur ein Startzeitpunkt festgelegt werden.

2.2.2 Medienelemente

Es können Bilder, Audio-Dateien, Animationen, Videos und Text-Spuren eingebunden werden. Listing 13 beschreibt die Syntax der jeweiligen Elemente:

```
1 <img src="bild.png" region="r1" begin="2s" dur="5s"/>
2 <audio src="ton.wav" begin="2s" title="Willkommen!"/>
3 <animation src="ani.svg" region="r2" begin="0s"/>
4 <video src="clip.mpg" region="r3" begin="0s" clipBegin="4s"
     clipEnd="60s"/>
5 <textstream src="untertitel.rt" region="r4"/>
```

Listing 13: Ausschnitt aus `media.smil`

Hervorzuheben ist hier die Möglichkeit, mithilfe von `clipBegin` und `clipEnd` nur einen Ausschnitt aus einem Video auszuwählen. Liegt ein Medienelement in mehreren Versionen vor, von denen die richtige ausgewählt werden soll, so verwendet man das `switch`-Element, welches sich als so gute Idee herausstellte, dass es später von SVG übernommen wurde:

```
1 <switch>
2     <audio src="a_espanol.wav" systemLanguage="es"/>
3     <audio src="a_francais.wav" systemLanguage="fr"/>
4     <audio src="a_deutsch.wav" systemLanguage="de"/>
5     <audio src="a_english.wav"/>
6 </switch>
```

Listing 14: Ausschnitt aus `switch.smil`

In der ursprünglichen Versionen konnte jedoch anhand von einer deutlich größeren Auswahl von Attributen das korrekte Element bestimmt werden, es gibt Attribute zur Übertragungsrate (`systemBitrate`), zu ausgewählten Untertiteln (`systemCaptions`), zur Auflösung (`systemScreenSize`) und zu vielem mehr.

2.2.3 Animationen

Zwar war über das *Document Object Model* theoretisch auch die Möglichkeit gegeben, mit einer imperativen Programmiersprache auf SMIL zuzugreifen; dennoch war es das Paradigma, so etwas zu vermeiden und Animationen so weit wie möglich deklarativ umzusetzen. Der Befehlssatz dazu wurde komplett in SVG übernommen und dient dort noch heute zu einfachen Animationen. Listing 13 demonstriert einige Möglichkeiten mit einem sich drehenden, blinkenden Rechteck, das einen roten Rand bekommt, sobald man mit der Maus darüber fährt.

Dazu wird das `animate`-Tag verwendet, mit dem beliebige zahlwertige Attribute animiert werden können; sie durchlaufen innerhalb der durch `dur` festgelegten Zeit linear den Wertebereich, der durch `from` und `to` spezifiziert wird. Mit `animateTransform` kann speziell das `transform`-Attribut aufgrund seiner komplexeren Syntax animiert werden.

Schließlich kann auch mit dem `set`-Tag einige imperative Programmierung vermieden werden: Hiermit können auch Attribute animiert werden, die nicht zahlwertig sind, dazu ist ein neuer Wert und der Zeitbereich anzugeben, in dem der Wert gesetzt werden soll.

```
1 <svg xmlns="http://www.w3.org/2000/svg" width="300" height="300">
2     <g transform="translate(150,150)">
3         <rect x="-50" y="-50" width="100" height="100" fill="#00599b"
              stroke-width="5px">
4             <animate attributeType="CSS" attributeName="opacity" from="1"
                  to="0" dur="5s" repeatCount="indefinite" />
5             <animateTransform attributeName="transform"
                  attributeType="XML" type="rotate" from="0" to="360"
                  begin="0s" dur="2s" repeatCount="indefinite"/>
6             <set attributeName="stroke" to="red" begin="mouseover"
                  end="mouseout"/>
7         </rect>
8     </g>
9 </svg>
```

Listing 15: Ausschnitt aus `animation.svg` dabblet.com/gist/6225906

So zeigt sich an diesem Beispiel besonders schön, dass “Informatik-Experimente” wie SMIL zwar auch fehlschlagen können, aber dennoch nicht ohne Wirkung bleiben: Aus alten Fehlern wird in neuen Projekten gelernt und gelungene Elemente können auch übernommen werden. So braucht die Forschung auch solche oft nicht längerfristig verwendeten Sprachen, um Stück für Stück zu einer besseren Lösung zu gelangen.

A Quellen

SVG-Logo, Titelseite: Entnommen von der SVG-Website[6].

SMIL-Logo, Titelseite: Entnommen von der SMIL-Website[8].

B Referenzen

[1] Adobe Press, *PostScript Language Reference Manual*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition (1985), URL `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`

[2] *WMF – Windows Meta-File*, URL `http://msdn.microsoft.com/en-us/library/cc250370.aspx`

[3] *PGML – Precision Graphics Markup Language*, URL `http://www.w3.org/TR/1998/NOTE-PGML-19980410`

[4] *VML – Vector Markup Language*, URL `http://www.w3.org/TR/NOTE-VML`

[5] *W3C – World Wide Web Consortium*, URL `http://www.w3.org`

[6] *SVG – Scalable Vector Graphics*, URL `http://www.w3.org/Graphics/SVG/`

[7] *XML – Extensible Markup Language*, URL `http://www.w3.org/TR/REC-xml/`

[8] *SMIL – Synchronized Multimedia Integration Language*, URL `http://www.w3.org/AudioVideo/`

[9] *SVG 1.1 – Recommendation*, URL `http://www.w3.org/TR/SVG/`

[10] *SVG 1.2 Tiny – Recommendation*, URL `http://www.w3.org/TR/SVGTiny12/`

[11] *SVG 1.2 Full – Working Draft*, URL `http://www.w3.org/TR/SVG12/`

[12] *SVG 2.0 – Working Draft*, URL `http://www.w3.org/TR/SVG2/`

[13] *XML Namespaces – Recommendation*, URL `http://www.w3.org/TR/xml-names/`

[14] *XHTML 1.0 – Recommendation*, URL `http://www.w3.org/TR/xhtml1/`

[15] *XML Linking Language 1.1 – Recommendation*, URL `http://www.w3.org/TR/xlink11/`

[16] *CSS – Cascading Style Sheets*, URL `http://www.w3.org/TR/CSS`

[17] *XSL Transformations – Recommendation*, URL http://www.w3.org/TR/xslt

[18] *GIMP – GNU Image Manipulation Program*, URL http://www.gimp.org/

[19] *Inkscape – Open Source SVG Editor*, URL http://inkscape.org/

[20] *Adobe Illustrator*, URL http://www.adobe.com/de/products/illustrator.html

[21] *SVG-Edit – Ein Online-SVG-Editor*, URL http://svg-edit.googlecode.com/svn/trunk/editor/svg-editor.html

[22] *Dabblet.com – Ein Online XHTML Side-By-Side Editor*, URL http://dabblet.com

[23] *W3Schools – SVG Tutorial*, URL http://www.w3schools.com/svg/

[24] J. D. Eisenberg, *SVG Essentials*, O'Reilly, 1st edition (2002)

[25] W. May, *Information Extraction and Integration with* Florid*: The* Mondial *Case Study*, Technical Report 131, Universität Freiburg, Institut für Informatik (1999), available from http://dbis.informatik.uni-goettingen.de/Mondial

[26] *Adobe Flash – Dokumentation*, URL http://www.adobe.com/support/documentation/de/flash/

[27] *SMIL 3.0 – Recommendation*, URL http://www.w3.org/TR/SMIL3/

[28] *Ambulant Player – Website*, URL http://www.ambulantplayer.org/

[29] D. C. Bulterman, L. Rutledge, *SMIL 3.0*, Springer, 2nd edition (2008)